



**SAMS**  
**Освой самостоятельно**

# Программирование р Microsoft® **Access 2002**

Пол Киммел



за **24**  
**часа**

**SAMS**

**Освой**

**самостоятельно**

Программирование  
для Microsoft®

**Access 2002**

за **24**  
часа



**SAMS**  
***Teach Yourself***

---

**Microsoft®**

# **Access 2002 Programming**

***in* 24  
Hours**

---

**Paul Kimmel**

**SAMS**

*201 West 103rd St., Indianapolis, Indiana, 46290 USA*

**SAMS**

**Освой самостоятельно**

Программирование  
для Microsoft®

**Access 2002**

**за 24  
часа**

**Пол Киммел**



Издательский дом "Вильямс"  
Москва « Санкт-Петербург » Киев  
2003

ББК 32.973.26-018.2.75

К40

УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *С.Н. Тригуб*

Руководитель проекта *В.В. Александров*

Перевод с английского и редакция канд.физ.-мат.наук *Е.Н. Дериевой*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:  
info@williamspublishing.com, http://www.williamspublishing.com

**Киммел, Пол.**

**К40** Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа. : Пер. с англ. — М. : Издательский дом "Вильямс", 2003. — 480 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0453-6 (рус.)

Система управления базами данных Access давно заняла прочную позицию в современном мире информационных технологий и завоевала признание приверженцев простых и эффективных программных решений. Самая свежая из версий системы, Access 2002, поддерживает разнообразные инструменты программирования — от традиционных средств ODBC и SQL до новейших объектных протоколов ActiveX Data Objects (ADO). Наибольшая степень гибкости и переносимости кода реализуется при использовании Visual Basic for Applications — уникальной по мощности, простоте и универсальности среды программирования, служащей неотъемлемой частью большинства программных продуктов из состава пакета Microsoft Office XP. Эта книга будет полезна в первую очередь новичкам, поскольку с ее помощью читатель-непрофессионал сможет ознакомиться с основами программирования и получить исчерпывающие ответы на интересующие вопросы. Данное издание станет ценным пособием для всех, кто стремится освоить передовые информационные технологии.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Sams Publishing.

Authorized translation from the English language edition published by Sams Publishing Copyright © 2002

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2003

ISBN 5-8459-0453-6 (рус.)

ISBN 0-672-32098-3 (англ.)

© Издательский дом "Вильямс", 2003

© Sams Publishing, 2002

# Оглавление

Введение	22
<b>Часть I. Азбука программирования в Access</b>	<b>23</b>
1-й час. Новинки Access 2002	25
2-й час. Познакомимся с VBA	39
3-й час. Как программа работает с данными	51
4-й час. Последовательность действий и выполнение вычислений	69
<b>Часть II. Основы конструирования программ</b>	<b>85</b>
5-й час. Программирование управляющих структур	87
6-й час. Управление базами данных	105
7-й час. Расширенные типы данных	123
<b>Часть III. Использование ресурсов Access</b>	<b>139</b>
8-й час. Декомпозиция задач	141
9-й час. Работа с макросами	159
10-й час. Как использовать готовые решения	179
<b>Часть IV. Определение типов данных. Использование массивов и коллекций</b>	<b>197</b>
11-й час. От сложного к простому: создание собственных типов данных	199
12-й час. Управление данными переменного объема	209
13-й час. Коллекции данных	227
<b>Часть V. Программирование и базы данных Access</b>	<b>237</b>
14-й час. Стиль программирования: „Что такое хорошо, и что такое плохо“	239
15-й час. ADODB — ваш верный помощник	251
16-й час. Применение языка SQL	273
<b>Часть VI. Работа над ошибками</b>	<b>295</b>
17-й час. Отладка кода	297
18-й час. Обработка ошибок во время выполнения программы	313
<b>Часть VII. Проектирование интерфейса пользователя</b>	<b>327</b>
19-й час. Создание экранных форм	329
20-й час. Как связывать Web-страницы с базами данных	355
<b>Часть VIII. Объектно-ориентированное программирование в Access</b>	<b>379</b>
21-й час. Основы программирования классов	381
22-й час. Совершенствование типов данных	399
23-й час. Настройки Access	413
24-й час. Управление информацией о контактах Outlook	427
Приложение. Ответы	441
Предметный указатель	467

# Содержание

Введение	22
Часть I. Азбука программирования в Access	23
<b>1-й час. Новинки Access 2002</b>	25
Область задач облегчает навигацию	26
Индивидуальная настройка меню и панелей инструментов	26
Элемент меню Развернуть	26
Индивидуальная настройка меню и панелей инструментов	27
Назначение гиперссылок интерфейсным объектам	29
Проверка правописания	30
Возможности редактирования	31
Команды Отменить и Вернуть	31
Помощник Office	31
Введите вопрос	31
Дополнительные возможности буфера обмена	32
Распознавание речи	32
Совместная работа над документами	32
Настройка NetMeeting	33
Соединение установлено. Что дальше?	33
Создание узлов	34
Удаление персональной информации	34
Знакомство с объектами ActiveX Data Objects	34
Новые объекты для обеспечения безопасности баз данных — ADOX	35
Поддержка хранимых процедур	35
Программные объекты	35
Связь Web-страниц с базами данных	36
Access Projects	36
Резюме	36
Вопросы и ответы	37
Задания	37
Тесты	37
Упражнения	38
<b>2-й час. Познакомимся с VBA</b>	39
Знакомство с Access VBA	39
Служебные слова Access VBA	40
Операторы и операнды Access	42
Данные: сведения, известные программе	43
А теперь все вместе	45
Арифметические операторы	45
Операторы сравнения	46
Логические операторы	47

Операторы конкатенации строк	48
Шаг вперед	49
Резюме	49
Вопросы и ответы	49
Задания	50
Тесты	50
Упражнения	50
<b>3-й час. Как программа работает с данными</b>	<b>51</b>
О компьютерной памяти	52
Объявление переменных	52
Переменные формата Dim	53
Переменные формата ReDim	53
Постоянные значения	54
Глобальные переменные	55
Присваивание данных и вычисления	55
Будьте точны и до конца откровенны	56
Тип данных Variant	57
Неявное определение переменных	57
Всплывающие подсказки в окне редактора модуля Access	58
Тестирование кода	59
Отладка с использованием всплывающих подсказок	60
Режим просмотра локальных переменных	61
Как пользоваться окном просмотра результатов	62
Добавление объекта наблюдения	63
Редактирование объекта наблюдения	64
Режим Quick Watch	64
Режим непосредственного исполнения кода	65
Стек вызовов	66
Резюме	67
Вопросы и ответы	67
Задания	68
Тесты	68
Упражнения	68
<b>4-й час. Последовательность действий и выполнение вычислений</b>	<b>69</b>
О понятии "уравнение"	69
Операнды уравнений	70
Правила "дорожного движения"	72
Классификация операторов по числу операндов	73
Применение арифметических операторов	74
Операторы сравнения	74
Логические операторы в истинном свете	76
Побитовые операции	79
Сложение строк	81
Кто здесь главный?	81
Специальные операторы	82

Резюме	83
Вопросы и ответы	84
Задания	84
Тесты	84
Упражнения	84
<b>Часть II. Основы конструирования программ</b>	<b>85</b>
<b>5-й час. Программирование управляющих структур</b>	<b>87</b>
Кто следит за порядком на дороге	88
Расширение выражения If с помощью Else	90
Вложенные условные конструкции	90
Будьте аккуратны	91
Как избавиться от вложенных условных выражений	92
Хотите немного покружиться?	94
Другие виды условных циклов	95
Итерационные циклы	97
Прямая и обратная итерации	99
Функции Lbound и Ubound	99
Циклическая обработка данных	100
Средства прерывания циклов	101
Использование функции Switch	102
Резюме	103
Вопросы и ответы	103
Задания	103
Тесты	104
Упражнения	104
<b>6-й час. Управление базами данных</b>	<b>105</b>
Совместимость Access 2002 с Access 2000	105
Новые форматы файлов, обеспечивающие повышенную производительность	106
Средства сжатия и восстановления	106
Создание базы данных	106
Создание базы данных	107
Построение таблицы	108
Создание программного модуля	110
Создание таблицы посредством программного кода	111
Управление таблицей из среды приложения	114
Базы данных: основные понятия	114
Получение информации о структуре таблицы	115
Циклический ввод данных в таблицу	117
Динамический поиск информации в таблице	119
Резюме	120
Вопросы и ответы	121
Задания	121
Тесты	121
Упражнения	122

<b>7-й час. Расширенные типы данных</b>	<b>123</b>
Знакомство со средствами OLE Automation	124
Добро пожаловать в мир объектов!	125
Некоторые полезные объекты	125
Сравнение ADO и DAO	126
Использование объектов ADODB	127
Connection	129
Recordset	130
Использование объектов ADOX	133
Catalog	133
Резюме	136
Вопросы и ответы	137
Задания	137
Тесты	137
Упражнения	138
<b>Часть III. Использование ресурсов Access</b>	<b>139</b>
<b>8-й час. Декомпозиция задач</b>	<b>141</b>
Механизмы создания процедур	142
Первая и последняя строки процедуры	143
Правила именования процедур	143
Аргументы — как их звать-величать	143
Определение типов аргументов	144
Как строить функции	147
Заповеди программирования	148
Заповедь 1: "Заключай повторяющиеся строки кода твоего в функции"	148
Заповедь 2: "Выражайся лаконично"	149
Заповедь 3: "Ограничивай число аргументов твоих"	149
Заповедь 4: "Проясняй мысль твою посредством квалификаторов"	149
Заповедь 5: "Обуславливай решения твои"	149
Заповедь 6: "Не ленись комментировать код твой"	150
А теперь все хором!	150
Создание таблицы	151
Импорт текста из файла	151
Поиск записи	153
Использование реестра Windows	154
Резюме	156
Вопросы и ответы	156
Задания	157
Тесты	157
Упражнения	157
<b>9-й час. Работа с макросами</b>	<b>159</b>
Азбука макропрограммирования	159
Создание таблицы с помощью SQL	160
Создание макроса	161
Задание имени и условия выполнения макроса	162



Тестирование и отладка макроса	165
Макросы группы КопироватьОбъект	166
Использование макросов УдалитьОбъект	167
Ключи от города	169
Импорт данных	170
Преобразование базы данных	170
Преобразование текста	171
Замена макрокоманд инструкциями на языке VBA	174
Макросы: быть или не быть	175
Резюме	176
Вопросы и ответы	176
Задания	177
Тесты	177
Упражнения	177
<b>10-й час. Как использовать готовые решения</b>	<b>179</b>
Функции для работы со строками	179
Взаимное преобразование строк и чисел	180
Строковые функции двойного назначения	181
Функции преобразования типов данных	183
Поиск строк	184
Динамическое заполнение строк	185
Форматирование данных	185
Функции Date и Time	186
Операции файлового ввода-вывода	187
Команды Open и Close	187
Чтение и запись текстовых данных	190
Чтение и запись двоичных данных	192
Функции интерактивного ввода данных	193
Еще раз об оперативной справочной системе	194
Резюме	194
Вопросы и ответы	195
Задания	195
Тесты	195
Упражнения	196
<b>Часть IV. Определение типов данных. Использование массивов и коллекций</b>	<b>197</b>
<b>11-й час. От сложного к простому: создание собственных типов данных</b>	<b>199</b>
О понятии агрегации	200
Объявление новых типов	200
Какие данные может содержать пользовательский тип	202
Создание экземпляров пользовательских типов данных	204
Перечислимые типы	205

А теперь повторим	206
Резюме	207
Вопросы и ответы	207
Задания	207
Тесты	207
Упражнения	208
<b>12-й час. Управление данными переменного объема</b>	<b>209</b>
Знакомство с массивами	209
Объявление массивов	210
Массивы фиксированного размера	211
Динамические массивы	211
Статические массивы	213
Задание точки отсчета индекса массива	213
Использование массивов для хранения данных	213
Функции для управления массивами	216
Проверка переменных с помощью функции <code>IsArray</code>	217
Передача массивов параметров в процедуры	218
Функции, возвращающие массивы	219
Что еще следует знать о массивах	220
Сортировка данных	220
Сортировка методом "пузырька"	220
Сортировка посредством выбора	222
Быстрая сортировка	223
Резюме	225
Вопросы и ответы	226
Задания	226
Тесты	226
Упражнения	226
<b>13-й час. Коллекции данных</b>	<b>227</b>
Знакомство с коллекциями	227
Использование коллекций	228
Термины объектно-ориентированного программирования	228
Создание объектов коллекций	229
Управление коллекциями данных	231
Добавление элементов в коллекцию	231
Удаление данных из коллекции	233
Метод <code>Item</code> и свойство <code>Count</code>	233
Циклическая обработка элементов коллекции	234
Где еще применяются коллекции	235
Резюме	235
Вопросы и ответы	235
Задания	236
Тесты	236
Упражнения	236

**14-й час. Стил ь программирования: „Что такое хорошо, и что такое плохо”**

**239**

Соглашения об именах	240
Использование целых слов при именовании	241
Использование словосочетаний при именовании подпрограмм и функций	242
Избегайте нестандартных аббревиатур	243
Правила выравнивания текста программы	243
Как упростить код	245
"Нет" — вложенным условным конструкциям	245
"Да" — процедурам и функциям	246
Создавайте короткие и уместные процедуры	246
Как комментировать код	247
Используйте полноценные предложения	247
Комментируйте длинные фрагменты кода	247
Сопровождайте примечаниями неоднозначные участки кода	248
О возможностях повторного использования кода	248
Советы по тестированию и отладке кода	249
Еще раз о работе с данными	249
Резюме	249
Вопросы и ответы	250
Задания	250
Тесты	250
Упражнения	250

**15-й час. ADODB — ваш верный помощник**

**251**

Соединение с базой данных	252
Аргументы процедуры открытия базы данных	253
Создание источника данных ODBC	255
Провайдер: что это такое?	257
Задание имени и пароля пользователя	257
Выбор режима соединения	257
Управление наборами данных	258
Открытие и закрытие наборов данных	258
Разновидности наборов данных	260
Добавление и изменение данных	262
Редактирование полей	263
Удаление строк из набора данных	264
Поиск записей	265
Поиск в таблице	265
Поиск с использованием запроса	267
Свойство Filter	268
Сохранение данных в коллекциях	268
Использование AddItem и RemoveItem	269
Использование информационных ресурсов и поиск готовых решений	270

Резюме	270
Вопросы и ответы	271
Задания	271
Тесты	271
Упражнения	272
<b>16-й час. Применение языка SQL</b>	<b>273</b>
Использование выражения SELECT	274
Простые формы SELECT	274
Фильтрация данных с помощью предложения WHERE	275
Операторы, применяемые в предложении WHERE	276
Предложение WHERE и вложенные команды SELECT	278
Сортировка данных	279
Группировка столбцов	280
Использование предложения HAVING	280
Объединение таблиц	281
Объединение запросов	283
Переименование столбцов результата	283
Добавление записей	284
Добавление данных в указанные поля	284
Добавление полной строки данных	285
Вызов команды Insert с параметрами	285
Добавление записи с помощью SELECT	286
Обновление данных	287
Удаление данных	288
Вызов функций из команд SQL	289
Хранимые процедуры	289
Добавление хранимой процедуры в каталог	290
Выполнение хранимой процедуры	291
Создание запросов в базах данных формата SQL Server	292
Резюме	293
Вопросы и ответы	293
Задания	294
Тесты	294
Упражнения	294
<b>Часть VI. Работа над ошибками</b>	<b>295</b>
<b>17-й час. Отладка кода</b>	<b>297</b>
Тестирование кода	297
О важности промежуточного тестирования	298
Что именно следует тестировать	299
Как тестировать программный код для Access	299
Использование ловушек	301
Что такое ловушка	301
Как применять ловушки	302
Трассировка кода	303
Верификация исходных условий	304

Использование соглашений	305
Повторное использование кода отладки	305
Использование директив компилятора	306
Отладочный код — только для чтения	309
Различные типы кода обработки ошибок	309
Резюме	310
Вопросы и ответы	310
Задания	311
Тесты	311
Упражнения	311
<b>18-й час. Обработка ошибок во время выполнения программы</b>	<b>313</b>
Сравнение технологий обработки ошибок	314
Обработка ошибок с помощью условных выражений: старый подход	314
Обработка исключений: современная технология	314
Как строить обработчики ошибок	316
Конструкция заголовка обработчика ошибок	316
Общие правила создания обработчиков ошибок	317
Очистка состояния обработчика ошибок	318
Пассивные обработчики ошибок	318
Использование команды Resume	318
Применение команды Resume Next	320
Использование объектов класса Егг	321
Свойства объекта Егг	321
Методы объекта Егг	321
Применение обработчика ошибок для проверки данных, вводимых пользователем	322
Проблема экономии ресурсов компьютера	323
Применение объекта Debug	324
Резюме	325
Вопросы и ответы	325
Задания	326
Тесты	326
Упражнения	326
<b>Часть VII. Проектирование интерфейса пользователя</b>	<b>327</b>
<b>19-й час. Создание экранных форм</b>	<b>329</b>
Использование мастеров создания форм	330
Обзор режимов проектирования форм	330
Создание формы в режиме Мастер форм	334
Управление данными с помощью команд меню	337
Фильтрация данных	337
Настройка параметров формы	340
Знакомство с атрибутами визуальных объектов	341
Размещение управляющих элементов на форме	344
Создание программного кода формы	346
Расширенные возможности редактирования	347

Расширенные возможности форм и отчетов	347
Тестирование формы	349
Построение отчета	349
Использование объекта Printer	350
Добавление элементов меню в Access для печати отчета	350
Установка параметров запуска приложения	352
Резюме	353
Вопросы и ответы	354
Задания	354
Тесты	354
Упражнения	354
<b>20-й час. Как связывать Web-страницы с базами данных</b>	<b>355</b>
Internet и intranet	356
Установка и использование приложения Internet Information Services	357
Создание Web-сайта	358
Знакомство с Web-страницами	358
Что представляет собой страница доступа к данным	359
Расширенные возможности проектирования Web-страниц	360
Создание демонстрационной базы данных	360
Создание Web-страниц с помощью мастеров	362
Выбор данных из нескольких таблиц	363
Работа с мастером страниц	364
Инструменты Web-дизайна в среде Access	365
Использование конструктора страниц доступа к данным	365
Меню и панели инструментов для настройки страницы доступа к данным	368
Размещение дополнительных компонентов на странице	371
Диалоговое окно свойств	371
Привязка полей данных к компонентам интерфейса	372
Сортировка и группировка данных	372
Добавление сводных таблиц на Web-страницы	374
Universal Naming Convention	375
Office Data Connection	376
Роль XML в Access	376
Резюме	377
Вопросы и ответы	377
Задания	378
Тесты	378
Упражнения	378
<b>Часть VIII. Объектно-ориентированное программирование в Access</b>	<b>379</b>
<b>21-й час. Основы программирования классов</b>	<b>381</b>
В чем состоит важность объектного подхода	382
В первый раз - первый класс	383
Скрытие информации	384
<b>Содержание</b>	<b>15</b>

Определение методов класса	386
Определение свойств класса	388
Использование свойств объектов	390
Статические свойства	390
Обработчики событий создания и удаления объекта	391
Создание нового класса	392
Тестирование класса	395
Расширение возможностей существующих классов	395
Создание экземпляра класса	395
Резюме	396
Вопросы и ответы	397
Задания	398
Тесты	398
Упражнения	398
<b>22-й час. Совершенствование типов данных</b>	<b>399</b>
Определение цели	399
Когда именно следует создавать классы	400
Принципы объектно-ориентированного проектирования	401
Проектирование класса ассоциативных массивов	401
Несколько советов по реализации класса	402
Расширение возможностей существующих классов	403
Использование коллекции для хранения данных ассоциативного массива	403
Реализация функций файлового ввода-вывода	406
Тестирование нового класса	408
Применение класса Strings	409
Резюме	409
Вопросы и ответы	410
Задания	410
Тесты	410
Упражнения	411
<b>23-й час. Надстройки Access</b>	<b>413</b>
Знакомство с надстройками Access	413
Создание базы данных для ведения протокола ошибок	414
Динамическое создание журнальной таблицы	415
Ведение протокола ошибок	416
Интерфейс просмотра ошибок	418
Тестирование кода надстройки	419
Установка и отключение надстроек	420
Создание таблицы с данными о регистрации	422
Установка надстройки	423
Модификация и удаление надстройки	424
Применение надстройки для ведения протокола ошибок	424
Подведем итоги	425
Резюме	425

Вопросы и ответы	426
Задания	426
Тесты	426
Упражнения	426
<b>24-й час. Управление информацией о контактах Outlook</b>	<b>427</b>
Знакомство с Outlook 2002	428
Объектная модель Outlook	428
Создание экземпляра Outlook	429
Использование объекта NameSpace	430
Использование коллекции Folders	431
Просмотр информации о контактах Outlook в Access	433
Обновление информации Outlook из внешних приложений	434
Поиск заданной строки в сообщениях электронной почты	435
Восстановление удаленных сообщений	437
Автоматическая рассылка почтовых сообщений	438
Резюме	439
Вопросы и ответы	440
Задания	440
Тесты	440
Упражнения	440
<b>Приложение. Ответы</b>	<b>441</b>
1-й час. Новинки Access 2002	441
Тесты	441
Упражнения	442
2-й час. Познакомимся с VBA	443
Тесты	443
Упражнения	443
3-й час. Принципы работы программы с данными	443
Тесты	443
Упражнения	444
4-й час. Последовательность действий и выполнение вычислений	444
Тесты	444
Упражнения	445
5-й час. Программирование управляющих структур	445
Тесты	445
Упражнения	446
6-й час. Управление базами данных	446
Тесты	446
Упражнения	447
7-й час. Расширенные типы данных	447
Тесты	447
Упражнения	448
8-й час. Декомпозиция задач	448
Тесты	448
Упражнения	449



9-й час. Работа с макросами	449
Тесты	449
Упражнения	450
10-й час. Как использовать готовые решения	450
Тесты	450
Упражнения	450
11-й час. От сложного к простому: создание собственных типов данных	451
Тесты	451
Упражнения	451
12-й час. Управление данными переменного объема	452
Тесты	452
Упражнения	453
13-й час. Коллекции данных	453
Тесты	453
Упражнения	454
14-й час. Стиль программирования: "Что такое хорошо, и что такое плохо"	455
Тесты	455
Упражнения	455
15-й час. ADODB — ваш верный помощник	455
Тесты	455
Упражнения	456
16-й час. Применение языка SQL	456
Тесты	456
Упражнения	457
17-й час. Отладка кода	457
Тесты	457
Упражнения	457
18-й час. Обработка ошибок во время выполнения программы	458
Тесты	458
Упражнения	459
19-й час. Создание экранных форм	460
Тесты	460
Упражнения	460
20-й час. Как связывать Web-страницы с базами данных	461
Тесты	461
Упражнения	461
21-й час. Основы программирования классов	462
Тесты	462
Упражнения	462
22-й час. Совершенствование типов данных	463
Тесты	463
Упражнения	464
23-й час. Настройки Access	464
Тесты	464
Упражнения	465
24-й час. Управление информацией о контактах Outlook	465
Тесты	465
Упражнения	466
<b>Предметный указатель</b>	<b>467</b>

## Об авторе

Пол Киммел — основатель компании *Software Conceptions*, занимающейся созданием программного обеспечения и предоставлением консультационных услуг для самых разных организаций и фирм, разбросанных по всему миру. Он более 10 лет руководил проектами на основе Microsoft Access, осуществляемыми по заказам транснациональных промышленных компаний, брокерских контор и страховых фирм ([pkimmel@softconcepts.com](mailto:pkimmel@softconcepts.com)).

Пол — автор и соавтор ряда серьезных книг по программированию на Visual Basic и в Access, среди которых *Visual Basic.NET Unleashed* и *Microsoft Access Development Unleashed* (издательство *Sams*). Пол также принимает участие в еженедельной рассылке *Code Guru Visual Basic Tech Notes* от Internet.com. Пол Киммел и вся его семья — жена Лори, дети Тревор, Дуглас, Алекс, Ноа и пес Хоган — проживают в городе Окемос, штат Мичиган.

# Посвящение

Посвящаю эту книгу отцу, Джеральду Киммелу (Gerald Kimmel).

## Благодарности

Прежде всего следует поблагодарить Анжелу Козловски (Angela Kozlovski), с которой я работал над предыдущей книгой (для Que). Работа с ней над новым проектом была настоящим удовольствием. Благодарю всех сотрудников *Sams*, которые помогали создавать исходный текст, листинги и копии экранов для новой книги.

Здесь мне предоставлена счастливая возможность выразить свою признательность всем, кто помог превратить обрывки мыслей и текста в цельную книгу. Читая их имена и описание заслуг, вы тем самым разделите мою почетную и радостную обязанность.

Благодарю всех своих учителей. Заранее приношу свои извинения за допущенные неточности в именах. Благодарю учителей г-на Кейница (Keinitz) и г-жу Шоу (Shaw) школы Washington Elementary в Оуоссо (Owosso), Мичиган (Michigan); г-на Раиса (Rice) из Owosso Junior High. Спасибо г-же Креддок (Creddock) за то, что не "завалила" меня на экзамене по аналитической геометрии в Lansing's Sexton High School, хотя я, вероятно это заслужил, поскольку в подростковом возрасте наука наводила на меня дремоту. Спасибо Джеймсу Бонду (James Bond), учителю вождения из Sexton. Благодарю учителей истории и английского языка д-ра Богмана (Baughman) и г-жу Монтгомери (Montgomery). Диссертация д-ра Богмана помогла мне осознать, что написанию книг тоже можно научиться. Несмотря на то, что я великолепно изучил C, а мои грамматические навыки по-прежнему хромают, я пришел к выводу, что чтение биографий и поэзии — приятное времяпрепровождение. Д-р Богмана и г-жа Монтгомери преподавали в Jefferson Community College в Луизвилле (Louisville), Кентукки (Kentucky). Благодарю д-ра Форсита (Foryth) и д-ра Стиклена (Sticklen) из Мичиганского университета (Michigan State University), профессоров в области компьютерных технологий за прекрасный опыт обучения, а д-ра Форсита — за поддержку, оказанную мне в процессе обучения. И, наконец, лучший мой учитель — моя жена Лори, преподававшая мне немало уроков о любви, терпимости, состраданию и воспитанию детей.

Отдельное спасибо Михаэлю Байцу (Michael Beitz) из Германии, который нашел несколько технических ошибок в листингах главы 22 и благодаря которому в данном издании код был исправлен.

Благодарю Стефана Кинга (Stephen King), за *On Writing*, интересную книгу, в которой содержится множество полезных советов о создании книг, и на мой взгляд, они помогли сделать эту книгу более ясной и четкой.

И наконец, выражаю сердечную благодарность всем издателям, редакторам, покупателям и читателям, которые создавали, покупали и читали предыдущее издание данной книги. По всем вопросам, связанным с книгой *Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа* (Teach Yourself Access 2002 Programming in 24 Hours), обращайтесь по адресу [pkimmel@softconcepts.com](mailto:pkimmel@softconcepts.com).

## Библиография

Fowler, Martin. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley. Reading, MA, 2000.

Purdum, Jack. *C Programmer's Toolkit*. Que Corporation. Carmel, IN, 1989.

# Сообщите нам ваше мнение

Вы, читатель этой книги, и являетесь главным ее критиком и комментатором. Мы ценим ваше мнение и хотим знать, что было сделано правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным. Нам интересно услышать также любые другие замечания в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Пришлите электронное письмо или просто посетите наш Web-узел, оставив свои замечания — одним словом, любым удобным для вас способом дайте нам знать, нравится вам эта книга или нет, а также выскажите свое мнение о том, как сделать наши книги более подходящими для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш факс или номер телефона. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

E-mail: [info@williamspublishing.com](mailto:info@williamspublishing.com)

WWW: <http://www.williamspublishing.com>

# Введение

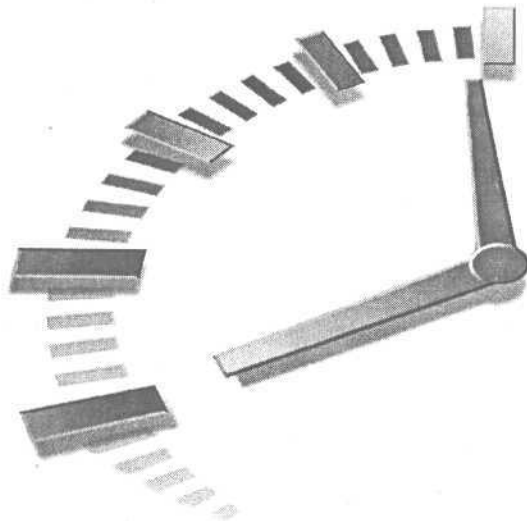
Наша книга, *Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа*, — для занятых профессионалов.

Она написана специально для тех, кто хочет максимально эффективно использовать Access. Данная книга является своего рода дополнением к *Teach Yourself Access 2002 in 21 Days* издательства Sams. В ней вы не найдете информации о создании баз данных и обычной работе с ними с помощью меню программы Access (для этого есть другие издания). Эта книга не для тех, кто относит себя к профессиональным программистам. А если возникла необходимость создать некоторый код, определяющий интерфейс пользователя, создающий зависимые от данных Web-страницы, а также написать SQL для решения некоторых проблем, — эта книга для вас. (Обратите внимание, что использование SQL ограничено лишь несколькими задачами. Подробнее о работе с SQL см. специальные издания.)

*Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа* не является пособием для новичков Access. Эта книга охватывает более сложные темы, предлагая решать возникающие задачи с помощью программирования. Пройдя 24-часовой курс, вы приобретете твердые навыки применения инструментов Access и программирования с помощью средств Visual Basic for Applications (VBA).

Настоящая книга является переработанным изданием *Освой самостоятельно программирование для Microsoft Access 2000 за 24 часа*. Читатели предыдущей версии найдут в ней немало полезного. Последние годы все большее значение имеет Internet, поэтому в данном издании значительно подробнее изложены вопросы создания страниц доступа к данным и их публикации, а также использование сводных таблиц в Web-страницах.

Приятного чтения. Надеюсь, вы получите ценные знания и опыт. Как обычно, с вопросами и предложениями обращайтесь по адресу [pkimmel@softconcepts.com](mailto:pkimmel@softconcepts.com).



# Часть I

## Азбука программирования в Access

### Темы занятий

1-й час. Новинки Access 2002

2-й час. Познакомимся с VBA

3-й час. Как программа работает с данными

4-й час. Последовательность действий и выполнение  
вычислений





# 1-й час

## Новинки Access 2002

Access 2002, а также возможности программирования в ее среде являются еще более впечатляющими, нежели средства, предлагавшиеся более ранними версиями системы. Access 2002 обладает встроенными инструментами, поддерживающими стандарты объектно-ориентированного программирования. Удобные инструменты, например автоматическая проверка правописания, индивидуальная настройка меню и панелей инструментов, значительно облегчают работу. Система поддерживает функции совместной работы в режиме on-line с помощью программы NetMeeting: пользователь программы и ее разработчик могут напрямую общаться посредством Internet. Значительно облегчают участь программистов мощные инструменты ActiveX Data Objects (ADO). Access 2002 позволяет создавать хранимые процедуры, связывать Web-страницы с базами данных и легко **масштабировать** последние до уровня Microsoft SQL Server с помощью Microsoft Access Projects.

Рассматриваемые в комплексе, все эти средства предоставляют возможности более быстрого создания качественных приложений для работы с базами данных, подготовки Web-ориентированных решений, масштабирования баз данных на платформу SQL Server, написания и переноса фрагментов объектно-ориентированного программного кода в среду других продуктов Microsoft Office, поддерживающих VBA.

В ходе этого занятия вы ознакомитесь с перечисленными инструментами и узнаете, как пользоваться некоторыми из них. Средствам, которые в этой главе не будут рассматриваться подробно (ввиду их сложности), посвящены отдельные разделы книги.

Основные темы занятия.

- Индивидуальная настройка меню и панелей инструментов.
- Средства проверки правописания.
- Настройка и использование функций совместной работы.
- Применение мастеров Data Page Wizards.



# Область задач облегчает навигацию

В Access 2002 появилась область (или панель) задач. Если при запуске Access эта панель не отображена на экране, щелкните правой кнопкой мыши на панели инструментов и в появившемся контекстном меню выберите команду Область задач (Task Pane). Данная панель упрощает процесс открытия существующих баз данных и создания новых (из шаблонов, имеющихся в Access или доступных на [Microsoft.com](http://Microsoft.com)). Также можно добавлять узлы (Network Place) — папки, находящиеся на локальном ПК или в другом месте, доступ к которым обеспечивается через URL.

Группа Открытие файла (Open File) на панели задач содержит ссылки на недавно использовавшиеся базы данных. Группа Создание (New File) предлагает возможность быстрого создания файлов баз данных. Ссылка Выбор файла (Choose File) группы Создание из имеющегося файла (New from Existing File) вызывает окно браузера, в котором можно выбрать нужный файл. Группа Создание с помощью шаблона (New from Template) позволяет выбрать шаблон, имеющийся на компьютере или доступный на узле корпорации Microsoft. Ссылка Добавление узла позволяет создавать папки, доступ к которым обеспечивается через URL.

Чтобы область задач не появлялась на экране при загрузке программы Access, снимите флажок Показывать при запуске (Show at Startup).

## Индивидуальная настройка меню и панелей инструментов

Access 2002 позволяет настраивать меню и панели инструментов в соответствии с собственными предпочтениями и потребностями. Система может отображать сокращенный набор элементов меню и кнопок панелей инструментов либо автоматически предлагать те из них, которыми вы пользуетесь наиболее часто. Можно изменять размеры панелей инструментов и располагать их в тех областях экрана, где вам удобно. (Например, в главе "19-й час. Создание экранных форм", описан процесс настройки элементов меню.)

Другая полезная новинка — возможность назначения интерфейсным объектам системы гиперссылок, обеспечивающих прямой доступ к ресурсам Internet или корпоративной сети.

## Элемент меню Развернуть

Access 2002 позволяет отображать ограниченное число элементов меню. В этом случае меню снабжаются дополнительным элементом Развернуть (More) (рис. 1.1). Щелчок на нем приводит к открытию оставшейся части меню. При выборе элемента, который отсутствовал в исходном множестве, он автоматически добавляется в список видимых, и в следующий раз вам уже не придется обращаться к элементу Развернуть.

Автоматическое сокрытие некоторых элементов меню упрощает его восприятие и использование. Access отображает только те элементы, к которым вы обращаетесь чаще всего.

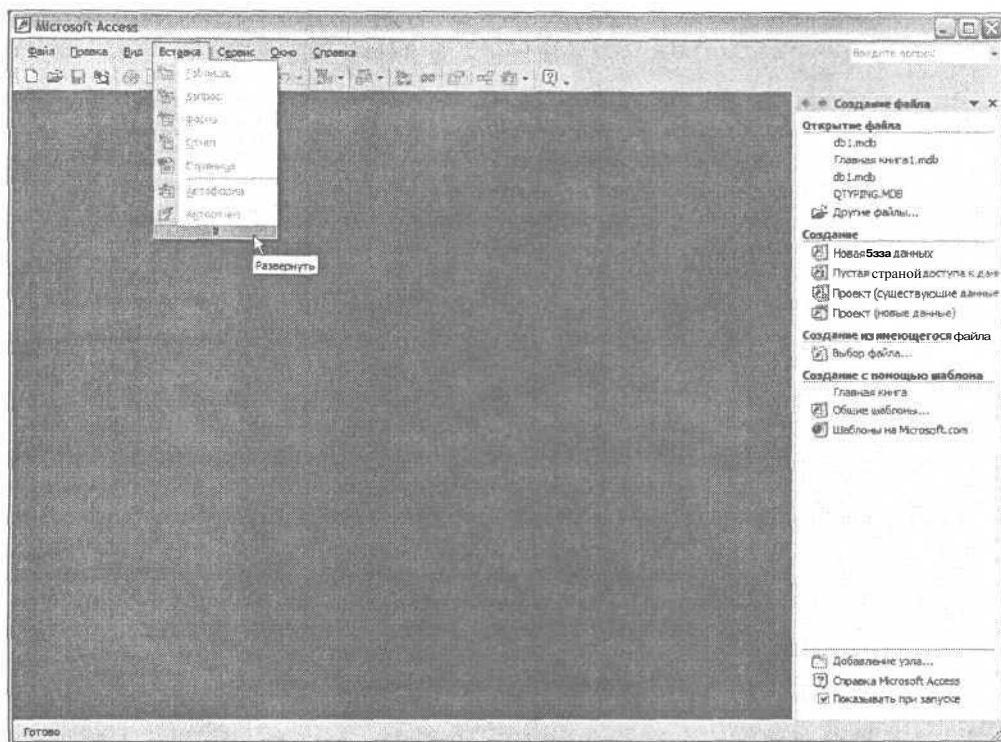


Рис. 1.1. Щелчок на элементе Развернуть позволяет увидеть полную версию меню

## Индивидуальная настройка меню и панелей инструментов

Строка меню и панели инструментов снабжены указателем перемещения, имеющим вид вертикальной черты. Расположите над ним курсор мыши (рис. 1.2), щелкните левой кнопкой и, не отпуская ее, перетащите курсор в требуемую позицию — интерфейсный объект Access займет назначенное вами место.

Как строка меню, так и панели инструментов могут быть закреплены у любой границы окна либо оставаться в состоянии "свободного плавания" — система помогает вам приспособить интерфейс к собственным потребностям. Чтобы сделать объект плавающим, щелкните над указателем перемещения и перетащите курсор в среднюю часть окна. Если же требуется закрепить объект, передвиньте его к нужной стороне рамки.

Состав строки меню и панелей инструментов отнюдь не ограничивается тем, что вы видите, когда открываете окно Access в первый раз, — существует множество других меню и панелей, которые можно принудительно отображать и скрывать (рис. 1.3). Впрочем, все эти дополнительные элементы обычно появляются на экране автоматически, отвечая контексту текущей ситуации.

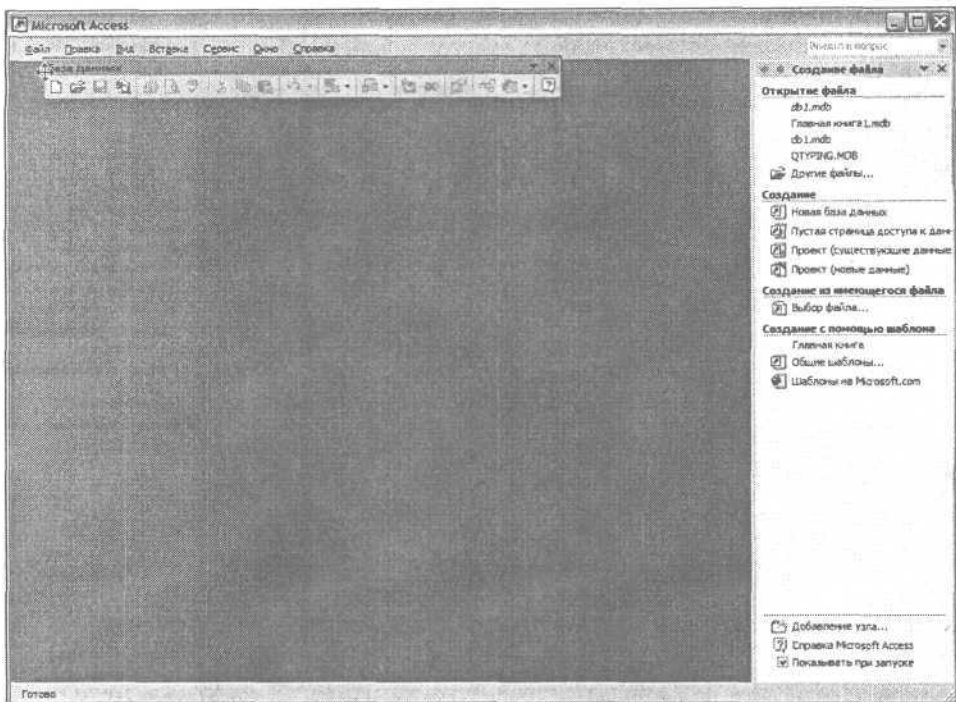


Рис. 7.2. Указатель перемещения позволяет управлять положением интерфейсного элемента системы на экране

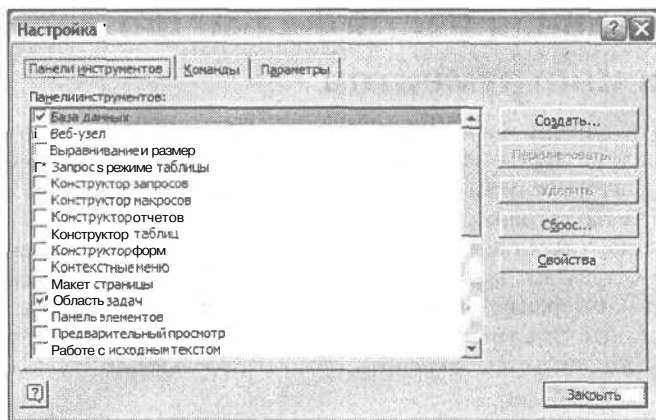


Рис. 1.3. Щелкните правой кнопкой мыши в пределах основной панели инструментов и выберите в контекстном меню элемент Настройка (Customize) — откроется одноименное диалоговое окно



Другой способ открытия диалогового окна Настройка — выбор в строке меню команды Сервис⇒Настройка (Tools⇒Customize).

Чтобы получить доступ к дополнительным интерфейсным элементам системы, расположите курсор мыши над строкой меню или любой открытой в данный момент панелью инструментов, щелкните правой кнопкой мыши и выберите в контекстном меню элемент Настройка. Перейдите на вкладку Панели инструментов (Toolbars) диалогового окна Настройка, отметьте флажками нужные элементы и снимите флажки тех, которые хотели бы скрыть.

## Назначение гиперссылок интерфейсным объектам

Любой кнопке панели инструментов или элементу меню может быть назначена соответствующая гиперссылка. Существует несколько способов использования подобной возможности. Вы можете, скажем, "привязать" к кнопке панели или к элементу меню некий документ в формате HTML либо создать совершенно новый интерфейсный объект, обеспечивающий доступ к тому или иному ресурсу непосредственно из среды приложения.

Чтобы назначить элементу меню или кнопке панели инструментов соответствующую гиперссылку, выполните следующие действия.

1. Выберите команду **Сервис**⇒**Настройка**, чтобы открыть диалоговое окно Настройка.
2. Поместите курсор мыши над объектом, которому хотите назначить гиперссылку. (Если объект в данный момент скрыт, предварительно отобразите его.)
3. Щелкните правой кнопкой мыши на элементе меню или панели инструментов (но не в области окна Настройка) и выберите в контекстном меню команду Назначить гиперссылку⇒Открыть (Assign Hyperlink⇒Open).
4. В диалоговом окне Назначить гиперссылку: Открыть (Assign Hyperlink: Open) (рис. 1.4) выберите нужный файл или введите в поле Адрес (Address) URL гиперссылки.

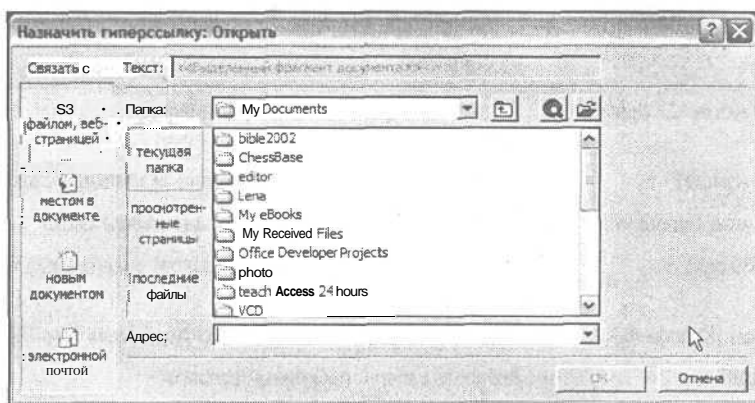


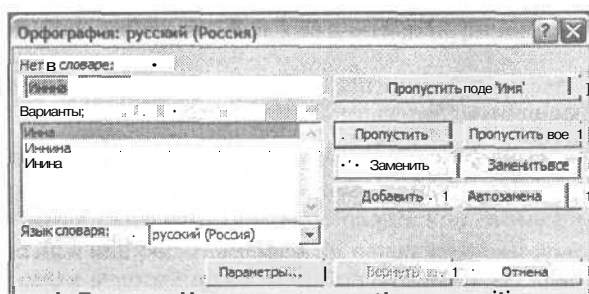
Рис. 1.4. Чтобы назначить гиперссылку соответствующему элементу интерфейса Access, воспользуйтесь средствами диалогового окна Назначить гиперссылку: Открыть

5. Щелкните на кнопке ОК.

Чтобы удалить существующую гиперссылку, повторите описанные выше действия, но при выполнении п.3 выберите в контекстном меню команду Изменить гиперссылку⇒ Удалить ссылку (Edit Hyperlink⇒Remove Link).

## Проверка правописания

Access 2002 предлагает удобные средства автоматической проверки правописания. Щелкните в пределах таблицы, запроса или формы, а затем выберите в строке меню команду Сервис⇒Орфография (Tools⇒Spelling). Система осуществит анализ содержимого всех полей выбранного объекта данных. Если встречается незнакомое слово, Access приостанавливает выполнение процедуры и открывает диалоговое окно Орфография (Spelling) (рис. 1.5), позволяющее внести необходимые исправления. Назначение всех опций окна Орфография описано ниже, в табл. 1.1.



*Рис. 1.5. Диалоговое окно Орфография предоставляет все необходимые рычаги управления процессом проверки правописания*

**Таблица 1.1. Опции проверки правописания в среде Access**

Наименование	Назначение
Пропустить поле '...' (Ignore '...' Field)	Исключить из рассмотрения указанное поле данных
Пропустить (Ignore)	Отказаться от исправления текущего экземпляра слова
Пропустить все (Ignore All)	Исключить из рассмотрения все экземпляры слова
Заменить (Change)	Заменить слово в поле "Нет в словаре" содержимым поля "Заменить на"
Заменить все (Change All)	Заменить все экземпляры слова содержимым поля "Заменить на"
Добавить (Add)	Добавить слово в выбранный словарь
Варианты (Suggest)	Отобразить список наиболее подходящих вариантов замены слова
Параметры (Options)	Выбрать язык словаря, настроить варианты замены и указать способы обработки слов, набранных в верхнем регистре и/или содержащих цифры
Вернуть (Undo Last)	Отменить последнюю операцию
Отмена (Cancel)	Закрыть окно без внесения каких-либо изменений

Процедура проверки орфографии игнорирует последовательности символов, которые не может распознать как цельное слово. Средства анализа правописания — мощный инструмент, доступный как на этапе проектирования приложения, так и в момент его выполнения; программисты и пользователи, которые не сильны в орфографии, получают гарантии того, что сохраняемые ими текстовые данные не содержат досадных ошибок и опечаток.



Подобно другим программам, снабженным функциями проверки орфографии, Access 2002 не в состоянии справиться с ошибками, имеющими отношение к использованию *омофонов* — слов, которые звучат одинаково, но пишутся по-разному и носят различную смысловую окраску. Если, например, вы введете слово "также", подразумевая, на самом деле, "так же", программа не сможет вам помочь, поскольку ваша ошибка с формальной точки зрения таковой не является.

То же справедливо и в отношении *паронимов* — созвучных слов, имеющих разную смысловую нагрузку и разное написание — "эффект" и "аффект", "разлив" и "розлив" и т.п. Другими словами, взяв на вооружение самые современные технологии, вы все равно должны быть внимательными, пользуясь словами родного языка.

## Возможности редактирования

В Access 2002 насчитывается много новых средств, позволяющих повысить производительность труда. Краткое их описание приведено ниже.

### Команды Отменить и Вернуть

Access 2002 поддерживает многоуровневые операции Отменить (Undo) и Вернуть (Redo), что позволяет изменять как объекты в режиме конструктора и других режимах, так и хранимые процедуры и функции, а затем возвращаться к исходным объектам.

### Помощник Office

В Office есть анимированный помощник, отключенный по умолчанию. Чтобы воспользоваться услугами Помощника, сначала подключите его, выбрав команду Справка⇒Показать помощника (Help⇒Show Office Assistant). Управлять поведением Помощника можно программными средствами (см. главу "24-й час. Управление информацией о контактах Outlook").

### Введите вопрос

Поле со списком Введите вопрос (Ask a Question), расположенное в правом верхнем углу экрана программы (см. рис. 1.1), позволяет быстро отыскать справочную информацию по нужной теме. Просто введите вопрос в области поля со списком.

Введите в этом поле слово **помощник** (Clippit), чтобы получить ссылки на разделы справочной системы, касающиеся работы с Помощником. Программа хранит список введенных ранее вопросов.

# Дополнительные возможности буфера обмена

Буфер обмена увеличен и может хранить до 24-х фрагментов. При копировании в буфер обмена большего числа фрагментов, новые данные помещаются в верхнюю часть списка, а данные из нижней части списка удаляются.



Быстро просмотреть содержимое буфера обмена можно, дважды нажав <Ctrl+C>.

Чтобы отобразить буфер обмена, выберите команду **Правка⇒Буфер обмена Office (Edit⇒Office Clipboard)**. По умолчанию панель задач **Буфер обмена (Clipboard)** появляется в правой части окна программы Access.

## Распознавание речи

Для использования возможностей распознавания речи необходимо подключить к компьютеру микрофон и динамики, и вы сможете голосовыми командами не только управлять работой Access и всего компьютера, но и успешно путешествовать в Web. После установки средств распознавания речи в меню **Сервис** станет доступной панель инструментов **Распознавание речи (Speech Recognition)**.

Данная технология особенно удачно работает при сочетании голосовых команд с использованием мыши и клавиатуры. После установки средств распознавания речи программа запросит прочесть текст, что займет примерно 10 минут. Данный текст разработан специально для того, чтобы в дальнейшем наиболее эффективно работать с особенностями вашего произношения. Чем больше времени вы потратите на тренировку, тем лучше программа распознавания речи будет вас понимать.

## Совместная работа над документами

Функции совместной работы в режиме on-line — это просто чудо! Заслуга принадлежит программным продуктам **NetMessenger** и **NetMeeting** (вы можете свободно загрузить их с Web-сайта [www.microsoft.com](http://www.microsoft.com) компании Microsoft). Команда **Сервис⇒Совместная работа⇒Начать собрание (Tools⇒Online Collaboration⇒Meet Now)** позволит, используя текущее подключение к Internet, обратиться непосредственно к другим пользователям сервиса **NetMeeting**.

Поскольку подобные средства относительно новы, я не возьму на себя смелость давать исчерпывающие инструкции по их использованию. Служба **NetMeeting** предоставляет в ваше распоряжение такие возможности: обмена голосовой, аудио- и видеoinформацией; организации общей "виртуальной классной доски"; совместного использования приложений и файлов. Например, вместо того, чтобы звонить по междугородной телефонной линии, вы можете установить аналогичное соединение посредством Internet. Если ваш собеседник находится в пределах той же локальной сети, вам вообще не придется тревожить телефонную компанию. Если ваш компьютер снабжен устройствами ввода-вывода видеоданных, во время разговора вы будете видеть на мониторе лицо своего собеседника.

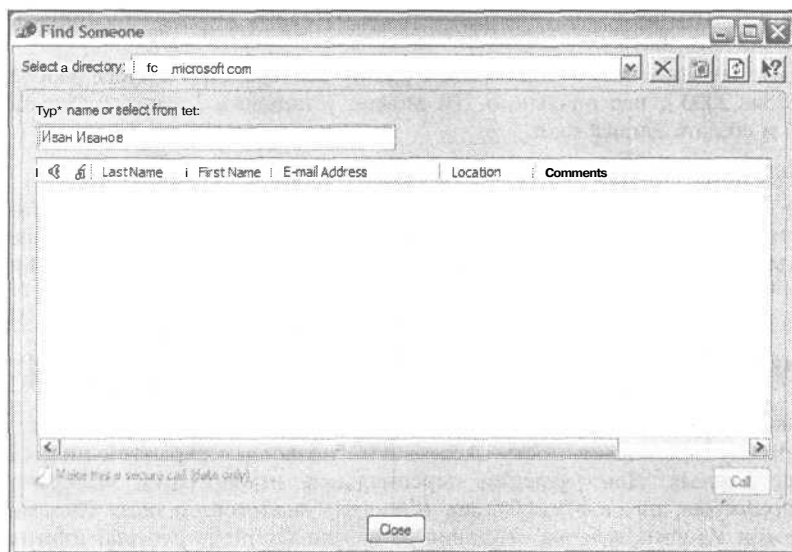
Качество передачи голоса пока не соответствует привычным стандартам, принятым в традиционной телефонии, но оно заметно улучшается. Кроме того, отрицательную роль может сыграть и проху-сервер — например, **WinProxu**. Если, однако, вы пользуетесь прямым подключением к Internet, все происходит достаточно гладко. Окончательный результат развития подобных технологий предвидеть довольно трудно, но, по моему мнению, со временем отнюдь не исключен вариант существенного падения спроса на традиционные телефонные услуги в пользу применения средств Internet.

# Настройка NetMeeting

Чтобы воспользоваться средствами NetMeeting, вам необходимо подключение к Internet либо IP-адрес, обозначающий ваш компьютер в локальной сети. Если Internet-соединение установлено, достаточно выбрать команду **Сервис**⇒**Совместная работа**⇒**Начать собрание**, чтобы влиться в сообщество участников виртуальной конференции. Если с NetMeeting вы ранее еще не работали, воспользуйтесь услугами мастера настройки Microsoft NetMeeting. Мастер, в частности, поинтересуется тем, какой сервер каталогов вы намерены использовать. Сервер каталогов — это профамма, которая может найти в Internet нужных собеседников и установить связь с ними.

## Соединение установлено. Что дальше?

NetMeeting — серьезная самостоятельная программа. После ввода команды, предписывающей начать собрание, открывается диалоговое окно **Вызов (Find Someone)**, в котором необходимо указать реквизиты будущего собеседника (рис. 1.6). Если искомого имени нет в перечне пользователей текущего сервера каталогов, выберите в раскрывающемся списке **Каталог (Select a directory)** название другого сервера.



*Рис. 1.6. В этом диалоговом окне укажите сервер каталогов и выберите подключенных к нему пользователей, с которыми вы хотели бы пообщаться*

Выделив имя интересующего вас человека, щелкните на кнопке **Вызвать (Call)**. Откроется окно программы NetMeeting, и вы сможете приступить к совместной работе. Возможности общения определяются характеристиками аппаратного обеспечения вашего компьютера. Если он снабжен звуковой картой, микрофоном и громкоговорящими, вы сможете воспользоваться голосовыми средствами. Если у вас и вашего собеседника есть устройства обработки видеoinформации, вы будете видеть друг друга. В любом случае вы можете обратиться к средствам обмена сообщениями в режиме chat, передачи файлов и использования "виртуальной классной доски" с помощью инструментов, вызываемых из меню **Сервис**.



Команда **Сервис**⇒**Общие** приложения (**Tools**⇒**Sharing**) дает возможность совместного использования файлов, включая также базы данных Access. Это весьма удобно при разрешении возникших проблем, отладке процедур, взаимном обмене информацией между разработчиками и пользователями приложений.

## Создание узлов

УЗЛЫ — это папки на сервере, доступ к которым можно получить по локальной сети или через Internet. Хранящаяся в таких папках информация может быть как общедоступной, так и закрытой. Использование узлов позволяет упростить совместную работу над базами данных, обеспечивая к ним доступ людей, находящихся в разных концах Земного шара.

Для создания узлов используйте ссылку **Добавление узла** (**Add Network Place**) на панели задач. Выполните следующие действия.

1. Если область задач не отображена на экране, выберите команду **Файл**⇒**Создать** (**File**⇒**New**).
2. Щелкните на ссылке **Добавление узла**, расположенной в нижней части панели задач.
3. Укажите URL (<http://www.адрес.com>) Web-узла. (Возможно, понадобится указать имя пользователя и пароль.)
4. Задайте название папки и завершите работу мастера. При запуске рабочей станции Windows 2000 с персонального ПК можно установить Internet Information Services (IIS) и создать intranet-сайт.

После создания сетевой папки в ней можно хранить страницы доступа к данным (созданные в Access и сохраненные как внешние Web-страницы). Указанная новая возможность программы Access 2002 существенно упрощает процесс создания зависящих от данных Web-сайтов (подробнее см. главу "20-й час. Как связывать Web-страницы с базами данных").

## Удаление персональной информации

По разным причинам при публикации баз данных и страниц доступа к данным может потребоваться анонимность. Access 2002 позволяет скрывать информацию об авторе базы данных. Для удаления персональной информации выберите команду **Сервис**⇒**Параметры**, на вкладке **Общие** (**General**) диалогового окна **Параметры** установите флажок **Удалить личные сведения из файла** (**Remove personal information from this file**).

## Знакомство с объектами ActiveX Data Objects

ActiveX Data Objects (ADO) — это новейший стандарт объектов баз данных, позволяющий значительно упростить задачи программирования. Объекты ADO могут использоваться в Access, Visual Basic, приложениях Web и в любой другой среде программирования, позволяющей создавать объекты COM. Речь идет о том, что большинство языков программирования для Windows (если не все они) способны обращаться к функциям, реализованным в составе динамической библиотеки Msado15.dll.



ADO.NET — новейший протокол баз данных, который пока еще не доступен в Access. Microsoft занимается объединением Office в среду .NET, но поскольку Office XP не является частью .NET, в нем ADO.NET не используется.

За более подробной информацией о .NET-технологиях обратитесь на Web-сайт компании Microsoft.

ADO содержат объекты, помогающие легко создавать источники данных и управлять ими. Подробнее об объектах ADO см. главу "15-й час. ADODB — ваш верный помощник".

## Новые объекты для обеспечения безопасности баз данных — ADOX

Объекты нового стандарта — ActiveX Data Objects Extensions 2.1 for DDL and Security — призваны облегчить задачу обеспечения безопасного доступа к базам данных. Эти объекты будут упоминаться в различном контексте в нескольких разделах книги, но особое внимание мы уделим их рассмотрению в ходе 15-го занятия.

## Поддержка хранимых процедур

Среди реализованных в составе библиотеки Msadox.dll средств для работы с ADOX имеется класс Procedure. Он обеспечивает создание хранимых процедур и запись их в файл базы данных. О том, как создавать и сохранять процедуры, вы узнаете из раздела "Хранимые процедуры" главы 15. В той же главе приведены примеры использования других объектов ADOX.

## Программные объекты

В состав Microsoft Access 2002 входят средства для работы с языком программирования Visual Basic for Applications (VBA). VBA — ближайший "родственник" языка Visual Basic (предшественника Visual Basic.NET) — с недавних пор стал поддерживать парадигму объектно-ориентированного программирования (как и VB6). Основным элементом стандарта объектно-ориентированного программирования — класс.



В версию 7.0 Visual Basic внесены радикальные изменения, он является частью структуры .NET. Последняя версия пакета Office не использует VB.NET, но, скорее всего, это вопрос недалекого будущего.

*Класс* — это определяемый пользователем тип данных, который лежит в основе объектного подхода. Модель класса позволяет программисту объединить данные и функции для их обработки в целостную сущность. Экземпляры классов, называемые *объектами*, — основной "материал", из которого строятся программы, написанные в объектном стиле.

Если бы классов не было, их стоило бы придумать, поскольку создание и повторное использование функций для работы с графическими интерфейсами и базами данных оказалось бы чрезвычайно сложной и неблагодарной задачей. Сведения о различных аспектах объектно-ориентированного программирования изложены в последующих главах книги. О том, как создавать и использовать классы, вы узнаете из глав "21-й час. Основы программирования классов" и "22-й час. Совершенствование типов данных".

# Связь Web-страниц с базами данных

Самые последние достижения в области информационных технологий так или иначе связаны с задачами программирования для среды Internet. Web-страница — это текстовый файл, содержащий данные в форматах Hypertext Markup Language (HTML) либо Active Server Pages (ASP). Тексты HTML и ASP способны восприниматься программными Web-браузерами, такими как Netscape Communicator или Internet Explorer. В этом контексте файлы HTML и ASP можно рассматривать как законченные фрагменты кода, а браузеры — как разновидность интерпретаторов языков программирования.

До недавнего времени создание Web-страниц, содержащих и отображающих информацию из баз данных, являлось непростой задачей. Но теперь реализованные в Access 2002 мастера Data Pages Wizards позволяют строить Web-страницы с полями, представляющими актуальную информацию из баз данных. Поле страницы, таким образом, оказывается напрямую связанным с соответствующим объектом базы данных, позволяя как считывать информацию, так и изменять ее. В главе "20-й час. Как связывать Web-страницы с базами данных" мы расскажем о приемах создания Web-страниц в Internet либо корпоративной сети (intranet или extranet), способных отображать информацию из баз данных в окне стандартного браузера.

## Новый термин

*Intranet* — это частная сеть, построенная в соответствии с технологиями Internet. Ее пользователи обычно имеют отношение к определенной компании или группе и подключены в единую локальную вычислительную сеть.

## Новый термин

*Extranet* — расширенный вариант intranet. Если intranet предполагает доступ к сети для членов территориально ограниченной группы пользователей, то extranet позволяет осуществлять подключение тех компьютеров, которые не охвачены локальной сетью.

## Access Projects

В Microsoft Access 2002 реализована новая модель проектов. Работая с прежними версиями системы, вы создавали базы данных в виде файлов формата MDB. Access 2000 позволяет строить файлы проектов формата ADP — Microsoft Access Project.

Файлы проектов Access предоставляют возможность масштабирования баз данных на платформу Microsoft SQL Server 2000. Это означает, что вы можете продолжать пользоваться привычным интерфейсом Access, но применять программный сервер SQL Server вместо сервера Jet, предлагаемого в Access по умолчанию. Единственное, что вам потребуется, — установить на локальном компьютере программное обеспечение SQL Server 2000 либо подключиться к соответствующему выделенному серверу баз данных.

## Резюме

На этом занятии вы смогли кратко ознакомиться с рядом новинок, реализованных в составе Access 2002. Возможности совместной работы в режиме on-line, объекты ActiveX Data Objects, доступные инструменты создания Web-ориентированных проектов, средства масштабирования баз данных до уровня Microsoft SQL Server — все это характеризует Access 2002 как привлекательный и мощный продукт.

Access 2002 поможет вам автоматизировать решение мелких повседневных задач. Кроме того, система предоставляет возможности создания замысловатых и развитых объектно-ориентированных приложений для среды клиент-сервер, реализующих общепринятый графический интерфейс Windows либо Web, что дает возможность пользователям применять стандартные программы-браузеры.

Материал остальных глав книги посвящен вопросам программирования в среде Access. Вы научитесь писать разнообразные программы — от простейших до достаточно сложных, поддерживающих объектно-ориентированный стиль, — и быстро предлагать качественные решения для Windows или другой операционной среды. Прежде чем приступить к использованию полученных знаний на практике, ознакомьтесь с приведенными ниже разделами "Вопросы и ответы" и "Задания".

## Вопросы и ответы

**Вопрос.** Целесообразно ли тратить время на настройку меню и панелей инструментов?

**Ответ.** В этом деле постарайтесь максимально учесть собственные практические цели. Мое мнение таково: не стоит серьезно изменять содержимое стандартных меню и панелей инструментов, если это не помогает решению конкретной задачи либо не облегчает выполнение каких-либо часто повторяющихся рутинных операций.

**Вопрос.** Служит ли наличие подключения к Internet обязательным условием возможности использования функций совместной работы?

**Ответ.** Нет. Те же функции можно с успехом применять и в условиях локальной вычислительной сети. Все, что действительно необходимо, — это знание IP-адреса компьютера вашего коллеги или иных сведений, которые помогут отыскать корреспондента среди множества пользователей сервера каталогов.

**Вопрос.** Необходимо ли платить за пользование службой NetMeeting?

**Ответ.** Нет. Программа NetMeeting, вызываемая из среды Access 2002, доступна также при работе с другими приложениями Microsoft Office XP. (Если в вашей редакции Office она отсутствует, ее можно бесплатно загрузить с Web-сайта компании Microsoft.) Вам необходимо всего лишь стать абонентом провайдера Internet либо подключиться к корпоративной сети intranet.

**Вопрос.** Смогу ли я применять все возможности Access 2002, работая с базами данных старого формата, MDB?

**Ответ.** Вы сможете применять почти все из них. В вашем распоряжении остаются инструменты, которые были доступны в прежних версиях системы, и многие из новых средств, включая возможности создания классов и процедур. Однако некоторые из функций системы специально ориентированы на использование совместно с мощным и более развитым программным сервером Microsoft SQL Server, который приходит на смену Jet — стандартному серверу баз данных Access.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Для чего предназначен элемент меню Развернуть?
2. Если элементу меню или кнопке панели инструментов назначить гиперссылку, то "поведение" этого интерфейсного объекта изменится. Верно ли это?
3. Функция проверки правописания подвергает анализу выражения SQL и такие простые образцы текстовых данных, как сокращения и произвольные последовательности символов. Верно ли это?

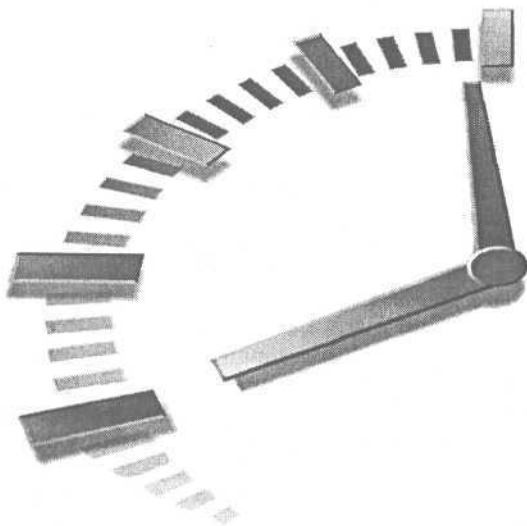
4. Каково наименование библиотеки, содержащей объекты Procedure, предназначенные для создания хранимых процедур?
5. Access 2002 имеет в своем составе функции-мастера, позволяющие создавать Web-страницы, связанные с объектами баз данных. Верно ли это?

## Упражнения

1. Откройте панель инструментов Собрание по сети (Online Meeting) и закрепите ее у верхней границы окна Access.
2. Добавьте новое меню в конец строки меню.
3. Присвойте созданному меню новое имя — Контакты.
4. Назначьте интерфейсному объекту гиперссылку, указывающую на Web-сайт издательства, которое выпустило оригинальную версию данной книги (<http://www.sampublishing.com>).
5. Установите посредством NetMeeting соединение с компьютером пользователя в Internet либо в пределах корпоративной сети.

## 2-й час

# Познакомимся с VBA



Пищевая сода, мука, соль, сахар, вода, яйца, молоко и т.д. — основные составляющие множества привычных нам обеденных блюд (скажем, хлеба). На этом занятии вам придется войти в образ пекаря или кондитера. Я научу вас тому, как, пользуясь некоторым набором ингредиентов, готовить вполне "съедобные" программы. Решаете ли вы задачи автоматизации повседневных действий (скажем, пополнения личной адресной книги) либо боретесь с более сложными проблемами (например, расчетом зарплаты для сотрудников своей компании) — Access способна прийти на помощь. Со всеми деталями, о которых будет рассказано на этом занятии, придется сталкиваться каждый раз, когда вы решите обратиться к услугам Access.

Чтобы написать любую, даже самую простую, программу для Access, вы должны знать о пяти составляющих. Ниже, в этой и следующих главах, мы их рассмотрим — основы синтаксиса языка программирования Visual Basic for Applications (VBA), служебные слова и операторы, поддерживаемые Access, правила объявления переменных, способы создания функций и подпрограмм, приемы построения пользовательских типов данных.

Основные темы занятия.

- Синтаксис языка программирования Access VBA.
- Как пользоваться операторами.
- Объявление переменных и их применение для хранения данных в программе.
- Выражения языка.

## Знакомство с Access VBA

Access VBA — это язык программирования. Или в более широком смысле — просто язык. Осознание данного факта не может не радовать, поскольку каждый из нас в той или иной степени владеет, по меньшей мере, одним языком (скажем, русским). Поскольку Access VBA — язык, он, как и любой другой "уважающий себя" язык, диктует пользователю определенный набор основных правил. Такие правила в академической терминологии называются грамматикой. Вам наверняка знакомо слово *грамматика*, но программисты чаще говорят *синтаксис*, поэтому мы, следуя традиции, будем поступать так же.

Каждый язык характеризуется тем или иным синтаксисом, или множеством правил, которые его определяют. Access VBA охватывает несколько категорий языковых объектов.

Языки программирования в первую очередь воспринимаются и применяются людьми, во вторую — компьютерами; затем может возникать новая цепочка взаимодействий между языком и его пользователями.

Первым делом человек с помощью компьютера решает задачу на соответствующем языке программирования. Язык программирования часто выглядит неким подмножеством английского языка с собственными грамматическими правилами. (Существует ряд исключений — например, машинный код или язык ассемблера. Хотя такие языки все еще находят широкое применение, они гораздо более сложны для освоения и практического использования в сравнении с языками высокого уровня, подобными VBA.)

Текст, написанный в соответствии с правилами синтаксиса языка программирования, обычно называют *кодом*. После того как код создан, Access проверяет его правильность и преобразует в такую форму, в которой он становится понятным компьютеру. Вот почему мы причисляем компьютеры к разряду пользователей второго эшелона.

Наконец, написанный код поступает в распоряжение других людей, компьютеров или человеко-машинных систем — состав конкретного сочетания пользователей зависит от особенностей задачи или способов ее реализации. Операционная система Windows 98 — это набор программ, используемых и компьютерами, и людьми. Предложим другой пример: как-то мне пришлось создавать программную систему для Access, назначением которой было периодическое обновление данных посредством загрузки их с внешнего сервера. Процесс был автоматизирован целиком. Операционная система Windows NT с помощью программы Task Scheduler (Планировщик заданий) запускала приложение Access в соответствии с указанным расписанием; таким образом, единственным потребителем кода оказывался компьютер, выполнявший задачу.

Access "знает", как обращаться с написанным вами кодом. Если правила программирования соблюдены, система обязательно поймет, что именно вы имели в виду. Существует несколько моментов, которые необходимо понимать, чтобы суметь благополучно донести до сведения Access свои мысли и намерения. В оставшейся части этой главы мы рассмотрим основные "кирпичики", из которых строится здание программы для Access.

## Служебные слова Access VBA

Access оперирует рядом служебных слов. Следует отметить, что объем словаря системы не очень велик. Если сравнивать Access с живым человеком, то уместным — с точки зрения богатства языка — окажется сопоставление с пятилетним ребенком.

Хорошая новость — чтобы успешно общаться с Access, не понадобится заучивать слишком много слов. Но есть и плохая — число сочетаний этих слов безгранично, и программисты все еще движутся вперед, пытаясь развить словарь системы.

Наилучший источник, перечисляющий все служебные слова Access, — оперативная справочная система. Тем не менее, я счел необходимым включить в текст главы таблицу, содержащую описание наиболее употребительных служебных слов VBA.

**Таблица 2.1. Служебные слова VBA**

Слово	Значение
As	Используется в качестве разделителя между наименованиями переменных и их типами в объявлениях переменных и аргументов функций
Binary	Используется для того, чтобы задать тип сравнения строк в операторе <b>Option Compare</b> и тип доступа к файлу в операторе <b>Open</b>
ByRef	Указывает, что переменная передается в процедуру по ссылке
ByVal	Указывает, что переменная передается в процедуру по значению
Call	Обозначает вызов подпрограммы
Dim	Используется с целью выделения памяти для хранения переменной
DO	Используется совместно со словами <b>while</b> и <b>Loop</b> для организации циклов
End	Обозначает команду завершения программы
Else	Применяется для обозначения альтернативного условия в условных операторах <b>If</b>
Error	Выявляет возникающие ошибки. Это функция, которая выдает сообщение, сообщая номер ошибки, а также условия, при которых эта ошибка возникает
Exit	Используется в конструкциях <b>Exit For</b> , <b>Exit Sub</b> , <b>Exit Function</b> и т.п. для обеспечения непосредственного выхода из текущего программного блока
False	Имеет значение ноль. Используется для проверки булевых выражений
For	Используется совместно со словом <b>Next</b> для организации циклов
Function	Применяется для объявления или определения функции
Get	Используется для чтения файла с диска или получения отдельной записи
If	Используется совместно со словосочетанием <b>End If</b> и необязательным выражением <b>Else</b> для построения условного оператора, проверяющего истинность выражения булевой логики
Let	Префикс при присвоении переменной или записи оператора
Loop	Употребляется в качестве завершающей фразы цикла <b>DO while</b>
Me	Способ ссылки на объект. Эквивалентен оператору <b>This</b> в C++ и <b>Self</b> в Object Pascal
Next	Употребляется в качестве завершающей фразы цикла <b>For</b>
ReDim	Применяется для повторного выделения памяти для элементов массива
Set	Используется с целью присвоить объект переменной
step	Определяет значение, на которое увеличивается переменная-счетчик на каждом шаге циклов <b>For Next</b> и <b>For Each</b>
stop	Обозначает команду приостановки работы, используемую при тестировании программы. (Затем выполнение программы может быть возобновлено)
Sub	Употребляется при объявлении или определении подпрограммы
Then	Используется в составе условного оператора <b>If</b>
True	Имеет значение -1. Используется для проверки булевых выражений
Type	Применяется при определении нового типа
While	Употребляется в составе конструкции <b>Do While</b> для организации цикла





При наборе в окне текстового редактора модуля Access служебное слово по умолчанию выделяется синим цветом. Незавершенные служебные слова отмечаются красным цветом, а остальной текст — черным

Несмотря на то, что в табл. 2.1 перечислены не все элементы словаря системы, в ней вы найдете описание большинства часто используемых слов, и во многих случаях их окажется вполне достаточно. При необходимости можно, конечно, обращаться к файлу оперативной справки, копировать оттуда необходимые служебные слова и вставлять их в текст программы, над которой трудитесь в данный момент. Но смеем предположить, что после изучения нескольких глав этой книги вы с удивлением обнаружите, насколько легко запоминаются многие конструкции языка программирования — просто в процессе работы, без специальных видимых усилий. Я поощряю именно такой стиль обучения — приобретать знания естественно, без зубрежки и заучивания.

Служебные слова сами по себе нельзя назвать особенно полезными. Применять их следует лишь в определенном контексте.

## Операторы и операнды Access

Служебные слова — только часть общей картины. Другое необходимое множество объектов — операторы и их операнды. Оператор обычно выглядит в виде единственного символа. Например, символом + обозначается оператор сложения. В табл. 2.2 приведен перечень операторов VBA, воспринимаемых Access.

### Новый термин

*Оператор* — это средство языка, обозначаемое определенным символом и служащее для выполнения единой операции. Например, оператор умножения \* применяется для умножения чисел.

Таблица 2.2. Операторы VBA

Обозначение	Наименование	Описание
#	Фунт	Применяется для построения условных директив компилятора
\$	Доллар	Используется для указания на возвращение функцией текстовой строки
&	Амперсанд	Оператор конкатенации ("склеивания") строк
*	Знак умножения	Оператор умножения
()	Круглые скобки	Используются для группирования членов арифметического или условного выражения либо в операциях над массивами
-	Минус	Оператор вычитания
+	Плюс	Оператор сложения
\	Знак деления	Оператор целочисленного деления
/	Знак деления	Оператор вещественного деления
'	Апостроф	Оператор комментария
"	Двойные кавычки	Используются в паре для обозначения строки текста

Обозначение	Наименование	Описание
=	Символ присваивания	Оператор присваивания левой части выражения значения правой части
.	Точка	Используется для доступа к членам объекта
<	Меньше	Оператор, проверяющий, меньше ли значение левой части выражения по сравнению со значением правой части
<=	Меньше или равно	Оператор сравнения
>	Больше	Оператор сравнения
>=	Больше или равно	Оператор сравнения
=	Равно	Оператор сравнения
<>	Не равно	Оператор сравнения

Каждый оператор должен использоваться совместно с одним или несколькими операндами. Большинство операторов предполагает наличие двух операндов — левого, т.е. расположенного слева от символа оператора, и правого. Такие операторы называются *бинарными*. Существуют *унарные* операторы, воздействующие на единственный операнд. Операторы других типов в Access VBA не применяются (хотя в языке C++, например, используются и *тернарные* операторы, работающие с тремя операндами).



Равенство — великолепный пример оператора, действие которого зависит от контекста. В некоторых случаях это оператор присваивания, а в других — оператор сравнения. Если знак = появляется там, где производится проверка, например в операторе Do While, он выполняет тест на эквивалентность. А если слева от знака равенства стоит переменная, то, вероятнее всего, = является оператором присваивания, который присваивает этой переменной значение, находящееся справа от знака.

#### Новый термин

*Операнд* — это объект воздействия оператора. Например, арифметическое выражение 5+3 содержит два операнда — 5 и 3 — и оператор сложения +. Оператор осуществляет сложение двух заданных операндов.

Оператор отрицания — Не (Not) — пример унарного оператора, который требует наличия единственного операнда. Оператор целочисленного деления, обозначаемый с помощью символа \, относится к разряду бинарных. (Таков же и оператор вещественного деления /.)

Язык Access VBA обладает тем неоспоримым преимуществом, что многие из используемых в нем операторов уже давно и хорошо вам знакомы — еще со школьной скамьи. Арифметические операторы, например, действуют совершенно так же, как вы и предполагаете. Если вам известно, как пользоваться калькулятором, с арифметическими операторами VBA вы несомненно справитесь.

## Данные: сведения, известные программе

Довольно образна такая аналогия: служебные слова и операторы языка программирования подобны атомам физического мира. Служебные слова и операторы — это мельчайшие части кода. Стоит присовокупить к ним данные, и вы получите выражение языка. Выражения подобны предложениям человеческого языка — они схожи с молекулами, состоящими из атомов.

**Выражение** — это аналог предложения естественного (например, английского или русского) языка. Мы с вами можем оперировать простыми предложениями (скажем, "Поддай книгу") или более сложными — наподобие того, которое вы сейчас читаете: Выражения, написанные в соответствии с правилами VBA, также могут быть короткими и простыми либо пространными и сложными.

Данные, которыми оперирует программа, хранятся в памяти компьютера в виде переменных. **Переменные** — это именованные фрагменты памяти, предназначенные для хранения данных определенного типа. **Тип** данных указывает на род информации, которую следует хранить, и диапазон изменения ее значений. • Чтобы написать выражение, потребуются, как минимум, переменная (т.е. имя, обозначающее место хранения объекта данных), оператор и значение. Прежде чем приступить к составлению выражений, необходимо усвоить, каким образом представляются данные.

В табл. 2.3 описаны типы данных, поддерживаемые Access VBA.

Таблица 2.3. Типы данных VBA

Наименование	Объем	Интервал допустимых значений
Byte	1 байт	От 0 до 255
Boolean	2 байта	True (истина) ИЛИ False (ложь)
Currency	8 байт	От -922337203685477.5808 до 922337203685477.5807
Date	8 байт	От 1 января 100 г. до 31 декабря 9999 г.
Decimal	12 байт	79228162514264337593543950335 без десятичной точки; либо +/-7.9 и 28 десятичных разрядов после точки
Double	8 байт	От -922337203685477.5808 до 922337203685477.5807
Integer	2 байта	От -32768 до 32767
Long	4 байта	От -2147483648 до 2147483647
Object	4 байта	Ссылка на любой объект
Single	4 байта	От -3.402823E38 до -1.401298E-45 для отрицательных величин и от 1.401298E-45 до 3.402823E38 для положительных величин
String	Произвольный	Длина строки — приблизительно до 2 млрд символов
Type	Произвольный	Пользовательский тип данных
Variant	16 байт	Любое числовое или символьное значение



Если необходимо сохранить какие-либо данные для использования их в программе, следует объявить соответствующую переменную. Ниже показан пример синтаксической конструкции для объявления переменной.

`Dim ИмяПеременной As ТипДанных`

Объявление переменной начинается со служебного слова Dim (обратитесь к табл. 2.1). За ним следует наименование (или как еще говорят — идентификатор) переменной. В качестве наименования переменной может использоваться любая после-

довательность символов алфавита и цифр (не более 255), начинающаяся с буквы. После наименования и служебного слова `As` указывается обозначение типа (см. табл. 2.3). Целесообразно давать переменным какие-либо осмысленные названия. Рекомендуем использовать целые слова. (Несмотря на то, что Access VBA формально позволяет применять в идентификаторах буквы национального алфавита, в практике реального программирования пользоваться подобной возможностью нежелательно. — *Прим. перев.*) Выбранный тип должен соответствовать природе объекта данных.

Далее предлагаем попробовать объявить несколько переменных. Чтобы объявить переменную, предназначенную для хранения имени человека, можно записать:

```
Dim Name As String
```

Если необходима переменная, которая должна содержать значение даты рождения, допустимо такое объявление:

```
Dim BirthDate As Date
```

Попытайтесь припомнить любые данные, с которыми вам приходится иметь дело, и записать на бумаге выражения объявлений переменных для их хранения. При этом вы невольно воспользуетесь основными языковыми конструкциями, входящими в состав объявлений переменных.

## А теперь все вместе

Выражения — наименьшие части кода, которые Access способна воспринимать и использовать для решения задачи. Вспомните материал предыдущего раздела: объявления переменных — примеры выражений.

Употребляя служебное слово `Dim` при объявлении переменных, вы уже реально приступили к самостоятельному построению выражений. Хотя переменные сами по себе еще не решают задачу, они выступают важным звеном в общем технологическом цикле программирования.

## Арифметические операторы

Арифметические операторы могут быть и простыми, и достаточно сложными, отвечающими условиям задачи и намерениям программиста. Листинг 2.1 демонстрирует примеры использования операторов сложения, вычитания, присваивания, деления и умножения.

**Листинг 2.1.** Примеры арифметических выражений с использованием переменных, операторов и операндов

```
1: Dim A As Integer
2: A = 0
3: A = 5 + 3
4: A = 6 \ 2
5: A = 5 - 3
6: A = 16 * 16
```

В строке 1 объявляется переменная `A`. Строка 2 иллюстрирует присвоение переменной `A` значения 0. В правой части выражения находится значение 0, а слева от оператора присваивания (`=`) расположен идентификатор переменной — `A`. В процессе присвоения в переменную, стоящую слева от оператора, заносится значение, указанное справа. Левую часть операторов присваивания принято называть *lvalue*, а правую — *rvalue*. При выполнении операций присваивания в качестве *rvalue* может ис-

пользоваться константа (скажем, 5), значение другой переменной или результат операции. Строка 3 демонстрирует присваивание результата операции сложения. Вначале Access выполняет сложение  $(5 + 3)$ , а затем его результат присваивает *lvalue*-переменной А. Таким образом, после выполнения строки 3 кода в переменной А будет храниться число 8. В процессе операции, определяемой строкой 4, переменной А будет присвоено значение 3. Выполнение следующей строки — 5 — приведет к занесению в А значения 2, а после завершения операции, указанной в строке 6, А приобретет значение 196.

Листинг 2.1 демонстрирует основы арифметических операций. Хотя подобный набор строк кода вряд ли может иметь смысл в практике реального программирования, он полезен с методической точки зрения. Впрочем, вы можете запросто объединить несколько операторов в единую цепочку. Например,  $A = 5 + 3 - 6 \setminus 2 + 16 * 16$  — вполне допустимое выражение. Однако будет лучше, если вы постараетесь упрощать выражения.

Если арифметическое выражение содержит много переменных, констант и операторов, целесообразно применять круглые скобки, которые помогут недвусмысленно разъяснить системе ваши намерения. Например, выражение  $A = 5 + 3 - 6 \setminus 2 + 16 * 16$ , — это, на самом деле,  $A = (5 + 3) \cdot (6 \setminus 2) + (16 * 16)$ .

Арифметические операторы Access подчиняются тем же правилам приоритета выполнения, что и в традиционной математике. Операции умножения и деления имеют более высокий приоритет (т.е. выполняются раньше), чем сложение и вычитание.

## Операторы сравнения

Операторы сравнения —  $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $=$  и  $o$  — были упомянуты ранее, в табл. 2.2. Все они относятся к разряду бинарных. Каждый из операторов сравнения предполагает наличие значений *lvalue* и *rvalue*. Результат выполнения оператора сравнения — величина типа Boolean, принимающая одно из значений, True (истина) или False (ложь).

### Новый термин

Переменные типа Boolean способны хранить одно из двух значений — True или False. Своим происхождением и именем они обязаны английскому математику Джорджу Булю (George Boole), разработавшему основы алгебры логики, именованной позднее *булевой*. Булева алгебра устанавливает законы поведения объектов, которые способны принимать два состояния — True ИЛИ False.

Листинг 2.2 демонстрирует, каким образом можно присвоить результат выполнения оператора сравнения переменной типа Boolean.

### Листинг 2.2. Примеры операторов сравнения

```
1: Dim Result As Boolean
2: Result = 5 > 4
3: Result = 6 = 7
4: Result = 4 <= 3
```



Смысл некоторых языковых объектов VBA зависит от контекста их употребления. Так, например, строку 3 листинга 2.2 следует трактовать таким образом: результат выполнения оператора сравнения ( $=$ ) двух чисел, 6 и 7 (а это значение False) присваивается ( $=$ ) переменной Result, которая ранее была объявлена как булева.

Строка 1 содержит объявление переменной Result типа Boolean. В строке 2 осуществляется проверка, действительно ли число 5 превосходит число 4. Результат, True (истина), присваивается переменной Result. Итог False (ложь) проверки равенства чисел 6 и 7 присваивается переменной после выполнения строки 3 кода. Осуществляя операцию сравнения, содержащуюся в строке 4, система обнаружит, что число 4 на самом деле не больше числа 7, и поэтому присвоит переменной Result значение True.

## Логические операторы

Логические операторы, перечисленные в табл. 2.4, применяются при выполнении действий над булевыми величинами. Логические выражения — как и арифметические — могут быть довольно простыми или настолько сложными, насколько это необходимо. Некоторые примеры логических операторов приведены ниже, в тексте листинга 2.3.

**Таблица 2.4. Логические операторы VBA**

Обозначение	Описание
And	Бинарный оператор; если оба его операнда равны True, результат также равен True; в противном случае результат равен False
Eqv	Оператор побитового сравнения
Imp	Оператор импликации (бинарной логической связи)
Not	Унарный оператор отрицания (например, Not True есть False, а Not False — True)
Or	Бинарный оператор; если хотя бы один из его операндов равен True, то результат также равен True
Xor	Бинарный оператор; результат равен True, если выражения левой (lvalue) и правой (rvalue) частей различны

**Листинг 2.3. Примеры логических операторов**

```
1: Dim Test As Boolean
2: Test = True And False
3: Test = False Or False
4: Test = Not (5 > 4)
5: Test = ((MyValue > 3.1459) And (YourValue <= (6 + 5))) Or True
```

В строке 1 листинга 2.3 объявляется булева переменная Test. После выполнения строки 2 Test принимает значение False. Результат вычисления выражения False Or False, приведенного в строке 3, равен False. Выражение (5 > 4) строки 4 истинно, но его отрицание дает итог False, который и присваивается переменной Test. Результат вычисления выражения строки 5 всегда истинный, независимо от значений числовых переменных MyValue и YourValue, поскольку выражение завершается конструкцией Or True.



Старайтесь не создавать слишком сложных логических выражений — с каждым вновь добавленным оператором они обнаруживают тенденцию выйти за пределы понимания. Вычисляйте каждое элементарное логическое выражение в отдельной строке кода.

Результаты выполнения логических операторов обычно представляют в виде *таблиц истинности*, которые вы сможете отыскать в оперативной справочной системе Access. Таблицы истинности помогут проверять создаваемые логические выражения. Конечно, не помешают и знания основ математической логики (например автор данной книги, будучи студентом университета штата Мичиган, изучал курс дискретной математики). При построении логических выражений целесообразно по возможности их упрощать и использовать круглые скобки, уточняющие смысл высказывания.

## Операторы конкатенации строк

К операциям со строками придется обращаться очень часто. String — это тип данных, позволяющий хранить непрерывные последовательности символов. При конкатенации (или сложении строк) одна строка присоединяется в конец другой.

Существует два оператора сложения строк. Они обозначаются символами *плюс* (+) и *амперсанд* (&). Оба оператора бинарны. Просмотрите фрагмент кода, представленный листингом 2.4.

### Листинг 2.4. Примеры сложения строк

```
1: Dim A As String
2: A = "Здравствуй, " + "мир!"
3: A = "Здравствуй, " & "мир!"
```

В строке 1 объявляется переменная A типа string. В строках 2 и 3 используются операторы сложения, "склеивающие" две строки ("Здравствуй, " и "мир!") в единое целое, и результат — строка "Здравствуй, мир!" — присваивается переменной A. В этом примере конечные результаты применения обоих операторов сложения строк — + и & — совершенно равноценны.

Отличие оператора & состоит в том, что он способен превращать значения любого встроенного типа данных в строку. Поэтому сложение строки "Мне уже " с числом 5 посредством оператора & даст в результате строку "Мне уже 5". Оператор & осуществляет неявное преобразование значений всех встроенных типов данных в значение типа String. Примеры вы найдете в листинге 2.5.

#### Новый термин

*Встроенным* называется тип данных, изначально определенный в языке программирования (в частности, Access VBA).

### Листинг 2.5. Примеры использования оператора &

```
1: Dim A As String
2: A = "Мне уже " & 5
3: MsgBox A
4: A = "Мне уже " + Str( 5 )
5: MsgBox A
```

Результаты выполнения строк 2 и 4 равносильны. В обоих случаях переменной A присваивается строка "Мне уже 5". Символ амперсанда исключает необходимость обращения к встроенной функции Str ( ), осуществляющей явное преобразование числа в строку. (Подробнее о встроенных функциях см. главу "8-й час. Декомпозиция задач".) Оператор конкатенации строк, обозначаемый символом &, процесс преобразовании типов проводит самостоятельно.

# Шаг вперед

Служебные слова, операторы и переменные — обязательные компоненты любых программ. Помимо них при создании программы вам понадобятся и другие "строительные" материалы — функции и подпрограммы, пользовательские типы данных, классы и их объекты.

Ознакомившись с материалом главы 8, вы научитесь создавать и использовать подпрограммы и функции. Из главы "10-й час. Как использовать готовые решения" вы узнаете об обширной библиотеке стандартных функций Microsoft Access и о способах их применения. Глава "11-й час. От сложного к простому: создание собственных типов данных" рассказывает о тонкостях определения пользовательских типов данных. А главой "21-й час. Основы программирования классов" начинается раздел, посвященный вопросам объектно-ориентированного программирования в среде Access.

Научившись создавать функции и подпрограммы, вы сможете успешно решать многие практические задачи бизнеса. Но чтобы получить доступ к самым мощным и эффективным средствам программирования для Access, вам, однако, понадобятся знания в области объектно-ориентированного программирования, владение такими понятиями, как класс, объект и компонент.

## Резюме

Решение задачи можно представить в виде схемы, соответствующей принципу "от простого к сложному": выражение → блок кода → готовая программа. В ходе этого занятия вы имели возможность ознакомиться с элементами низшего структурного уровня программы — выражениями. Выражения обычно состоят из служебных слов, переменных и операторов. На основе выражений строятся все программы.

На этом занятии вы научились пользоваться операторами (сведения об арифметических операторах, собственно говоря, наверняка не стали для вас большим откровением). Мы рассмотрели наиболее употребительные служебные слова Access VBA и рассказали о способах объявления и инициализации переменных.

*Переменная* — это именованная область компьютерной памяти, предназначенная для хранения объекта данных определенного типа. Выбор типа данных зависит от условий задачи и путей ее эффективной практической реализации. Наиболее часто используемые типы данных — String и Integer. Таким образом, вы получили сведения об основных операторах для работы с числами и строками.

Прежде чем приступить к решению практических задач, целесообразно ознакомиться со способами создания функций и подпрограмм.

## Вопросы и ответы

Вопрос. Мне необходимо объявить переменную для хранения чисел с плавающей точкой. Что следует делать?

Ответ. К счастью, правила объявления переменных в VBA едины и просты (вспомните материал раздела "Данные: сведения, известные программе"). Начните выражение со служебного слова Dim, затем укажите имя переменной, введите служебное слово As и, наконец, задайте тип Double.

Вопрос. Можно ли преобразовать строку символов (например, такую как "02/12/1999") в значение типа Date?

Ответ. Да. Access VBA предлагает целый ряд встроенных функций, предназначенных для взаимного преобразования данных различных типов. Функция Format — наиболее



универсальна. Для решения же вашей конкретной задачи удобнее воспользоваться функцией CDate. Код в этом случае мог бы выглядеть так: CDate ("02/12/1999").

**Вопрос. Каким образом программа может узнать и использовать значения текущих даты и времени?**

**Ответ.** В составе Access VBA есть функции, возвращающие системные дату и время: Date и Time соответственно.

**Вопрос. Могли бы вы порекомендовать удобный способ запоминания всех служебных слов и операторов языка?**

**Ответ.** Многие из них вам уже хорошо известны. Если вы пользовались калькулятором, то наверняка знакомы со многими арифметическими операторами. По мере необходимости обращайтесь за дополнительной информацией и примерами к оперативной справочной системе Access. Совершенствуйте свои знания на практике.

## Задания

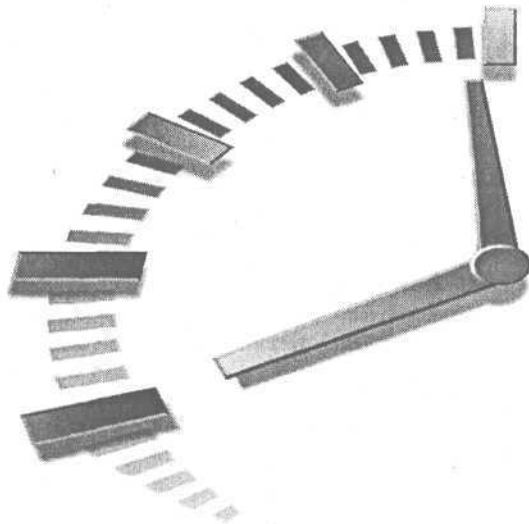
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Каков наиболее предпочтительный тип данных для хранения чисел с плавающей точкой?
2. К какому типу должна относиться переменная, предназначенная для присваивания результата вещественного деления?
3. В чем состоит отличие операторов (+ и &) сложения строк?
4. Чем отличается объявление переменной от ее инициализации?
5. Какое служебное слово применяется для объявления и инициализации постоянных значений (констант)?

## Упражнения

1. Напишите выражение объявления переменной для хранения даты и инициализации этой переменной значением текущей даты.
2. Составьте выражение для вычисления суммы числа и того же числа, помноженного на дробь.
3. Напишите выражение для сложения строк, выполняющее неявное преобразование типов данных.



## 3-й час

# Как программа работает

## с данными

Во всех программах используются определенные данные. Access — это приложение, которое обращается к данным двух разновидностей. Информацию первого рода вы можете сохранить в базе данных, и она останется в том же виде даже в случае, когда Access выгружается из памяти компьютера. Кроме того, программа оперирует некоторыми данными в процессе своего выполнения, и они не сохраняются в памяти после завершения работы.

Данные можно записывать на магнитные или иные носители, поддерживающие возможности *долговременного* хранения, — жесткие диски, дискеты, компакт-диски, магнитные ленты и т.п. Оперативное запоминающее устройство (ОЗУ) — это, в свою очередь, пример *временной* компьютерной памяти.

На этом занятии речь пойдет о хранении данных в ОЗУ компьютера. Результаты всех вычислений первым делом сохраняются именно в оперативном запоминающем устройстве. Только затем вы сможете поместить нужную информацию на магнитный носитель для последующего постоянного использования.

Большую часть забот, связанных с управлением ОЗУ, принимают на себя Access и операционная система Windows. Но это не значит, что на вашу долю не остается ничего — данными, с которыми работает ваша собственная программа, должны манипулировать вы сами, и именно о таких вещах мы расскажем в этой главе.

Основные темы занятия.

- Подробнее об объявлении переменных.
- Как код обращается с данными.
- Всплывающие подсказки в окне редактора модуля Access.
- Применение режима непосредственного исполнения кода.
- Как пользоваться окнами просмотра результатов.

# О компьютерной памяти

Для программиста, работающего в среде Access, одинаково важны ресурсы долговременной и оперативной памяти компьютера. Если ваша программа для Access бездействует, файл базы данных сохраняется на внешнем носителе, но если программа активизируется, информация из базы данных порциями копируется в ОЗУ.

Для хранения данных на внешних носителях используются различные физические эффекты, чаще всего — магнитные. ОЗУ же представляет собой набор электрических микропереключателей. Современные компьютеры умеют обращаться с магнитными носителями и электрическими полупроводниковыми переключателями, получая доступ к различным устройствам долговременного хранения данных и оперативной памяти. Программы операционной системы считывают информацию из ОЗУ и дорожек дисков. Но подлинная "рабочая лошадка", везущая "воз" ответственности за все происходящее, — это микропроцессор. Процессор вашего компьютера — такой как, скажем, Intel Pentium 800 — управляет действиями по считыванию данных из оперативной памяти и записи информации в нее.

Набор электрических проводников, по которым "путешествуют" информационные сигналы, носит название *шины*. Адресная шина микропроцессора хранит сведения о том, в каком месте памяти размещены данные. Между физическим адресом памяти и доступными программисту способами доступа к ячейке данных существуют определенные зависимости, однако этот вопрос выходит за рамки проблем, рассмотренных в данной книге.

Чтобы воспользоваться данными в памяти, процессору нужны соответствующие инструкции и адрес. В роли "советчика" выступает код, который вы написали, а также компилятор Access. Излагая свои мысли в виде кода, вы придерживаетесь правил языка программирования Visual Basic for Applications (VBA). В составе Access имеется компилятор, превращающий написанное вами в адреса и данные, которые компьютер способен верно воспринимать.

## Новый термин

*Компилятор* — это программа, которая преобразует текст, написанный на языке программирования, в машинные команды, доступные компьютеру.

Компилятор — необходимый посредник между текстом на языке программирования и физической оперативной памятью компьютера, откуда процессор должен черпать инструкции, регламентирующие его работу. Оперативная память — как раз то место, где происходят самые важные события. Обработывая написанную вами команду объявления переменной, компилятор назначает этой переменной адрес памяти.

## Объявление переменных

Объявление переменной сводится к написанию строки кода, содержащей имя переменной и ее тип. Из материала главы "2-й час. Познакомимся с VBA" вы уже узнали, как строить выражения для объявления переменных с помощью служебного слова Dim, но существует и несколько других форматов объявления, которые необходимо знать. Напомним, как следует объявлять переменные с помощью инструкции Dim:

`Dim ИмяПеременной As ТипДанных`

ИмяПеременной — это значимое название переменной, на которое можно ссылаться в программе, а ТипДанных — существующий тип данных, подходящий для размещения информации определенного рода.

Вы уже достаточно хорошо осведомлены относительно способа объявления переменных с помощью служебного слова Dim, поэтому теперь стоит узнать о новых форматах объявлений. Существует четыре основных типа объявлений. Им соответствуют служебные слова Dim, ReDim, Const и Global. В следующих разделах мы расскажем более подробно о каждой из названных конструкций объявлений, проиллюстрировав изложенную информацию примерами их использования.

## Переменные формата Dim

Формат объявления на основе служебного слова Dim является наиболее употребительным. Посредством Dim вы указываете компилятору на необходимость отведения блока памяти для хранения значений переменной заданного типа.

Например, для записи в память числа типа Long необходимо 4 байта (или 32 бита). Такой объем позволяет разместить в переменной число из диапазона, простирающегося приблизительно от -2 млрд до 2 млрд.



1 байт состоит из 8 бит. Поэтому под переменную 2-байтового типа отводится 16 бит. Законы комбинаторной математики гласят, что для подсчета количества различных комбинаций нулей и единиц в последовательности битов заданной длины достаточно возвести 2 в степень, равную длине последовательности (в нашем примере — 16), т.е.  $2^{16} = 65536$ . Принимая во внимание необходимость хранения как положительных, так и отрицательных значений, получим диапазон допустимых значений от -32768 до 32767. Простейший способ получения сведений об ограничениях, присущих каждому типу данных, — обратиться к оперативной справочной системе Access 2002.



Для начала целесообразно запомнить хотя бы такие типы данных: String (для хранения символьных последовательностей), Integer (для небольших целых чисел) и Double (для вещественных чисел).

Хотя не так уж и важно знать, сколько именно памяти отводится под переменную того или иного типа, иметь представление о наборе доступных типов и соответствующих им диапазонов допустимых значений необходимо. Указанные сведения находятся в статьях оперативной справочной системы Access.

Наиболее широкое применение находят такие типы данных — String, Integer и Double. String предназначен для хранения фрагментов текста, Integer — целых чисел в диапазоне от -32768 до 32767, а Double — очень малых или больших чисел, отрицательных или положительных, а также чисел с плавающей точкой.

## Переменные формата ReDim

Тип объявления посредством служебного слова ReDim применяется при работе с массивами данных. *Массив* — это последовательность частиц информации, занимающая в памяти непрерывный ряд ячеек, каждый элемент которой адресуется с помощью индекса. Массивы полезны в том случае, когда вы имеете дело с несколькими значениями одного типа. Например, если необходимо объявить переменные для хранения десяти номеров лицевых счетов, удобно воспользоваться массивом значений типа String. Ниже приведен пример, иллюстрирующий синтаксис объявления с помощью служебного слова ReDim:

ReDim ИмяМассиваПеременных(ЧислоЭлементов) As ТипДанных

Вначале вводится слово ReDim, за ним следует имя (идентификатор) массива, которое сопровождается числом, обозначающим количество элементов. Это число заключается в круглые скобки. Конструкция завершается знакомым вам словом AS, после которого указывается обозначение типа данных. Следующая строка кода — пример реального объявления массива с помощью ReDim:

```
ReDim Counters( 10 ) As String
```

Трактовать ее следует как инструкцию объявления массива из десяти переменных символьного типа, причем число их может меняться. Слово ReDim употребляется исключительно для объявления массивов — с его помощью вы уведомляете Access, что число элементов массива может варьироваться.



Обратите внимание, что для объявления массива допускается использование и служебного слова Dim, но при этом в дальнейшем вы не сможете изменить число элементов массива. Применение же ReDim гарантирует последующую возможность увеличения или уменьшения количества элементов.

Массивы имеют большое значение при работе с несколькими однотипными элементами данных. Служебное слово ReDim — это первое, что следует вспомнить, если вы решите обратиться к возможностям построения массивов. Подробнее о массивах см. главу “12-й час. Управление данными переменного объема”.

## Постоянные значения

Dim используется при объявлении объектов данных, значения которых могут изменяться во время выполнения программы. А служебное слово Const подразумевает совершенно противоположное — содержимое объявленной с его помощью величины всегда остается неизменным.



Окружающий мир предлагает убедительные и наглядные примеры постоянных величин. Число  $\pi$ , как известно, приблизительно равно 3.14159. В случае необходимости использования числа  $\pi$  в вычислениях удобно объявить в программе соответствующую величину с помощью слова Const. Предлагаем описание синтаксиса подобной конструкции:

```
Const ИмяПостояннойВеличины = Значение
```

Служебное слово Const сопровождается наименованием постоянной величины, за которым следуют оператор присваивания и значение соответствующего типа. Компилятор самостоятельно определяет тип значения, поэтому вам не нужно указывать его явно. Определение числа  $\pi$  в тексте программы могло бы выглядеть следующим образом:

```
Const PI = 3.14159
```



В сообществе программистов бытует практика называть константы именами, состоящими целиком из прописных букв. Access формально этого не требует, но подобная схема именования достаточно удобна. Если вы с нею согласны, придерживайтесь такого правила последовательно и неукоснительно. Ваш код приобретет профессиональный вид и станет понятен другим.

В практике программирования часто возникают неприятные ситуации, связанные с непредвиденными изменениями значений переменных. Но константы не подвержены подобной опасности, поскольку в ходе выполнения программы их значения — по определению — остаются постоянными. Константы — это величины, на которые всегда можно положиться. Применяйте их настолько активно, насколько это возможно и целесообразно. Только оставив за плечами сложные программные проекты, вы сможете по-настоящему оценить, что такое надежность кода.

## Глобальные переменные

Синтаксис

Переменные, объявленные посредством служебного слова `Global`, могут адресоваться в любом месте кода программы. Синтаксис объявления глобальных переменных таков:

`Global ИмяГлобальнойПеременной As ТипДанных`

Объявляя собственную глобальную переменную, дайте ей подходящее имя и укажите нужный тип данных. Значение переменной `Global` может изменяться в любой строке кода программы. Программа, содержащая большое число глобальных переменных, становится трудной для восприятия, поэтому имеет смысл использовать их только в тех случаях, когда это действительно необходимо.

В ходе программирования конкретного участка кода вы задумываетесь над текущими значениями переменных. Если переменная глобальна, ее значение может быть изменено командами другого блока кода. Поэтому ваши исходные предположения подвержены угрозе непредсказуемого нарушения. Чтобы уменьшить вероятность случайного искажения содержимого глобальных переменных, вы как программист должны предпринять дополнительные меры.

## Присваивание данных и вычисления

Переменные в программе могут использоваться в качестве обоих операндов (левого или правого) различных операторов, а также параметров функций и подпрограмм. Если переменная находится слева (справа) от символа оператора, ее называют левым (правым) операндом.

Новый термин

*Функция* — это содержащий одну или несколько команд блок кода, который адресуется по имени. Функция обычно возвращает значение определенного типа.

Новый термин

*Подпрограмма* (или *процедура*) — это также блок кода, который содержит одну или несколько команд. Однако подпрограмма не может возвращать значения.

Новый термин

*Параметр* (или *аргумент*) — это переменная, которая передается в функцию или процедуру. Параметр может принимать участие в вычислениях и изменяться командами функции (процедуры).

Оператор присваивания обозначается символом равенства (=) и заносит в левый операнд-переменную значение правого операнда-выражения. В тексте листинга 3.1 приведено пять примеров операторов присваивания и вычислений.

### Листинг 3.1. Примеры операторов присваивания и вычислений

```
1: Dim SquareOfCircle As Double
2: Dim Radius As Double
3: Const PI = 3.14159
4: Radius = 10
5: SquareOfCircle = PI * Radius ^ 2
```



Строка кода — это буквально одна строка текста, но выражение может быть простым или составным, т.е., другими словами, оно может содержать более одной строки текста.

Строка 1 листинга 3.1 служит для объявления переменной `SquareOfCircle` типа `Double`. В строке 2 объявляется переменная `Radius` того же типа. Строка 3 содержит объявление константы `PI`, равной числу  $\pi$ . В строке 4 переменная `Radius` инициализируется значением 10. В строке 5 вычисляется значение площади круга заданного радиуса ( $\text{PI} * \text{Radius}^2$ ), и результат присваивается переменной `SquareOfCircle`. (Символ  $\wedge$  (знак вставки, или, как его называют в обиходе, крышки) — это обозначение оператора возведения в степень.)



Можно объявить несколько переменных одного типа посредством единственного выражения, в котором имена переменных отделяются запятыми. Например, `Dim SquareOfCircle, Radius As Double`.

Часто полагают, что все переменные, объявленные таким образом, имеют один и тот же тип, но это неверно. В предыдущем примере можно предположить, что и `SquareOfCircle`, и `Radius` имеют тип `Double`. На самом деле переменная `Radius` имеет тип `Double`, а `SquareOfCircle` — тип `Variant`.



Объявление постоянной величины с помощью служебного слова `Const`, сопровождаемое присваиванием ей исходного значения, называют *инициализацией*.

Многие такие термины способны ввести в заблуждение, но они изобретены программистами с целью ясного и лаконичного выражения мыслей. Хотя вы не обязаны освоить всю терминологию, все же лучше ее знать — особенно в том случае, если приходится часто общаться с коллегами-профессионалами.

Строки 3, 4 и 5 листинга 3.1 иллюстрируют операции присваивания. Элементы строки 5 — переменные `PI`, `Radius` и константа-литерал `2` — это члены вычисляемого выражения.

## Будьте точны и до конца откровенны

Очень важно, чтобы написанный вами код в процессе выполнения "вел себя" именно так, как вы предполагали. Существуют два правила, которым необходимо следовать: старайтесь не применять переменные типа `Variant` и не пользоваться неявно определенными переменными.

## Тип данных Variant

О типе данных Variant вкратце рассказывалось в главе 2. Variant — встроенный тип данных, применение которого приведет к лишним накладным расходам. Встретив в тексте программы объявление переменной типа Variant, компилятор присоединяет дополнительный код, который во время исполнения программы должен определить действительный тип объекта данных. Переменные Variant находят применение в коде стандартных библиотек Microsoft и объектов COM. Впрочем, в большинстве случаев предпочтительно избегать использования переменных variant и употреблять точные и однозначные объявления. В этом случае ваши намерения будут выражены наиболее четко и недвусмысленно, обеспечивая гарантии соответствия текущих значений типам данных и допустимым диапазонам их изменения.

Новый термин

*COM* (сокращение от Common Object Model — общая модель объектов) — это стандарт построения составных типов данных, которые могут быть оформлены в виде отдельных самостоятельных блоков исполняемого кода. Часто объекты COM называют компонентами.

## Неявное определение переменных

Access допускает использование переменных, определенных неявно. Неявное определение подразумевает, что переменная просто, без дополнительных объявлений, вводится в точке ее использования. Листинг 3.2 представляет собой исправленный вариант листинга 3.1 — все переменные теперь определяются неявно.

Листинг 3.2. Пример неявного определения переменных

```
1: Const PI = 3.14159
2: Radius = 10
3: SquareOfCircle = PI * Radius ^ 2
```

Обратите внимание, что в тексте листинга отсутствуют какие бы то ни было выражения объявлений. Переменные Radius и SquareOfCircle вводятся в действие без дополнительных разъяснений относительно их типов. На первый взгляд, ничего негативного в таком подходе нет — программа, как и прежде, справится с задачей вычисления площади круга. Но если код приобретет более внушительные размеры — скажем, до нескольких сотен строк, — ситуация может резко измениться к худшему. Вы должны принять к сведению, что подобная практика может осуществляться, но постарайтесь избежать ее использования.

В первой строке модуля программы, как правило, должна содержаться команда Option Explicit. Пользуйтесь ею всегда. Она гарантирует вам и коллегам, работающим с вашим кодом, невозможность введения в обращение неявно определенных переменных.

Новый термин

*Модуль* — это отдельный файл, содержащий текст на языке программирования Access VBA.

Если, введя в начало текста программы команду Option Explicit, вы затем попытаетесь воспользоваться неявно определенной переменной, Access откроет окно сообщения об ошибке, показанное на рис. 3.1.



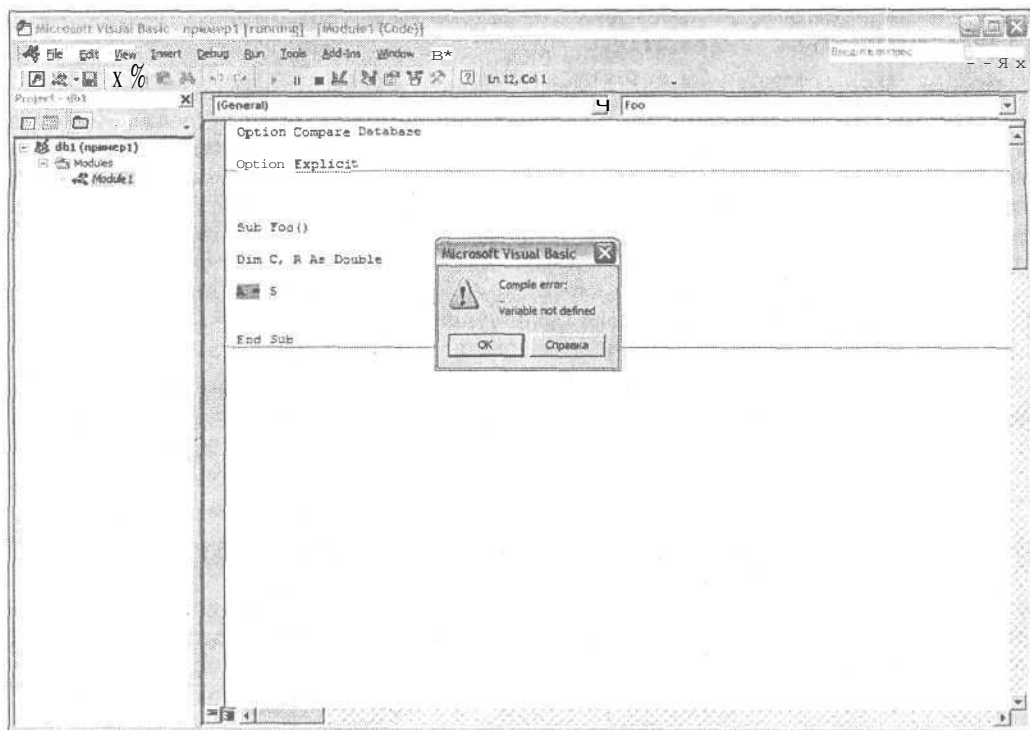


Рис. 3.1. При попытке неявного задания переменной система отображает окно сообщения об ошибке

Конечно, легко воздержаться от использования неявно определенных переменных в том случае, если программа невелика. Но решение любой серьезной задачи требует написания немалого количества строк кода, и тенденция к его росту — величина постоянная. Если ваша программа достаточно пространна, важно привлечь на помощь столько полезных средств, сколько окажется возможным. Одно из них — явное задание типов переменных.

## Всплывающие подсказки в окне редактора модуля Access

Теперь вы уже знаете, как объявлять переменные и составлять выражения. Прежде чем перейти к более сложным темам, рассмотрим один важный инструмент среды программирования Access, который поможет вам в управлении переменными.

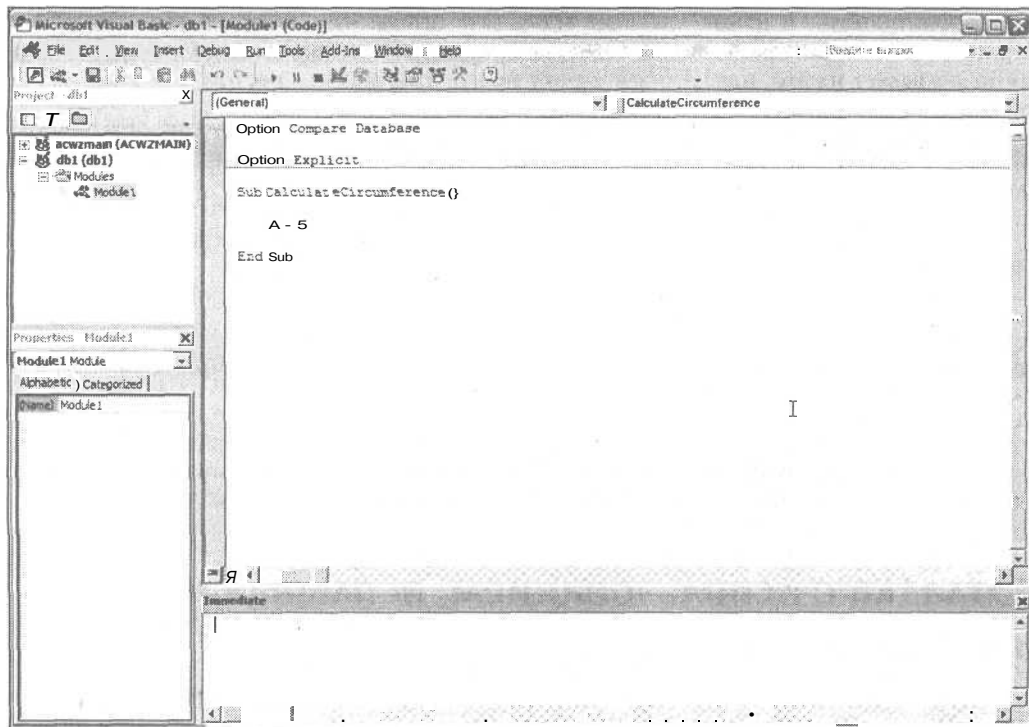
*Интегрированная среда разработки Access* (Integrated Development Environment, или IDE) имеет в своем составе механизм *всплывающих подсказок*. Всплывающая подсказка — это небольшое окно сообщения, которое автоматически появляется на экране всякий раз, когда курсор мыши оказывается над фрагментом текста с наименованием переменной. Механизм доступен только в режиме отладки. Итак, некоторые краткие пояснения.

Код сохраняется в файле модуля. *Модуль* — это файл Access, содержащий законченный блок текста на языке VBA. Почти весь код программы — за исключением выражений, подобных Option Explicit, — размещается внутри функций или подпро-

грамм. (Более подробно функции и подпрограммы будут рассмотрены в главе "8-й час. Декомпозиция задач".) Сразу после написания хотя бы одной функции или подпрограммы можно подвергнуть проверке текст строка за строкой.

## Тестирование кода

*Режим отладки* — это способ выполнения программы с возможностью одновременного просмотра состояний объектов программы. Подобную задачу можно выполнить, не покидая окна редактора Microsoft Visual Basic. Поскольку VBA, помимо Access, находит применение в составе других офисных продуктов компании (например Excel), Microsoft предлагает единый интерфейс программирования и отладки кода (рис. 3.2). В окне редактора Visual Basic — значительное количество заслуживающих внимания компонентов — немудрено и растеряться, — но несколько примеров вам наверняка помогут.



*Рис. 3.2. Код программ для Access набирается в окне редактора Microsoft Visual Basic*

Чтобы открыть окно редактора, выполните следующие действия.

1. Запустите Access 2002 на выполнение.
2. Выберите в строке меню **Файл**→**Создать** или щелкните на ссылке **Новая база данных** (Blank Database) на панели задач.
3. В диалоговом окне **Файл новой базы данных** (File New Database) задайте название базы данных и щелкните на кнопке **Создать** (Create).
4. Выберите в списке **Объекты** (Objects) окна **База данных элемент Модули** (Modules).

- Щелкните на кнопке Создать (Create) панели инструментов того же окна — откроется окно редактора Microsoft Visual Basic, содержащее текст нового модуля (см. рис. 3.2).

Теперь все готово для работы. Если точно выполнены все указанные выше рекомендации, на экране компьютера вы увидите окно редактора Microsoft Visual Basic с текстом файла, именующегося *Module1*. В пределах этого окна вам потребуется набрать код, приведенный в листинге 3.3. (Это, собственно, тот же текст, который отображен в листинге 3.1, но сейчас его обрамляют команды определения начала и конца подпрограммы.)

Набирая текст примера в окне редактора, уделите особое внимание строкам со второй по пятую, введя все шесть строк дословно. Мы обратимся к этому примеру еще не раз, чтобы ознакомиться с несколькими полезными инструментами, которые Access предоставляет в распоряжение программиста.

Перейти в режим отладки кода можно, щелкнув в пределах текста, который следует протестировать, и нажав клавишу <F8>. (Убедитесь, что вы верно выполнили все приведенные выше инструкции.) Программа начинает выполнение со строки 1. Система указывает на это, подсвечивая строку программы желтым цветом.

### Листинг 3.3. Подпрограмма вычисления площади круга

```
1: Sub CalculateCircumference()  
2:   Dim Circumference As Double, Radius As Double  
3:   Const PI = 3.14159  
4:   Radius = 10  
5:   Circumference = PI * Radius ^ 2  
6: End Sub  
7:  
8: Sub CallCalculate()  
9: Call CalculateCircumference()  
10: End Sub
```

На страницах данной главы обсуждается код, приведенный в строках со второй по шестую. Код строк 8–10 используется для тестирования процедуры CalculateCircumference().

## Отладка с использованием всплывающих подсказок

Чтобы отладить код, щелкните на нем и нажмите клавишу <F8>. Сначала выполняется код строки 1. При этом в окне редактора данная строка будет выделена желтым цветом.

После каждого нажатия клавиши <F8> отладчик переходит к очередной строке текста. Нажмите <F8> несколько раз, пока в желтый цвет не окрасится пятая строка кода. Подсветка отмечает текущее положение отладчика и ту строку кода, которая будет выполнена следующей.

Сейчас переменным PI и Radius уже присвоены надлежащие значения, поскольку соответствующие строки кода выполнены. Проверить текущее содержимое переменных PI и Radius вам помогут всплывающие подсказки. Поместите курсор мыши поверх названия переменной, и на экране отобразится окно подсказки (рис. 3.3).

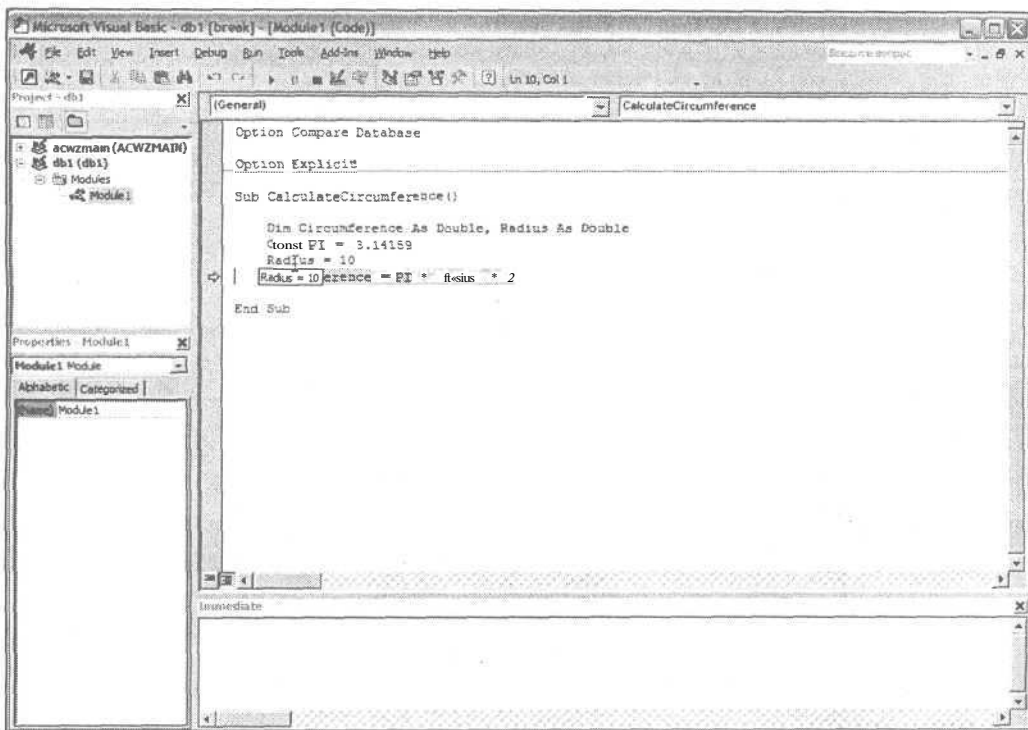


Рис. 3.3. В окне всплывающей подсказки демонстрируется текущее значение переменной *Radius*

## Режим просмотра локальных переменных

Локальными называются переменные, определенные в контексте функции или подпрограммы. Все переменные подпрограммы, представленной листингом 3.3 (Circumference, Radius и PI), локальны. Окно просмотра состояния локальных переменных открывается командой меню View → Locals Window. На рис. 3.4 показано окно Locals, содержащее сведения о состоянии переменных нашей подпрограммы.



Термин *переменная* используется нами для ссылки и на собственно переменные, и на постоянные величины, определенные в тексте программы.

Окно Locals обладает тем преимуществом, что позволяет одновременно ознакомиться с содержимым всех локальных переменных, определенных в подпрограмме или функции. Кроме того — и это еще более важно, — оно предоставляет информацию о состоянии объектов пользовательских типов данных. Объект пользовательского типа данных может содержать несколько переменных и иных объектов различных типов. Более подробные сведения по этой теме приведены в главе "11-й час. От сложного к простому: создание собственных типов данных".

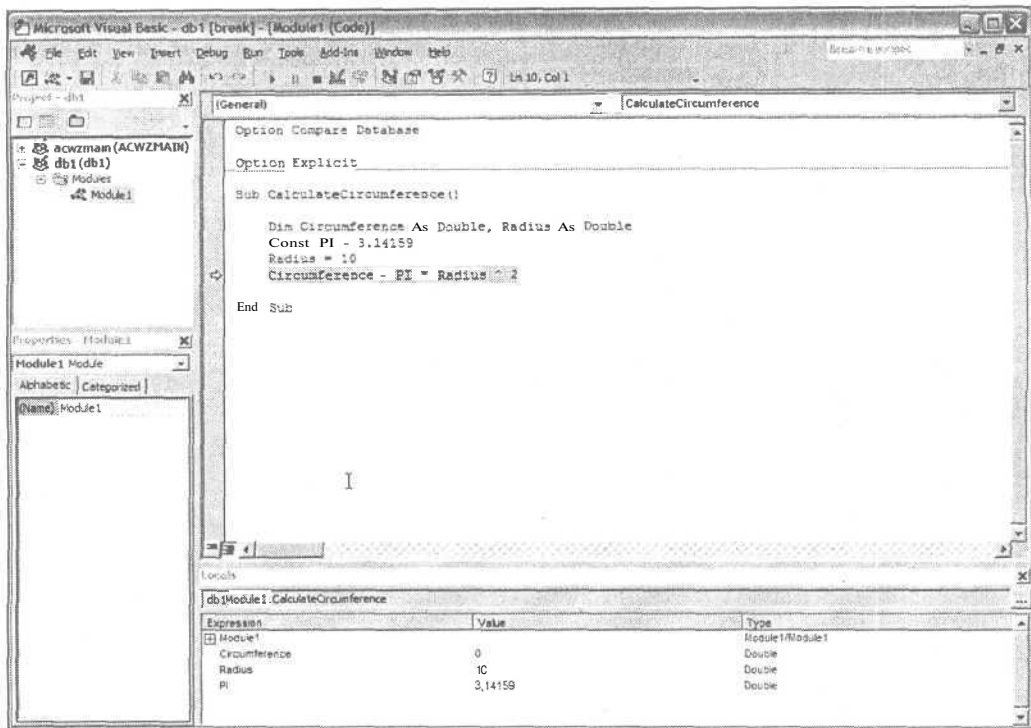


Рис. 3.4. Окно *Locals* отображает состояние локальных переменных программного блока

## Как пользоваться окном просмотра результатов

Окно *Watches*, служащее для просмотра результатов вычислений, подобно окну *Locals*, позволяет увидеть сразу несколько значений.

Окно *Watches* (рис. 3.5) отображает информацию только о тех объектах наблюдения, которые вы укажете. Среди переменных, за которыми ведется наблюдение в окне *Watches*, могут быть как локальные (принадлежащие контексту текущей подпрограммы или функции), так и те, что относятся к другим блокам программы.

Окно *Watches* построено в виде таблицы. Столбец *Expression* (Выражение) может содержать названия отдельных объектов и вычисляемые выражения, которые содержат операторы. В столбце *Value* (Значение) располагается значение переменной или результат вычисления. Столбец *Type* (Тип) предназначен для указания типа значения. В столбце *Context* (Контекст) программа отображает названия модуля и функции (подпрограммы), которые разделены оператором точки и которым принадлежит объект данных. В нашем примере было использовано имя модуля (*Module1*), предложенное системой по умолчанию, а подпрограмме присвоено имя *CalculateCircumference*. Поэтому в столбце *Context* будут содержаться значения, равные *Module1.CalculateCircumference*.

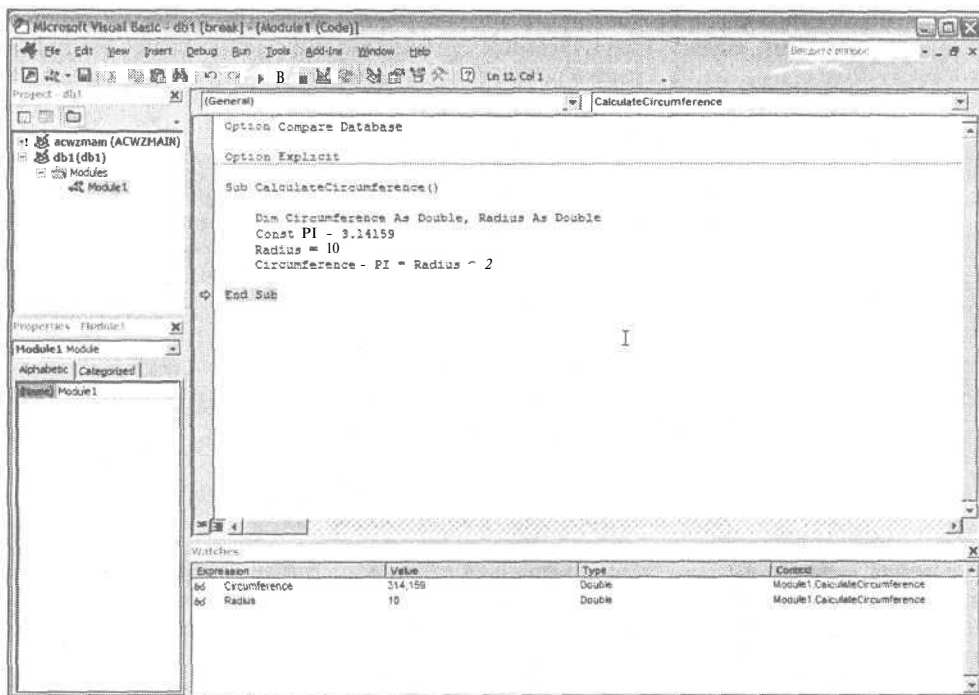


Рис. 3.5. Окно *Watches* позволяет вести наблюдение за содержимым только тех объектов данных, которые вас интересуют

## Добавление объекта наблюдения

Наиболее важное различие между окнами *Watches* и *Locals* состоит в том, что первое позволяет следить за результатами вычисления выражений. Мы будем пользоваться примером, приведенным на рис. 3.6, как отправной точкой для дальнейшего изложения преимуществ окна *Watches*.

Чтобы добавить в таблицу *Watches* новый объект наблюдения, достаточно выбрать в строке меню окна редактора Microsoft Visual Basic команду **Debug**⇒**Add Watch**. Механизм добавления объектов наблюдения удобно использовать во время исполнения кода в режиме отладки, но описания объектов наблюдения можно готовить и заранее, еще до старта программы. Рассмотрите диалоговое окно *Add Watch*, показанное на рис. 3.6: чтобы добавить объект наблюдения, необходимо ввести требуемое выражение в текстовое поле *Expression*. В качестве выражений допускаются идентификаторы переменных либо конструкции, содержащие операторы (например, `SquareOfCircle > 300`). Контекст выражения (группа интерфейсных элементов *Context*) можно указать особо; по умолчанию в качестве такового принимается текущий контекст. Контекст, обозначаемый подсветкой строки в окне редактора, — это место кода, где в данный момент "находится" отладчик. Раскрывающийся список *Procedure* позволяет уточнить контекст, если вы выберете любую из подпрограмм или функций, определенных в тексте модуля. Список *Module* дает возможность выбрать тот или иной модуль проекта; по умолчанию в нем указано наименование текущего модуля.

Теперь обратите внимание на группу элементов *Watch Type*, предназначенных для задания типа объекта наблюдения. Опции *Watch Type* позволяют указать системе, при каких условиях (с учетом значения выражения, заданного в поле *Expression*) следует приостановить выполнение программы. Рис. 3.6 соответствует ситуации, когда выбра-

на опция Break When Value Is True, предоставляющая возможность прервать выполнение программы в том случае, когда заданное логическое выражение (в нашем примере — `SquareOfCircle > 300`) даст в результате вычисления истинное значение.

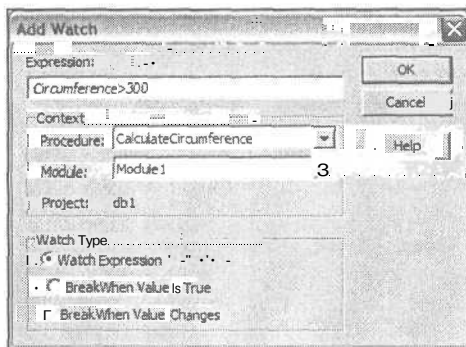


Рис. 3.6. Диалоговое окно Add Watch позволяет добавить объект наблюдения

Если для отладки вы пользуетесь клавишей <F8>, выполнение программы приостанавливается на каждой строке кода. Если же необходимо приостановить программу только в том случае, когда будет удовлетворено указанное условие, следует нажать <F5>.

## Редактирование объекта наблюдения

Диалоговое окно Edit Watch, предназначенное для редактирования объекта наблюдения, внешне схоже с окном Add Watch. Единственное отличие состоит в том, что при открытии окна Edit Watch содержит объект наблюдения, предварительно выбранный в окне Watches. Чтобы отредактировать объект наблюдения, выполните следующие действия.

1. Щелкните на имени нужного объекта наблюдения в окне Watches.
2. Откройте меню Debug.
3. Выберите команду Edit Watch.

Откроется диалоговое окно Edit Watch, ссылающееся на объект наблюдения, который вы выбрали в окне Watches. Если вы внимательно присмотритесь к содержимому диалоговых окон Edit Watch и Add Watch, то заметите, что они различаются только заголовками.

## Режим Quick Watch

Режим Quick Watch позволяет быстро ознакомиться со значением любой переменной или результатом вычисления выражения, ранее определенных в тексте программы. Например, легко увидеть результат вычисления `Radius ^ 2` — части всего выражения. Чтобы воспользоваться средствами Quick Watch, достаточно выполнить следующие действия.

1. Если программа, находящаяся в режиме отладки, приостановила свое выполнение, выделите в окне редактора выражение, которое вас интересует.
2. Откройте меню Debug.
3. Выберите команду Quick Watch.

Если вы решите воспроизвести пример, касающийся просмотра результата вычисления выражения `Radius ^ 2`, то увидите на экране компьютера диалоговое окно Quick Watch, подобное тому, которое приведено на рис. 3.7.



Рис. 3.7. Диалоговое окно Quick Watch — более эффективный инструмент просмотра содержимого переменных или результатов вычисления любых выражений, нежели средства добавления объектов наблюдения

Если вы помните, в примере, соответствующем листингу 3.3, переменной Radius было присвоено значение 10. Квадрат этой величины — разумеется, 100.

Режимы наблюдения за состоянием объектов данных программы — удобные инструменты отладки кода любого назначения и сложности.

## Режим непосредственного исполнения кода

Окно Immediate, предназначенное для непосредственного исполнения кода, — один из моих любимых рабочих инструментов. Окно Immediate (рис. 3.8) позволяет выполнять коды различной степени сложности. Чтобы открыть окно, следует выбрать в строке меню редактора команду View⇒Immediate Window.

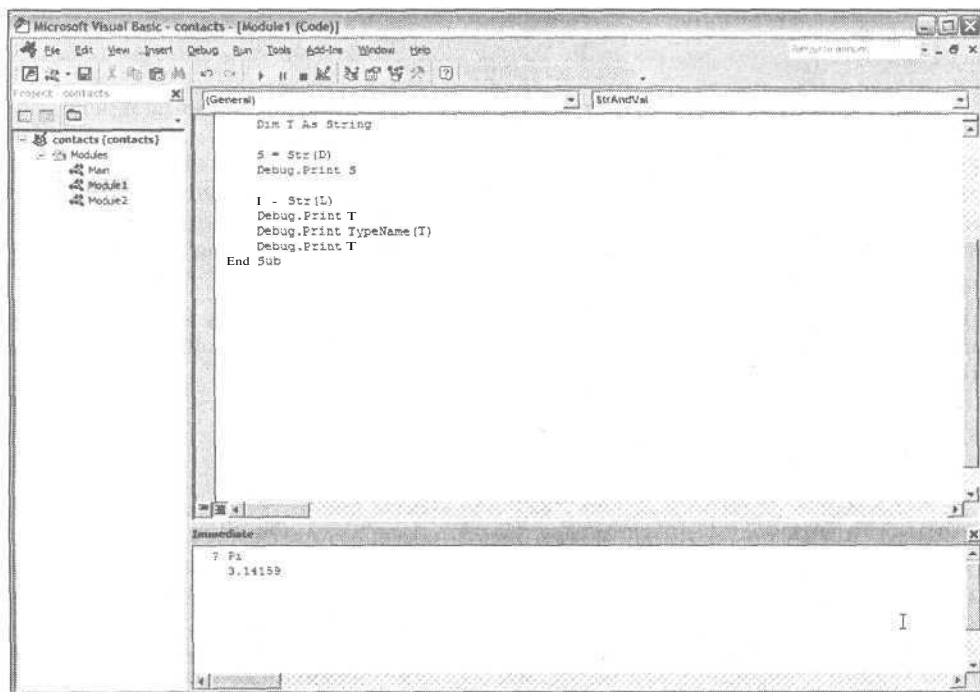


Рис. 3.8. В окне Immediate отображается содержимое константы PI, определенной в тексте программы



Приемы использования окна Immediate требуют дополнительных пояснений, поскольку существуют некоторые ограничения, касающиеся разновидностей того кода, который может быть исполнен.

Если необходимо воспользоваться средствами окна Immediate, откройте его, наберите текст нужной команды или выражения и нажмите клавишу <Enter>. Чтобы ознакомиться с содержанием константы PI, выполните следующие действия.

1. В режиме отладки выполните код листинга 3.3, дойдя до строки 5.
2. Выберите в строке меню редактора команду View⇒Immediate Window, чтобы открыть окно Immediate.
3. Щелкните в пределах окна Immediate, чтобы задать ему фокус ввода.
4. В окне Immediate наберите ? PI (? — краткая версия команды print, оставшаяся от предыдущих версий Visual Basic. Редактор Visual Basic самостоятельно преобразует ? в print. Например, команда Debug.? PI будет преобразована Debug.Print PI).
5. Нажмите клавишу <Enter>.

Ниже появится значение 3.14159, ранее присвоенное константе PI (см. рис. 3.8). В листинге 3.4 показан пример более сложного кода, который может быть исполнен с помощью средств окна Immediate.

#### Листинг 3.4. Пример непосредственного исполнения кода

```
1: Open "c:\winnt\win.ini" for Input As #1
2: Line Input #1, s
3: Print s
4: Close #1
```

Строка 1 содержит команду открытия файла `c:\winnt\win.ini`, который присутствует на диске каждого компьютера, работающего под операционной системой Windows. Строка 2, ссылаясь на открытый файл с помощью номера #1, осуществляет чтение первой строки файла и присваивание полученного значения неявно заданной переменной `s` типа String. В строке 3 выполняется вывод содержимого переменной `s` в окно Immediate. На компьютере автора книги результат вывода выглядел так:

```
; for 16-bit app support
```

Команда в строке 4 закрывает файл. Окно Immediate — удобный инструмент предварительной проверки кода перед его практической реализацией в составе программного модуля.

## Стек вызовов

Использование служебного слова Call для вызова процедур восходит еще к языку ассемблера. В VBA по-прежнему можно использовать его перед именем процедуры. В техническом смысле инструкция вызова подпрограмм означает, что регистры сегмента кода и указателя инструкций в ЦПУ должны начать выполнение инструкций, определенных адресом CS:IP (это 16-битный адрес, но аналогично происходит работа и в 32-битной Windows). Для размещения локальных переменных создается стековый фрейм, в стеке хранится текущий CS:IP-адрес, а локальные переменные размещаются непосредственно после выполнения CS:IP-инструкции.

В нашем случае вызов можно понимать как обращение к программе с целью выполнить именованную процедуру. Поскольку вызов происходит по имени (и адресу), отладчик может отследить порядок вызова процедур и выявить, где вызывались про-

цедуры. Список вызываемых процедур помещается в стек вызовов (Call Stack). В редакторе Visual Basic этот стек приводится в одноименном диалоговом окне (рис. 3.9).

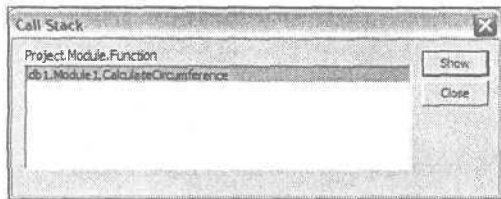


Рис. 3.9. В диалоговом окне Call Stack содержится список вызванных процедур — от последней к первой. В данном окне можно отследить поток операций, выполняемых программой

Когда же следует обращаться к окну Call Stack? При возникновении ошибок часто необходимо отследить работу программы по частям, чтобы выявить место появления ошибок. Последняя исполнявшаяся процедура находится сверху списка. Это своего рода след из хлебных крошек, как в сказке о Ганзеле и Гретель, по которому можно вернуться назад и выяснить, где программа работает неверно.

Чтобы переместиться к нужной подпрограмме, дважды щелкните на ее имени в окне Call Stack, и редактор Visual Basic переместит курсор на эту процедуру.

## Резюме

Сейчас вы уже можете уверенно обращаться с переменными. Все, что осталось сделать, — это разложить по полочкам в памяти знания, полученные на этом занятии. Вы ознакомились со способами объявления переменных различных типов. Испытав на практике приемы объявления переменных и присваивания им значений, вы сможете усовершенствовать навыки владения данными — одной из главных составных частей любой программы.

Из данной главы вы узнали о применении механизма всплывающих подсказок, о способах работы с окнами Locals, Watches и Immediate — эффективными и удобными средствами отладки кода и "шлифовки" его на уровне данных простых типов и выражений. К этим инструментам мы будем неоднократно обращаться и на следующих занятиях.

## Вопросы и ответы

**Вопрос.** Можно ли в пределах одного выражения объявить несколько переменных?

**Ответ.** Да. Однако помните, что переменные, перечисленные без явного указания типа, будут иметь тип Variant.

**Вопрос.** Разрешается ли объявлять константу в одном месте кода, а затем обращаться к ней неоднократно из различных частей программы?

**Ответ.** Да. Вы можете объявить константу в верхней части текста модуля — точно так же, как и переменные Global. Константы, определенные вне контекста любой функции или подпрограммы, система воспринимает как глобальные. Это самый безопасный вид объектов Global, поскольку их значения всегда остаются неизменными.

**Вопрос.** Возможен ли вариант экспериментирования с различными реализациями одного и того же решения, не требующий написания новых функций или процедур?

**Ответ.** Да. Это как раз одно из достойных применений средств окна Immediate.

# Задания

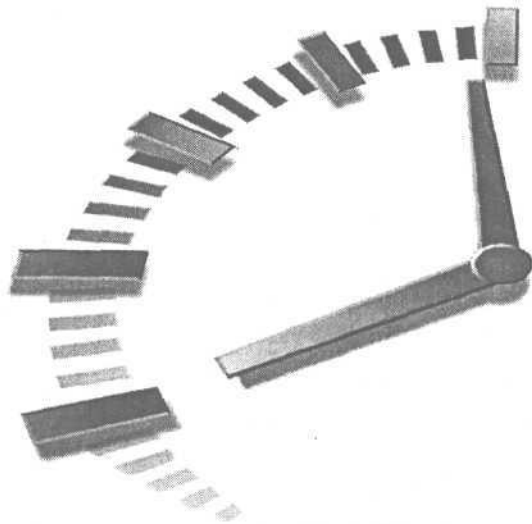
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Какое отличие существует между явным и неявным определением переменной?
2. С помощью каких средств языка можно исключить потенциально опасные возможности неявного определения переменных?
3. Для каких целей предназначен формат объявления переменных, использующий служебное слово `ReDim`?
4. В каком месте кода следует объявлять глобальные переменные?
5. Данные каких типов могут быть присвоены переменным типа `Variant`? Следует ли широко использовать объекты типа `Variant`?

## Упражнения

1. Напишите выражение объявления константы `PI` за пределами подпрограммы `SquareOfCircleCalc`, приведенной в листинге 3.3.
2. Исправьте код листинга 3.3 таким образом, чтобы в выражениях использовалась новая глобальная константа `PI`.
3. Постройте выражение объявления динамического массива для хранения десяти чисел двойной точности.



## 4-й час

### Последовательность действий и выполнение вычислений

В предыдущих главах вы ознакомились с приемами использования операторов, служебных слов, переменных и выражений. Это занятие мы посвятим рассмотрению выражений специального вида — так называемых *уравнений*. В контексте материала этой главы под уравнением мы будем понимать выражение, содержащее оператор(ы) и операнд(ы).

Практически в каждом выражении любой программы используются операторы. Операторы позволяют управлять последовательностью выполняемых программой действий, осуществлять арифметические операции, складывать и форматировать строки данных.

Основные темы занятия.

- Разновидности операторов.
- Типы выражений.
- Как осуществлять сравнение данных.
- Логические операторы.

## О понятии „уравнение”

*Уравнение* — это выражение, содержащее оператор. В практике реального программирования вам придется сталкиваться с подобными выражениями — арифметическими, логическими, условными — буквально на каждом шагу. В ходе этого занятия вы получите исчерпывающие сведения об операторах различных типов. Но вначале поговорим об операндах — данных, подвергающихся воздействию со стороны операторов.

Как вы, вероятно, помните, в разделе "Операторы и операнды Access" главы "2-й час. Познакомимся с VBA" мы говорили, что операнды — это элементы данных, с которыми работает оператор. Например, оператор сложения, обозначаемый символом *плюс* (+), имеет дело с двумя операндами — левым и правым. Так, в уравнении  $2 + 3$  константа 2 — это левый операнд, а 3 — правый.

## Операнды уравнений

Каждое уравнение содержит, по меньшей мере, один операнд. Операндами могут служить объекты данных различных видов — константы-литералы, переменные, именованные константы, глобальные значения и результаты вычисления функций.

В роли операндов могут выступать объекты любого типа, а каждый оператор способен воздействовать на данные конкретных типов. Другими словами, типы операндов должны соответствовать типу оператора. Подробнее об этом — далее в главе.

Предлагаем рассмотреть операнды различных видов.

### Константы-литералы

*Литерал* — это постоянное значение, введенное в текст программы непосредственно, без объявления и указания имени. Листинг 4.1 содержит несколько примеров констант-литералов.

Листинг 4.1. Примеры констант-литералов

```
1: 5
2: "Здравствуй, мир!"
3: "(517) 347-7170"
4: 10#
5: "A"
6: 3.14159
7: True
```



Литерал — это значение, на которое нельзя сослаться по имени.

Литерал 5, заданный в строке 1, — это просто число 5. Литералы могут иметь любой тип — лишь бы он совпадал с типом оператора, к которому относится операнд. Литералы в строках 2, 3 и 5 — это значения типа `string`. Значение в строке 4 выглядит несколько необычно. Символ фунта (`#`) после числа 10 указывает компилятору Access, что значение следует трактовать как число двойной точности (`Double`), а не просто как целое (`Integer`). В строке 6 в виде литерала задано число двойной точности, равное значению  $\pi$ . Строка 7 содержит логический литерал `True`.

### Переменные

С переменными вы уже ознакомились — мы достаточно подробно говорили о них в главе "3-й час. Как программа работает с данными". *Переменная* — это именованный объект, позволяющий хранить данные определенного типа. Операторы, как правило, "неравнодушны" к типам данных, которые им предлагаются. Если в уравнении типы данных не соответствуют тем, которые предусматриваются оператором, компилятор Access выдаст сообщение об ошибке (рис. 4.1).

Используйте переменные в том случае, если, по вашему мнению, объекты данных могут изменять свои значения. Если же заведомо известно, что содержимое элемента данных будет оставаться постоянным, уместно определить его в виде константы.

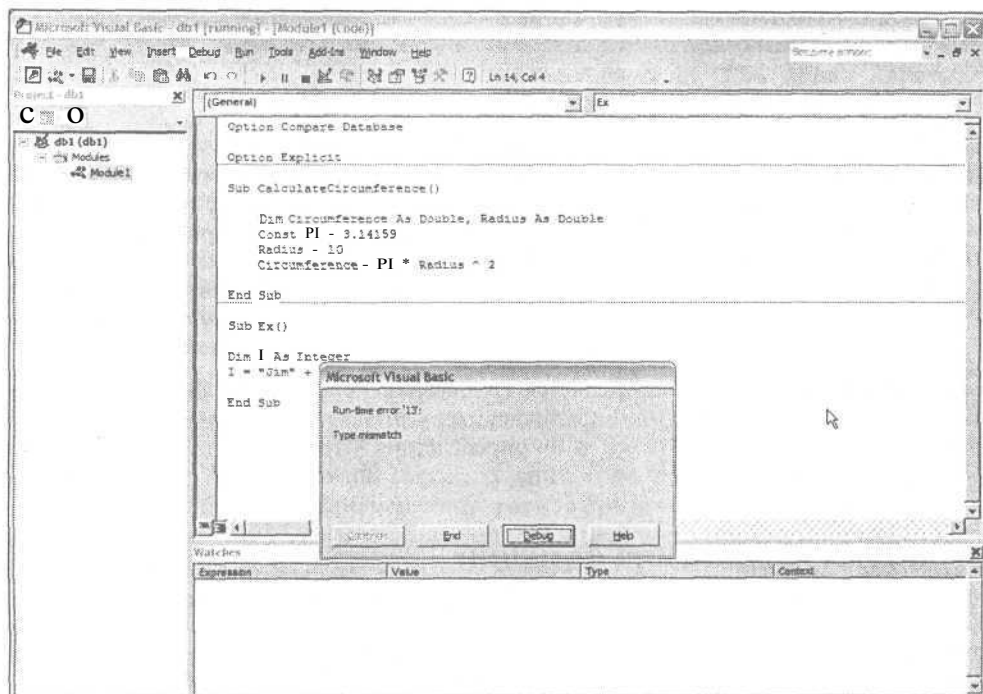


Рис. 4.1. Если типы данных не соответствуют тем, которые поддерживаются конкретным оператором, появляется окно сообщения об ошибке

## Константы

**Константы** — это именованные постоянные значения. Константы, по определению, не могут изменяться — ни преднамеренно, ни случайно. Используйте константы, если вам необходимы твердые гарантии целостности постоянных значений, адресуемых по имени.

Константы соответствующих типов могут использоваться в качестве операндов для любых операторов; исключение составляет оператор присваивания — употребление констант в его левой части не допускается. После начальной инициализации константа не может быть присвоено новое значение. Константа инициализируется в точке ее первого определения — как в следующем примере:

```
Const MyConst = 5
```

Тип данных константы задается неявно — компилятор определяет его автоматически в момент инициализации. Выражение `Const MyConst = 5` неявно предполагает использование типа `Integer`.

## Глобальные элементы данных

Определение элемента данных с помощью служебного слова `Global` оказывает влияние на возможный контекст его использования, но никак не связано с заданием типа. Глобальные элементы — переменные и константы — могут относиться к любому типу. Определение элемента данных посредством слова `Global` просто предоставляет возможность доступа к нему из разных мест кода. В листинге 4.2 приведены примеры определения глобальных элементов данных.

#### Листинг 4.2. Примеры определения глобальных элементов данных

```
1: Global Const FILE_NAME = "Module1.bas"  
2: Global MyInteger As Integer
```

##### Новый термин

**Контекст** — это совокупность признаков, определяющих возможности доступа к объектам программы. Глобальные данные могут адресоваться из любого места программного кода. Объекты, лишенные признака Global (т.е. локальные), разрешается использовать только внутри того блока кода, в котором они определены. Контекст может быть задан самыми разными конструкциями языка программирования. Поэтому считаем целесообразным на некоторое время отложить обсуждение вопросов определения контекста. Мы вернемся к ним в главе "8-й час. Декомпозиция задач".

В строке 1 листинга 4.2 определяется глобальная константа `FILE_NAME`. Обратите внимание, что единственное отличие локальных констант от глобальных состоит в отсутствии служебного слова `Global` в их определении. Строка 2 содержит определение глобальной переменной `MyInteger` типа `Integer`. Значение `FILE_NAME` постоянно; напротив, значение переменной `MyInteger` легко изменить в любой строке кода.

## Результаты вычисления функций

В качестве операндов уравнений могут применяться и значения, возвращаемые функциями. **Функция** — это набор строк кода, адресуемый по имени и возвращающий значение определенного типа. Мы более подробно поговорим о функциях на 8-ом занятии. Ниже приведем пример, иллюстрирующий использование функций в уравнениях.

#### Листинг 4.3. Пример использования функций в качестве операндов уравнений

```
1: Function GetUserName () As String  
2:   GetUserName = InputBox ("Введите имя пользователя", _  
   "Имя пользователя", "Пол Киммел")  
3: End Function  
4: Sub Main()  
5:   Dim Greeting As String  
6:   Greeting = "Здравствуйте, " + GetUserName  
7:   MsgBox Greeting  
8: End Sub
```

Строки 1, 2 и 3 содержат функцию, возвращающую имя пользователя в виде строки. В строке 4 объявлена переменная `Greeting`. Строка 5 содержит оператор присваивания, сохраняющий в переменной `Greeting` результат сложения литерала "Здравствуйте " и строки с именем пользователя, возвращенной функцией `GetUserName`. В строке 6 применена стандартная процедура `MsgBox Access`, предназначенная для вывода окна сообщения. (Оператор сложения строк, используемый в строке 5 листинга, подробно рассмотрен ниже, в одноименном разделе этой главы.)

## Правила „дорожного движения“

В уравнениях, как вам уже известно, могут функционировать самые разные типы данных и применяться разные способы их определения. Постигание таинств программирования в `Access` требует времени, причем существенную его часть следует потратить на изучение соответствий между типами данных и операторами, способными с ними взаимодействовать. Перечислим основные правила, которым необходимо следовать, чтобы успешно достичь конечной цели, путешествуя в мире `Access`.

Во-первых, литералам имеет смысл предпочесть именованные константы и переменные. Если константа определена единожды и применяется во многих местах текста, при необходимости ее изменения вам потребуется обратиться только к одной строке кода, в которой выполняется инициализация. Но если вы применяете литералы, вам придется отыскивать и исправлять их по всему тексту программы.

Второе, не менее полезное, правило гласит: избегайте использования глобальных переменных. Глобальные данные — потенциальный источник недоразумений и ошибок. В определенный момент времени все мы — и я в том числе — способны надежно хранить в памяти только незначительный объем информации. Если переменные используются только тогда и в тех местах кода, когда и где они определены, текущий контекст их употребления будет понятен и вам, и читателям вашей программы как сейчас, так и много дней спустя. Кроме того, если объект данных локализован, легче избежать случайных ошибок в его применении.

Наконец, старайтесь не создавать чрезмерно сложных выражений. Длинные фразы могут показаться признаком высокого профессионального уровня, но их сложно составлять, они с трудом поддаются отладке и часто приводят к результатам, далеким от исходного замысла.

## Классификация операторов по числу операндов

Операторы, применяемые в языках программирования, можно разделить на три группы в зависимости от числа используемых ими операндов: унарные, бинарные и тернарные. *Унарные* операторы обращаются с одним, правосторонним, операндом. *Бинарные* операторы имеют дело с двумя операндами, левым и правым. *Тернарные* операторы взаимодействуют с тремя аргументами. Единственный известный мне тернарный оператор — это конструкция из служебных символов `?` и `:`, используемая в языках C и C++. Поскольку в Access VBA тернарные операторы не реализованы, нам более не стоит о них говорить.



В Access VBA поддерживаются только два унарных оператора — `Not` и `AddressOf`. Оба они предполагают использование правосторонних аргументов. Все другие операторы VBA бинарные. Это означает, что их синтаксис можно описать следующим образом:

ЛевыйОперанд БинарныйОператор ПравыйОперанд

Подставьте вместо аргументов ЛевыйОперанд и ПравыйОперанд выражения, а вместо словосочетания БинарныйОператор — любой реальный бинарный оператор (естественно, следуя определенным правилам и ограничениям) — вы получите простое уравнение с одним оператором.



Аналогичным образом можно представить синтаксис унарного оператора: УнарныйОператор ПравыйОперанд

Чтобы получить уравнение с унарным оператором, достаточно заменить словосочетание УнарныйОператор одним из допустимых операторов — `Not` или `AddressOf`, а в качестве правого операнда указать соответствующее выражение. Оператор `Not` мы рассмотрим более подробно в разделе "Логические операторы в истинном свете", а примеры использования оператора `AddressOf` приводятся в разделе "Специальные операторы".



# Применение арифметических операторов

Говоря об арифметических операторах, я имею в виду те же самые средства выполнения арифметических действий, которые предоставлены любым калькулятором. Все мы начинали знакомиться с ними еще в начальной школе. (А ну-ка, вспомните, как звали вашу первую учительницу математики!) Все арифметические операторы относятся к разряду бинарных. Не будем тратить время на дополнительные разъяснения — они излишни. На мой взгляд, достаточно привести сведения о символах, посредством которых обозначаются арифметические операторы, и особо отметить те из них, которые используются относительно редко. В табл. 4.1 представлены обозначения, наименования и описания всех арифметических операторов языка VBA.

Таблица 4.1. Арифметические операторы

Символ	Наименование	Описание
$\wedge$	Возведение в степень	Формат: $X \wedge Y$ . Результат — $X$ , возведенное в степень, равную $Y$
$*$	Умножение	Формат: $X * Y$ . Результат — произведение $X$ и $Y$
$/$	Вещественное деление	Формат: $X / Y$ . Результат — вещественное частное от деления $X$ на $Y$
$\backslash$	Целочисленное деление	Формат: $X \backslash Y$ . Результат — целое частное от деления $X$ на $Y$
Mod	Взятие остатка	Формат: $X \text{ Mod } Y$ . Результат — остаток от целочисленного деления $X$ на $Y$ (например, $5 \text{ Mod } 3 = 2$ )
$+$	Сложение	Формат: $X + Y$ . Результат — сумма $X$ и $Y$
$-$	Вычитание	Формат: $X - Y$ . Результат — разность $X$ и $Y$

Символ прямой косой черты ( $/$ ) — это обозначение оператора вещественного деления. В качестве левого и правого операндов могут использоваться любые числа, и в результате деления вы получите вещественное число. Например,  $5 / 3 = 1.666$ . Но если в том же выражении употребить символ обратной косой черты ( $\backslash$ ), то  $5 \backslash 3 = 1$ . Применяйте тот оператор деления, который отвечает условиям конкретной задачи.

Вам наверняка знаком и оператор возведения в степень ( $\wedge$ ), хотя его нельзя причислить к тем, которыми приходится пользоваться ежедневно. Наиболее часто выполняются операции возведения числа в квадрат и куб. Оператор возведения в произвольную степень находит применение в приложениях, связанных с решением научно-технических задач.

Последний из группы наиболее экзотических операторов — это оператор взятия остатка (деления по модулю) (Mod), который используется для определения признака целочисленной делимости одной величины на другую и лежит в основе целого раздела математики — теории колец. Чтобы определить, делится ли без остатка одно число на другое, необходимо проверить выполнение тождества  $X \text{ Mod } Y = 0$ .

## Операторы сравнения

Операторы сравнения позволяют проверить, каким образом соотносятся значения левого и правого операндов. Результат сравнения — это булева величина, принимающая одно из двух значений — True (истина) или False (ложь). В табл. 4.2 приведены

обозначения, описания операторов сравнения и возможные варианты их значений (слова *Левый* и *Правый* используются нами для ссылки, соответственно, на левый и правый операнды выражения).

Таблица 4.2. Операторы сравнения

Обозначение и описание	True	False	Null
< (Меньше)	Левый < Правый	Левый >= Правый	Левый или Правый = Null
<= (Меньше или равно)	Левый <= Правый	Левый > Правый	Левый или Правый = Null
> (Больше)	Левый > Правый	Левый <= Правый	Левый или Правый = Null
>= (Больше или равно)	Левый >= Правый	Левый < Правый	Левый или Правый = Null
= (Равно)	Левый = Правый	Левый ≠ Правый	Левый или Правый = Null
<> (Не равно)	Левый <> Правый	Левый = Правый	Левый или Правый = Null

Если читать выражение оператора сравнения слева направо, указывая вначале левый операнд, название оператора, а затем правый операнд, то результат становится очевидным. Рассмотрим следующий пример:

5 > 6

Строка этого кода читается так: "Пять больше шести". Число 5, разумеется, не больше числа 6; поэтому результат сравнения равен False. Если вам трудно запомнить названия операторов сравнения, сделайте шпаргалку и прикрепите ее к монитору компьютера. Еще один хороший прием запоминания таков: воспринимайте символы < и > как стрелки и следуйте правилу — чтобы результат оказался истинным, стрелка своей вершиной должна указывать на меньшую из двух сопоставляемых величин. Листинг 4.4 содержит несколько примеров, иллюстрирующих применение операторов сравнения литеральных выражений. Далее приводится анализ каждого примера.

Листинг 4.4. Примеры операторов сравнения

```
1: Sub TestComparisonOperators()  
2: Dim Result As Boolean  
3: Result = (6 > 5)  
4: Result = (6 >= 5)  
5: Result = ("Hello" < "World")  
6: Result = ("Hello" <= "World")  
7: Result = (True = False)  
8: Result = (False <> False)  
9: End Sub
```

В строке 2 объявляется булева переменная Result — далее она будет использоваться для хранения результатов выполнения всех операций сравнения. (Вообще, операторы сравнения чаще применяются в логических выражениях. Более подробные сведения по этой теме приведены в главе "5-й час. Программирование управляющих структур".) В строке 3 и всех остальных скобки использованы только для удобочитаемости; формально они не обязательны, но существенно проясняют смысл выражений. В результате выполнения строки 3 переменной Result будет присвоено значение True, поскольку число 6 действительно больше, чем 5. Строка 4 оставляет значение

Result без изменений — выражение  $6 \geq 5$  истинно. Result продолжает сохранять значение True и после присваивания в строке 4, так как "Hello" меньше, чем "World". Итог вычислений, выполняемых в строке 5, аналогичен. В строке 6 осуществляется сравнение литеральных булевых величин и переменной Result присваивается значение False, поскольку True не может быть равно False. Ситуация не изменится и после выполнения оператора в строке 8 — действительно, False не может не равняться False.

Листинг 4.4 демонстрирует еще одно применение операторов сравнения — сопоставляться могут не только целые числовые величины. Ничто не мешает сравнивать булевы значения, числа различных типов и даже строки. При выполнении операций сравнения значений типа string компилятор Access не учитывает регистр символов. Символы, занимающие в строках одинаковые позиции, сопоставляются в порядке их следования в алфавите, независимо от регистра. Поэтому результаты выполнения строк 4 и 5 листинга 4.4 именно такие, как описаны выше. Независимость результатов сравнения от регистра означает, что, например, строки "hello", "HELLO", "Hello" и любые другие, обозначающие то же слово и образованные из строчных и прописных букв, считаются одинаковыми.

## Логические операторы в истинном свете

Логические операторы, вероятно, несколько сложнее для восприятия. Если вы никогда не имели с ними дела, вам потребуется практика. В табл. 4.3 приведены обозначения, наименования и описания всех логических операторов, поддерживаемых в VBA.

Таблица 4.3. Логические операторы

Обозначение	Наименование	Описание
And	И	Логическая конъюнкция
Eqv	Эквивалентность	Логическая эквивалентность
Imp	Импликация	Логическая импликация
Not	НЕ	Логическое отрицание
Or	ИЛИ	Логическая дизъюнкция
Xor	Исключающее ИЛИ	Логическое исключение

"Вам уже хорошо известны логические операторы, или вы не осведомлены о них". Последнее предложение представляет собою логическое выражение. Первое высказывание в нем — "вам уже хорошо известны логические операторы", а второе — "вы не осведомлены о них". Высказывания связаны между собою союзом (в контексте нашей темы — оператором) "или".

Бинарные логические выражения на основе оператора And истинны (т.е. вычисляются с результатом True) только в том случае, если оба высказывания-операнда равны True. Выражения, использующие оператор Or, обращаются в False только тогда, когда оба операнда равны False. Унарный оператор отрицания (Not) изменяет значение своего операнда на противоположное. С целью облегчения задачи исчисления высказываний для каждого логического оператора были разработаны так называемые *таблицы истинности*. В табл. 4.4 собраны воедино таблицы истинности всех бинарных логических операторов.

Таблица 4.4. Таблицы истинности бинарных логических операторов

Значения операндов	And	Eqv	Imp	Or	Xor
False-False	False	True	True	False	False
False-True	False	False	True	True	True
True-False	False	False	False	True	True
True-True	True	True	True	True	False

## Новый термин

*Таблица истинности* — это сжатое представление результатов вычисления выражения с использованием определенного логического оператора и различных комбинаций операндов.

Левый столбец табл. 4.4 содержит четыре возможных сочетания значений левого и правого операндов (так, например, пару *True-False* следует воспринимать как обозначение такой комбинации, когда левый операнд равен *True*, а правый — *False*). В следующих пяти столбцах представлены результаты вычисления выражений на основе определенного оператора для каждого из четырех сочетаний операндов. Например, в ячейке на пересечении столбца с оператором *And* и первой строки (соответствующей сочетанию *False-False*) находится значение *False*. Это значит, что если оба аргумента оператора *And* равны *False*, то результат всего выражения также равен *False*. Другими словами, если каждое высказывание ложно, то и их конъюнкция также ложна. Как нетрудно понять, конъюнкция (выражение на основе оператора *And*) истинна тогда и только тогда, когда истинны одновременно оба высказывания (операнда, или аргумента). Для достижения истинности логической дизъюнкции (выражения на основе оператора *Or*) достаточно, чтобы, по меньшей мере, один из операндов был равен *True*. Так же несложно узнать о закономерностях "поведения" и всех остальных бинарных логических операторов, описанных в табл. 4.4. Если у вас возникают затруднения с запоминанием правил исчисления логических высказываний, напишите коротенькую шпаргалку с таблицами истинности операторов и держите ее на виду.

Единственный унарный логический оператор — *Not* — описывается собственной таблицей истинности (табл. 4.5).

Таблица 4.5. Таблица истинности логического оператора *Not*

Значение операнда	Not
False	True
True	False

Результат выполнения выражения на основе оператора отрицания *Not* — это величина, обратная значению операнда.

В листинге 4.5 приведены примеры вычисления различных логических выражений. Комментарии даны ниже.

Листинг 4.5. Примеры вычисления логических выражений

```
1: Dim Result As Boolean
2: Result = True Or False
3: Result = True And True
4: Result = False Xor True
```

```
5: Result = False Imp False
6: Result = True Eqv False
7: Result = Not True
```

Строка 1 содержит выражение объявления булевой переменной Result — она будет использоваться для хранения результатов вычисления логических выражений, приведенных далее. Результат выполнения оператора в строке 2 — True. Значение Result не меняется и после вычисления следующего оператора. Результат выполнения строки 4 также равен True, поскольку выражение на основе *Xor* истинно в тех случаях, когда значения операндов различны. Результат логической импликации (строка 5) равен True, так как оба операнда выражения равны False (обратитесь к данным табл. 4.4). Посмотрим на строку 6: True, разумеется, не эквивалентно False, и поэтому результат равен False. Наконец, оператор Not, заданный в строке 7, отрицает значение своего операнда True — таким образом, результат равен False.

## Оператор And

Логическому оператору And соответствует союз "и" русского языка. Запомните: результат логической конъюнкции будет истинным тогда и только тогда, когда истинны одновременно оба ее операнда. В качестве операндов могут выступать как простые переменные, именованные константы или литералы, так и выражения. Листинг 4.6 демонстрирует несколько примеров использования логического оператора And.

### Листинг 4.6. Примеры использования оператора And

```
1: Dim Result As Boolean
2: Result = False And False
3: Result = (6 > 5) And (7 = 7)
```

Выражение в строке 2 можно воспринимать как модель конъюнкции двух выражений, дающих в результате вычисления значения False. False And False равно False. Оба операнда уравнения в строке 3 — это выражения на основе операторов сравнения. Каждому оператору языка Access VBA поставлено в соответствие то или иное значение приоритета выполнения (об этом речь пойдет ниже, в разделе "Кто здесь главный?"), но порядок вычислений легко изменить с помощью скобок. Левый операнд,  $6 > 5$ , равен True; правый,  $7 = 7$ , также равен True. Поэтому общий результат конъюнкции True And True равен True.

## Оператор Eqv

Вы вправе спросить, зачем нужен оператор эквивалентности (Eqv), если уже есть общеизвестный оператор равенства (=). Верно, в логических выражениях оба эти оператора дают один и тот же результат. Отличие между ними состоит в том, что Eqv употребляется также при выполнении побитовых операций (подробнее о них рассказывается ниже, в одноименном разделе). Рассмотрим код листинга 4.7.

Новый термин

*Побитовой* называется операция, выполняемая над одноименными разрядами двух чисел в двоичном представлении и дающая в результате новое число.

### Листинг 4.7. Сравнение операторов = и Eqv

```
1: Dim I As Integer
2: Dim B As Boolean
3: B = (5 = 7)
4: I = (5 Eqv 7)
```

Результат сравнения двух целых чисел с помощью оператора `=` — это значение типа `Boolean`. Однако, если к тем же операндам применяется оператор `Eqv`, результат его выполнения — целое число. Так, переменная в строке 3 получает значение `False`, а переменной `I` в строке 4 присваивается значение `-3`.

При сравнении булевых значений операторы равенства и эквивалентности действуют одинаково. Но когда в качестве операндов выступают целые числа, то оператор `=` по-прежнему выполняет логическую проверку, а `Eqv` осуществляет побитовую операцию. Напомним: побитовым операциям посвящен одноименный раздел текущей главы.

## Оператор `Imp`

Оператор импликации — это логический инструмент доказательства истинности утверждений. Например, во фразе "Если на Васе — оранжевый галстук, это свидетельство дурного настроения" содержится утверждение о том, что процесс и результат завязывания на Васиной шее оранжевого галстука напрямую связаны с тем, что персонаж не в духе. Если на Васе в данный момент нет злополучного галстука, то, согласно приведенной выше формуле, может показаться, что наш герой радуется жизни. К несчастью, Вася может впасть в уныние даже в том случае, если его оранжевый галстук две недели назад был сдан в химчистку.

Результат импликации становится ложным в единственном случае — когда правый операнд равен `False`. Если операция импликации для вас не очевидна, обращайтесь за помощью к таблице истинности (см. табл. 4.4).

## Оператор `Or`

Оператор `Or` ("ИЛИ"), как и его ближайший родственник, `And` ("И"), относительно легко поддается осмыслению и не нуждается в подробных разъяснениях, поскольку все мы ежедневно и неоднократно пользуемся им в разговорной речи.

Подобно всем рассматриваемым здесь логическим операторам, `Or` может применяться также и в побитовых операциях (см. раздел "Побитовые операции").

## Оператор `Xor`

Оператор `Xor` называют еще "исключающим ИЛИ" либо даже "не ИЛИ". Последний вариант — просто недоразумение. Если воспринять его буквально, то может показаться, что вначале необходимо вычислить результат операции "ИЛИ", а затем применить к нему оператор отрицания. В итоге — nonsens. Однако все гораздо проще: результат выполнения `Xor` равен `True` только в тех случаях, когда значения операндов различны.

## Оператор `Not`

`Not` — единственный унарный логический оператор. Кроме него, в словаре Access VBA есть еще один унарный оператор — `AddressOf`. Используйте `Not` в тех случаях, когда необходимо получить противоположное логическое значение.

## Побитовые операции

Побитовыми называются операции, выполняемые над одноименными разрядами чисел в двоичном представлении. Вы, вероятно, знаете, что один байт, грубо говоря, соответствует символу. Байт состоит из восьми битов. Бит — это двоичное число, способное принимать одно из двух значений: 0 или 1.

Когда речь идет о логических операторах, применяемых к целым числам, мы подразумеваем, что компилятор выполняет действия над битами и сохраняет результат в целочисленной переменной. Чтобы понять смысл побитовых операций, необходимо преобразовать пару обычных десятичных чисел в двоичную форму и сравнить значения их разрядов, расположенных на одинаковых позициях.



Компьютеры работают именно с двоичными числами, поскольку память электронной машины состоит из миллионов миниатюрных полупроводниковых переключателей, каждый из которых может находиться в одном из двух состояний — *включен* или *выключен*. Двоичная система счисления — родная стихия компьютеров; двоичные вычисления выполняются гораздо быстрее десятичных.

Важно понимать, что двоичная арифметика основана на использовании только двух цифр (0 и 1), а все мы в обычной жизни пользуемся десятичной системой счисления. Система счисления определяется количеством цифр, применяемых для представления чисел. Поэтому в двоичной системе употребляются две цифры (0 и 1), а в десятичной — десять (от 0 до 9).

Что же на самом деле стоит за рядом цифр, представляющим собой запись значения в определенной системе счисления? Для объяснения воспользуемся наглядным примером. Возьмем, скажем, десятичное число 13. Его можно представить следующим образом:

$$(1 * 10^1) + (3 * 10^0) = 10 + 3 = 13$$

Отдельные цифры в записи числа 13 (т.е. 1 и 3) — это коэффициенты слагаемых, которые являются последовательным рядом убывающих степеней числа, равного основанию системы счисления.

То же десятичное число 13 в двоичной записи будет выглядеть как 1101 (произносится именно так — один-один-ноль-один), поскольку

$$(1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 8 + 4 + 0 + 1 = 13$$

Теперь становится очевидной следующая общая формула:

$$(C_{n-1} * B^{n-1}) + (C_{n-2} * B^{n-2}) + \dots + (C_0 * B^0),$$

где  $n$  — число разрядов числа,  $C_i$ ,  $0 \leq i < n$  — коэффициенты слагаемых (представляемые цифрами в записи числа), а  $B$  — основание системы счисления. Нумеруются разряды числа справа налево, начиная с нуля.

Для выполнения бинарных побитовых операций **необходимо** предварительно преобразовать числа-операнды в двоичную систему счисления, а затем сопоставить каждую пару их одноименных разрядов. На помощь может прийти стандартное Windows-приложение Калькулятор, работающее в режиме Инженерный. Листинг 4.8 содержит короткий пример.

#### Листинг 4.8. Пример выполнения побитовой операции And

```
1: Dim R As Integer
2: R = 5 And 4
3: MsgBox R
```



Побитовые операции применимы только к целым числам.

Число 5 в двоичной записи выглядит как 101, а 4 — это 100. В результате выполнения оператора побитовой конъюнкции, приведенного в строке 2 листинга, переменной  $R$  присваивается значение 4. Сопоставим каждую пару одноименных разрядов двух чисел, полагая, что 1 (единица) эквивалентна логическому значению True, а 0 (ноль) — это False. Для нулевого разряда имеем 1 And 0 (т.е. True And False); результат (очевидно) равен 0. Конъюнкция первых разрядов (0 And 0) также дает 0. Наконец, для пары вторых разрядов логическое выражение выглядит как 1 And 1 и равно 1. Общий результат — 100, или 4 в десятичной системе счисления. Данный результат можно просмотреть в окне сообщения, выводимого на экран с помощью стандартной процедуры MsgBox, которая указана в третьей строке листинга.

# Сложение строк

Оба оператора сложения строк, реализованные в Access VBA, находят самое широкое практическое применение. Операторы + и &, описанные ниже в табл. 4.6, добавляют строку правого операнда в конец левого.

Таблица 4.6. Операторы сложения строк

Символ	Наименование	Описание
+	Плюс	Сложение строк
&	Амперсанд	Сложение строк с неявным преобразованием типов

Если используется оператор +, сторонний пользователь вашей программы может быть введен в заблуждение, поскольку текущий контекст не обязательно дает однозначный ответ на вопрос, какие операнды складываются — числовые или символьные. Но оператор & недвусмысленный, поскольку применяется исключительно для конкатенации строк. Листинг 4.9 содержит несколько примеров употребления обоих операторов.

Листинг 4.9. Примеры использования операторов сложения строк

```
1: Dim MyString As String
2: MyString = "Здравствуй, " + "мир!"
3: MyString = "Здравствуй, " & "мир!"
4: MyString = "Мне " + 5 + "лет"
5: MyString = "Мне " & 5 & "лет"
```



Оператор & одновременно со сложением строк выполняет неявное преобразование в строку значений типа, отличного от string.

Результаты выполнения операторов, указанных в строках 2 и 3 листинга, одинаковы. Однако, анализируя строку 4, компилятор выдаст сообщение об ошибке несоответствия типов (см. рис. 4.1). Те же данные при использовании оператора &, заданного в строке 5, будут обработаны успешно, поскольку числовое значение 5 подвергнется операции неявного преобразования в строку "5". Дадим небольшой совет — при необходимости сложения строк всегда пользуйтесь оператором &.

## Кто здесь главный?

Каждому оператору соответствует определенный уровень приоритета — конкретный оператор выполняется после операторов одной группы и перед другими. Например, операторы умножения и деления имеют более высокий приоритет, нежели операторы сложения и вычитания, т.е. выполняются в первую очередь.

Компилятор Access VBA не требует от вас обязательного явного указания порядка выполнения операторов — он руководствуется заранее оговоренными правилами, изложенными в табл. 4.7. Операторы расположены в таблице в порядке убывания приоритета выполнения, причем арифметические операторы имеют высший приоритет, операторы сравнения выполняются после арифметических, а логические находятся на низшей ступени.



Таблица 4.7. Приоритеты выполнения операторов

Арифметические операторы	Операторы сравнения	Логические операторы
-	=	Not
* , /	<>	And
\	<	Or
Mod	>	Xor
+ , -	<=	Eqv
&	>=	Imp

Таблицу следует читать сверху вниз, а затем слева направо. Все арифметические операторы имеют более высокий приоритет по сравнению с операторами сравнения и логическими операторами. Среди арифметических операторов наивысшим приоритетом обладает оператор возведения в степень. Оператор конкатенации строк, условно отнесенный к группе арифметических операторов, замыкает ее. Тем не менее, сложение строк все равно выполняется прежде любых операций сравнения или логического исчисления. Запоминать данные табл. 4.7 вовсе не обязательно — вы всегда с не меньшим успехом сможете поведать о своем замысле, расставив соответствующим образом круглые скобки.

Еще одно хорошее правило, которое стоит взять на вооружение, таково: не старайтесь выразить свое намерение в виде единого громоздкого выражения — разделите предложение на несколько более простых и расположите их в нужном порядке.

## Специальные операторы

В арсенале VBA есть еще три оператора, которые при определенных обстоятельствах могут вам пригодиться. Два из них (Is и Like) — бинарные, а третий (AddressOf) — унарный. Названные операторы описаны в табл. 4.8.

Таблица 4.8. Специальные операторы

Наименование	Синтаксис	Описание
Is	Объект1 Is Объект2	Оператор проверяет, ссылаются ли две переменные на один и тот же объект
Like	Результат = Строка Like Образец	Оператор проверяет, удовлетворяет ли строка заданному образцу
AddressOf	AddressOf ИмяПроцедуры	Возвращает числовое значение, представляющее собой адрес процедуры в памяти

### Оператор Is

Оператор Is применяется с целью проверки, ссылаются ли обе переменные-операнды на один и тот же объект. Объект в данном случае — это экземпляр определенного пользователем типа данных, который содержит элементы-свойства и функции-методы их обработки. Подробнее об объектах и приемах их использования см. главу "21-й час. Основы программирования классов".

### Оператор Like

Оператор Like возвращает логический результат проверки такого факта: действительно ли указанная строка (левый операнд) сравнима с заданным текстовым образцом (правым операндом). Образец, с которым сопоставляется строка, может содержать:

обычные литеральные символы; вопросительные знаки (?), служащие для представления любого единственного символа; символы звездочки (\*), обозначающие строку произвольной (в том числе и нулевой) длины; символы фунта (#), представляющие любую цифру; а также заключенные в квадратные скобки ([]) списки символов, которые следует принять во внимание или, наоборот, исключить из рассмотрения. Листинг 4.10 содержит несколько примеров сравнения строк с образцом с помощью оператора Like.

#### Листинг 4.10. Примеры использования оператора Like

```
1: Dim Result As Boolean
2: Dim ZipCode As String
3: ZipCode = "48864"
4: Result = ZipCode Like "488##"
5: Result = "Сидорчук" Like "Сидор*"
6: Result = "(555) 555-1212" Like "(555) 555-####"
7: Dim CustomerName As String
8: CustomerName = "Jones"
9: Result = CustomerName Like "[!K-Z]*"
```

Оператор сравнения в строке 4 определяет, что строка, хранящаяся в переменной ZipCode, действительно содержит префикс "488". В строке 5 выполняется сопоставление литерала "Сидорчук" с образцом "Сидор\*", и результат этой операции также равен True. Подобный код можно использовать для поиска различных вариантов имен и фамилий, например *Сидор*, *Сидорчук*, *Сидоров*, *Сидоренко* и т.п., поскольку звездочка, завершающая образец, обозначает любую последовательность символов, в том числе и нулевой длины. Строка 6 служит для проверки принадлежности заданного телефонного номера множеству номеров, начинающихся с последовательности цифр "(555) 555". Результат операции равен True. Оператор сравнения с образцом, приведенный в строке 9, возвращает значение True для строк, начинающихся с любой буквы латинского алфавита, за исключением тех, которые попадают в промежуток от K до Z.

## Оператор AddressOf

В состав стандартного интерфейса программирования Windows-приложений (Windows Application Programming Interface, API) входят специальные средства использования так называемых *callback-переменных*, содержащих адреса функций или процедур. Поскольку идентификаторы всех переменных, функций и процедур программы преобразуются компилятором в определенные адресные значения, наличие адреса процедуры позволяет использовать его для непосредственного обращения к этой процедуре. В некоторых случаях функции Windows API должны иметь возможность обращения к процедурам, определенным пользователем.

Вам следует знать о существовании оператора AddressOf, хотя с вероятностью 99.9% его никогда не придется использовать на практике. Подробности, касающиеся применения средств Windows API в целом и оператора AddressOf в частности, вы найдете в таких книгах, как, например, *Peter Norton's Complete Guide to Windows 98, Second Edition* издательства Sams.

## Резюме

На этом занятии мы подробно рассмотрели все операторы языка программирования Access VBA. Они будут использоваться вами постоянно — в любом выражении программного кода VBA, которое когда-либо придется написать. Вы узнали о критериях классификации операторов по числу используемых ими операндов, типах данных и приоритетах выполнения.

Операторы сравнения и логического исчисления обычно применяются в структурах программного кода, управляющих последовательностью действий. Арифметические и логические операторы применяются при расчетах. Вы будете сталкиваться с ними вновь и вновь — на протяжении всех оставшихся двадцати занятий — и таким образом приобретете богатый опыт их практического использования.

В следующей главе мы расскажем о применении операторов сравнения и логического исчисления в прогаммных управляющих структурах. Управление потоком вычислений — один из основополагающих аспектов профаммирования.

## Вопросы и ответы

**Вопрос.** Каким образом можно превратить несколько разрозненных значений различных типов в единую строку?

**Ответ.** Объявите переменную типа `string` и примените оператор конкатенации (&) с неявным преобразованием типов данных.

**Вопрос.** Как преобразовать десятичное число в двоичное?

**Ответ.** Если вы затрудняетесь сделать это самостоятельно (вспомните материал раздела "Побитовые операции" текущей главы), обратитесь за помощью к стандартному Windows-приложению Калькулятор в режиме Инженерный.

**Вопрос.** Как упростить сложное выражение?

**Ответ.** Используйте круглые скобки и/или разделите сложную конструкцию на последовательность из нескольких простых.

## Задания

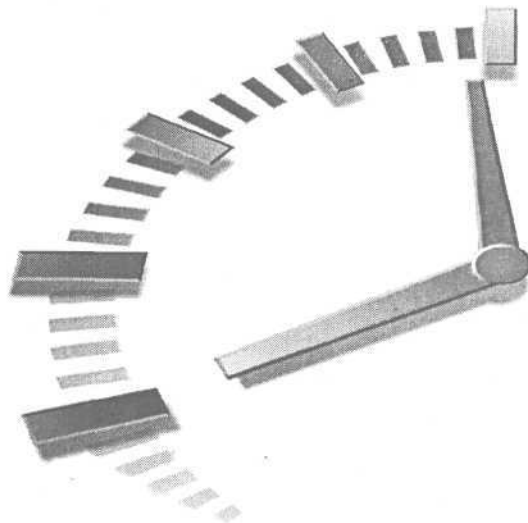
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Для каких целей применяется оператор `+`?
2. Назовите четыре основные фуппы операторов.
3. Каков приоритет выполнения фупп операторов?
4. Как изменить естественный порядок выполнения операций?
5. Операторы какой фуппы обладают наивысшим приоритетом выполнения?

## Упражнения

1. Напишите текст подпрограммы, демонстрирующей все возможные сочетания значений аргументов оператора `And`.
2. Используя окно непосредственного исполнения кода (Immediate) интегрированной среды профаммирования Microsoft Visual Basic, вычислите значения следующих выражений на основе побитового оператора: `Or 2 Or 3; 4 Or 5; 6 Or 7`.
3. С помощью средств окна Immediate вычислите остаток от деления пар чисел, приведенных в предыдущем упражнении.



# Часть II

## Основы конструирования программ

### Темы занятий

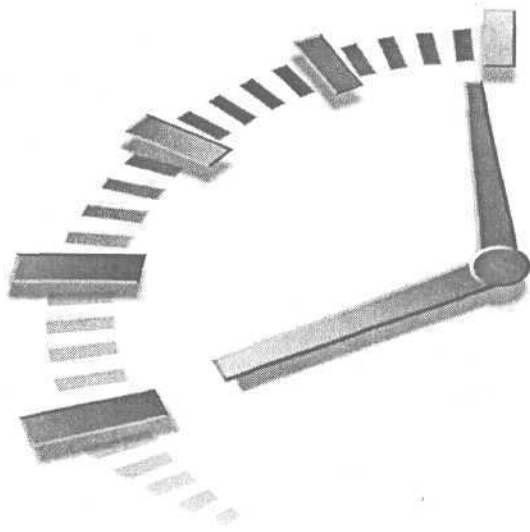
5-й час. Программирование управляющих структур

6-й час. Управление базами данных

7-й час. Расширенные типы данных



## 5-й час



# Программирование управляющих структур

Управляющие структуры в программном коде подобны постовому, регулирующему дорожное движение. Каждая отдельная строка кода решает свою маленькую задачу — вычисляет выражение, выводит данные в файл или загружает их с Web-страницы Internet. Необходимо лишь, чтобы все участники действовали согласованно в достижении общей цели.

Предложенная метафора с постовым-регулирующим может быть заменена и другими. В процессе написания программы вам, возможно, удобнее представлять себе оркестр, где каждый музыкант под неусыпным руководством дирижера исполняет собственную партию, достигая гармонии с остальными. Подойдет и модель действий авиадиспетчера, водителя автобуса и т.п. Все перечисленные примеры с достаточной степенью точности отражают сущность программных управляющих структур, которые предоставляют возможность изменения порядка выполнения операций в соответствии с алгоритмом решения задачи и текущим содержимым объектов данных.

Вы вправе спросить, каким образом программе удастся выполнять действия в разном порядке, успешно достигая конечного результата. На самом деле, программа не обязательно выполняет операции последовательно, в четком порядке, поскольку она работает с данными, и определенные ее действия подходят для одного набора значений данных, а другие уместны в иных случаях. Хороший и понятный пример — расчет подоходного налога. Доходы различной величины облагаются налогом на основании соответствующей ставки; с ростом уровня доходов увеличивается и ставка налогообложения. Таким образом, значения данных, можно сказать, "управляют" поведением программы, которая их обрабатывает.

В этой главе мы рассмотрим вопросы логической организации программного кода. Основные темы занятия.

- Как создавать управляющие структуры.
- Использование конструкции Select Case.
- Приемы построения нелинейного программного кода.
- Применение массивов и коллекций данных.

# Кто следит за порядком на дороге

В своих рассуждениях об управляющих структурах программы будем следовать метафоре постового-регулирующего — автору книги она по душе (нет, отдельные сотрудники автоинспекции мне как раз не симпатичны, но абстрактная модель, представляющая личность, которая следит за происходящим в программе, — совсем другое дело).

**Одна из самых употребительных конструкций языка Access VBA — это выражение**  
If ... Then ... End If.:

If (условие) Then  
    ' одна или несколько строк кода  
End If

## Новый термин

Далее употребляется термин *выражение*. Оператор If является выражением, поскольку выступает стандартной формой кода, выполняемого по условию.

С помощью комментария *Одна или несколько строк кода* мы обозначили место — между первой строкой конструкции, содержащей служебные слова If, Then, и последней, End If, — в котором можно расположить любое число строк кода. Условное выражение — самая главная часть управляющей структуры, которая содержит любое выражение, возвращающее результат типа Boolean. В условном выражении могут использоваться операторы сравнения и логические операторы (о них речь шла на прошлом занятии), а также результаты вызова функций, возвращающих булевы значения. (Подробнее о функциях см. главу "8-й час. Декомпозиция задач".)

Сущность выражения If состоит в том, что последующие строки кода (до строки End If) выполняются только в том случае, когда результат вычисления условного выражения, заключенного в скобки, равен True. Листинг 5.1 содержит несколько полезных примеров использования конструкции If, которые вам следует внимательно изучить. (Приведенный код можно проверить, набрав и откомпилировав его в окне редактора Microsoft Visual Basic, либо исполнить непосредственно с помощью средств окна Immediate.)

## Листинг 5.1. Примеры использования конструкции If ... Then ... End If

```
1: Sub IfTest( )
2:   Const PROMPT = "Введите значение дохода:"
3:   Const TITLE = "Подходящий налог"
4:   Const HIGH_TAX_RATE = "Поздравляем, с вас возьмут больше!"
5:
6:   Dim TaxableIncome As Double
7:
8:   TaxableIncome = Val( InputBox( PROMPT, TITLE, "40000" ))
9:
10:  If (TaxableIncome > 40000) Then
11:    MsgBox HIGH_TAX_RATE
12:  End If
13:
14:  Const PROMPT_1 = "Введите имя пользователя:"
15:  Const TITLE_1 = "Имя пользователя"
16:  Const WELCOME_ADMIN_USER = "Администратор, добро пожаловать!"
17:  Dim UserName As String
18:  UserName = "guest"
19:  UserName = InputBox( PROMPT_1, TITLE_1, UserName)
20:  If (UserName = "Admin") Then
```

```

21:      MsgBox WELCOME_ADMIN_USER
22: End If
23: End Sub

```



Напомним: вы сможете протестировать код листинга 5.1, если введете текст процедуры в окне редактора Microsoft Visual Basic и воспользуетесь клавишей < > для последовательного исполнения по строкам.

## Анализ

Первое, что нельзя не отметить, взглянув на содержимое листинга 5.1 - текст выглядит как полноценная, довольно сложная программа. В строках 1 и 23 находятся операторы, определяющие начало и конец процедуры `Sub IfTest`. В строках 2, 3 и 4 объявляются и инициализируются именованные символьные константы; первая из них (`PROMPT`), например, получает значение "Введите значение дохода:". В строке 6 объявлена переменная `TaxableIncome` типа `Double`. Тип `Double` соответствует числам с плавающей запятой. В строке 8 используются две функции, причем код выполняется справа налево, т.е. первой вызывается вложенная стандартная функция `InputBox`. (В данный момент воспринимайте формат обращения к функции `InputBox` таким, каким он есть; если же вам не терпится вникнуть в подробности, обратитесь за помощью к оперативной справочной системе Access.) При вызове функции `InputBox` открывается диалоговое окно, показанное на рис. 5.1.

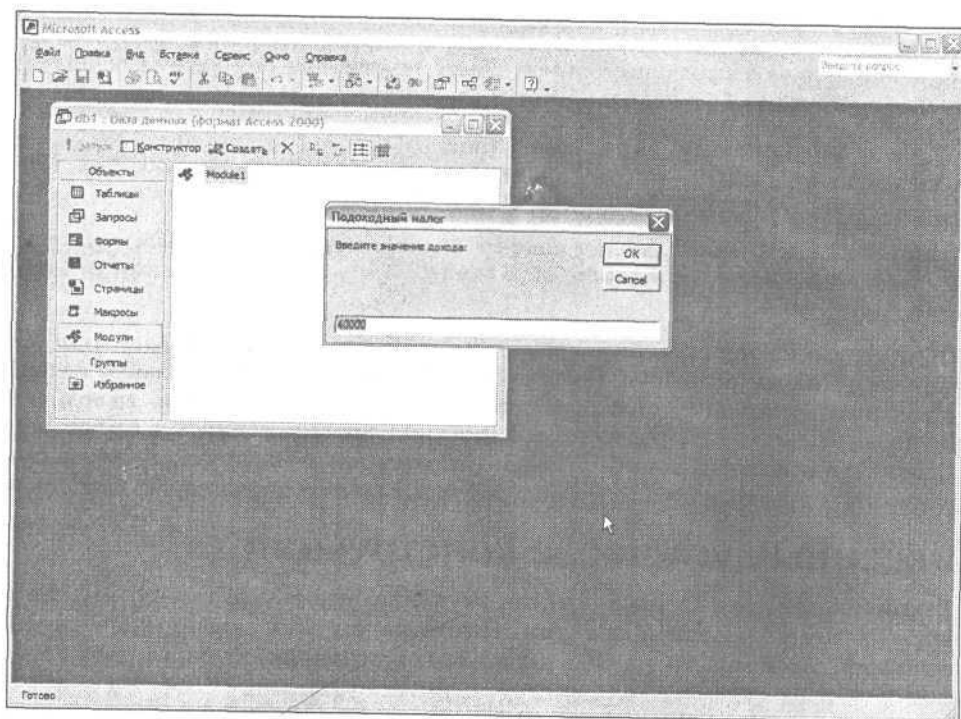


Рис. 5.1. Вызов стандартной функции `InputBox` приведет к открытию соответствующего диалогового окна



Функция `InputBox` возвращает строку символов, введенных пользователем в поле редактирования (оно расположено в нижней части диалогового окна — см. рис. 5.1). Внешняя функция выражения в строке 8 листинга (`Val`) выполняется после завершения работы функции `InputBox`. (Если необходимо уточнить детали, обратитесь к справочной системе Access.) `Val` осуществляет преобразование строки символов, возвращенной функцией `InputBox`, в число двойной точности. Поскольку результат работы `InputBox` — это строка, а в объявлении переменной `TaxableIncome` указан тип `Double`, `Val` — это как раз тот инструмент, который нам нужен. В строках с 10-й по 12-ю содержится выражение `If`, в котором полученная числовая величина дохода сопоставляется с константой-границей, и в случае превышения последней на экран выводится окно сообщения.

В строках 14–16 объявляется новый набор именованных констант, две из которых будут использованы в качестве аргументов в вызове функции `InputBox`, содержащейся в строке 19, а третья — далее. В строках 17 и 18 объявляется и инициализируется символьная переменная `UserName`. Строки 20–22 содержат выражение `If` — в нем значение переменной `UserName`, которое редактируется пользователем в диалоговом окне, вызванном функцией `InputBox` (см. строку 19), сопоставляется с литеральной константой `"Admin"`. В случае совпадения открывается окно сообщения.

Итак, с помощью конструкции `If` допускается анализировать и числовые величины, и строки. Помните, что при использовании конструкции `If` программа должна реагировать как на положительный результат сравнения `True`, так и на отрицательный `False`.

## Расширение выражения If с помощью Else

Служебное слово `Else` используется с целью разбить выражение `If` на два блока кода. Строки кода верхнего блока `If` выполняются, если результат сравнения равен `True`, а нижнего — в противном случае. Далее приводится синтаксис расширенного формата конструкции `If` с использованием слова `Else`:

<div style="writing-mode: vertical-rl; transform: rotate(180deg);">             синтаксис         </div> <div style="font-size: 2em; margin: 5px 0;">             ⤵         </div> <div style="font-size: 2em; margin: 5px 0;">             ⬇         </div> <div style="font-size: 2em; margin: 5px 0;">             A         </div>	<code>If</code> (условное выражение) <code>Then</code> ' одна или несколько строк кода, выполняемых при значении <code>True</code> условного выражения <code>Else</code> ' одна или несколько строк кода, выполняемых при значении <code>False</code> условного выражения <code>End If</code>
--	---

Обратите внимание: служебное слово `Else` занимает отдельную строку и разделяет выражение `If` на две половины. В каждой из них может содержаться любое количество строк кода. Запомните: строки, расположенные над словом `Else`, выполняются, если Условное Выражение равно `True`, а нижний блок выражений — в случае `False`.

Слово `Else` использовать необязательно — обращайтесь к нему в тех случаях, когда необходимо отреагировать на оба возможных результата вычисления условного выражения.

## Вложенные условные конструкции

Выражение `If` допускается размещать внутри другого выражения `If`, как над служебным словом `Else`, так и под ним. Подобные условные конструкции называют вложенными. Несмотря на то, что с формальной точки зрения такие трюки вполне правомерны и на первый взгляд кажутся оправданными, эффективными и даже забавными, на практике применять их не рекомендуется. Листинг 5.2 содержит пример условной конструкции, построенной на основе нескольких вложенных выражений `If`.

```

1: Sub NestedIfTest ( )
2:     Const PROMPT = "Введите имя пользователя:"
3:     Const TITLE = "Имя пользователя"
4:     Dim UserName As String
5:     UserName = InputBox( PROMPT, TITLE, UserName )
6:     If (UserName = "Admin") Then
7:         ' одна или несколько строк кода
8:     Else
9:         If (UserName = "ADMIN") Then
10:            ' одна или несколько строк кода
11:        Else
12:            If (UserName = "guest") Then
13:                ' одна или несколько строк кода
14:            End If
15:        End If
16:    End If
17: End Sub

```

Строки листинга демонстрируют применение вложенных условных выражений, анализирующих содержимое переменной `UserName`. Хотя по существу ничего сложного в этом коде нет, увлечься им не стоит. А если вас все-таки угораздило написать что-то подобное, по крайней мере, позаботьтесь о выравнивании соответствующих слов `If`, `Else` и `End If` по горизонтали. Отметим, что такой стиль программирования нельзя назвать положительным, поэтому нужно избавляться от него всеми возможными средствами.

Существует альтернативная конструкция `Select Case`, более простая и доступная для понимания. С ее помощью вам удастся избежать нагромождений вложенных выражений `If`.

## Будьте аккуратны

Код, приведенный в листинге 5.2, заслуживает внимания, но это, вероятно, простейший пример программ, содержащих вложенные выражения `If`. Если конструкция усложняется, она превращается в ребус, недоступный для понимания.


Впрочем, существуют способы программирования, позволяющие уклониться от необходимости обращения к вложенным условным выражениям. Сделайте первый шаг — для того, чтобы осознать негативные стороны подобного стиля работы. И еще: осваивайте практические приемы решения проблемы.

В следующем разделе будет рассказано об использовании выражения на основе служебного слова `Case`. Мы также рассмотрим приемы упрощения кода, расположенного внутри условной конструкции — в промежутках между словами `If` и `Else`; `Else` и `End If` с помощью вынесения его в отдельные функции. Чтобы детальнее ознакомиться с функциями, обратитесь к главе 8.

Листинг 5.2 — прекрасная иллюстрация того, как может выглядеть код, упорядоченный с помощью функций. Достаточно только вместо строк комментариев, обозначенных оператором апострофа (`'`), вставить вызовы соответствующих функций. (Напомним — более подробные сведения о функциях приведены в главе "8-й час. Декомпозиция задач".)

# Как избавиться от вложенных условных выражений

ЕСЛИ программа содержит несколько вложенных конструкций If, подумайте о замене их условным выражением Select Case. (Далее для краткости будем называть его выражением Select либо Case.) Конструкция Select имеет следующий синтаксис.



```
Select Case значение
    Case условие1
        ' код для Case1
    Case условие 2
        ' код для Case2
    ...
    Case условие N
        ' код для CaseN
    [Case Else]
        ' необязательный код для Case Else
End Select
```

Конструкция Select начинается со служебных слов Select Case, за которыми следует наименование переменной любого простого типа данных, например string, integer или Double. В последней строке содержится словосочетание End Select. Между первой и последней строками может находиться произвольное число строк вида Case Значение, где в качестве значения используется именованная или литеральная константа того же типа, что и тестируемая переменная. Каждая такая строка обозначает ситуацию равенства переменной и указанной здесь константы. Далее приводится набор строк кода, отвечающий конкретному случаю равенства. Частный случай конструкции Select с единственной строкой Case в точности аналогичен выражению If. После перечисления всех случаев равенства разрешается включить в выражение необязательный раздел Case Else (факт ее необязательности отмечен в описании синтаксиса квадратными скобками). Этот раздел — рекомендуем *всегда* особо изучать целесообразность его использования — отвечает тому случаю, когда содержимое переменной не равно ни одному из значений, явно оговоренных выше посредством выражений Case.

Листинг 5.3 демонстрирует пример упрощения кода, рассмотренного нами ранее (см. листинг 5.2). Многочисленные выражения If заменены единой доходчивой конструкцией Select.

Листинг 5.3. Пример замены вложенных выражений If конструкцией Select

```
1: Sub SelectCaseTest ( )
2:   Const PROMPT = "Введите имя пользователя:"
3:   Const TITLE = "Имя пользователя"
4:   Const WELCOME_ADMIN_USER = "Администратор, добро пожаловать!"
5:   Const WELCOME_GUEST_USER = "Гость, добро пожаловать!"
6:   Const WELCOME_SUPER_USER = "Суперпользователь, добро пожаловать!"
7:   Const BAD_USER = "А вам сюда нельзя!"
8:   Dim UserName As String
9:   UserName = "guest"
10:  UserName = InputBox ( PROMPT, TITLE, UserName )
11:  UserName = UCase( UserName )
12:  Select Case UserName
```

```

13:      Case "ADMIN"
14:          MsgBox WELCOME_ADMIN_USER
15:      Case "GUEST"
16:          MsgBox WELCOME_GUEST_USER
17:      Case "pkimmel"
18:          MsgBox WELCOME_SUPER_USER
19:      Case Else
20:          MsgBox BAD_USER, vbExclamation
21:  End Select
22: End Sub

```

## Анализ

В строках с 2-7 объявляются и инициализируются именованные константы, которые будут использованы далее. Строка 8 содержит объявление переменной `UserName` типа `String`, а строка 9 — выражение ее инициализации значением `"guest"`. Рекомендуется по возможности инициализировать переменные сразу же после их объявления. Строка 10 содержит вызов функции `InputBox`, которая открывает диалоговое окно запроса. В строке 11 используется стандартная функция `UCase`, служащая для преобразования символов строки аргумента к верхнему регистру — это позволит ограничиться сопоставлением значения переменной со строковыми литералами, набранными в верхнем регистре. В строке 12 берет начало выражение `Select Case`, которое предназначено для анализа содержимого переменной `UserName`. Программа будет последовательно проверять каждое `Case`-условие равенства до тех пор, пока не найдет совпадения; в противном случае выполнится код секции `Case Else`. Если `UserName` равна `"ADMIN"`, откроется окно сообщения, отображающее значение константы `WELCOME_ADMIN_USER`. Если `UserName` равна `"GUEST"`, в окне появится приветствие гостю. Если же введенное пользователем имя не совпадает ни с `"ADMIN"`, ни с `"GUEST"`, ни с `"pkimmel"`, тогда выполнится код раздела `Case Else`, т.е. откроется окно, отображающее значение константы `BAD_USER` (рис. 5.2).



Рис. 5.2. Пример окна сообщения, вызванного посредством процедуры `MsgBox` с указанием имен стандартной пиктограммы предупреждения

Access предоставляет в ваше распоряжение обширную библиотеку стандартных подпрограмм и функций. Многие из них, включая знакомую вам процедуру `MsgBox`, предусматривают возможность задания большого числа дополнительных необязательных параметров. Если вы внимательно изучите код листинга 5.3, то обратите внимание, что `MsgBox` дважды вызывается с указанием единственного параметра типа `String`, а в строке 18 ей передается уже два параметра. Второй из них — это предопределенная константа `vbExclamation`, обозначающая одну из многочисленных стандартных пиктограмм (ее легко заметить в левой части рис. 5.2). Чтобы научиться пользоваться встроенными процедурами и функциями, необходима практика работы с материалами оперативной справочной системы Access. Если бы в моем распоряжении было не менее тысячи страниц книги, я рассказал бы обо всем гораздо подробнее, хотя, вероятно, это лишено особого смысла, поскольку запомнить такой объем информации почти невозможно. Нельзя запастись впрок знаниями такого характера — почаще обращайтесь с вопросами к справочной системе Microsoft Office.

# Хотите немного покружиться?



Access VBA, как и все другие языки программирования, наделен возможностями организации циклических вычислений. Одна из конструкций, позволяющих решить подобную задачу, — While ... Wend. Она позволяет выполнять одну и ту же последовательность строк кода до тех пор, пока результат вычисления логического выражения, расположенного после служебного слова while, остается истинным. Синтаксис управляющей структуры While ... Wend таков:

```
While (условное выражение)
    одна или несколько строк кода
Wend
```

В первой строке конструкции находится служебное слово While, которое сопровождается заключенным в круглые скобки логическим выражением, возвращающим результат типа Boolean.

Если результат вычисления логического выражения равен False, код, расположенный внутри конструкции, не будет выполняться вовсе. Повторим — структура вида while ... Wend позволяет выполнить одну и ту же последовательность действий нуль или более раз. Пример использования While ... Wend приведен ниже, в тексте листинга 5.4.



В качестве условного выражения в конструкции While ... Wend могут использоваться не только логические выражения, дающие в результате расчета значения типа Boolean, но и арифметические, которые возвращают значения Integer. В последнем случае целочисленное значение 0 будет трактоваться системой как False, а любые целые числа, не равные 0, — как True. Например, конструкция вида While (Целочисленная Переменная) вполне работоспособна.

## Листинг 5.4. Пример использования управляющей структуры While ... Wend

```
1: Sub WhileWendDemo ( )
2:   Dim User As String
3:   User = InputBox("Добавьте имя пользователя", "Добавление_
   пользователей ", User )
4:   While (Len( User ) > 0)
5:     MsgBox "Добавление пользователя" & User, vbInformation
6:     User = InputBox("Добавьте имя пользователя", "Добавление)_
   пользователей ", User )
7:   Wend
8: End Sub
```

Листинг 5.4 демонстрирует процесс ввода имен пользователей, выполняемый до тех пор, пока не выбрана кнопка Cancel (Отмена) либо не введено имя нулевой длины. Строки 1 и 8 задают начало и конец процедуры WhileTest. В строке 2 объявляется переменная User типа String, предназначенная для хранения текущего имени. В строке 3 вызывается знакомая вам функция InputBox, предоставляющая пользователю программы возможность ввести первое имя. Строка 4 — начало конструкции цикла while. В качестве условного выражения выступает оператор сравнения Len( User ) > 0. Здесь используется встроенная функция Len, вычисляющая длину строки-аргумента. Строка 5

содержит код, обрабатывающий введенное значение; в данном случае оно просто высвечивается на экране с помощью процедуры `MsgBox`. Далее пользователь получает возможность ввести новое имя. Выражение `Wend`, расположенное в строке 7, заставляет компилятор вернуться к строке 4 программы и вновь проверить результат условного выражения. Если пользователь щелкнет на кнопке `Cancel` (Отмена) либо введет строку нулевой длины, выполнение цикла прервется, и управление будет передано строке, следующей за `Wend`. В противном случае строки кода 4 и 5 будут выполняться циклически.



Помните о том, что ваш код должен корректно обрабатывать обе ситуации — и истинности, и ложности результата вычисления условного выражения.

Легко составить такое условие, при котором цикл `While ... Wend` будет выполняться бесконечно. Условное выражение вычисляется в начале цикла, и когда результат становится равным `False`, выполнение цикла завершается. Но если в качестве условия задать такое выражение, как, скажем, `1 = 1`, его результат будет всегда оставаться равным `True`. Ниже приведен пример бесконечного цикла.

```
Sub InfiniteLoop()  
    While (1)  
        MsgBox "Это бесконечный цикл"  
    Wend  
End Sub
```



Если вы попытаетесь выполнить данную процедуру, для остановки выполнения кода нажмите `<Ctrl+Break>`. Используйте эту комбинацию клавиш при работе в Access в случае, если какой-либо процесс окажется бесконечным.

В языке **VBA** предусмотрены специальные способы выхода из бесконечных циклов — мы расскажем о них в разделе "Средства прерывания циклов", завершающем эту главу.

## Другие виды условных циклов

Помимо управляющей структуры `While ... Wend`, в вашем распоряжении имеются и другие средства организации циклических вычислений. Одно из них — конструкция `Do ... Loop`, которая может применяться в двух формах. Первая из них предполагает проверку условия в начале цикла, а другая — в конце него. Так выглядит синтаксис цикла `Do ... Loop` с начальной проверкой условия:

```
Do [{While I Until} условное выражение]  
    ' одна или несколько строк кода  
Loop
```

Условное выражение, подлежащее тестированию, расположено в начале цикла, и конструкция действует почти так же, как и ранее рассмотренная структура `While ... Wend`. Если применяется служебное слово `While`, сходство обеих структур становится полным.

Служебное слово `Until` предполагает задание обратного по смыслу условного выражения — если с `Do While` использовалось, скажем, выражение на основе оператора сравнения *больше* (`>`), то в конструкции `Do Until`, выполняющей те же действия, придется употребить оператор *меньше или равно* (`<=`).



Поскольку условное выражение находится в начале цикла, не исключена ситуация, что код внутри цикла не выполнится ни разу. При использовании второй формы цикла `Do ... Loop`, с проверкой условия в конце, код будет гарантированно выполнен, по меньшей мере, один раз. Синтаксис конструкции с последующим тестированием условного выражения выглядит так:

```
Do
    ' одна или несколько строк кода
Loop [{While | Until} условное выражение]
```

Как и ранее, допускается применять служебные слова `While` либо `Until`. При использовании `Until` не забудьте о необходимости "обращения" условного выражения. В принципе, варианты применения `while` или `Until` равнозначны — выбор того и иного из них зависит только от вас.

Рассматривая конструкцию `While ... Wend`, мы говорили, что результатом вычисления условия может быть как булево значение, так и величина типа `Integer`. То же справедливо и в отношении циклов формата `Do ... Loop`. Если вы намереваетесь тестировать целочисленные значения, имейте в виду, что 0 равносителен `False`, а любые ненулевые значения — `True`. Приведенный ниже листинг 5.5 представляет собой исправленный вариант листинга 5.4 и решает ту же задачу.

#### Листинг 5.5. Пример использования `Do ... Until`

```
1: Sub AddUsers ( )
2:   Dim User
3:   Do
4:     User = InputBox("Добавьте имя пользователя", "Добавление_
пользователей ", User )
5:     MsgBox "Добавление пользователя" & User, vbInformation
6:     Loop Until (Len(User) = 0)
7:   End Sub
```

Между двумя вариантами процедур, представленных в листингах 5.4 и 5.5, имеется ряд различий. Прежде всего, примите к сведению, что в листинге 5.5 функция `InputBox` вызывается только один раз. Но теперь возникает новая проблема — необходим дополнительный код, который гарантирует, что в строке 5, выполняющей обработку введенного имени, не будет приниматься во внимание последовательность нулевой длины. Условие из строки 4 листинга 5.4 переместилось в строку 5 листинга 5.5, но теперь здесь применяется оператор равенства — ведь мы воспользовались служебным словом `Until`.

Применяя `Until`, не забудьте о необходимости "обращения" условного выражения. В листинге 5.6 приведен несколько усовершенствованный вариант цикла `Do Loop`, учитывающий возможные ошибки при вводе данных.

#### Листинг 5.6. Исправленный вариант процедуры листинга 5.5

```
1: Sub DoLoopDemo ( )
2:   Dim User
3:   Do
4:     User = InputBox("Добавьте имя пользователя", "Добавление_
пользователей ", User )
5:     If (Len( User ) > 0) Then
6:       MsgBox "Добавление пользователя" & User, vbInformation
7:     End If
8:     Loop Until (Len( User ) = 0)
9:   End Sub
```

После ввода дополнительного контролирующего условия код стал намного более безопасным, поскольку он исключает возможность обработки имени нулевой длины. Обратите внимание: программы, представленные в листингах 5.4 и 5.6, функционально идентичны.

## Итерационные циклы

Вы уже ознакомились с управляющими структурами вида `If ... End If`, `While ... Wend` и `Do ... Loop`. В двух последних, циклических, в качестве условия завершения используется логическое и — реже — целочисленное выражение. Но довольно часто встречаются задачи, когда необходимо знать и задавать точное количество шагов выполнения одного и того же блока кода. В подобных ситуациях активно применяются структуры данных в виде массивов и коллекций.

### Новый термин

*Массивом* называется непрерывная последовательность ячеек памяти, адресуемая по имени. Доступ к элементам массива осуществляется по значению целочисленного индекса.

### Новый термин

*Коллекция* — это определяемый пользователем тип данных. Пользователем в данном случае считается программист, создающий код на языке Access VBA. Коллекции подобны массивам, но они обладают некоторыми дополнительными возможностями, выходящими за пределы совокупности методов обычной адресации элементов данных. Подробнее о коллекциях и способах их применения см. главу "6-й час. Управление базами данных".

Язык VBA поддерживает еще одно средство организации циклов. Речь идет о конструкции `For ... Next`, используемой в тех случаях, когда число шагов выполнения одного и того же блока кода (итераций) заранее известно. Управляющая структура `For ... Next` в своей самой простой форме описывается следующим выражением:

```
For Индекс = НижняяГраница To ВерхняяГраница
    ' одна или несколько строк кода
Next [Индекс]
```

Обязательные атрибуты первой строки конструкции — **служебные** слова `For`, `To` и знак равенства (`=`). Запись `For ... Next` следует понимать так: целочисленной переменной `Индекс` цикла присваивается начальное значение `НижняяГраница`; после этого выполняются строки кода, расположенные ниже, до строки со словом `Next`; далее текущее значение переменной `Индекс` увеличивается на единицу и сравнивается с величиной `ВерхняяГраница`; если граница не превышена, строки цикла выполняются в очередной раз; в противном случае цикл завершается. В качестве величин `НижняяГраница` и `ВерхняяГраница` могут использоваться литеральные значения, именованные константы, а также арифметические выражения либо результаты выполнения функций типа `Integer`. (В приведенном ниже разделе "Функции `Lbound` и `Ubound`" рассмотрены специальные средства языка VBA, облегчающие задание нижних и верхних границ изменения переменной цикла.) Последняя строка кода цикла содержит служебное слово `Next`, за которым допускается указывать наименование переменной, используемой в качестве индекса. Несмотря на то, что идентификатором переменной в данном случае можно пренебречь, для облегчения восприятия кода программы все же рекомендуется задавать его, поскольку итерационные циклы, как и иные управляющие структуры, могут быть вложенными, и задача сопоставления строк `For` и `Next` значительно упрощается, если после `Next` указано наименование переменной цикла.

Между начальной и конечной строками конструкции `For ... Next` может быть размещен любой код. Степень его сложности определяется условиями задачи и выбранным методом ее решения. Впрочем, чем более простым окажется код, тем лучше; старайтесь выносить блоки кода, решающие отдельные подзадачи, за пределы управляющих структур и оформляйте их в виде процедур или функций. (Подробнее о



функциях речь пойдет на 8-м занятии.) Листинг 5.7 содержит пример использования структуры For . . . Next; нетрудно заметить его сходство с "текстом листинга 5.5.

#### Листинг 5.7. Пример цикла For ... Next

```
1: Sub AddUsers( )
2:   Dim Users(10) As String
3:   Dim I As Integer
4:   For I = 1 to 10
5:     Users(I) = InputBox( "Введите имя пользователя:", _
        "Имя пользователя" )
6:   Next I
7: End Sub
```



Для наименований массивов рекомендуется использовать существительные множественного числа, как это сделано в строке 2 листинга 5.7.

Процедура AddUsers предполагает ввод в точности десяти имен пользователей (в предыдущих примерах это число не ограничивалось). Строка 2 содержит объявление массива Users, каждый элемент которого представляет собой переменную типа String. В строке 3 объявлена целочисленная переменная цикла I. Строки 4 и 6 ограничивают конструкцию цикла For . . . Next, а строка 5, предписывающая ввод данных, будет повторена указанное число раз (10). На каждом шаге цикла очередному элементу массива, адресуемому индексной переменной I, присваивается значение, введенное пользователем в диалоговом окне. На рис. 5.3 приведено окно Locals, отображающее содержимое массива Users после его заполнения.

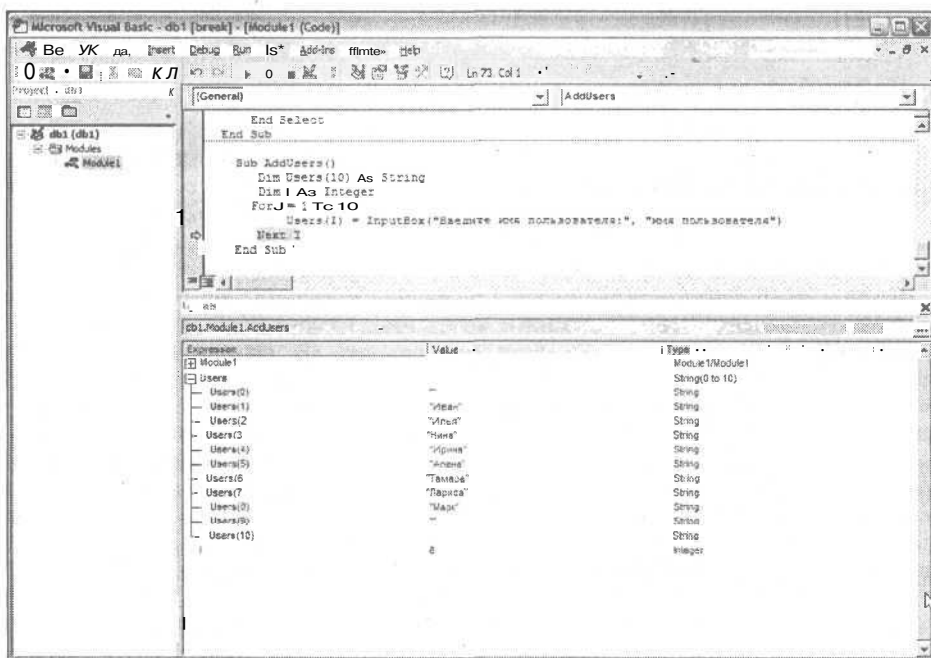


Рис. 5.3. Окно Locals редактора Visual Basic отображает содержимое массива, заполненного пользователем

# Прямая и обратная итерации



Структура For ... Next предполагает возможность выполнения прямой и обратной итераций. Этой цели служит слово step — часть конструкции, позволяющая задавать как положительное, так и отрицательное значение шага (интервала) **приращения** переменной цикла.

For Индекс = НижняяГраница To ВерхняяГраница Step Шаг  
' одна или несколько строк кода  
Next [Индекс]

Выражение синтаксиса почти не изменилось. Единственное отличие состоит в том, что в конец строки For ... To добавлено служебное слово Step, за которым следует значение **шага** приращения переменной цикла. В качестве величины Шаг могут использоваться целочисленные литералы, именованные константы, переменные и арифметические выражения. Теперь цикл может выполняться даже в том случае, если значение выражения ВерхняяГраница меньше значения. НижняяГраница — достаточно после слова step задать отрицательную величину. Листинг 5.8 служит примером цикла с отрицательным шагом изменения индекса.

## Листинг 5.8. Пример цикла For ... Next с отрицательным шагом изменения индекса

```
1: Sub AddUsers( )
2:   Dim Users(10) As String
3:   Dim I As Integer
4:   For I = 10 to 1 Step -1
5:     Users(I) = InputBox( "Введите имя пользователя:", _
   "Имя пользователя". )
6:   Next I
7: End Sub
```

Код листинга 5.8 отличается от предыдущего примера только содержимым строки 4. Теперь массив Users заполняется в обратном порядке, начиная с 10-го элемента.

## Функции Lbound и Ubound

Во многих случаях количество элементов массива или коллекции заранее известно, но существуют ситуации, когда в момент написания кода (или его исполнения) такая информация недоступна. В реальности трудно представить пример точного и заведомо неизменного количества итераций цикла, использующего массивы (коллекции) данных.

Код должен быть гибким — с этим трудно поспорить, — и поэтому создатели Access и VBA предусмотрительно предоставили в распоряжение программистов специальные функции — LBound и Ubound. Lbound возвращает номер первого значимого элемента указанного массива, а Ubound — последнего.

Мы вновь обратимся к знакомому примеру, связанному с заполнением массива имен пользователей в диалоговом режиме, — листинг 5.9 представляет вариант кода, в котором применены функции LBound и Ubound.

## Листинг 5.9. Пример использования функций Lbound и Ubound

```
1: Sub AddUsers ( )
2:   Dim Users(10) As String
3:   Dim I As Integer
4:   For I = Ubound( Users ) to Lbound( Users ) Step -1
```

```

5:      Users(I) = InputBox( "Введите имя пользователя:", _
    "имя пользователя" )
6:      Next I
7: End Sub

```

Обратите внимание — единственное отличие кода листинга 5.9 от предыдущих примеров состоит в том, что литеральные значения, задающие нижнюю и верхнюю границы изменения переменной цикла (и индекса массива) заменены вызовами функций `Lbound` и `Ubound`. Каждой из них в качестве параметра передается имя массива `Users`. Теперь становится вполне понятным общий синтаксис вызова этих функций:



```

Lbound( ИмяМассива )
Ubound( ИмяМассива )

```

Каждая из функций возвращает целочисленное значение. Замените `ИмяМассива` идентификатором реального массива, который используется в программе, — и вы получите гибкий код, который не будет зависеть от количества элементов массива.

Раньше говорилось о том, что использования литеральных значений следует — насколько это возможно в конкретной ситуации — избегать. Профессиональные программисты в шутку называют литералы *магическими числами*, имея в виду их способность к преподнесению "сюрпризов" в самый неожиданный момент. Если все же вместо вызова функций `Lbound` и `Ubound` вы указываете постоянные значения, применяйте именованные константы, а не литералы.

## Циклическая обработка данных

Конструкция `For Each` специально предусмотрена для циклической обработки данных, хранящихся в виде массивов или коллекций. Более подробные сведения о коллекциях приведены в следующей главе, а сейчас предлагаем ознакомиться с приемами использования управляющей структуры `For Each` применительно к массивам. Синтаксис конструкции `For Each` не зависит от того, с каким объектом данных — массивом или коллекцией — вы работаете. Он выглядит следующим образом:



```

Dim Элемент As Variant
For Each Элемент In ИмяМассива
    ' одна или несколько строк кода
Next Элемент

```

В описание синтаксиса специально включено объявление переменной `Элемент`, поскольку конструкция `For Each` должна — это обязательное требование — ссылаться на переменную типа `Variant`. На каждом шаге цикла переменной `Элемент` присваивается значение очередного элемента массива, а в следующих строках у вас появляется возможность обработки этой переменной — вы словно явно ссылаетесь на конкретный элемент массива. Строка `Next Элемент` заставляет компилятор перейти к очередному элементу массива и присвоить его значение указанной переменной. Листинг 5.10 служит примером кода, предназначенного для отображения содержимого каждого элемента массива типа `String` в окне сообщения.

### Листинг 5.10. Пример цикла `For Each`

```

1: Sub ShowUsers ( )
2:   Dim Users(2) As String
3:   Users(0) = "Иван"

```

```

4: Users(1) = "Петр"
5: Users(2) = "Федор"
6: Dim User As Variant
7: For Each User In Users
8:     MsgBox User
9: Next User
10: End Sub

```



По умолчанию в Access VBA нумерация элементов массивов начинается с нуля. Таким образом, если в объявлении массива содержится число *n*, в нем насчитывается *n+1* элемент. Вам, однако, предоставлена возможность выбора границы отсчета с помощью директивы `Option Base n`, где в качестве *n* допускается указывать одно из двух значений — 0 или 1. Строку `Option Base n` необходимо располагать в самом начале текста модуля.

#### Анализ

Строки 2-5 листинга 5.10 содержат инструкции объявления символического массива `Users` и инициализации его элементов. В строке 6 объявлена переменная `User` типа `Variant`, предназначенная для использования в конструкции `For Each`. Строки 7-9 предусматривают циклическое прохождение по всем элементам `Users`, на каждом шаге которого значение очередного элемента (подчеркиваем, *элемента*, а не *индекса*!) присваивается переменной `User` и выводится в окне сообщения, открываемом с помощью стандартной процедуры `MsgBox`.

## Средства прерывания циклов

В практике программирования встречаются ситуации, когда необходимо покинуть цикл до его естественного завершения. Такое случается, если, например, по условиям задачи невозможно построить тестовое выражение, позволяющее корректно завершить выполнение цикла. Язык VBA предлагает для подобной цели специальную инструкцию, обозначаемую служебным словом `Exit`. Она дает возможность выйти из цикла в любой момент, когда это необходимо. Существует две формы инструкции принудительного прерывания циклов:

```

Exit For
Exit Do

```

Первая, `Exit For`, как легко догадаться, используется внутри циклов вида `For ... Next`, а вторая, `Exit Do`, — в циклах `Do ... Loop`. Наличие средства прерывания цикла дает основания предпочесть конструкцию `Do ... Loop` аналогичной ей управляющей структуре `While ... Wend`, лишенной подобной возможности.

Далее вновь обратимся к примеру, приведенному в листинге 5.7. Внутри цикла используется условная конструкция `If`, предназначенная для проверки длины введенного имени и исключающая возможность обработки пустой строки. Теперь мы понимаем, что решение можно усовершенствовать. Соответствующий исправленный вариант кода приведен в листинге 5.11.

#### Листинг 5.11. Пример использования инструкции прерывания цикла

```

1: Sub AddUsers( )
2:     Dim User As String
3:     Do
4:         User = InputBox("Добавьте имя пользователя", _
5:             "Добавление пользователей ", User )
6:         If (Len( User ) = 0) Then Exit Do

```

```

6:      MsgBox "Добавлен пользователь " & User, vbInformation
7:      Loop Until (Len( User ) = 0)
8: End Sub

```

В конец строки 5 добавлена инструкция Exit Do. Если пользователь введет пустое имя либо щелкнет на кнопке Cancel (Отмена) диалогового окна, вызываемого функцией InputBox, выполнение цикла будет принудительно прервано. Итак, объем кода уменьшился на одну строку, и теперь в случае ввода пустого имени выполняется только одна проверка, а не две, как раньше. (В принципе, в строке 7 условие Len( User ) = 0 можно заменить литеральной булевой (True) или целочисленной (1) константой, поскольку выход из бесконечного цикла гарантируется инструкцией Exit Do. — Прим. перев.)



В 1995 году я провел исследование, в результате которого было установлено, что значение средней стоимости строки кода коммерческих программных проектов колеблется от \$5 до \$8. Есть все основания предполагать, что за прошедший период оно существенно выросло. На первый взгляд может показаться — чем больше строк я напишу, тем больше заработаю. Это мнение, однако, небесспорно — разумное и обоснованное уменьшение объема программы с лишней окупит издержки, связанные с сопровождением избыточного и неэффективного кода. (Приведенные цифры отвечают реалиям рынка США. Автор, видимо, не предполагал, что для не избалованного жизнью русскоязычного читателя-профессионала эта информация может представлять особый интерес. — Прим. перев.)

## Использование функции Switch

Функция Switch в качестве параметра использует разделенный запятыми список пар, сравнивает переданное значение с элементами списка и возвращает парное значение. Пример использования данной функции приведен в листинге 5.12.

**Листинг 5.12. Пример использования функции Switch**

```

1: Sub DemoSwitch( )
2: Dim ProductName As String: Dim MatchUp As Variant
3: ProductName = InputBox( _
4:   "Введите название программы (Access, Visual Basic, Delphi)", _
5:   "Соответствие программы языку программирования")
6:   MatchUp = Switch (ProductName = "Access", "VBA", _
7:     ProductName ="Visual Basic", "Visual Basic", _
8:     ProductName ="Delphi", "Object Pascal", _
9:     Len(ProductName) >= 0, "<Не найдено>"
10: End Sub

```

В этой программе пользователь вводит значение переменной ProductName, которое передается в подпрограмму и для которого находится соответствующая пара. В функцию Switch передаются три пары: ProductName = "Access", ProductName = "Visual Basic", ProductName = "Delphi". Если ProductName содержит одно из данных значений, функция Switch возвращает соответствующую ему пару. Например, если в окне ввода указать "Access", функция Switch вернет значение "VBA". Указанная функция возвращает значение типа Variant, что позволяет использовать ее с самыми разными данными.

## Резюме

На этом занятии мы рассмотрели разнообразные средства языка Access VBA, предназначенные для организации циклических вычислений. Конструкции `While ... Wend`, `Do ... Loop` и `For ... Next` позволяют выполнять один и тот же блок кода столько раз, сколько это необходимо или обусловлено текущим состоянием обрабатываемых данных. Выражение `If` дает возможность изменять порядок вычислений в зависимости от результата тестирования заданного условия. Не забывайте о том, что ваш код должен корректно обрабатывать все возможные ситуации.

Чтобы сделать примеры более конкретными и наглядными, автор книги ознакомил вас с несколькими дополнительными приемами, конструкциями и объектами — бесконечными циклами, средствами их принудительного прерывания, массивами, коллекциями и встроенными функциями Access VBA. Я обещаю научить вас программированию — только не ленитесь прочесть эту книгу и испытать на практике все то, о чем узнаете. (Зачем далеко ходить — выполните задания, которые приведены ниже!)

## Вопросы и ответы

**Вопрос.** В каких случаях целесообразно использовать вложенные циклы `For ... Next`?

**Ответ.** Вложенные циклы `For ... Next` оказываются весьма полезны в тех случаях, когда приходится выполнять сложные вычисления с индексированными значениями (что делается, скажем, при реализации алгоритмов сортировки) либо элементами многомерных массивов.

**Вопрос.** Какую из циклических конструкций следует предпочесть — `while ... wend` или `Do While ... Loop`?

**Ответ.** `While ... Wend` поддерживается в языке из соображений обратной совместимости со старыми приложениями. Более предпочтительна конструкция `Do While ... Loop`.

**Вопрос.** Вы упомянули о нескольких полезных встроенных функциях VBA. Как ознакомиться с остальными?

**Ответ.** Файлы оперативной справочной системы Access содержат немало нужных примеров, но зачастую в них бывает трудно отыскать именно то, что требуется в данный момент. Наилучший способ знакомства со всем многообразием стандартных процедур и функций Access VBA — изучение больших "кусков" реального эффективного кода.

**Вопрос.** Что лучше — коллекция или массив?

**Ответ.** Вопрос не совсем корректен, ибо полезными в конкретном случае могут оказаться и коллекции, и массивы. Создавая совершенно новый проект, вы, видимо, предпочтете обратиться к коллекциям. Об этой мощной и эффективной структуре данных мы расскажем в следующей главе.

## Задания

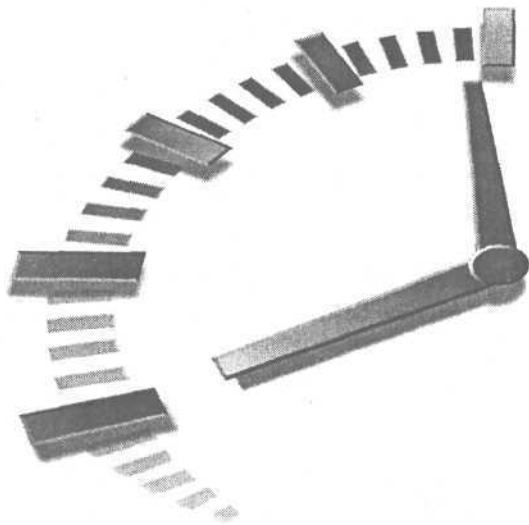
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Как определить цикл для обработки элементов массива произвольной длины?
2. Если необходим такой цикл, код которого должен выполняться по меньшей мере один раз, какую конструкцию вы выберете?
3. Как убедиться, что код выражения `If` работает верно?
4. Для каких целей предназначена инструкция `Exit For`?
5. Что представляют собой средства прерывания циклов? При каких обстоятельствах их следует использовать?

## Упражнения

1. Применяя средства оперативной справки Microsoft Visual Basic, отыщите информацию о функции `Switch`. Напишите код на основе `Switch`, позволяющий найти имя человека по заданной фамилии среди нескольких пар строк вида *имя-фамилия*.
2. Обратившись к системе оперативной справки, найдите сведения о функции `Iff`. Приведите конкретный пример ее использования.
3. Объявите объект коллекции и примените конструкцию `For Each` для отображения содержимого каждого элемента коллекции с помощью процедуры `MsgBox`.



## 6-й час

# Управление базами данных

Мои поздравления! Вы успешно одолели целых пять глав книги, а ведь это немало! Да-а-а, нечего сказать! Эту главу можно считать определенной вехой. Теперь вы научитесь, применяя полученные знания, управлять базами данных, таблицами, столбцами и полями данных.

До сих пор вы знакомились с базовыми конструкциями языка и образцами кода, которые можно использовать для решения самых разнообразных задач. На этом занятии вы сможете реализовать приобретенный опыт, работая с таблицами Access, которые сами же и построите.

В этой главе вы будете иметь дело с примерами кода двух видов. Первые из них, по моему мнению, следует обязательно изучить, а вторые будут играть необходимую вспомогательную роль. Я постараюсь особо оговаривать, на что именно следует обращать внимание в каждом конкретном случае, а какие элементы кода можно в данный момент игнорировать — им будут посвящены отдельные разделы нашей книги.

В этой главе мы рассмотрим вопросы программирования в контексте управления базами данных Access.

Основные темы занятия.

- Совместимость с Access 2000.
- Динамическое создание таблиц.
- Программный код для пополнения базы данных.
- Циклы и условные конструкции для управления столбцами и полями данных.
- Процедуры динамического поиска информации в базе данных.

## Совместимость Access 2002 с Access 2000

Access 2002 и Access 2000 используют совместимые форматы файлов. Access 2002 может читать и изменять файлы баз данных формата Access 2000, конвертировать их в свой формат, а также конвертировать файлы своего формата в формат Access 2000.

Если все пользователи базы данных работают в Access 2002, вероятно, вам захочется преобразовать файлы в один из новых форматов, описанных в следующих разделах.



# Новые форматы файлов, обеспечивающие повышенную производительность

Преобразование базы данных в формат Access 2002 позволяет полностью воспользоваться всеми преимуществами новейшей версии данной программы. Access 2002 предлагает усовершенствованные форматы файлов. Базы данных Access 2002 можно хранить в форматах MDE и ADE.

## Базы данных формата MDE

Увеличить безопасность базы данных Access можно двумя способами: заштитив базу данных паролем с помощью команды **Сервис⇒Защита (Tools⇒Protection)** или создав MDE-файл командой **Сервис⇒Служебные программы⇒Создать MDE-файл (Tools⇒Database Utilities⇒Create MDE File)**.

Сохранение базы данных в формате MDE не позволяет пользователям просматривать, изменять или создавать формы, отчеты и модули. Более того, пользователи не могут средствами программирования добавлять, удалять или изменять ссылки на другие библиотеки объектов ActiveX и вообще менять код.

## Базы данных формата ADE

Если проект Access создается на базе SQL-сервера, а не Jet, будет создан проект ADP (а не база данных MDB). Повысить безопасность такого проекта баз данных можно, преобразовав базу данных ADP в базу данных ADE. Такое преобразование совершается с помощью меню **Сервис⇒Служебные программы⇒Преобразовать базу данных (Tools⇒Database Utilities⇒Convert Database)** только в работе с ADP-проектами.



Для создания ADP-проектов и использования SQL-сервера с целью хранения данных необходимо иметь SQL Server 6.5 Service Pack 5 или его более поздние версии.

## Средства сжатия и восстановления

Access 2002 предлагает усовершенствованные возможности сжатия и восстановления, которые позволяют с минимальными потерями восстанавливать поврежденные формы и отчеты.



Сжатие и восстановление базы данных в Access удаляет NTFS разрешения, если база данных расположена на диске, использующем NT разрешения.

## Создание базы данных

Для получения практических навыков программирования в среде Access вам не обойтись без базы данных. Мы построим ее вместе, а затем будем использовать на протяжении всего занятия.

Научившись создавать базы данных и таблицы в ней с помощью традиционных диалоговых инструментов Access, вы сможете узнать, как та же задача решается динамически, средствами программы. Главное назначение Access и программных систем на ее основе — это управление базами данных. Что ж, приступим.

# Создание базы данных

Access 2002 — это программный продукт, предназначенный преимущественно для управления базами данных. Он, как и другие компоненты пакета Microsoft Office XP, поддерживает язык программирования Visual Basic for Applications (VBA), который и является предметом нашего изучения. VBA для Access отличается, скажем, от VBA для Excel тем, что предлагает специальные средства для работы с базами данных.

В ходе дальнейшего изложения мы будем ссылаться на базу данных, содержащую одну таблицу под названием CONTACTS. Прежде чем построить таблицу, необходимо создать базу данных. Если вы хотели бы освежить в памяти информацию о способах создания баз данных Access, прочтите приведенную ниже инструкцию; вы также можете сразу перейти к следующему разделу.

Чтобы создать базу данных, выполните следующие действия.

1. Запустите программу Access.
2. На панели задач Создание файла (New) щелкните на ссылке Новая база данных (Blank Database) (рис. 6.1).

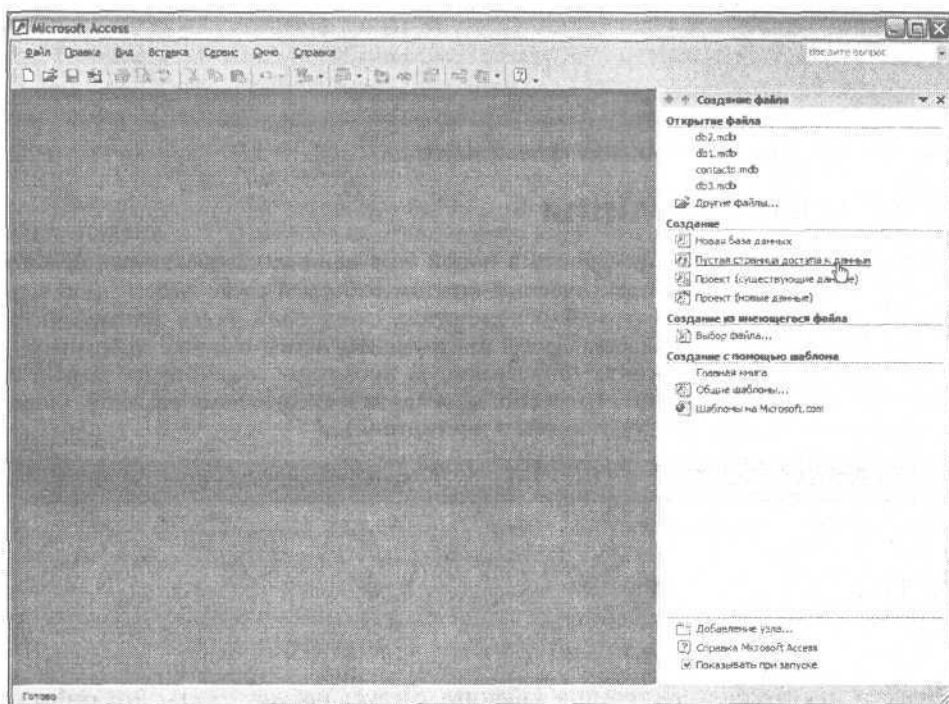


Рис. 6.1. Создайте новую базу данных, пользуясь областью задач Создание файла

3. В диалоговом окне Файл новой базы данных сохраните новую базу данных под именем Contacts.mdb.

После успешного создания файла базы данных откроется дочернее рабочее окно программы, показанное на рис. 6.2.

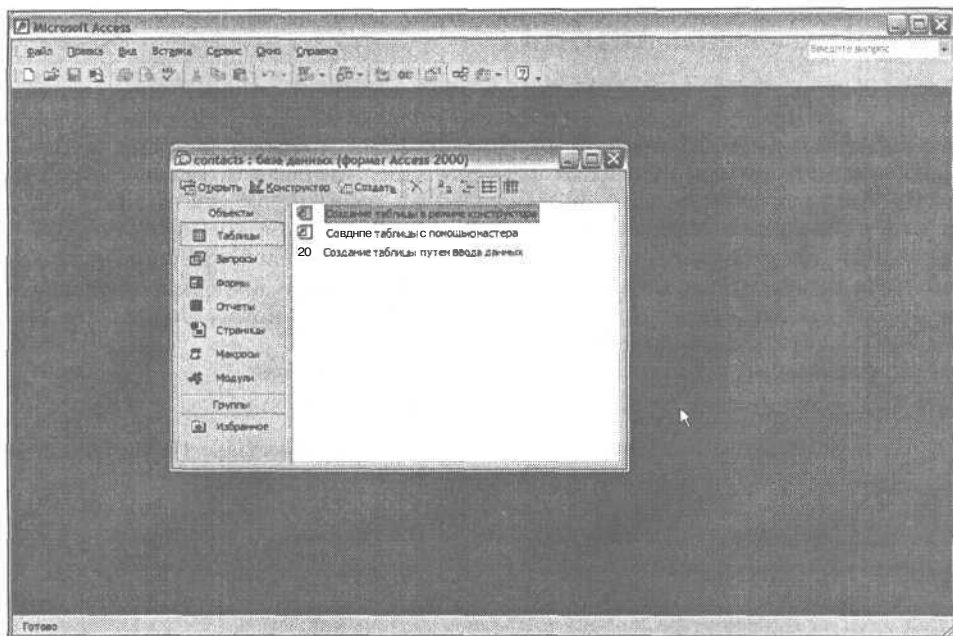


Рис. 6.2. Так выглядит окно базы данных Access

## Построение таблицы

Наша следующая цель — построить в новой базе данных CONTACTS одноименную таблицу. Если вы обладаете практическим опытом работы в среде Access либо знакомы с соответствующей литературой (рекомендуем книгу *Sams Teach Yourself Microsoft Access 2000 in 24 Hours* издательства Sams), инструменты Access вам уже знакомы.

Итак, предлагаем инструкцию, описывающую процедуру создания таблицы в базе данных Access (если вам это не интересно, перейдите к следующему разделу). Определения полей таблицы CONTACTS указаны в листинге 6.1.

### Листинг 6.1. Определения полей таблицы Contacts

```
ID, AutoNumber, Primary Key
FIRST_NAME, Text, 20
LAST_NAME, Text, 20
PHONE_NUMBER, Text, 36
EMAIL, Text, 50
WWW, Text, 50
```

Приводя инструкцию построения таблицы, следует предположить, что ранее созданный файл `Contacts.mdb` базы данных открыт. (Если это не так, обратитесь к предыдущему разделу, "Создание базы данных", текущей главы.) Чтобы создать таблицу CONTACTS, выполните следующие действия.

1. Обратитесь к окну открытой базы данных `Contacts` и выберите элемент Таблицы (Tables) списка Объекты (Objects) (рис. 6.3).
2. В списке, расположенном в правой части окна, выберите элемент Создание таблицы в режиме конструктора (Create Table in Design View) и щелкните на кнопке Создать (New) панели инструментов (см. рис. 6.3).

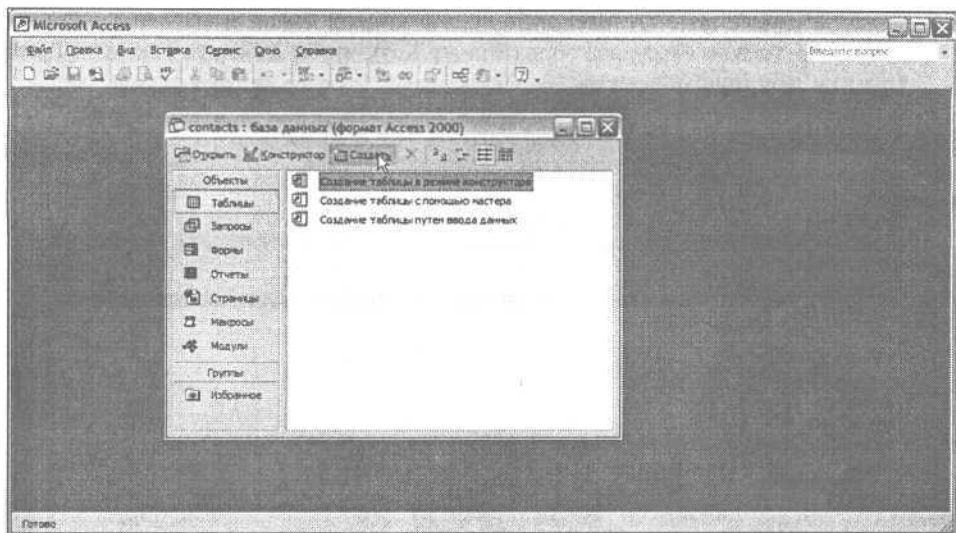


Рис. 6.3. В окне базы данных выберите элемент *Таблицы* списка *Объекты*

3. В списке режимов диалогового окна Новая таблица (New Table) выберите элемент Конструктор (Design View) и щелкните на кнопке ОК.
4. Откроется окно таблицы в режиме конструктора (рис. 6.4). Руководствуясь данными листинга 6.1 и изображением на рис. 6.4, введите сведения о структуре таблицы.

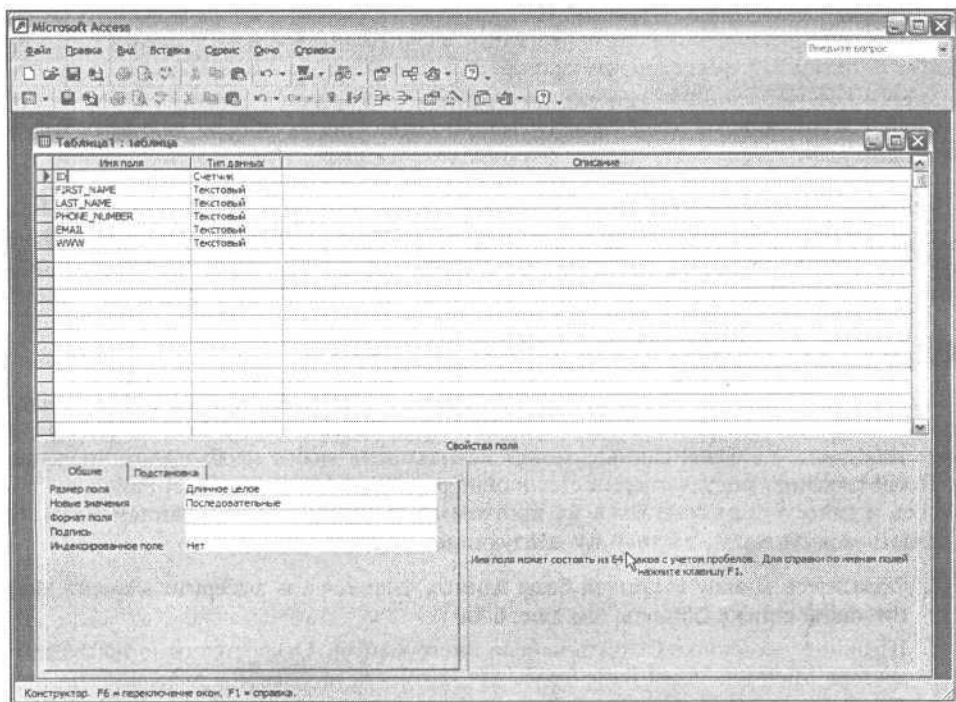


Рис. 6.4. Так задается описание структуры таблицы

5. Указав в списке поле ID, щелкните правой кнопкой мыши и выберите в контекстном меню элемент **Ключевое поле** (Primary Key), чтобы добавить в поле ID признак Primary Key (рис. 6.5).

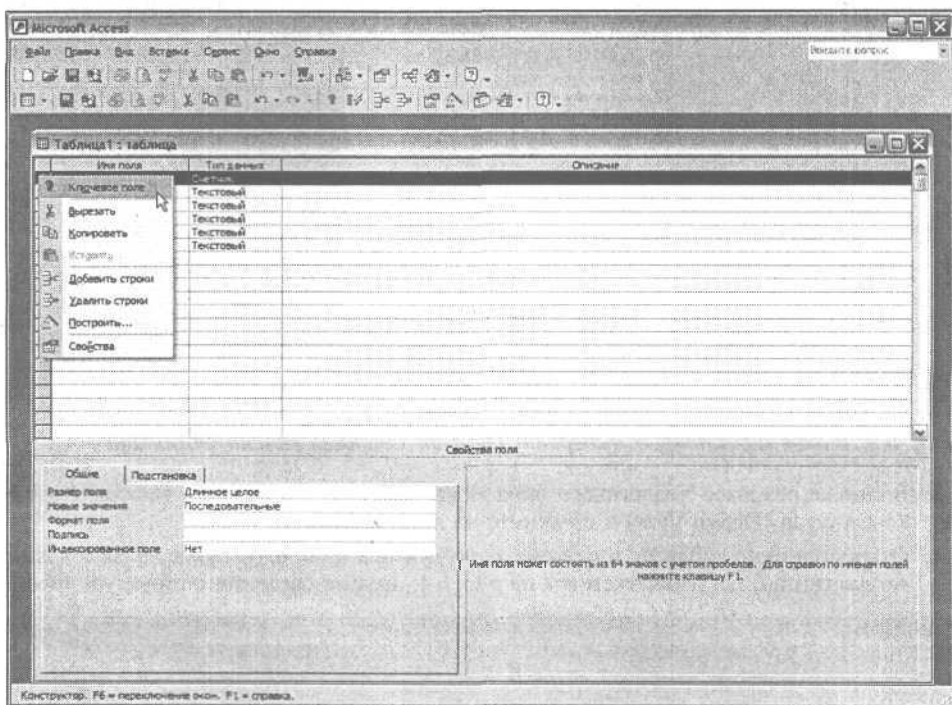


Рис. 6.5. Контекстное меню предоставляет возможность выбора ключевых полей таблицы

Построенной таблицей мы будем пользоваться на протяжении всего занятия. Владея инструментами диалогового проектирования, важно уметь создавать таблицы, подобные CONTACTS, с помощью программного кода. О том, как решать такие задачи, речь пойдет позже. А сейчас поведаем о создании модуля, в котором позже будет размещен код.

## Создание программного модуля

Весь программный код на языке VBA, который вам придется писать, создается внутри модулей. Модуль — это объект Access, который сохраняется в файле базы данных. Приложение, работающее с базой данных, может использовать любое необходимое число модулей. После создания модуль открывается в окне редактора Microsoft Visual Basic.

Здесь и далее будем ссылаться на программный модуль под названием Main. Чтобы создать модуль Main, выполните следующие действия.

1. Обратитесь к окну открытой базы данных Contacts и выберите элемент Модули (Modules) списка Объекты (см. рис. 6.3).
2. Щелкните на кнопке Создать панели инструментов. Откроется окно текстового редактора Microsoft Visual Basic (далее для краткости называемое *редактором*).
3. В строке меню редактора выберите команду **File**⇒**Save Contacts**.

4. В диалоговом окне **Сохранение(Save As)** замените имя **Module1**, предложенное по умолчанию, строкой **Main** и щелкните на кнопке **ОК**.

Модуль **Main** будет служить хранилищем кода всех примеров, которые рассматриваются в ходе этого занятия. При необходимости создания новых модулей выполните рекомендации приведенной выше инструкции. Чтобы открыть окно редактирования существующего модуля, дважды щелкните на имени модуля в окне базы данных.

## Создание таблицы посредством программного кода

Access предлагает несколько инструментальных средств создания таблиц. Ранее мы уже пользовались, например, окном конструктора таблицы. Существует и широко применяется способ создания/изменения объектов базы данных с помощью предложений на языке **SQL**.

Зачастую возникает необходимость динамического создания таблиц непосредственно с помощью прикладной программы. Листинг 6.2 содержит пример кода, предназначенного для построения таблицы **CONTACTS** (описание ее структуры приведено в тексте листинга 6.1).

### Листинг 6.2. Построение таблицы с использованием объекта **ADOX**

```
1 Sub CreateTable( )
2 Const DatabasePath = _
3 "c:\Books\Teach Yourself Access 2002" + _
4 "Programming\Chapter 6\CONTACTS.mdb"
5 Const ProviderStr="Provider=Microsoft.Jet.OLEDB.4.0;" + _
6 " Data source = " + DatabasePath
7: Dim Table As New Table
8: Dim Catalog As New ADOX.Catalog
9: Dim Key As New ADOX.Key
10: Catalog.ActiveConnection = ProviderStr
11: Table.Name = "CONTACTS"
12: Table.ParentCatalog = Catalog
13: Table.Columns.Append "ID", adInteger
14: Table.Columns( "ID" ).Properties( "AutoIncrement" ) = True
15: Table.Columns.Append "FIRST_NAME", adVarChar, 20
16: Table.Columns.Append "LAST_NAME", adVarChar, 20
17: Table.Columns.Append "PHONE_NUMBER", adVarChar, 36
18: Table.Columns.Append "EMAIL", adVarChar, 50
19: Table.Columns.Append "WWW", adVarChar, 50
20: Catalog.Tables.Append Table
21: Key.Name = "ID"
22: KeyType = adKeyPrimary
23: Key.Columns.Append "ID"
24: Catalog.Tables( "CONTACTS" ).Keys.Append Key, kyPrimary
25: Set Catalog.ActiveConnection = Nothing
26:End Sub
```

Листинг 6.2 относится к той разновидности примеров — об этом говорилось в самом начале главы, — которые не обязательно изучать детально. При первом их чтении у вас могут возникнуть сложности — не надо паники, это нормально! Листинг 6.2 содержит текст процедуры **CreateTable**. В строках 7-9 приведены объяв-

ления переменных Table, Catalog и Key. Воспринимайте новое для вас служебное слово New как должное — для объявлений объектов специальных типов данных, классов, требуются дополнительные языковые средства. Соответствующие пояснения вы найдете в главе “21-й час. Основы программирования классов”. ADOX — это объект ActiveX, разработанный специалистами компании Microsoft. На некоторое время мы отложим обсуждение сложных аспектов ADOX — впрочем, если вам невтерпеж, вы можете, не мешкая обратиться к оперативной справочной системе Access; если вы решили не опережать события, дождитесь, пока мы плавным и естественным образом перейдем к главе 21.

Чтобы получить возможность обращения к функциям ADOX, вы должны включить в список поиска внешних библиотек и объектов ActiveX соответствующую ссылку. Для этого выполните следующие действия.

1. Откройте базу данных Contacts.
2. В списке Объекты окна базы данных выберите элемент Модули.
3. Дважды щелкните на имени модуля Main, чтобы открыть окно редактора.
4. Выберите в строке меню команду Tools⇒References; откроется диалоговое окно References.
5. Прокрутите список Available References и установите флажок Microsoft ADO Ext. 2.7 for DDL and Security (рис. 6.6).

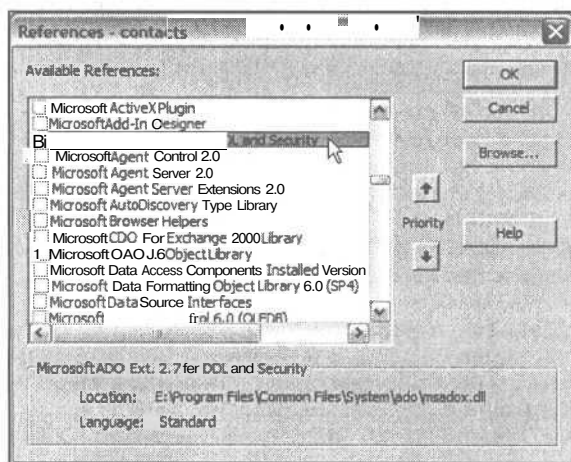


Рис. 6.6. Диалоговое окно References позволяет создавать ссылки на внешние объекты ActiveX

6. Щелкните на кнопке OK.

Выполнив указанные операции, вы получите доступ к объектам ADOX. Строка 5 листинга 6.2 содержат информацию, необходимую для обеспечения возможности подключения к базе данных Contacts.mdb. Если вы набираете рассматриваемый нами код в окне редактора, в точности повторите весь текст, за исключением фрагмента DatabasePath = — здесь необходимо ввести путь к каталогу, в котором находится ваш собственный экземпляр файла Contacts.mdb.

## Я нашел!

Это происходило в III веке до н.э. Правитель Сиракуз как-то заподозрил, что заказанная им корона выполнена не из чистого золота. Владыка вызвал Архимеда (да-да, того самого, которого называют отцом геометрии) и попросил либо доказать, либо опровергнуть правомерность своих опасений. Архимед должен был установить истину, не причинив ущерба короне, Мудрец размышлял над проблемой несколько дней, но ответ был найден неожиданно. Во время посещения общественной бани Архимед заметил, что как только он садится в ванну, уровень воды в ней резко возрастает. Архимед, словно ошпаренный, выпрыгнул из купели и, забыв про одежду, понесся по улицам, оглашая город криками: "Эврика! Эврика!" ("Я нашел!")

Мыслитель нашел решение. Принцип, известный теперь как закон Архимеда, гласит: "На тело, погруженное в жидкость, действует сила выталкивания, равная весу жидкости, вытесненной телом". Сравнив смещения уровней жидкости при погружении в нее короны и известного количества золота, Архимед доказал, что подозрения Сиракуза не были напрасны.

По мнению Гради Буча (Grady Booch), главного исследователя компании Rational Corporation и ведущего редактора отдела литературы по объектно-ориентированному программированию издательства Addison-Wesley, в наше время подобные вспышки озарения обычно сопровождаются восклицаниями "Aha!!". Как бы то ни было, со времен Архимеда человеческая природа мало изменилась — людьми до сих пор руководит жажда открытий и постижения новых таинственных горизонтов знания.

Сейчас вам вовсе не обязательно всецело и доподлинно разбираться в том, что именно написано в тексте листинга 6.2, — понимание, безусловно, придет, но чуть позже, когда в этой и последующих главах книги вы постепенно ознакомитесь со всей необходимой информацией.

Вернемся к листингу 6.2. Строки 13-19 содержат определения столбцов таблицы — те же, которыми мы руководствовались ранее при ее создании с помощью диалоговых средств Access. Строка 14 демонстрирует способ присваивания столбцу признака автоматического приращения (`AutoIncrement`). Инструкция строки 20 добавляет созданную таблицу в каталог (`catalog`). Каталог ссылается на объекты таблиц, именovaných курсоров, учетных карточек пользователей и групп пользователей. Строки 21-24 служат примером определения первичного ключа (`Primary Key`) и добавления его в таблицу. Строка 25 закрывает текущее соединение с каталогом.

Хотя код листинга 6.2, возможно, не до конца вам понятен, воспринимайте неясные фрагменты спокойно. При необходимости создания новой таблицы вы можете просто взять и применить указанные команды, следуя такой обобщенной схеме.

1. В строке 11 замените имя "CONTACTS" новым.
2. В строках 13–19, содержащих определения столбцов, введите новые значения имен столбцов, их типов и дополнительных признаков.
3. В строках 21–23 выберите в качестве ключевых требуемые столбцы.
4. Замените имя таблицы в строках 11 и 24.

Выполнив процедуру, исправленную в соответствии с указанными рекомендациями, вы создадите в базе данных новую таблицу. Сейчас достаточно выявить те элементы кода, которые для вас очевидны. Внимательно просмотрите текст процедуры и убедитесь, что вы способны различать строки, в которых приведены объявления переменных, а также фрагменты с использованием операторов. Вы бесспорно научитесь создавать подобный код к тому моменту, когда перевернете последнюю страницу завершающей главы нашей книги.



# Управление таблицей из среды приложения

Ваш жизненный опыт подсказывает — и это правильно! — что таблица состоит из столбцов и строк. Таблица Access позволяет сохранять данные в промежутках между сеансами работы программных приложений.

Чтобы мы смогли говорить на одном языке, вам необходимо знать несколько основных терминов. Одни из них вам уже известны, а другие вы просто могли забыть. Я представлю вам описание таких терминов, а затем мы перейдем к вопросам управления таблицей посредством прикладного программного кода. Если рассматриваемая система понятий вами уже прочно усвоена, можете смело переходить к следующему разделу.

## Базы данных: основные понятия

Чтобы получить возможность использования таблицы, *хранящейся* в базе данных, необходимо выполнить несколько операций. Прежде всего, следует подключиться к базе данных. Затем с помощью соответствующих инструкций программного кода вы сможете открыть таблицу, выполнить надлежащие операции с определенными данными и закрыть таблицу. Перед завершением работы программы необходимо выполнить операции по очистке памяти.

### Каталог

*Каталог* (catalog) — это объект, служащий синонимом имени файла базы данных. Каталог содержит ссылки на таблицы, курсоры, учетные записи пользователей и групп пользователей.

### Соединение

*Соединение* (Connection) — это объект, осуществляющий связь прикладной программы с файлом базы данных. При создании соединения необходимо указать местоположение файла. Соединение указывает Access, с какой базой данных будет работать ваше приложение. Наиболее важный параметр соединения — наименование файла базы данных.

### Набор данных

*Набор данных* (Recordset) — это общий термин, применяемый при обращении к таблице, запросу или курсору. Объект Recordset применяется для получения информации, хранящейся в группе записей (строк) данных. Термин Recordset в Access равнозначен понятию курсора, принятому в системах управления базами данных с использованием программных серверов Oracle, DB2 или Microsoft SQL Server.

### Записи, столбцы и поля

Набор данных (Recordset), вероятно, легче всего представить в виде страницы привычной электронной таблицы. *Запись* (ROW) — это одна горизонтальная строка страницы, а *столбец* (Column) — набор данных, упорядоченных по вертикали. Элемент данных на пересечении строки и столбца называют *полем* (Field). (В электронных таблицах принят термин *ячейка*.) Открыв таблицу базы данных в среде Access, вы сможете убедиться в ее заметном сходстве с визуальным представлением электронной таблицы.

# Получение информации о структуре таблицы

Вы уже научились создавать таблицы с помощью инструментов Access и средств программного кода. Но наша таблица CONTACTS пока пуста. Прежде чем приступить к ее наполнению, мы еще раз вспомним о том, как открывать и закрывать соединение с базой данных и объект набора данных.

Знания о способах создания соединения и открытия набора данных вам просто необходимы. Эти операции должны предшествовать любым действиям с базами данных. Листинг 6.3 содержит пример кода, иллюстрирующего приемы создания соединения с базой данных, открытия/закрытия набора данных и отображения информации о структуре таблицы.

## Листинг 6.3. Пример открытия таблицы и отображения данных о ее структуре

```
1: Sub DisplayFields( )
2: Const DatabasePath = _
3: "c:\Books\Teach Yourself Access 2002" + _
4: "Programming\Chapter 6\CONTACTS.mdb"
5: Const ProviderStr="Provider=Microsoft.Jet.OLEDB.4.0;" + _
6: " Data source = " + DatabasePath

1: Dim Connection As New ADODB.Connection
2: Dim Catalog As New ADOX.Catalog
3: Dim RecordSet As New ADODB.RecordSet
4: Dim Field As Field
5: Connection.Open ProviderStr
6: Set Catalog.ActiveConnection = Connection
7: RecordSet.Open "CONTACTS", Catalog.ActiveConnection, adOpenKeyset
8: RecordSet.Fields.Refresh
9: For Each Field In RecordSet.Fields
10:     Debug.Print Field.Name & ", " & Field.Type & ", " &
        Field.ActualSize
11: Next
12: RecordSet.Close
13: Set RecordSet = Nothing
14: Set Catalog = Nothing
15: Connection.Close
16: Set Connection = Nothing
17:End Sub
```

Обратите внимание: процедура названа подходящим именем, DisplayFields, поскольку предназначена для отображения сведений о структуре таблицы на экране (строка 1). Строки 2-6 содержат объявления объектов стандартных классов (Connection, Catalog, RecordSet и Field), разработанных Microsoft. (Более подробные сведения о них вы сможете почерпнуть из нескольких последующих глав нашей книги и файлов оперативной справочной системы Access.)

Строка 12 открывает соединение с базой данных. Набирайте этот код в том виде, в каком он приведен, за исключением фрагмента Data Source = — здесь необходимо ввести путь к каталогу, в котором находится ваш собственный экземпляр файла Contacts.mdb. Строка 13 может быть применена вами без каких-либо изменений при открытии этой и любой другой таблицы. Строка 14 содержит инструкцию непосредст-

венного открытия таблицы. Чтобы применить код строки 14 для открытия другой таблицы, достаточно заменить наименование "CONTACTS" требуемым. Выполнение команды Refresh, содержащейся в строке 15, гарантирует получение актуальных данных. (Не забывайте, что с той же таблицей одновременно могут работать и другие программы.)

Строки 16-18 содержат циклическую конструкцию For Each — мы говорили о ней на прошлом занятии, "5-й час. Программирование управляющих структур". Команда в строке 17 выводит на экран данные о наименовании поля, его типе (в виде целого числа) и размере. Чтобы увидеть результаты, обратитесь к средствам окна Immediate (пример вывода данных показан на рис. 6.7). В оставшихся строках кода (19-24) выполняются операции очистки и удаления объектов данных программы. Объекты набора данных и соединения закрываются командами Close. Инструкции присваивания предопределенного значения Nothing осуществляют возврат выделенных переменным фрагментов памяти в общую область (или, как еще говорят, нул) динамически распределяемой памяти. (Подробнее о назначении Nothing и приемах его использования см. в системе оперативной справки Access.)

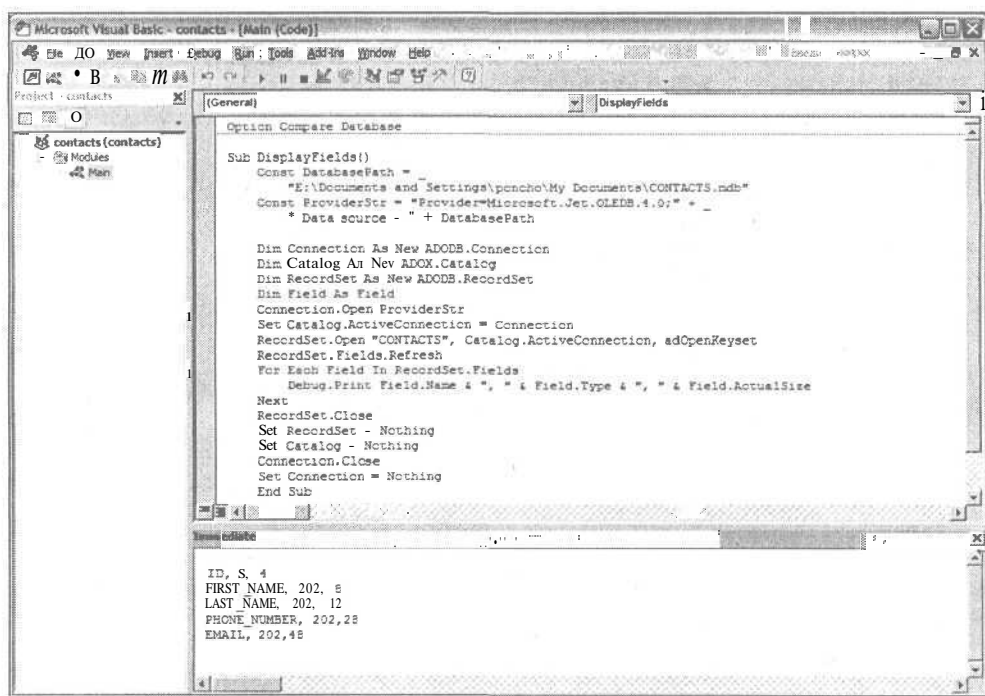


Рис. 6.7. Окно Immediate позволяет увидеть информацию о полях таблицы, которая хранится в виде коллекции, принадлежащей объекту RecordSet



Для корректной работы процедуры листинга 6.3 необходимо, чтобы база данных имела по крайней мере одну запись. На самом деле приведенный код, помещая данные о структуре таблицы в окне Immediate, не должен требовать наличия самих записей, тем не менее в Access 2002 с пустой таблицей он работать не будет.

Столкнувшись с любой задачей управления базой данных, вы неизбежно обратитесь к фрагментам кода, представленного в листинге 6.3. Впрочем, частота и целесообразность использования тех или иных команд зависят от многих факторов. Например, операции открытия и закрытия таблицы обычно выполняются при каждом обращении к ней, но базу данных достаточно открыть один раз, в начале программы, а закрывать имеет смысл в самом конце сеанса работы.

В следующем разделе мы рассмотрим пример заполнения таблицы данными, вводимыми пользователем.

## Циклический ввод данных в таблицу

С целью решения задачи ввода данных в таблицу целесообразно воспользоваться циклической конструкцией. Ниже мы рассмотрим пример, использующий цикл на основе выражений `For . . . Next`. Чтобы упростить изложение, будем применять самые очевидные средства ввода данных. О более сложных инструментах организации графического интерфейса пользователя вы узнаете, прочитав главу “19-й час. Создание экранных форм”.

Ознакомившись с текстом листинга 6.3, вы изучили приемы создания соединения с базой данных, открытия таблицы и получения данных о ее структуре. Чтобы наполнить таблицу данными, программа должна содержать инструкции вставки записи и сохранения информации, введенной пользователем, в памяти, а затем в таблице. Листинг 6.4 представляет исправленный вариант предыдущей процедуры, демонстрирующий способы ввода данных в таблицу и сохранения их.

Листинг 6.4. Пример процедуры ввода данных в таблицу

```
1: Sub InputData ( )
2: Const DatabasePath = _
3:     "c:\Books\Teach Yourself Access 2002" + _
4:     "Programming\Chapter 6\CONTACTS.mdb"
5: Const ProviderStr="Provider=Microsoft.Jet.OLEDB.4.0;" + _
6:     " Data source = " + DatabasePath
7:
8: Dim Connection As New ADODB.Connection
9: Dim Catalog As New ADOX.Catalog
10: Dim RecordSet As New ADODB.RecordSet
11: Dim Field As Field
12:
13: Connection.Open ProviderStr
14: Set Catalog.ActiveConnection = Connection
15: RecordSet.Open "CONTACTS", Catalog.ActiveConnection, _
16:     adOpenDynamic, adLockOptimistic
17: RecordSet.Fields.Refresh
18:
19: Dim Temp As String
20: Do While (1)
21:     RecordSet.AddNew
22:     For Each Field In RecordSet.Fields
23:         If (Field.Name <> "ID") Then
24: Temp = InputBox("Введите значение поля " & _
25: Field.Name & "(Q=Выход):", Field.Name)
26:         If (Temp = "Q") Then Exit For
```

```

27:             Field.Value = Temp
28:         End If
29:     Next
30:
31:     If (Temp = "Q") Then
32:         Exit Do
33:     Else
34:         RecordSet.Update
35:     End If
36: Loop
37:
38: Set RecordSet = Nothing
39: Set Catalog = Nothing
40: Connection.Close
41: Set Connection = Nothing
42:End Sub

```



Если при работе программы листинга 6.4 щелкнуть на кнопке ОК, введенные данные будут сохранены в поле. По щелчку на кнопке Отмена (Cancel) в поле сохранится пустая строка. Для выхода из цикла необходимо нажать клавишу <Q>.

Многое из того, что представлено в тексте листинга 6.4, вы уже видели. Строки 1-12 выполняют операции создания соединения с базой данных и открытия таблицы. Главное в процедуре содержится во внешнем цикле Do While ... Loop (расположенном в строках 20-36) и внутреннем цикле For ... Next (строки 22-29).

Инструкция Do While строки 20 открывает бесконечный цикл. Литерал 1 равнозначен логическому значению True, и поэтому подобный цикл, не предусматривающий возможности естественного завершения, должен содержать команду принудительного прерывания. Она существует — это директива Exit Do, размещенная в строке 32. Внешний цикл, Do While ... Loop, контролирует операции добавления новой записи в таблицу и ее обновления после выполнения внутреннего цикла. Внутренний цикл обеспечивает возможность и прохождения по всем полям новой записи (кроме поля "ID", заполняемого Access автоматически), и сохранения в них значений, введенных пользователем. В качестве средства ввода используется уже хорошо известная функция InputBox. Temp — это символьная переменная для хранения результата ввода. С целью обеспечения возможности принудительного выхода из внутреннего и внешнего циклов используются условные выражения If (Temp = "Q") Then Exit, т.е. единственный символ "Q", введенный пользователем в диалоговом окне, трактуется программой как сигнал прерывания цикла.

В конце процедуры расположены команды закрытия набора данных, каталога и соединения (именно в таком, обратном, порядке), а также освобождения памяти, отведенной под объекты.

Код листинга 6.4 вы сможете использовать при работе с любым набором данных — планируете ли вы осуществлять ввод информации, форматировать значения полей или выполнять вычисления. Операции открытия/закрытия таблиц и циклического прохождения по записям данных относятся к разряду самых употребительных и универсальных.

# Динамический поиск информации в таблице

Циклические конструкции могут быть использованы также для поиска информации в таблице. В предыдущем разделе в качестве критерия прерывания цикла применялось условное выражение `If (Temp = "Q") Then`. Подобная конструкция применима и для решения задач анализа данных.

Посредством внесения незначительных изменений мы превратили процедуру `InputData`, приведенную выше в тексте листинга 6.4, в новую процедуру, `FindData`, предназначенную для поиска данных в таблице (листинг 6.5).

## Листинг 6.5. Пример процедуры поиска данных в таблице

```
1: Sub FindData ( )
2:   Dim Connection As New ADODB.Connection
3:   Dim Catalog As New ADOX.Catalog
4:   Dim RecordSet As New ADODB.RecordSet
5:   Dim Field As Field
6:   Const DatabasePath = _
7:     "c:\Books\Teach Yourself Access 2002" + _
8:     "Programming\Chapter 6\CONTACTS.mdb"
9:
10:  Connection.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
11:    "Data Source" & DatabasePath

1:   Set Catalog.ActiveConnection = Connection
2:   RecordSet.Open "CONTACTS", Catalog.ActiveConnection, _
   adOpenDynamic, adLockOptimistic
3:   RecordSet.Fields.Refresh
4:
5:   Dim Temp As String
6:   RecordSet.MoveFirst
7:
8:   Do While (RecordSet.EOF = False)
9:     Temp = InputBox ( "Введите искомые данные (Q=Выход) :",
   "Поиск данных")
10:    If (Temp = "Q") Then Exit Do
11:    For Each Field In RecordSet.Fields
12:      If (Field.Value Like Temp) Then
13:        MsgBox "Найдено: " & Field.Value & " в " & _
   RecordSet ( "ID" ).Value
14:        Exit For
15:      End If
16:    Next
17:
18:    RecordSet.MoveNext
19:  Loop
20:
21:  RecordSet.Close
22:  Set RecordSet = Nothing
23:  Set Catalog = Nothing
```

```
24: Connection.Close
25: Set Connection = Nothing
26:End Sub
```

Тексты листингов 6.4 и 6.5 почти идентичны. Наиболее существенные отличия вы заметите в содержимом цикла `Do while ... Loop` (см. строки 20–31). В качестве критерия завершения цикла используется условие достижения последней записи таблицы (служебное слово `EOF` обозначает конец файла, *End Of File*). Прежде чем приступить к выполнению цикла, помните, что поиск начнется с первой записи таблицы (см. команду `MoveFirst` в строке 18). Вновь, как и в прошлый раз, символ "Q" играет роль признака завершения цикла. Но если ранее цикл был бесконечен, то теперь выход из него гарантируется, помимо условного выражения `If (Temp = "Q") Then Exit Do`, еще и обязательным событием достижения последней записи таблицы. В строке 24 значение, введенное пользователем, сопоставляется с содержимым очередного поля текущей записи. Как мы говорили на прошлом занятии, оператор `Like` позволяет сравнивать символьные значения с заданным образцом (образец в нашем случае — это содержимое переменной `Temp`, заполненной пользователем). (Вызов функции `InputBox` и оператор сравнения `Temp` с "Q" следовало бы вынести за пределы цикла `Do while ... Loop`, расположенного в строках 20–31, и построить при этом еще один, самый "внешний" цикл, который предназначается исключительно для ввода очередной строки поиска и сопоставления ее с литералом "Q". — *Прим. перев.*)

Поиск такого рода, конечно, нельзя считать эффективным. Я назвал бы его "слепым". Представьте, что в вашем распоряжении имеется некий фрагмент числа, состоящий всего из нескольких цифр, причем вы можете не знать, к какого рода информации он относится — наименованию банковского счета, адресу клиента или номеру его телефона. Подобный алгоритм поиска следует считать оправданным только в таких условиях. Но если имеется дополнительная информация о том, к каким столбцам таблицы относятся искомые данные, тогда результат может быть достигнут гораздо быстрее. Основная цель примера, приведенного в листинге 6.5, — познакомить вас с конструкциями языка и способами обработки информации из баз данных.

## Резюме

На этом занятии мы вновь обратились к вопросам использования условных выражений и циклических конструкций, но уже применительно к задачам управления базами данных — а ведь это, собственно говоря, основная область приложения мощи `Access VBA`. Вы изучили понятия каталога, набора данных, соединения и поля. Каталог содержит ссылки на объекты таблиц, курсоров, учетных записей пользователей и групп пользователей. Набор данных — это обобщенный термин, применяемый для обозначения таблиц, запросов и курсоров. Соединение — объект, позволяющий связать код приложения с физическим файлом базы данных. Наконец, поле — это единый фрагмент данных.

В последующих главах мы более подробно рассмотрим различные объекты, обеспечивающие программный интерфейс к базе данных и ее элементам. А сейчас рекомендуем обратиться к разделу "Задания" и выполнить предлагаемые упражнения.

# Вопросы и ответы

**Вопрос.** Целесообразно ли изучить объекты ActiveX Data Objects (ADO) прежде, чем браться за решение любой задачи, связанной с базами данных Access?

**Ответ.** В ваше распоряжение предоставлены и другие альтернативы, но ADO — новейший и самый эффективный набор инструментальных средств программирования баз данных.

**Вопрос.** Какими именно инструментами, помимо ADO, можно воспользоваться?

**Ответ.** Во-первых, вы можете управлять данными в интерактивном режиме непосредственно из среды Access. В вашем арсенале имеются также механизмы создания и исполнения макросов, объекты DoCmd (за подробностями обращайтесь к системе оперативной справки Access), а также несколько устаревшие, но до сих пор поддерживаемые протоколы Direct Access Objects (DAO) и Remote Data Objects (RDO). Microsoft рекомендует, однако, пользоваться объектами ADO.

**Вопрос.** Должен ли я открывать и закрывать в программе объект класса Recordset всякий раз, когда необходимо выполнить действия с использованием информации из базы данных?

**Ответ.** Нет, только в определенных случаях. Например, единожды открыв объект класса RecordSet, вы можете затем использовать его многократно, но перед завершением программы объект следует обязательно закрыть.

**Вопрос.** Каким образом, по вашему мнению, следует подходить к вопросу дальнейшего изучения тем, освещаемых в ходе занятия?

**Ответ.** Мое мнение таково — информацию следует искать и изучать по мере необходимости. В принципе, если вы в состоянии самостоятельно сформулировать интересующий вопрос или описать наблевшую проблему, вам наверняка удастся с успехом воспользоваться средствами поиска, предоставляемыми оперативной справочной системой Access. Не стесняйтесь также обращаться за помощью к коллегам; кроме того, в вашем распоряжении новейшая литература, сеть Internet и ее многочисленные виртуальные конференции — на любой вопрос вы найдете квалифицированный и исчерпывающий ответ.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Какие объекты данных, помимо таблиц, могут адресоваться средствами RecordSet?
2. Как в среде Access создать новый программный модуль?
3. Каким образом можно протестировать код модуля?
4. Для каких целей применяется объект класса Catalog?
5. Каким образом установить указатель записей объекта класса RecordSet на первую запись?
6. Какую разновидность циклов предпочтительно использовать для обработки заранее известного количества объектов данных?

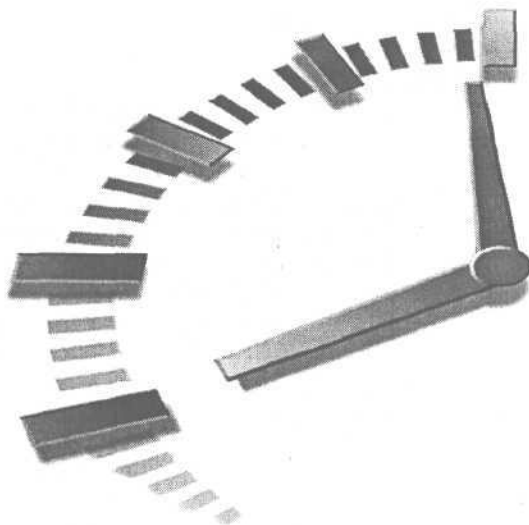


## Упражнения

1. Исправьте код процедуры CreateTable, приведенной в листинге 6.2, таким образом, чтобы добавить в определение таблицы CONTACTS столбцы для хранения адреса (ADDRESS), названия города (CITY), области (REGION) и почтового индекса (ZIP\_CODE).
2. Определите SQL-запрос, отображающий данные из всех столбцов таблицы CONTACTS, упорядоченные по возрастанию почтового индекса (POSTAL\_CODE).
3. Применив в качестве примера код листинга 6.5 и воспользовавшись выражением SQL, построенным при выполнении предыдущего упражнения, напишите процедуру поиска записи по заданному почтовому индексу (POSTAL\_CODE).

## 7-й час

# Расширенные типы данных



В главе "6-й час. Управление базами данных" мы рассматривали приемы использования циклических и условных конструкций в контексте управления объектами базы данных с помощью прикладных программ. На этом занятии мы сузим тему и обратимся к вопросам применения протокола ActiveX Data Objects (ADO).

На протяжении последних пяти-семи лет специалистами Microsoft разработано несколько протоколов, обеспечивающих возможность обмена информацией между Windows-приложениями, и возможность повторного использования откомпилированного кода. Самый ранний из протоколов — это стандарт *динамического обмена данными* (Dynamic Data Exchange — *DDE*). Позже была реализована гораздо более продуктивная и *гибкая* модель *связывания и внедрения объектов* (Object Linking and Embedding — *OLE*). В 1996 году Microsoft внесла в нее некоторые изменения и переименовала OLE в *ActiveX*.

Все эти протоколы, в частности, помогают программистам повторно использовать ранее разработанный код. Подобная идея в процессе ее реализации может быть воплощена в несколько форм. Самая простая и наиболее традиционная из них — модель *динамической библиотеки* (Dynamic Link Library — *DLL*), которая и сегодня находит самое широкое применение. (Просмотрите массив файлов с расширением *.DLL*, расположенных в каталогах *C:\Windows\System* или *C:\WinNT\System32* на диске своего компьютера. Я, например, использую поочередно обе операционные системы, Windows 2000, и в названных папках у меня содержится около 3120 таких файлов.)

С целью упрощения доступа к базам данных компанией Microsoft в рамках концепции ActiveX были разработаны специальные протоколы. Хронология их появления такова: Remote Data Objects (*RDO*), Direct Access Objects (*DAG*) и, наконец, ActiveX Data Objects (*ADO*). (Еще один протокол, который широко используется до сих пор, — это Open DataBase Connectivity (*ODBC*).) ActiveX — общий термин для обозначения средств повторного использования кода, а ODBC, RDO, DAO и ADO — это протоколы, специально предназначенные для работы с базами данных.

Возможно, названные аббревиатуры привели вас в легкое (?) замешательство. Если сейчас вы отправляетесь в свое первое путешествие в мир программирования, считайте, что вам крупно повезло. В первую очередь, вам следует уделить самое пристальное

внимание изучению объектов ADO, отбросив в сторону все остальное. Я же вступил на профессиональную стезю еще в то время, когда не существовало даже динамических библиотек DLL, и мне пришлось продираться сквозь джунгли LIB, DLL, DDE, OLE, ODBC, RDO и DAO, чтобы добраться, наконец, до ADO.



В связи с бурным развитием Internet компания Microsoft разрабатывает .NET-среду, которая включает Common Language Runtime (CLR) и другие технологии. В настоящее время Microsoft Office не поддерживает технологию .NET. На последующие версии Microsoft Office этот факт повлияет в том плане, что язык программирования Visual Basic.NET не поддерживает VBA, таким образом, беспокоиться об ADO+ вам не придется. В настоящее время языки программирования VBA для Microsoft Office и Visual Basic развиваются в разных направлениях.

Доподлинно не известно, зачем нужно было разрабатывать такое количество приблизительно равноценных средств программирования (хотя это можно рассматривать как гримасы некоей диалектической природы творчества), но один факт не вызывает сомнений — решение повторяющихся задач и процедуры управления базами данных значительно упростились.

Обращаясь к файлам оперативной справки Access и образцам кода, написанного сторонними разработчиками, будьте готовы к встрече с примерами использования *всех* названных выше протоколов. Но мы с вами, избрав один путь — новейший и самый мощный стандарт ADO, — будем неукоснительно его придерживаться. За исключением короткого примера сравнения DAO с ADO, во всех листингах этой и следующих глав мы будем ссылаться только на средства ADO.

В ходе прошлого занятия некоторые аспекты рассматриваемого кода вы принимали "на веру", таким образом сосредоточив внимание на других темах. Но теперь мы внесем более подробные разъяснения по поводу конструкций с использованием свойств и методов ADO, ранее введенных в обращение.

Основные темы занятия.

- Знакомство со средствами OLE Automation и сравнение архитектур DAO и ADO.
- ADODB и ADOX — инструменты управления данными.
- Примеры кода на основе ActiveX Data Objects.

## Знакомство со средствами OLE Automation

*OLE Automation* (или, в более современной и короткой трактовке, просто *Automation*) — это стандарт, который действует в рамках технологии Component Object Model (COM) и регламентирует способы совместного доступа к программным объектам, облеченным в форму откомпилированного кода. Программы, поддерживающие стандарт Automation и предоставляющие свои объекты в распоряжение других приложений, называют *OLE-серверами*. Приложения, которые обращаются к ресурсам OLE-серверов, носят название *клиентов*.

Клиент обладает возможностями активизации сервера и использования предоставляемых сервером услуг по своему усмотрению. Вспомните команду Dir, исполняемую в сеансе MS DOS. Вы указываете для Dir соответствующие параметры, и она выполняет свою задачу — выводит на экран список файлов. OLE-сервер способен предоставлять самые разнообразные возможности, а клиент может использовать все или только некоторые из них в различных сочетаниях. Access — это пример приложения-сервера. В других программах (клиентах) можно пользоваться инструментами Access для решения определенных задач.

Access предоставляет в распоряжение других программ — таких как Microsoft Excel, среды программирования Borland Delphi, Microsoft Visual Basic и т.п. — некоторые средства интерфейса.

#### Новый термин

*Интерфейс* — это обобщенный термин, употребляемый в программировании для обозначения кода и данных приложения, которые применяются другими приложениями или пользователем.

Программный интерфейс Access весьма обширный и многообразный. Со временем вы ознакомитесь с интерфейсными объектами *Application*, *DoCmd* и *CurrentDB*. Каждая из названных составных частей интерфейса Access обладает собственным интерфейсом подчиненного уровня. Хорошая новость — вам изначально не обязательно вникать во все подробности. Но есть и плохая — вся эта система объективно существует, и вам либо придется изучать ее, чтобы выяснить, удастся ли воспользоваться чем-то готовым для решения собственной частной задачи, либо делать все с нуля и самостоятельно.

## Добро пожаловать в мир объектов!

Термин *интерфейс* в своей современной интерпретации предполагает использование *объектов*. Объект — это экземпляр специального (составного) типа данных. До сих пор мы применяли преимущественно переменные простых типов, предназначенные для хранения одной порции данных. Впрочем, в листингах, рассмотренных на прошлом занятии, нам пришлось употребить, например, запись вида *ADODB*. Так вот, *ADODB* — это наименование специального типа данных, класса.

Мы назвали объект экземпляром *составного* типа данных, так как он содержит, как правило, *более* одной порции информации. В составе объекта могут находиться как собственно данные, так и методы их обработки. *Метод* — это термин для обозначения функции или процедуры, принадлежащей объекту (подробнее см. главу "8-й час. Декомпозиция задач").

#### Новый термин

*Объект* — это экземпляр составного типа данных, содержащий собственно данные и методы их обработки.

Давайте теперь немного порассуждаем. Переменные — это понятно. Мне столько-то лет, и подобную информацию я могу "воплотить" в некую переменную. То, что я умею делать, — это мои способности, мои возможности, мои методы. Далее объединим всю информацию воедино — как количественные сведения, которые меня характеризуют (возраст, рост, вес, семейное положение, номер телефона и т.п.), так и формулировки моих способностей. Получится некий "объект", описывающий меня как физическую и социальную сущность с той или иной степенью достоверности и полноты.

Более подробные сведения об объектах приведены в следующей главе.

## Некоторые полезные объекты

Используя объекты готовых мощных классов, вы значительно облегчите свою участь. В составе Access можно назвать три таких класса — *Application*, *DoCmd* и *CurrentDB*. Каждый из них содержит длинный перечень возможностей, и все они могут быть использованы вами при решении прикладных задач.

Объект класса *Application* позволяет ссылаться на приложение Access как таковое и содержит множество средств управления Access.

Класс *DoCmd* предоставляет возможности выбора и управления таблицами, запросами и курсорами, а также загрузки электронных таблиц, копирования баз данных и импорта текста.

Access — это набор инструментальных средств разработки баз данных. Класс `CurrentDB` дает возможность ссылаться на объект базы данных в файле базы, открытом в текущей сессии Access.

Все названные классы настолько сложны и обширны, что их описание, возможно, заслуживает отдельной книги. В ходе дальнейшего повествования мы будем рассматривать различные свойства и методы объектов `Application`, `DoCmd` и `CurrentDB`. За дополнительной информацией и примерами их использования обращайтесь к оперативной справочной системе Access.

## Сравнение ADO и DAO

Среди всего множества архитектур и технологий стандарты ADO и DAO заслуживают особого внимания. Оба они обеспечивают программиста схожим набором инструментов управления буквально всеми аспектами "поведения" баз данных. На этом занятии мы рассмотрим некоторые особенно полезные средства ADO, а глава "15-й час. ADODB — ваш верный помощник" предоставит вам счастливую возможность отточить свое мастерство на практике.

DAO — это предыдущий, сейчас уже несколько устаревший, стандарт средств управления базами данных. Хотя он до сих пор еще находит достаточно широкое применение, Microsoft предлагает использовать новейшие технологии ADO. Стандарт ADO предоставляет более мощные инструменты, а его применение приводит к существенному сокращению и упрощению кода. Листинги 7.1 и 7.2 демонстрируют два примера кода для решения одной и той же задачи; первый построен на основе объектов DAO, а во втором применены средства ADO. Объемы кода, как вы видите, заметно различаются — в пользу ADO.



Для работы подпрограммы листинга 7.1 необходимо подключить Microsoft DAO 3.6 Object Library, а для работы процедуры листинга 7.2 — Microsoft ActiveX Data Objects 2.6 Library (ADO). Чтобы добавить ссылки на эти библиотеки, откройте редактор Visual Basic, выполните команду **Tools**⇒**References** и в появившемся диалоговом окне установите флажки возле данных библиотек.

### Листинг 7.1. Пример использования объектов DAO

```
1: Sub DAOAddRow( )
2:
3: Const TITLE = "Sams Teach Yourself Access 2002 " & _
4:     " Programming in 24 Hours"
5:     Dim DB As Database
6:     Dim RS As DAO.Recordset
7:     Set DB = CurrentDb( )
8:     Set RS = DB.OpenRecordset( "LIBRARY" )
9:
10: RS.AddNew
11:
12: RS( "TITLE" ).Value = TITLE
13: RS( "AUTHOR" ).Value = "Paul Kimmel"
14: RS( "ISBN" ).Value = "0-672-32098-3"
15: RS( "PAGECOUNT" ).Value = 400
16: RS( "PUBLISHER" ).Value = "Sams"
17: RS( "PUBLICATIONDATE" ).Value = Date
```

```

18:
19  RS.Update
20:
21  RS.Close
22  Set RS = Nothing
23  Set DB = Nothing
24  End Sub

```

## Листинг 7.2. Пример использования объектов ADO

```

1: Sub ADOAddRow ( )
2: Const Title = "Sams Teach Yourself Access 2002 " & _
3:   " Programming in 24 Hours"
4:
5:   Dim RS As New ADODB.Recordset
6:   RS.Open "LIBRARY", CurrentProject.Connection, _
7:     adOpenKeyset, adLockOptimistic
8:   RS.AddNew
9:
10:  RS( "TITLE" ).Value = Title
11:  RS( "AUTHOR" ).Value = "Paul Kimmel"
12:  RS( "ISBN" ).Value = "0672316617"
13:  RS( "PAGECOUNT" ).Value = 400
14:  RS( "PUBLISHER" ).Value = "Sams"
15:  RS( "PUBLICATIONDATE" ).Value = Date
16:
17:  RS.Update
18:  RS.Close
19:  Set RS = Nothing
20:End Sub

```

### Анализ

Оба примера кода решают задачу добавления новой записи в таблицу, содержащую сведения об опубликованных книгах. Сравнивая тексты листингов 7.1 и 7.2, обратите внимание, что второй из них содержит на три строки меньше. Чем меньше строк в программе, тем, как правило, она проще. Кроме того, в коде листинга 7.2 создается только один составной объект данных (RS класса Recordset) против двух (RS класса Recordset и DB — Database) в примере листинга 7.1, что также способствует снижению сложности кода.

При создании новых проектов, наряду с объектами ADO, до сих пор активно применяются и объекты DAO, поэтому вам, возможно, придется обращаться и к тем, и к другим. Но в нашей книге внимание будет сосредоточено только на объектах ADO.

## Использование объектов ADODB

Объект класса ADODB — это объект COM. Иными словами, его интерфейс может использоваться приложением Access, любой программой, способной исполнять код на языке VBA (такой как Excel или Word) либо даже инструментальными средами программирования от сторонних производителей (не Microsoft) (скажем, Borland Delphi). Листинг 7.3 содержит фрагмент кода проекта Delphi с использованием функций ADODB.

### Листинг 7.3. Пример использования функций ADODB в проекте Delphi

```

1  procedure TForm1.Button1Click( Sender : TObject)
2  const
3      sTitle = 'Sams Teach Yourself Access 2002 ' & _
4              ' Programming in 24 Hours'
5:
6      sProvider = 'Provider=Microsoft.Jet.OLEDB.4.0; ';
7      sDataSource = 'Data Source= 661707.mdb';
8  var
9      RecordSet : Variant;
10     Connection : Variant;
11  begin
12:
13:     RecordSet :=CreateOLEObject( ' ADODB.RecordSet ' );
14:     Connection := CreateOLEObject( 'ADODB.Connection' );
15:     Connection.Open( sProvider + sDataSource );
16:     RecordSet.Open( 'LIBRARY',Connection, 1, 3 ).;
17:     RecordSet.AddNew;
18:     RecordSet.Fields.Item('TITLE').Value := sTitle;
19:     RecordSet.Fields.Item('AUTHOR').Value := 'PaulKimmel' ;
20:     RecordSet.Fields.Item('ISBN').Value := '0672316617';
21:     RecordSet.Fields.Item( ' PAGECOUNT' ).Value:=400;
22:     RecordSet.Fields.Item( 'pUBLISHER' ).Value := 'Sams';
23:     RecordSet.Fields.Item( ' PUBLICATIONDATE' ).Value:= Date;
24:
25:     RecordSet.Update;
26:     RecordSet.Close;
27:
28:     RecordSet := varNull;
29:     Connection := varNull;
30:
31:     End;

```

#### Анализ

И Даже если вы до сих пор никогда не видели программного кода на языке Object Pascal, просматривая текст листинга 7.3, вы легко выявите в нем многие из конструкций, которые применялись нами в листинге 7.2. Для справки: процедура листинга 7.3 отображает в окне сообщения содержимое одного поля второго столбца таблицы LIBRARY. Однако доступ к полям в Object Pascal достаточно громоздкий:

```
Recordset.Fields.Item(имя поля)
```

Процедура листинга 7.3 отображает в окне сообщения содержимое одного поля второго столбца таблицы LIBRARY.



Delphi — программная среда для Object Pascal, а Access — для VBA.


Хотя среда Borland Delphi предоставляет в распоряжение программиста собственные мощные средства для работы с базами данных, тот факт, что с помощью всего нескольких строк кода можно легко воспользоваться функциями сервера ADODB, значителен сам по себе. Далее рассмотрим некоторые аспекты интерфейса ADODB более конкретно.

# Connection

*Соединение* (Connection) — это блок информации, указывающий Access, как общаться с базой данных. В мире существует и конкурирует между собой достаточное количество компаний, производящих разнообразные системы управления базами данных. Корпорация Microsoft, помимо Access, поддерживает продукт MS-SQL Server. Другие крупные участники рынка — это Sybase, Oracle и Informix. Ваше приложение Access способно обращаться к базам данных любого из названных поставщиков, а это становится возможным именно благодаря объекту Connection. Если вам достаточно сослаться только на наборы данных внутри текущей базы, в которой хранится программный код, используйте объект `CurrentProject.Connection`, как было продемонстрировано в строке 3 листинга 7.2.

Чтобы открыть новое соединение, вначале необходимо объявить объект класса Connection. Синтаксис подобного объявления таков:

```
Dim ИмяСоединения As New ADODB.Connection
```



Сразу после набора фразы ADODB, завершающейся символом точки, в окне редактора программного модуля всплывает контекстное меню, в котором перечисляются возможные продолжения конструкции. Если этого не происходит, значит, в диалоговом окне References, вызываемом командой **Tools** → **References**, не установлен флажок Microsoft ActiveX Data Objects 2.1 Library списка Available References.

Как и во всех других случаях, выражение объявления переменной начинается со служебного слова `Dim`. Замените `ИмяСоединения` требуемым именем и сопроводите его служебными словами `As` и `New`. Слову `Connection` следует предпослать обязательную аббревиатуру `ADODB`, поскольку существуют и другие типы данных `Connection` — один из них, например, содержится в классе OLE-сервера `DAO`.

Объявив объект соединения, вы можете (и, видимо, должны) затем открыть его.

## Открытие

Команда открытия соединения предполагает наличие информации о содержимом так называемой *строки соединения*, имени пользователя, его пароле и значениях дополнительных параметров, которые передаются методу `Open` защищенной паролем базы данных. Имя пользователя и пароль являются необязательными параметрами.

Строка соединения включает в себя сведения о производителе системы управления базой данных. Здесь содержатся наименование, номер версии и другие особенности конкретной системы управления, представленные в специальном формате, описание которого вы сможете найти в фирменной документации от поставщика программного продукта. Строка соединения с секцией `Provider`, представляющей новейшую версию программного сервера Jet из состава Access 2002, выглядит так:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=ПутьКБазеДанных"
```

Вместо шаблона `ПутьКБазеДанных` в секции `Data Source`, вам следует указать полный путь к конкретному файлу базы данных. Соединение с базой данных Access может быть открыто даже в том случае, если строка соединения не содержит данных об имени пользователя и его пароле — в этом случае будут использованы значения, принятые по умолчанию.

Помимо строки соединения, функции открытия соединения с базой данных Access могут быть переданы дополнительные значения перечислимого типа — `adConnectUnspecified` и `adAsyncConnect`. (Подробности вы найдете в оперативной справочной системе Access.)



## Заккрытие

Завершив работу с объектом соединения, не забывайте закрыть его. Листинг 7.4 содержит короткий пример применения команд объявления, открытия и закрытия объекта `Connection`.

Листинг 7.4. Пример использования объекта `ADODB.Connection`

```
1: Sub OpenConnection( )
2:   Const Provider = "Provider=Microsoft.Jet.OLEDB.4.0;"
3:   Const DataSource =
4:     Data Source=E:\Books\Teach Yourself Microsoft Access 2002 " & _
5:     "Programming in 24 Hours\Chapter 7\661707.mdb"
6:
7:   Dim Connection As New ADODB.Connection
8:   Connection.Open Provider + DataSource
9:   Connection.Close
10: End Sub
```



Знак & и символ подчеркивания в строках 4 и 5 листинга 7.4 используются для конкатенации строки, расположенной в нескольких физических строках кода.

### Анализ

В строках 2-5 задается информации о соединении и источнике данных. В строке 7 содержится объявление объекта. В строках 8 и 9 выполняется открытие и закрытие соединения. Чтобы процедура выполняла что-то действительно результативное, вставьте необходимый код между строками 8 и 9.

## Recordset

После объявления объекта `Connection` и успешного открытия соединения будет естественным обратиться к объекту `Recordset`, который задает некий набор данных и позволяет приступить к решению конкретной задачи. Объект класса `Recordset` способен содержать данные из таблицы, запроса, курсора или хранимой процедуры. Открытый объект `Recordset` может одновременно ссылаться на одну запись (строку) набора данных. Объекты класса `Recordset` — одни из самых популярных и часто используемых средств управления данными.



Поверьте, целью разнесения по различным классам функций соединения с базой данных и управления ею является отнюдь не стремление ввести вас, программиста, в заблуждение и смятение. Основной довод, которому следовали разработчики Microsoft, — это разумное распределение ответственности. Задача управления базой данных в общей постановке весьма трудна. Наличие отдельных классов (таких как `Connection` или `Recordset`), решающих частные задачи, значительно упрощает технологию программирования. Но если собрать все данные и методы их обработки в единое целое, такой объект наверняка выйдет за пределы восприятия и потребует чрезмерных системных ресурсов. Возможность разбиения целого на части — одно из фундаментальных преимуществ объектно-ориентированного стиля программирования.

Наборы данных (объекты `Recordset`) следует открывать, а затем закрывать. Они предоставляют возможности добавления, циклического просмотра и редактирования записей. Вложенные объекты класса `Fields` еще более упрощают задачи обработки данных (см. листинги 7.1, 7.2 и 7.3).

## Открытие

Прежде чем воспользоваться средствами объекта `Recordset` для доступа к данным, его надлежит открыть. Чтобы открыть объект, следует объявить переменную `ИмяНабораДанных` типа `ADODB.Recordset`, а затем записать конструкцию вида

`ИмяНабораДанных.Опен ИмяОбъектаДанных, ОбъектСоединения, _  
ТипКурсора, ТипБлокировки, ДополнительныеОпции`

`ИмяНабораДанных` — это ранее объявленная переменная типа `ADODB.Recordset` (см. строку 5 листинга 7.2). В качестве `ИмяОбъектаДанных` следует задать имя таблицы (см. строку 6 листинга 7.2) либо символьной переменной, содержащей код SQL. Вместо `ОбъектСоединения` можно подставить имя объекта `CurrentProject.Connection`, если достаточно использовать ту же базу данных, которая содержит текст текущего модуля. Можно также использовать имя другого правильно определенного объекта типа `Connection` (обратитесь к разделу "Connection", приведенному выше).

Параметры `ТипКурсора` и `ТипБлокировки` принимают значения перечислимых типов `CursorTypeEnum` и `LockTypeEnum`, описание которых вы сможете найти в файлах оперативной справки Access. В строке 3 листинга 7.2 в качестве параметров `ТипКурсора` и `ТипБлокировки` указаны значения `adOpenKeyset` и `adLockOptimistic` соответственно.

## Закрывтие

После завершения работы с набором данных (объектом класса `ADODB.Recordset`) его необходимо обязательно закрыть. Возьмите за правило выполнять подобную операцию всегда. Строка 18 листинга 7.2 служит примером инструкции закрытия набора данных.

## Редактирование записи

Команды добавления записи и редактирования ее содержимого весьма просты. Строка 8 листинга 7.2 демонстрирует действия по добавлению записи. При необходимости сохранения внесенных изменений, связанных с добавлением или редактированием записи, используйте команду `Update`.

Предположим, что объект набора данных носит название `RS`. Тогда инструкция добавления записи будет выглядеть как `RS.AddNew`, редактирования — `RS.Edit`, а сохранения внесенных изменений — `RS.Update`. Примеры добавления, редактирования и сохранения записей приведены в строках 8–17 листинга 7.2.

## Условия достижения начала и конца набора данных

Объект класса `ADODB.Recordset` содержит два метода определения факта достижения начальной и конечной записей набора данных — соответственно, `BOF` (от *Beginning Of File* — начало файла) и `EOF` (от *End Of File* — конец файла).

### Новый термин

*Метод* — это термин, который указывает на процедуру или функцию, содержащуюся в составе класса и объектов на его основе. В наборе методов объекта заключены его функциональные возможности.

Предположим, что объект набора данных носит название `RS`. Тогда инструкция проверки условия достижения начала набора данных будет выглядеть как `RS.BOF`, а его конца — как `RS.EOF`.

Обе функции, BOF и EOF, возвращают результат типа Boolean: BOF = True при достижении начальной записи набора данных, а EOF = True — последней записи.

Поскольку значения типа Boolean — **подходящие** кандидаты на роль критериев завершения циклов и операндов условных выражений, их удобно использовать в циклических конструкциях, предполагающих прямое или обратное прохождение по записям набора данных. Строка 21 листинга 6.5, приведенного в главе 6, содержит пример использования функции EOF.

## Перемещение по записям

Класс `ADODB.Recordset` содержит ряд функций, позволяющих перемещаться по записям открытого набора данных: `Move`, `MoveFirst`, `MoveLast`, `MoveNext` и `MovePrevious`.

Удачное, красноречивое имя, данное процедуре или функции, обеспечивает ей дополнительные преимущества. Это в полной мере относится к функциям группы `Move`. `MoveFirst` смещает указатель текущей записи к началу набора данных, а `MoveLast` — к концу; `MoveNext` позволяет переместиться к следующей записи, а `MovePrevious` — к предыдущей. Наконец, `Move` обеспечивает возможность перемещения на произвольное число записей по направлению к концу или началу набора относительно текущей.

Листинг 6.5 главы 6 содержит пример использования методов `MoveFirst`, `MoveNext` и `EOF` для решения частной задачи прохождения по всем записям набора данных. Важно помнить, что все эти функции — "собственность" класса `ADODB.Recordset`. Поэтому при необходимости их использования обращение всегда следует предварять 1) наименованием объекта типа `ADODB.Recordset`, который должен быть ранее объявлен и открыт; 2) символом точки:

`ИмяНабораДанных.ИмяМетода`

Указанная синтаксическая формула справедлива в отношении любых объектов, к свойствам и методам которых необходимо обратиться. Слева от точки располагается имя переменной-объекта, а справа — название свойства (элемента данных), функции или процедуры, принадлежащих объекту данного класса.

## Использование полей

Набор данных (объект класса `ADODB.Recordset`) способен одновременно ссылаться только на одну (текущую) запись. Запись может состоять из одного или нескольких полей. В упоминавшейся ранее таблице `LIBRARY`, например, содержались **столбцы** `TITLE`, `AUTHOR`, `ISBN`, `PAGECOUNT`, `PUBLISHER` и `PUBLICATIONDATE`. **Объект** на пересечении записи (строки) и столбца называют *полем*.

Мы уже неявно пользовались объектами полей — просмотрите строки 10-15 листинга 7.2. Так, например, запись `RS("TITLE")` означает ссылку на поле столбца `"TITLE"` и текущей строки.

Более подробные сведения о свойствах и методах класса `Field` вы сможете найти в файлах оперативной справочной системы Microsoft Visual Basic, если введете строку поиска *Field Object (ADO)*. Наиболее важный атрибут объекта `Field` — свойство `Value`.

### Новый термин

*Атрибут* — это обобщенный термин, используемый для обозначения любого свойства или метода объекта класса.

Доступ к свойству `Value` осуществляется по общей схеме: `ИмяОбъекта.ИмяАтрибута`. Свойство `Value` — это переменная типа `Variant`. Другими словами, `Value` способно хранить и возвращать любые данные, и компилятор успешно определит их тип. В строках 10-15 листинга 7.2, например, переменным `Value` присваиваются значения символьных строк, чисел и дат.

К сожалению, мы не сможем рассмотреть здесь все атрибуты класса `Field` — их достаточно много. Примите к сведению, что некоторые из них будут упоминаться в ходе дальнейшего повествования. Кроме того; в вашем постоянном распоряжении оперативная справочная система с массой прекрасных примеров и подробнейших разъяснений — берите и пользуйтесь!

## Использование объектов ADOX

Библиотека *ADO Data Definition and Security Library* (ADOX) содержит определения классов, позволяющих управлять таблицами, курсорами и индексами, а также администрировать данные о пользователях и группах пользователей.

Чтобы получить доступ к библиотеке ADOX, откройте окно редактора Microsoft Visual Basic и выберите в строке меню команду `Tools⇒References`. Проклистайте список `Available References` диалогового окна `References` и установите флажок для элемента `Microsoft ADO Ext. 2.7 for DDL and Security`, а затем щелкните на кнопке `OK`. Теперь вы сможете создавать и использовать объекты ADOX.

## Catalog

*Каталог* (Catalog) — это контейнерный класс для хранения данных о таблицах, курсорах, хранимых процедурах, пользователях и группах. В тексте листинга 6.2, рассмотренного на прошлом занятии, мы использовали объекты `Catalog`, `Table` и `Key` для динамического создания таблицы `CONTACTS` и сохранения ее в одноименной базе данных.

Между объектами `Catalog` и `Connection` существует связь типа *один-к-одному*. Если необходимо создать с помощью программного кода таблицу и добавить ее в базу данных, определить новые или исправить существующие ключи и индексы, ввести или отредактировать атрибуты пользователей или групп — все эти операции вы сможете сделать с помощью объекта `Catalog`.

Приведенный ниже листинг 7.5 демонстрирует пример кода, предназначенного для отображения списка всех таблиц базы данных, и содержит строки, частично заимствованные из листингов, которые были рассмотрены на предыдущем занятии.

Листинг 7.5. Пример использования объекта `ADOX.Catalog`

```
1: Sub ListTables( )
2:   Dim Catalog As New ADOX.Catalog
3:   Set Catalog.ActiveConnection = CurrentProject.Connection
4:   Dim I As Integer
5:   For I = 0 To Catalog.Tables.Count - 1
6:     Debug.Print Catalog.Tables( I ).Name
7:   Next I
8:   Set Catalog = Nothing
9: End Sub
```

### Анализ

Строка 2 содержит объявление объекта `Catalog` типа `ADOX.Catalog`, а в строке 3 его атрибуту `ActiveConnection` присваивается значение `Current Project.Connection`, необходимое для привязки объекта к текущей базе данных. В строке 4 находится инструкция объявления переменной `I` типа `Integer`, которая далее будет использоваться в качестве счетчика итераций цикла. Строки 5-7 определяют цикл вида `For ... Next`

по числу таблиц, содержащихся в базе данных. На каждом шаге цикла выполняется процедура вывода на экран наименования очередной таблицы. Команда строки 8 осуществляет очистку памяти.

Основная нагрузка, которую несет подпрограмма ListTables, заключена в строке 6. Ее скрытые функции можно перечислить следующим образом:

```
Dim Name As String
Dim Table As Table
Table = Catalog.Tables ( I )
Name = Table.Name
Debug.Print Name
```

Операнд Catalog.Tables ( I ) возвращает I-й объект типа Table коллекции Tables. Атрибут Name хранит наименование таблицы, адресуемой объектом Table. Инstrukция Debug.Print Name отображает наименование таблицы в окне Immediate.

**Новый термин** Коллекция — это индексированный набор объектов определенного типа. Предопределенным коллекциям обычно даются названия в виде существительного — имени типа — во множественном числе.

Процедура листинга 7.5 отличается универсальностью и может использоваться с любыми базами данных Access. Примените ее по отношению к какой-либо базе данных, и вы получите результат, схожий с тем, который показан на рис. 7.1.

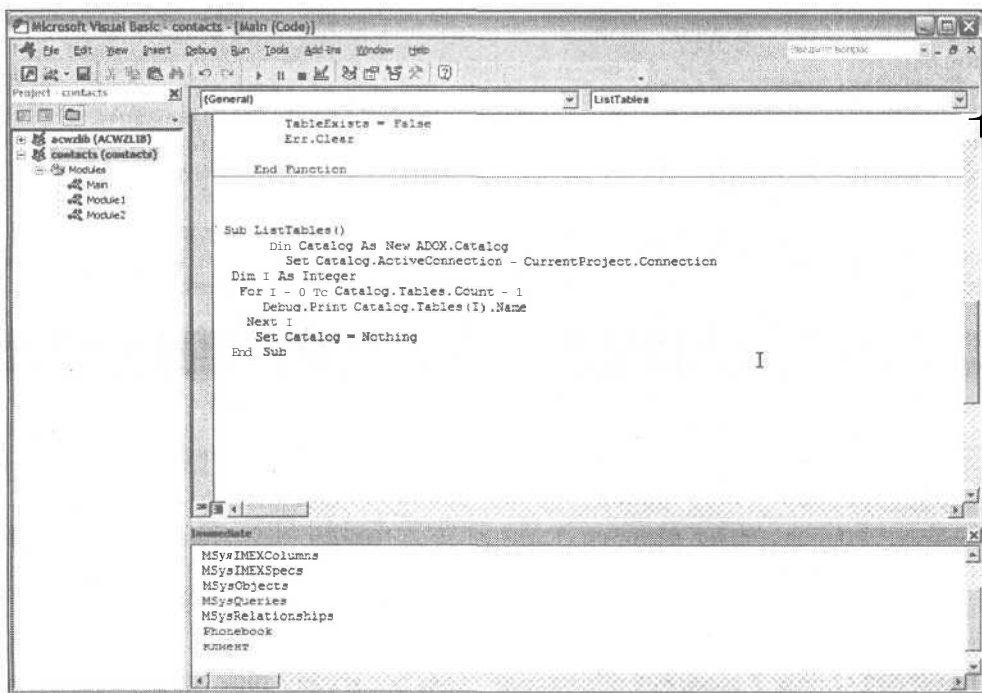


Рис. 7.1. Так может выглядеть перечень таблиц, хранящихся в базе данных

## Атрибут `ActiveConnection`

Свойство `ActiveConnection` объекта типа `Catalog` связывает последний с объектом соединения `Connection`. Если следует обеспечить соединение только с той базой данных, которой принадлежит текущий программный модуль, достаточно присвоить атрибуту `Catalog.ActiveConnection` значение `CurrentProject.Connection`. Впрочем, вы можете также построить объект `Connection` явно и привязать его к произвольной базе данных, а затем передать ссылку объекту `Catalog` тем же самым образом.

## Коллекция `Tables`

Как мы уже говорили, коллекция — это индексированный набор объектов одного типа. Принято соглашение, в соответствии с которым коллекции целесообразно называть существительными во множественном числе, производными от наименований типов. Так, например, коллекцию объектов `Connection` следовало бы назвать `Connections`.

В главе “13-й час. Коллекции данных” вы найдете общее описание коллекций и способов их использования — они просты, единообразны и последовательны, независимо от типов сохраняемых данных.

Коллекция `Tables` содержит упорядоченный набор объектов типа `Table`. Запись `Catalog.Tables ( I )` означает ссылку на *I*-й объект набора, где *I* — целочисленное значение в пределах от 0 до `Catalog.Tables.Count - 1`.

Каждый из объектов `Table` в свою очередь содержит коллекции объектов `Columns`, `Indexes`, `Keys` и `Properties`. `Columns` — набор объектов, описывающих столбцы таблицы, `Indexes` содержит сведения об индексах. `Keys` и `Properties` — хранилища данных соответствующих типов. Каждый из указанных типов найдет свое отражение на страницах нашей книги. Кроме того, в вашем распоряжении находятся файлы оперативной справочной системы, которые дадут вам исчерпывающие ответы на все вопросы. Например, чтобы открыть страницу справки, описывающую объект столбца таблицы, введите ключевое словосочетание поиска *Column Object (ADOX)*.

## Коллекции `Groups`, `Users` и `Views`

`Groups`, `Users` и `views` — это коллекции, принадлежащие объекту типа `Catalog`. Чтобы создать новую группу пользователей, достаточно добавить очередной объект класса `Group` в коллекцию `Groups`, воспользовавшись методом `Append`. Аналогичные действия возможны и в отношении объектов `User` и `View`.

Группам пользователей и всем их членам соответствуют личные учетные записи, хранящиеся в базе данных и регламентирующие права доступа к тем или иным объектам данных. Определяя учетную запись пользователя, вы должны указать, какие объекты данных пользователь, зарегистрированный под определенным именем, может просматривать и изменять. (Обратитесь к теме *Users Collection (ADOX)* оперативной справки.) Чтобы найти пример кода, касающийся задания прав пользователей, выполните следующие действия.

1. Запустите программу `Access`.
2. Пользуясь областью задач, откройте существующую базу данных или создайте новую.
3. Выберите элемент Модули (Modules) списка Объекты (Objects).
4. Щелкните на кнопке Создать (New) панели инструментов, чтобы запустить редактор `Visual Basic`.
5. Нажмите клавишу <F1> для вызова справки.
6. Перейдите на вкладку Указатель (Index) окна приложения `Microsoft Visual Basic Help`.

7. В поле Введите ключевые слова (Type Keywords) укажите словосочетание *Users Collection*.
8. Щелкните на кнопке Найти (Search).
9. В списке Выберите раздел (Choose a topic), помеченном справкой (найдено: 84) (84 found), щелкните на элементе *Users Collection (ADOX)* (рис. 7.2).
10. Обратитесь к странице темы *Users Collection* и щелкните на гиперссылке *Example* (Пример), выделенной голубым цветом.

Справочные файлы **MSDN** (службы Microsoft Developer Network) содержат гигабайты (миллиарды байт) информации, включая статьи и примеры, касающиеся самых разных инструментальных сред, технологий, задач и приемов программирования. Такой объем данных невозможно уместить в книге, подобной нашей, — не хватит даже сотни книг. Читайте, раскапывайте, ищите крупницы информации — это наилучший способ освоения новых технологий программирования.

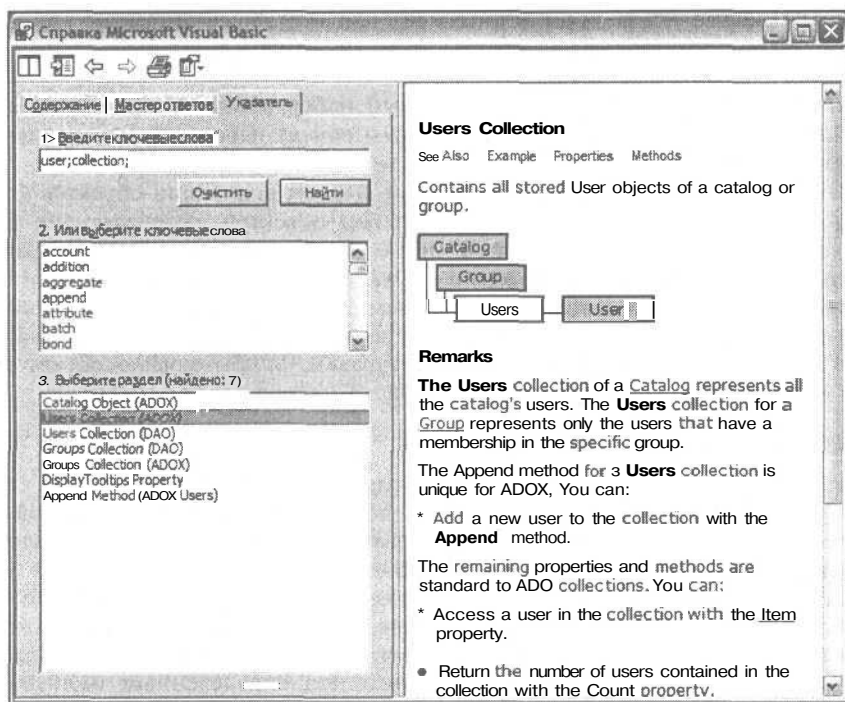


Рис. 7.2. Активно пользуйтесь средствами оперативной справочной системы, и она поможет вам отыскать все необходимые сведения и приобрести драгоценный опыт программирования

## Резюме

В ходе этого занятия мы более подробно рассмотрели содержимое объектов **ADODB** и **ADOX**, с которыми вы бегло ознакомились ранее, в главе 6. Современные технологии программирования предполагают активное повторное использование кода, ранее созданного сторонними разработчиками. Теперь вы можете, еще не написав ни

одной собственной строки кода, пользоваться готовыми решениями, причем от вас не требуется глубоких знаний о тонкостях их практической реализации.

Существуют тысячи программных классов. Вы сделали первый шаг к их постижению — теперь вы в общих чертах представляете, как обращаться с объектами Catalog, Table, Index, Key, Connection, Recordset и их коллекциями. На следующих занятиях вы ознакомитесь с другими приемами и примерами использования как ADO, так и других средств программирования для Access. Более подробные сведения о применении объектов ADODB, например, приведены в главе "15-й час. ADODB — ваш верный помощник".

## Вопросы и ответы

**Вопрос.** В каких случаях целесообразно использовать объекты ADO, а в каких — объекты иных классов (скажем, DAO)?

**Ответ.** Средства, предоставляемые протоколом ADO, вам будет вполне достаточно. Microsoft, не останавливаясь на достигнутом, продолжает развивать технологию ADO, и поэтому в большинстве случаев ADO окажется наилучшим выбором, требующим наименьших усилий с вашей стороны.

**Вопрос.** Я хотел бы подробнее изучить все те объекты, о которых вы рассказали. Каким образом можно это сделать?

**Ответ.** Изучайте их по мере необходимости, в процессе работы над конкретной практической задачей. Обращайтесь к дополнительной литературе, общайтесь с коллегами и обменивайтесь с ними накопленным опытом.

**Вопрос.** В каких именно местах кода следует создавать объекты Catalog, Connection и Recordset?

**Ответ.** Это зависит от обстоятельств. В наших демонстрационных примерах все эти операции выполнялись в пределах одной процедуры. Но в реальном приложении одни объекты могут создаваться единожды и затем использоваться многократно, а другие должны строиться и удаляться динамически. Критерий истины — только практика. Далее на страницах нашей книги вы найдете немало дополнительных примеров использования объектов.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

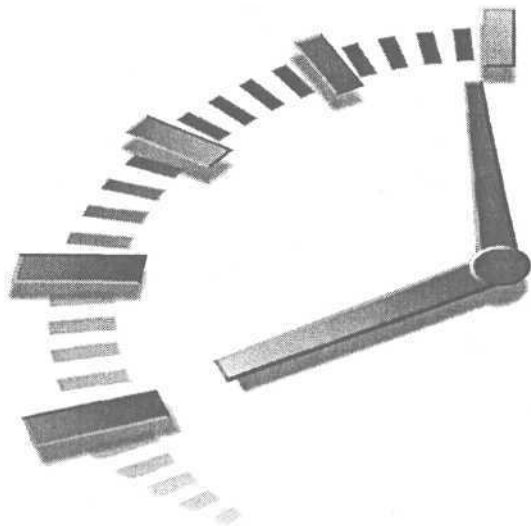
## Тесты

1. На какое количество записей способен одновременно ссылаться объект типа RecordSet?
2. Какому классу принадлежит коллекция Tables — Catalog или Connection?
3. С какими объектами данных позволяет работать объект Catalog?
4. Назовите два общих вида услуг (функций), предоставляемых объектами класса Catalog?
5. Какие объекты и методы применяются для циклического прохождения по набору записей таблицы?



## Упражнения

1. Создайте запрос для рабочей базы данных и сохраните его. Откройте запрос с помощью кода, использующего объект типа Recordset.
2. Напишите процедуру, предусматривающую возможность циклического прохождения по всем ключам, хранимым в объекте Catalog.
3. Создайте подпрограмму для прохождения по набору записей в обратном порядке, начиная с последней.



# **Часть III**

## **Использование ресурсов Access**

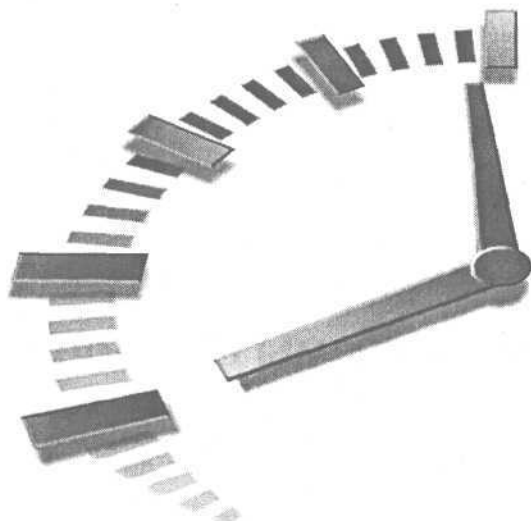
### **Темы занятий**

8-й час. Декомпозиция задач

9-й час. Работа с макросами

10-й час. Как использовать готовые решения





## 8-й час

# Декомпозиция задач

Вам наверняка известен очевидный, но надежный способ решения сложных задач, связанный с разбиением (или, как говорят профессионалы, — *декомпозицией*) их на мелкие подзадачи. Программа — это набор строк кода, каждая из которых, будучи связанной с остальными, предусматривает решение простой частной задачи. На предыдущих занятиях мы рассмотрели немало примеров кода, структурные единицы которых называли выражениями. Выражение может охватывать одну или несколько физических строк кода — точно так же, как и предложения естественного языка, которые могут быть простыми ("Послушай-ка, Вася!") или более сложными (как то, которое вы сейчас читаете). А вот примеры выражений на языке программирования: `A = 5` или `Dim A As Integer`. Выражения служат "строительным" материалом для создания более крупных конструкций — функций и подпрограмм (процедур).

На этом занятии вы научитесь выявлять подмножества строк кода, которые целесообразно группировать в одном месте и затем ссылаться на них по имени из других строк программы. Именованный блок кода, состоящий из нескольких строк, называют (в зависимости от способа его оформления) *функцией* или *процедурой*.

Определение функции или процедуры должно следовать определенным строгим синтаксическим правилам. Одно из них — это наличие служебных слов `Function` или `Sub`, которые обозначают, соответственно, начало блока функции или процедуры.

Во время работы с Access понадобится создавать функции и процедуры для решения таких задач, с которыми другими средствами справиться не удастся.

Не исключено, что при написании программ вам придется прибегнуть к помощи процедур или функций для решения повторяющихся задач, которые не могут или не должны выполняться в режиме диалога с пользователем. Хороший пример — задача автоматического обновления или резервного копирования базы данных, требующая автоматического выполнения ночью, в часы наименьшей загрузки компьютерного оборудования и каналов связи.

В таких ситуациях, когда необходимо автоматизировать решение рутинных задач и уменьшить количество ошибок, появляющихся из-за повторяемости или сложности операций, придет на помощь программа, выполняемая в среде Access. Программа для Access — это, в действительности, набор функций и процедур, написанных на языке VBA.

Основные темы занятия.

- Механизмы создания функций и процедур.
- Заповеди программирования.
- Практические примеры.
- Функции импорта текстовых данных.

## Механизмы создания процедур

Язык программирования VBA поддерживает две разновидности именованных блоков кода — процедуры (подпрограммы) и функции. Вначале обратимся к процедурам. Их проще создавать и удобнее использовать в качестве отправного пункта дальнейшего повествования.

В основе технологии процедурного программирования лежат очевидные и естественные предпосылки, связанные с ограниченными возможностями человеческой памяти и стремлением представить сложную проблему в виде набора простых именованных сущностей, каждая из которых ссылается на некую группу данных или подразумевает выполнение определенного законченного множества операций. Как я, например, запоминаю телефонные номера? Я мысленно разбиваю номер на три группы цифр: первая соответствует коду региона, вторую я называю префиксом, а третью — суффиксом. Вместо беспорядочного нагромождения цифр, в моей памяти сохраняется четко структурированная и поэтому легче воспринимаемая информация.

*Процедура* — это блок строк кода, адресуемый по имени. Общий принцип, на котором строится технология процедурного программирования, — агрегация, соединение частей в единое целое. Поскольку человеку свойственно воспринимать мир как множество связанных сущностей, нет ничего удивительного в том, что подобный подход нашел свое применение и в области программирования. Например, когда я говорю о Хогане, моем псе, то имею в виду животное, которое мы вызволили из мест "заключения". Это Лабрадор песочного цвета, смешанной породы, самец, пяти лет от роду. Он дружелюбный и преданный. Вот из таких отдельных характеристик состоит светлый образ нашего любимца.

Новый термин

Употребляя термин *агрегация*, подразумеваем процесс соединения строк программы в единый блок, адресуемый по имени.

В языках программирования подобная роль отводится процедурам. Процедуры содержат блоки кода для решения повторяющихся частных подзадач. Не стоит говорить о том, что намного легче ссылаться на блок строк по имени, нежели повторять все их заново при необходимости решения той же подзадачи.

В каждой процедуре легко заметить несколько основных частей. Первая — это наименование, которому предшествует служебное слово Sub. Далее могут быть перечислены значения *параметров*, которые передаются в процедуру. Такие значения называют *аргументами*. За первой строкой следует блок *тела* процедуры. Любая процедура завершается парой служебных слов End Sub. Ниже подробно рассмотрены все основные части процедуры, написанной на языке VBA.

# Первая и последняя строки процедуры



Первая и последняя строки процедуры настолько важны, что даже при отсутствии других строк процедура будет считаться правильной конструкцией языка VBA. Формат первой и последней строк любой процедуры неизменен. Его легко представить в виде следующей синтаксической формулы:

```
Sub ИмяПроцедуры ( [[Квалификатор] ИмяАргумента As ТипДанных, ... ] )  
End Sub
```

Строка заголовка процедуры начинается со служебного слова Sub, за которым следуют наименование процедуры и необязательный список аргументов в круглых скобках. Символ многоточия означает, что допустимо произвольное число аргументов. Процедура завершается служебными словами End Sub.



В языках программирования, как и в естественных языках, действуют строгие правила 'синтаксиса'. Синтаксис — это набор правил составления и использования языковых конструкций. Компилятор выполняет роль жесткого цензора, тщательно анализирующего допустимость употребления каждого символа в контексте его окружения и взаимосвязи строк и целых блоков кода программы. На данном этапе достаточно запомнить общие синтаксические правила конструирования процедур.

## Правила именования процедур

Удачные соглашения, касающиеся именования процедур, способны облегчить восприятие вашего кода другими и решить задачу документирования текста программы. Зачастую уместные и красноречивые названия процедур и их небольшой объем позволяют исключить необходимость вставки в код дополнительных строк комментариев. Можно предложить такую простую и удобную схему именования: название должно включать существительное, описывающее предмет, к которому имеет отношение процедура, и глагол, характеризующий выполняемую операцию. Внимательно изучите следующий пример:

```
Sub DocumentPrint( DocumentNumber As Long, DocumentDate As Date )  
    ***  
End Sub
```

Название процедуры — DocumentPrint. Оно говорит само за себя — процедура выполняет печать документа (например, платежного). Наименования аргументов, DocumentNumber и DocumentDate, также достаточно понятны. Первый из них — это номер документа (скажем, в базе данных), а второй — дата его сохранения. Если подпрограмма решает только задачу печати документа, специальные комментарии, вероятно, не понадобятся. (За дополнительными сведениями о правилах конструирования процедур и функций обращайтесь к разделу "Заповеди программирования", приведенному ниже в этой главе.)

## Аргументы — как их звать-величать

Переменные, перечисленные в круглых скобках после наименования процедуры, называют аргументами, или параметрами. Термин *аргумент* впервые был предложен математиками. Классики математики — эти первые программисты — давали имена всем сущностям, с которыми сталкивались. Впрочем, нам с вами понятен, вероятно,

далеко не весь математический жаргон. Но слово *аргумент* прижилось в среде программистов и стало общеупотребительными. Те же рекомендации, которые были даны выше при обсуждении правил именования процедур, вполне применимы и к их аргументам. Называйте переменные-аргументы внятно и недвусмысленно.

Избегайте использования нестандартных сокращений, неологизмов и бессмысленных последовательностей букв и цифр. Компьютеру, конечно, "до лампочки", как именно назван тот или иной объект программы, но вам самим и будущим читателям вашей программы это далеко не безразлично. Если когда-либо вашему преемнику (или даже вам самому — через пару месяцев) придется разбираться в назначении процедуры с душераздирающим названием *BrrMrrDrr\_1201*, вам не удастся избежать упреков от посторонних или приступов самобичевания.



Придумывая имена для объектов программы, используйте целые слова, начинающиеся с прописной буквы, последовательности слов или общепринятые аббревиатуры — и код будет оставаться понятным и доступным всем и всегда.

## Определение типов аргументов

Последняя важная вещь, которую следует обсудить, говоря об общих принципах создания подпрограмм, — вопрос о том, какие данные следует передавать процедурам при их вызове. Данные, получаемые процедурой, перечисляются в ее заголовке и заключаются в круглые скобки. Их называют аргументами, или параметрами. Я предпочитаю пользоваться термином *аргумент*, хотя оба названия вполне допустимы.

Аргумент определяется следующим образом: вначале указывается его наименование, за которым следуют служебное слово *As* и обозначение типа данных. Количество аргументов процедуры не ограничено. Если необходимо передать процедуре более одного аргумента, их следует разделять запятыми. (Количество запятых, таким образом, на единицу меньше числа аргументов.)

Ниже приведен пример короткой процедуры, которая вычисляет общую стоимость изделия с учетом налога. Результат сохраняется в глобальной переменной.

Global TotalSale As Double

Sub CalculateSalesTax( SaleAmount As Double, SalesTax As Double )

    TotalSale = SaleAmount \* (1 + SalesTax)

End Sub

Эта процедура использует два аргумента. Синтаксис ее правильный. Она также точна логически. Синтаксическая и логическая правильность — два обязательных условия, которых следует придерживаться. Если конструкция неверна с точки зрения синтаксиса, она просто не будет работать. Логическая правильность заключается в гарантии получения верного предсказуемого результата. Процедура *CalculateSalesTax* удовлетворяет обоим названным стандартам.

Существуют, однако, еще два требования. Первое из них — семантическая правильность — заключается в верной передаче вашего исходного замысла (алгоритма) средствами языка программирования. Второе, не менее важное, — это условие надежности и устойчивости кода по отношению к ошибкам времени выполнения.



Вопросам обеспечения устойчивости и надежности программного кода посвящена глава "18-й час. Обработка ошибок во время выполнения программы". Напомним еще раз — старайтесь избегать использования глобальных переменных.

Параметры `SaleAmount` и `SaleTax` в приведенном выше примере объявлены таким образом, что любые изменения их значений внутри процедуры `CalculateSalesTax` останутся в силе после ее завершения. Вряд ли вы предусматривали бы именно такое развитие событий, поэтому семантическая правильность подобного кода остается под вопросом.

Язык Access VBA позволяет предпослать любому аргументу процедуры определенное служебное **слово-квалификатор** (обратитесь к общему описанию синтаксиса процедур, приведенному выше), чтобы оговорить дополнительные требования и обеспечить верную передачу смысла. Хотя применять **квалификаторы** вовсе не обязательно, в определенных обстоятельствах они дают возможность уточнить семантику кода и уменьшить количество потенциальных ошибок (и время, которое придется провести в "обществе" отладчика). Подробные сведения о **квалификаторах** аргументов и приемах их использования приведены в нескольких следующих разделах этой главы. Вы увидите более надежные и верные варианты кода предыдущего примера.

## Квалификатор ByVal

**Квалификатор ByVal** (*передача параметра по значению*) применяют в тех случаях, когда необходимо исключить возможность изменения значения аргумента процедуры после ее завершения. Другими словами, после того как процедура будет выполнена, переменная-аргумент сохранит первоначальное значение, которое было передано процедуре в момент ее вызова.

Аргумент, помеченный **квалификатором ByVal**, *может* изменять свое значение в процессе выполнения процедуры, но эти изменения носят временный характер. На самом деле параметр, снабженный словом `ByVal`, получает и хранит *копию* передаваемых данных. Только ее вы и сможете изменить, а оригинал останется нетронутым. Внесем в код рассмотренной ранее процедуры исправления, позволяющие предохранить содержимое аргументов от случайного искажения.

```
Global TotalSale As Double
Sub CalculateSalesTax( ByVal SaleAmount As Double, ByVal SalesTax As Double )
    TotalSale = SaleAmount * (1 + SalesTax)
End Sub
```

Применяйте **Квалификатор ByVal** в тех случаях, когда необходимо гарантировать сохранность содержимого конкретного аргумента процедуры после ее завершения. Ничто не запрещает пользоваться аргументом по своему усмотрению внутри процедуры, но по окончании ее выполнения все изменения, внесенные в содержимое аргумента, будут утрачены. Чтобы проверить, как вы поняли смысл сказанного, рассмотрим следующий код.

### Листинг 8.1. Пример использования квалификатора ByVal в аргументах процедур

```
1: Sub TestByVal( ByVal A As Integer )
2:   A = 5
3:   MsgBox A
4: End Sub
5:
6: Sub CallFoo( )
7:   Dim A As Integer
8:   A = 10
9:   Call TestByVal( A )
10:  MsgBox A
11: End Sub
```



Если вы согласны с тем, что при выполнении строки 3 в окне сообщения будет отображено число 5, а результат вызова процедуры MsgBox в строке 10 — это число 10, тогда можно говорить о взаимопонимании.

## Квалификатор ByVal

Квалификатор ByVal (*передача параметра по ссылке*) уведомляет компилятор VBA (и читателя программы) о том, что значение аргумента разрешается изменять внутри процедуры с сохранением изменений после ее завершения. Приведем еще один исправленный вариант процедуры CalculateSalesTax — сейчас нам удастся избежать использования глобальной переменной TotalSale:

```
Sub CalculateSalesTax( ByVal SaleAmount As Double, ByVal SalesTax As
Double, ByVal TotalSale As Double )
    TotalSale = Sale * (1 + SalesTax)
End Sub
```

Квалификатор ByVal указывает, что в процедуру передается адрес исходной переменной, а не ее значение-копия. Поэтому результат изменения параметра не зависит от того, где оно было совершено — внутри процедуры или за ее пределами. Применяйте Квалификатор ByVal в тех случаях, когда процедуре должна быть предоставлена возможность изменения данных, объявленных извне и переданных в качестве аргумента.

## Квалификатор Optional

Квалификатор Optional (*необязательный аргумент*) позволяет создавать более гибкие процедуры, учитывающие конкретные обстоятельства вызова. Для аргумента процедуры, помеченного словом Optional, задавать значение необязательно. Этот квалификатор чрезвычайно удобен в тех ситуациях, когда в соответствии с условиями задачи аргумент в большинстве случаев принимает одно заранее известное значение и только изредка — какие-то другие.

Вернемся к нашему примеру, касающемуся расчета стоимости, и предположим, что нам дополнительно необходимо учитывать зависимость ставки налогообложения от места изготовления изделия. Если в большинстве случаев процедура должна вычислять полную стоимость изделий, произведенных в штате Мичиган, снабдите аргумент SalesTax квалификатором Optional и значением по умолчанию, соответствующим ставке налогов штата Мичиган. Теперь пользователю не придется заботиться о явном задании ставки, поскольку она заранее известна, а понадобится указывать ее только в тех редких случаях, когда речь идет об изделиях, произведенных за пределами штата. Исправленная версия процедуры представлена в тексте листинга 8.2.



Аргументы, снабженные квалификатором Optional, должны занимать место в конце списка параметров процедуры, иначе компилятор VBA выдаст сообщение об ошибке.

### Листинг 8.2. Пример использования квалификатора Optional

```
1: Sub CalculateTotalSale( ByVal TotalSale As Double, _
    ByVal SaleAmount As Double, _
    Optional TaxPersent As Double = .06 )
2:     TotalSale = SaleAmount * (1 + TaxPersent)
3: End Sub
4:
5: Sub Test( )
6:     Dim SaleAmount As Double
```

```

7: Dim TotalSale As Double
8: SaleAmount = 100
9: Call CalculateTotalSale( TotalSale, SaleAmount)
10: Dim WashingtonSalesTax As Double
11: WashingtonSalesTax = .08
12: Call CalculateTotalSale( TotalSale, SaleAmount, _
    WashingtonSalesTax )
13: End Sub

```

### Анализ

Квалификатор `Optional`, используемый в строке заголовка процедуры `CalculateTotalSale`, позволяет упростить код программы, поскольку наиболее часто используемое значение аргумента `SalesTax` — `.06` — известно заранее. Обратите внимание, что процедура `CalculateTotalSale` вызывается дважды — один раз с двумя параметрами (в строке 9), а другой — с тремя (в строке 12). Приемлемы оба варианта вызова.



Напомним — все аргументы с квалификатором `Optional` должны располагаться в конце списка параметров процедуры.

## Как строить функции

*Функции* (еще одна именованная конструкция) очень схожи с процедурами и различаются только в одном аспекте — они могут возвращать значения определенного типа. (Если вы не читали начало главы, рекомендуем сделать это сейчас.)

Ниже приведена информация о том, как задавать возвращаемый аргумент и использовать его в операторах присваивания. Общий синтаксис определения функции почти совпадает с правилами оформления процедуры и отличается только началом (теперь вместо служебного слова `Sub` следует задавать слово `Function`), суффиксом, указывающим тип возвращаемого значения, и конструкцией завершающей строки:

```

Function ИмяФункции ( [[Квалификатор] ИмяАргумента As ТипДанных, ... ] ) As ТипДанных
End Function

```

Напомним: при определении реальной функции вместо словосочетаний `ИмяФункции` и `ИмяАргумента` надлежит ввести четкие и запоминающиеся наименования (идентификаторы), а параметры `ТипДанных` заменить соответствующими служебными словами VBA, обозначающими допустимые типы данных. Символ многоточия означает, что разрешено задавать произвольное число аргументов.

Функции следует использовать в тех случаях, когда блок кода должен возвращать результат определенного типа. Процедуру `CalculateTotalSale` листинга 8.2 можно легко преобразовать в функцию — и решение задачи станет несколько более изящным. Взгляните на текст листинга 8.3.

### Листинг 8.3. Пример определения и использования функции

```

1: Function CalculateTotalSale ( ByVal SaleAmount As Double, _
    Optional TaxPercent As Double = .06 ) As Double
2:     CalculateTotalSale = SaleAmount * (1 + TaxPercent )
3: End Function

```

```

4:
5: Sub Test ( )
6:   Dim SaleAmount As Double
7:   Dim TotalSale As Double
8:   SaleAmount = 100
9:   TotalSale = CalculateTotalSale(SaleAmount)
10:  Dim WashingtonSalesTax As Double
11:  WashingtonSalesTax = .08
12:  TotalSale = CalculateTotalSale(SaleAmount, WashingtonSalesTax )
13: End Sub

```

#### Анализ

Изменения, внесенные нами в текст программы, служат цели пояснения различий между процедурами и функциями. Код функции с прежним именем — `CalculateSalesTax` — начинается со служебного слова `Function` и завершается им. Теперь вы можете не использовать параметр `TotalSale` — его роль исполняет возвращаемый аргумент, имя которого совпадает с названием функции (см. строку 2).



Думается, вы уже ощутили различия между процедурами и функциями. В ходе дальнейшего изложения будем использовать термины *процедура*, *подпрограмма* и *функция* как синонимы, если это не станет противоречить контексту обсуждаемой темы.

Итак, функции весьма схожи с процедурами. Различие состоит в синтаксических правилах оформления (вместо служебного слова `Sub` используется `Function`), конструкции завершения строки заголовка, а также наличии оператора присваивания, в левой части которого употребляется идентификатор возвращаемого аргумента, совпадающий с наименованием функции. В каких же случаях использовать процедуры, а в каких — функции? Ответ на этот вопрос весьма прост. Обращайтесь к процедурам, если вам не нужны возвращаемые значения. Впрочем, всегда можно безболезненно изменить свое решение.

## Заповеди программирования

Существует шесть известных заповедей программирования, которые необходимо принимать во внимание при использовании процедур и функций. Истинность этих эмпирических правил подтверждается десятилетиями развития теории и практики программирования. Неукоснительно им следуя, вы сможете с чистой совестью и чувством исполненного долга ежедневно покидать свой офис ровно в пять часов вечера. Ниже об этих правилах — более подробно.

### Заповедь 1: „Заклучай повторяющиеся строки кода твоего в функции“

Наилучшая программа — маленькая программа (естественно, при прочих равных условиях). Чтобы код работал лучше и выглядел более прозрачным, его должно быть по возможности меньше. Если вы видите, что определенные строки программы повторяются несколько раз, создайте на их основе функцию. Фрагмент кода, который встречается в тексте программы только единожды, проще протестировать, изменять и поддерживать в надлежащем состоянии.

С повторяющимся кодом работать сложнее. При необходимости внесения исправлений возрастает вероятность ошибок. Если, скажем, группа из пяти строк кода по-

вторяется в тексте программы двадцать раз, любое изменение в этой группе придется воспроизвести именно двадцать раз. Если хотя бы в одном месте вы допустите ошибку, программа не достигнет поставленной цели. Подобного не случится при использовании единой функции.

## **Заповедь 2: „Выражайся лаконично”**

Старайтесь писать короткие функции. Короткие — не более пяти-десяти строк. В большинстве случаев функции, объем которых превышает указанный предел, могут и должны считаться слишком большими. Практическая реализация данного правила требует от вас, вероятно, немалых усилий. Помните, код должен работать правильно — это самое главное. Впрочем, достаточно важны и другие его характеристики — например, стоимость сопровождения и возможность повторного применения. Так вот, чем более короткими и простыми будут функции, тем легче с ними совладать, тем ниже стоимость их сопровождения и выше вероятность повторного использования.

## **Заповедь 3: „Ограничивай число аргументов твоих”**

В самых лучших образцах функций используется всего лишь по несколько аргументов. Больше и не нужно, ведь функции просты. "Скромнее надо быть!" - этот житейский постулат вполне справедлив даже в программировании.

В набросках черновика программы старайтесь добиться, чтобы она просто решала поставленную задачу. Но во время второго, третьего и последующих сеансов правки текста уделите внимание лаконичности и надежности кода, а также убедитесь, что все параметры функций действительно нужны и используются по назначению.

## **Заповедь 4: „Проясняй мысль твою посредством квалификаторов”**

Активно применяйте квалификаторы аргументов процедур и функций — `ByVal`, `ByRef` и `Optional` (мы говорили о них выше, в разделе "Определение типов аргументов"), чтобы ясно выразить свою мысль и довести ее до сведения компилятора. Возможно, человек, читающий вашу программу, не сочтет целесообразным или возможным углубляться во все подобные тонкости, но компилятор должен это делать. Даже если код уже подвергался отладке и работает, не поленитесь явно указать нужные квалификаторы, чтобы до конца прояснить свой замысел.

## **Заповедь 5: „Обусловливай решения твои”**

В практике программирования часто встречаются ситуации, когда гарантией точных результатов служит выполнение некоторых условий относительно значений переменных, наличия файлов и т.п. Автор программы выдвигает определенные требования, а ее пользователь обязан их удовлетворить. Каждое условие необходимо снабжать комментариями и усиливать конструкциями, проверяющими его выполнение. Если требование не выполнено, автор программы должен предусмотреть адекватные действия.



Зачастую автор и пользователь программы — одно и то же лицо. Программист выступает в одной из ролей, и чтобы диалог персонажей оказался продуктивным, сценарий пьесы (код программы) должен быть безупречным.

Листинг 8.4 содержит пример кода, предполагающего выполнение определенного требования. Автор программы специально отметил его строкой комментария, а пользователь должен ему подчиниться.

Листинг 8.4. Пример кода с дополнительным условием

```
1: Function OpenFile( ByVal FileName As String ) As Double
2:   ' Условие: файл FileName должен существовать
3:   Debug.Assert Len( Dir( FileName ) ) > 0
4:   Open FileName for Input As #OpenFile
5: End Function
```

#### Анализ

Как мы говорили, условие должно снабжаться комментарием, и в примере таков имеется (см. строку 2). Впрочем, одной констатации мало — программа должна позаботиться о проверке выполнения требования (в нашем примере данная роль возложена на команду `Debug.Assert Len( Dir( FileName ) ) > 0` в строке 3). Если функция `Dir` определит, что файла не существует, т.е. возвратит строку нулевой длины, выполнение программы завершится в строке 3.

Подобный стиль работы поможет упростить код и разумно распределить ответственность между автором и пользователем программы.

## Заповедь 6: „Не ленись комментировать код твой”

Написаны миллионы строк кода, начисто лишенного комментариев. Часто можно слышать отговорки, наподобие такой: “Это трудно описать, и поэтому будет трудно читать”. Конечно, такой тезис просто несерьезен. Да, лучшие образцы программного текста не нуждаются в дополнительных комментариях — настолько умело они написаны. Это идеальный вариант, к которому следует стремиться. Но в большинстве случаев несколько дополнительных предложений, рассказывающих на нормальном человеческом языке о том, для чего предназначена функция, как достигается результат, какие условия должны быть выдержаны и т.п., сослужат хорошую службу. Позже вам или вашим преемникам уже не придется ломать голову, распутывая хитросплетения строк кода и вникая в суть вашего смелого творческого замысла.

## А теперь все хорошо!

Если группа строк кода повторяется несколько раз или в тексте программы легко выделить завершенные блоки — уместно обратиться к процедурам или функциям. Чтобы научиться писать хорошие функции, надежные и допускающие возможность повторного использования, вам понадобится определенная (надо сказать, немалая) практика. Вначале решите задачу, а затем уделите некоторое время “шлифовке” программы, если вы в состоянии это сделать. В настоящем разделе приводится несколько примеров, иллюстрирующих способы создания функций для решения ряда часто встречающихся задач.

Конкретные задачи были выбраны, можно сказать, почти случайно, они носят достаточно общий характер и удобны с методической точки зрения. Предлагаем вам примеры удачных решений, которые допускают повторное применение в ваших будущих реальных проектах.

## Создание таблицы

Первый уровень абстракции в общей модели управления информацией — база данных, а второй, весьма важный, — таблица. База данных состоит из одной или более таблиц. В некоторых ситуациях возникает задача динамического создания и удаления таблицы средствами прикладной программы в течение одного сеанса ее работы.

В Access 2002, наряду с VBA, поддерживается и язык структурированных запросов к базе данных (Structured Query Language — *SQL*) (подробнее см. главу "16-й час. Применение языка SQL"). На данном этапе вам достаточно знать, что для создания таблицы он вполне пригоден. (Конечно, при необходимости построения таблицы во все не обязательно обращаться именно к средствам языка SQL.) Подобный код удобно оформить в виде процедуры и передавать ей в качестве аргумента наименование таблицы, которую требуется создать. Просмотрите текст листинга 8.5.

Листинг 8.5. Пример процедуры построения таблицы в базе данных

```
1: Sub CreateTable(ByVal TableName As String)
2:   Dim SQLAsString
3:   ' Используется команда 'CREATE TABLE' SQL
4:   SQL = "CREATE TABLE " & TableName & "(TotalSales Double); "
5:   Call CurrentDb.Execute( SQL )
6: End Sub
```

### Анализ

Ну как — неплохо, не правда ли? Хотя до универсальности здесь далеко-далеко (любая создаваемая таблица будет содержать одно и то же единственное поле, TotalSales типа Double), некоторая гибкость все-таки обеспечивается: динамически строится таблица с именем, которое задается в качестве аргумента процедуры. Если вы будете придерживаться подобного подхода, то сократите объем кода программы, самое меньшее, на треть. Пример дает пищу для размышлений и простор для дальнейшего творчества.



Впрочем, простота простоте рознь. Решения должны быть конструктивными и целесообразными. Листинг 8.5 демонстрирует пример того уровня сложности, который допустим в функциях и процедурах. Воспринимайте отдельные выражения кода как атомы, а функции — как молекулы. Все мыслимые сущности физического мира состоят из иерархии атомов и молекул. Ваша программа сможет достичь степени сложности физического мира, если вы, ее создатель, будете последовательно и продуманно строить ее из маленьких "кирпичиков" — отдельных выражений, процедур и функций.

## Импорт текста из файла

В составе класса Application имеется вложенный класс DoCmd. Объект типа Application позволяет ссылаться на приложение Access как таковое. Он содержит инструменты управления Access. DoCmd (его следует воспринимать как интерфейс

к средствам Access) — весьма серьезный объект, насчитывающий множество свойств и методов, среди которых имеются функции импорта содержимого текстовых файлов.

#### Новый термин

*Свойство* — это переменная или объект определенного класса, находящийся в составе класса более высокого уровня иерархии.

Синтаксис

Процедура импорта содержимого текстовых файлов носит название `TransferText`. Ниже приведен синтаксис ее вызова.

```
Call DoCmd.TransferText( [ТипПреобразования] [, ИмяСпецификации], ИмяТаблицы, ИмяФайла [, ФлагЗаголовков] [, ИмяТаблицыHTML] [, КодоваяСтраница] )
```

Поскольку метод `TransferText` принадлежит классу `DoCmd`, этот факт необходимо явно оговаривать, используя при вызове процедуры префикс `DoCmd` с последующим символом точки. В качестве необязательного аргумента `ТипПреобразования` допустимо указывать одно из предопределенных значений (по умолчанию система устанавливает признак `acImportDelim` — импорт текста с разделителями). Аргумент `ИмяСпецификации` также необязателен. Параметры `ТипПреобразования` и `ИмяСпецификации` находят применение в случае импорта текста более сложной структуры. Аргументы `ИмяТаблицы` (наименование таблицы-получателя информации) и `ИмяФайла` (название файла-источника) обязательны. Остальные параметры — `ФлагЗаголовков`, `ИмяТаблицыHTML` и `КодоваяСтраница` — разрешается не задавать. (Исчерпывающие сведения о назначении, способах и примерах использования всех аргументов процедуры `DoCmd.TransferText` вы найдете в файлах оперативной справочной системы Microsoft Visual Basic.)



Поведение многих процедур и функций стандартных классов Access VBA существенно зависит от значений передаваемых им параметров и их количества. Зачастую имеет смысл заключать вызов стандартного метода VBA в тело пользовательской функции, чтобы упростить и прояснить интерфейс.

Рассматривая частную задачу импорта текста с разделителями, удобно определить новую функцию с меньшим числом аргументов и дополнительными средствами контроля. Рассмотрим пример простого текстового файла, содержащего разделенные запятыми имена, фамилии и почтовые адреса группы людей (листинг 8.6).

#### Листинг 8.6. Пример текстового файла с разделителями

```
Имя, Фамилия, Адрес Email  
Paul, Kimmel, Paul_Kimmel@hotmail.com  
Robert, Dearman, Robert.Dearman@jnli.com  
Robert, Golieb, RobertGolieb@hotmail.com
```

#### Анализ

Набор строк, приведенный в листинге 8.6, — это, по существу, уже готовая таблица. Если ее содержимое необходимо импортировать в таблицу базы данных, процедура `DoCmd.TransferText` придется весьма кстати. Листинг 8.7 отображает код пользовательской процедуры, упрощающей задачу импорта простого текста с разделителями.

### Листинг 8.7. Пример процедуры импорта текста с разделителями

```
1: Sub ImportContactFile( ByVal FileName As String, _  
    ByVal TableName As String )  
2:   Const HASFIELDNAMES = True  
3:   If (FileExists( FileName )) Then  
4:     Call DoCmd.TransferText (acImportDelim, , TableName, _  
        FileName, HASFIELDNAMES)  
5:   Else  
6:     MsgBox "Файл " & FileName & " не найден!"  
7:   End If  
8: End Sub  
9:  
10: Function FileExists (ByVal FileName As String) As Boolean  
11:   FileExists = Len(Dir(FileName)) > 0  
12: End Function
```

#### Анализ

Определение дополнительной процедуры, “охватывающей” вызов метода DoCmd.TransferText, позволяет расширить возможности кода (скажем, в нашем примере добавлены строки, анализирующие наличие файла с заданным именем) и скрыть те аспекты интерфейса TransferText, которые в данном случае просто не нужны.

## Поиск записи

В своей личной профессиональной практике я часто сталкивался с образцами кода, буквально перенасыщенного предложениями на языке SQL. Подобные программы, к сожалению, очень трудно воспринимать и тем более исправлять. Как и в предыдущем примере, мы постараемся скрыть особенности интерфейса SQL внутри тела разработанной нами функции.

Листинг 8.8 содержит код функции, в которой предложение на языке SQL используется для поиска записи в таблице CONTACTS (считайте, что с помощью процедуры листинга 8.7 мы уже ее заполнили) по заданному значению фамилии корреспондента.

### Листинг 8.8. Пример функции поиска записи в таблице

```
1: Function FindContact( ByVal LastName As String ) As String  
2:   Dim RecordSet As New AdoDB.RecordSet  
3:   Dim SQL As String  
4:   SQL = "SELECT [FirstName] + ' ' + [LastName] As" &  
5:   "[Contact Name]" From CONTACTS WHERE [LastName] = ' " &  
6:   LastName & "'"  
7:   Call RecordSet.Open (SQL, CurrentProject.Connection, _  
    dbOpenDynamic)  
8:   If ((RecordSet.BOF And RecordSet.EOF) = False) Then  
9:     FindContact = RecordSet( "Contact Name" )  
10:   Else  
11:     FindContact = "Запись не найдена!"  
12:   End If  
13:   RecordSet.Close  
14:   Set RecordSet = Nothing  
15: End Function
```





Одно из важнейших свойств процедур и функций состоит в том, что они представляют собой фрагменты кода, которые можно отлаживать и исправлять независимо от других частей программы. Функция, представленная в листинге 8.8, основана на использовании средств класса DAO. Если вам необходимо перейти к ADO, оптимизировать встроенные предложения SQL или отказаться от них вовсе, достаточно отредактировать только код тела функции — все остальные части программы остаются нетронутыми.

#### Анализ

Чтобы воспользоваться кодом листинга 8.8, понадобится единственная строка программы, содержащая вызов функции `FindContact` с указанием фамилии корреспондента, сведения о котором необходимо найти.

## Использование реестра Windows

Реестр Windows содержит иерархию ключей и значений, которые хранятся в отдельном файле. Реестр, в определенном смысле, — это база данных для хранения информации между выполнением приложений. С помощью инструментов баз данных можно размещать информацию в любом его месте.



Чтобы просмотреть реестр своего компьютера, щелкните на кнопке Пуск (Start), выберите команду Выполнить (Run), наберите `regedit` и нажмите <Enter>.

Непосредственное изменение реестра — далеко не самый мудрый поступок. Если вы случайно ошибетесь, система станет работать некорректно даже после перезагрузки. В любом случае перед внесением изменений в реестр сначала создайте его копию, выбрав в окне Редактор Реестра (Registry Editor) команду Файл⇒Экспорт (File⇒Export).

Вероятно, для приложений баз данных наиболее важна хранящаяся в реестре информация, которая помогает соединиться с базой данных. Вместо того, чтобы хранить информацию о соединении, как это было сделано в листингах предыдущих глав, в настоящих приложениях мы будем хранить информацию в реестре. Таким образом, при перемещении базы данных не придется изменять код — достаточно будет лишь изменить настройки реестра.



Хранение информации в реестре (в особенности информации о соединении) для Access более допустимо, чем для других приложений. В Access при работе с данными активной базы данных, чтобы получить доступ к объекту Connection, можно использовать объект CurrentProject, который является подключенным по умолчанию.

Для сохранения и изменения информации в реестре используют `SaveSetting` и `GetSetting`, которые позволяют непосредственно считывать и записывать информацию. Вообще, обращаться к этой информации можно независимо от инициализации приложения и того, открыто ли соединение. В листинге 8.9 приведен пример динамического чтения информации о соединении из реестра. На рис. 8.1 представлена копия экрана редактора реестра, в который добавлен ключ `ConnectionString`. Если из реестра получена пустая строка, происходит инициализация со значением, заданным по умолчанию.

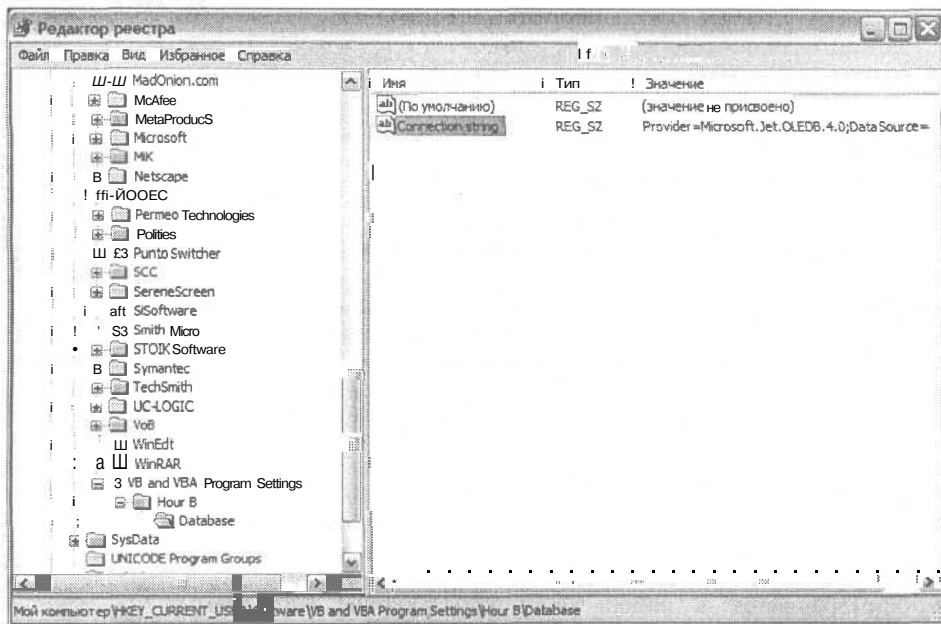


Рис. 8.1. В Windows 2000 с помощью функции *SaveSetting* в реестр добавлена строка соединения

**Листинг 8.9. Чтение строки соединения из реестра. Если строка соединения пуста, в реестр заносится значение, заданное по умолчанию**

```

1: Function GetDefaultConnectionString() As String
2:   GetDefaultConnectionString = _
3:     "Provider=Microsoft.Jet.OLEDB.4.0;" & _
4:     "DataSource=C:\Books\Sams\TeachYourself " & _
5:     "Access 2002 Programming in 24 Hours " & _
6:     "\Chapter 8\Hour8_1.mdb"
7: End Function
8:
9: Function GetConnectionString() As String
10:
11:   GetConnectionString = GetSetting("Hour 8", _
12:     "Database", "ConnectionString", " ")
13:
14:   If (GetConnectionString = vbNullString) Then
15:     Call SaveSetting("Hour 8", "Database", _
16:       "ConnectionString", GetDefaultConnectionString())
17:
18:     GetConnectionString = GetSetting("Hour 8", _
19:       "Database", "ConnectionString", " ")
20:   End If
21: End Function
22:
23: Sub ConnectWithRegistryInfo ()
24:   Dim Connection As New ADODB.Connection

```

```
25: Connection.Open (GetConnectionString())
26: MsgBox Connection.State = adStateOpen
27:End Sub
```

#### Анализ

В строке 23 определяется процедура `ConnectWithRegistryInfo`. В строке 25 с помощью `GetConnectionString()` считывается информация о соединении, установленном `GetSetting`. Если в регистре нет этой строки, используется значение по умолчанию, заданное в строке 15 данного листинга. В строке 26 производится проверка правильности соединения.

Помните, что `CurrentProject` — это глобальный объект, ссылающийся на активную базу данных. Если необходимо соединиться с внешним источником данных, понадобится сохранить информацию о конфигурации той базы данных.

Используйте реестр только в случае необходимости хранения такой информации о приложении, которая может понадобиться при следующем запуске приложения базы данных.

## Резюме

Функции и процедуры — это конструкции, которые позволяют группировать строки кода, предназначенные для решения небольших повторяющихся подзадач. Программа превращается в набор таких конструкций, управляемых из некоторой (главной) функции.

В ходе занятия вы узнали о том, как создавать процедуры и функции, передавать им значения-аргументы и уточнять смысл последних с помощью служебных слов-квалификаторов, придающих коду программы ясность и надежность.

Вы ознакомились с основными заповедями программирования, в сжатой форме аккумулирующими эмпирический профессиональный опыт, накопленный в течение нескольких десятилетий. Теперь вы в общих чертах представляете, как создавать лаконичный, выразительный и эффективный код.

Вашему вниманию были предложены практические примеры использования предложений на языке SQL и других средств управления базами данных в контексте применения процедур и функций. Теперь вам в определенной мере знакомы способы обращения к функциям из внешних динамических библиотек. Приведенный ниже раздел "Вопросы и ответы" освещает некоторые важные аспекты программирования, которым мы не смогли уделить надлежащего внимания.

## Вопросы и ответы

**Вопрос.** Мне необходимо протестировать код функции в среде Access. Каким образом это осуществить?

**Ответ.** Откройте модуль, содержащий функцию, в окне редактора Microsoft Visual Basic, установите текстовый курсор в требуемой строке кода функции и, нажимая клавишу <F8>, исполните код в режиме отладки.

**Вопрос.** Как заставить отладчик приостановить работу в определенной строке программы?

**Ответ.** Щелкните в левой колонке окна редактора напротив нужной строки кода. Строка будет помечена символом *точки останова* в виде небольшого красного кружка.

**Вопрос.** Во время отладки кода я хотел бы увидеть содержимое конкретной переменной. Как это сделать?

**Ответ.** Во время прохождения по строкам кода с помощью нажатия клавиши <F8> установите курсор мыши над наименованием нужной переменной — и в окне всплывающей подсказки вы увидите ее содержимое.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

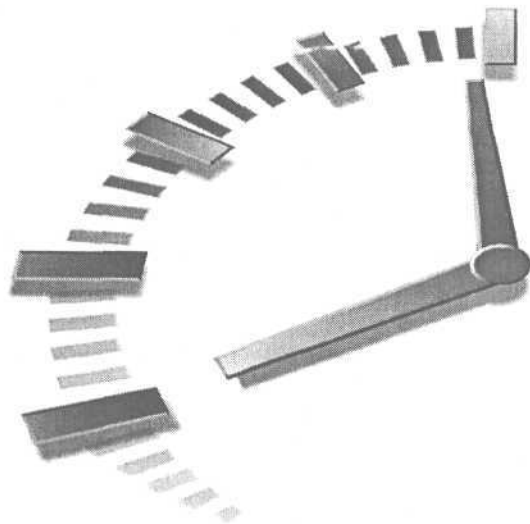
## Тесты

1. Назовите приемлемое среднее число строк кода функции или процедуры.
2. Предположим, что вы хотели бы снабдить некоторый аргумент функции (процедуры) **квалификатором** Optional. В каком месте списка параметров может находиться такой аргумент?
3. Придумайте название, уместное для функции, которая предусматривает операции чтения почтовых адресов из базы данных.
4. Каково имя объекта, указывающего на **текущую** открытую базу данных?
5. Как называется подкласс в составе класса Application, содержащий множество полезных свойств и методов?

## Упражнения

1. В начале раздела "Определение типов аргументов" был приведен пример процедуры, выполняющей расчет полной стоимости изделия. Реализуйте его в виде функции.
2. В разделе "Создание таблицы" мы рассматривали процедуру динамического построения таблицы. Напишите подпрограмму удаления таблицы из базы данных.
3. В разделе "Импорт текста из файла" мы говорили о процедуре импорта текста с разделителями в таблицу Access. Используя материалы оперативной справочной системы, перепишите процедуру, имея в виду возможности импорта данных из таблицы Excel.





## 9-й час

# Работа с макросами

Язык макросов — это язык программирования. Макрокод обрабатывается приложением-интерпретатором, которое считывает текст программы, написанный в соответствии с определенными синтаксическими правилами, анализирует и исполняет его.

Макросы Access предназначены для использования в двух направлениях. Во-первых, с их помощью можно создавать полноценные программы для решения реальных задач либо применять эпизодически в рамках крупных проектов. Во-вторых, макросы удобны как средство обучения основам программирования и быстрого получения черновых решений.

Каждая современная система программирования предоставляет в распоряжение пользователя определенный набор встроенных инструментальных средств — в частности, типов данных и функций. Язык макросов Access реализует некоторые из тех возможностей, которые доступны в VBA. При использовании макросов не требуются модули и тестовые функции; их удобно применять как средство быстрой реализации эскизных решений и экспериментальных образцов кода.

Основные темы занятия таковы.

- Как создавать макросы.
- Эксперименты с помощью макросов.
- Применение макросов для решения реальных задач.
- Замена макросов кодом VBA.

## Азбука макропрограммирования

Макросы хранятся в базе данных. Чтобы научиться создавать макросы, вам понадобится тестовая база данных. В ходе этого занятия мы воспользуемся базой CONCORDANCE, содержащей простой алфавитный указатель слов, который оформлен в виде двух взаимосвязанных таблиц. В первой из них, DOCUMENT, будут храниться наименования текстовых документов Word, а во второй, CONCORDANCE, — слова из этих документов и числовые показатели частоты их встречаемости.

# Создание таблицы с помощью SQL

В нашей учебной базе данных, CONCORDANCE, должны быть построены две таблицы. Первая таблица, DOCUMENT, будет содержать два столбца, а вторая, CONCORDANCE, — четыре. Чтобы построить таблицу DOCUMENT, достаточно выполнить следующую команду на языке SQL:

```
CREATE TABLE DOCUMENT (DOCUMENTID AUTOINCREMENT CONSTRAINT C1 PRIMARY KEY, FILENAME TEXT);
```

Таблицу CONCORDANCE необходимо создать с помощью такой команды:

```
CREATE TABLE CONCORDANCE (ID AUTOINCREMENT CONSTRAINT C1 PRIMARY KEY, DOCUMENTID LONG, WORD TEXT, COUNT INTEGER);
```

Чтобы построить таблицу с помощью конструкции CREATE TABLE языка SQL, выполните следующие действия.

1. Создайте новую базу данных.
2. В окне базы данных выберите элемент Запросы (Queries) списка Объекты (Objects).
3. Щелкните на кнопке Создать (New) панели инструментов.
4. В списке режимов диалогового окна Новый запрос (New Query) выберите элемент Конструктор (Design View) и щелкните на кнопке ОК.
5. Закройте диалоговое окно Добавление таблицы (Show Table).
6. Выберите в строке меню команду Вид⇨Режим SQL (View⇨SQL View).
7. В окне редактора Запрос на выборку введите текст команды создания таблицы DOCUMENT, приведенный выше. Повторите операцию, введя ниже команду построения таблицы CONCORDANCE.
8. Выберите в строке меню команду Запрос⇨Запуск.

Таблицы DOCUMENT и CONCORDANCE (именно в этом порядке) связаны соотношением типа *один-ко-многим*. Это означает, что каждой записи таблицы DOCUMENT может соответствовать несколько (нуль или более) записей таблицы CONCORDANCE.

Более подробные сведения о синтаксисе команды CREATE TABLE и примерах ее использования вы сможете почерпнуть в оперативной справочной системе Access, адресовав Помощнику Office ключевую фразу **CREATE TABLE**. Ниже приведено описание синтаксиса конструкции CREATE TABLE.



```
CREATE TABLE ИмяТаблицы [(ИмяСтолбца ТипДанных  
[(Размерность)] [, ...])]
```

После слов CREATE TABLE следует наименование таблицы и список описаний ее столбцов, заключенный в круглые скобки. Количество столбцов произвольно. Описания столбцов разделяются символом запятой.

Ниже представлена исправленная версия команды создания таблицы DOCUMENT, в которой для столбца FILENAME явно указана его размерность:

```
CREATE TABLE DOCUMENT (DOCUMENTID AUTOINCREMENT CONSTRAINT C1 PRIMARY KEY, FILENAME TEXT(255));
```

В последующих примерах мы будем ссылаться на таблицы, построенные в базе данных CONCORDANCE.

# Создание макроса

Макросы сохраняются в виде объектов базы данных. Чтобы создать макрос, в окне базы данных выберите элемент Макросы (Macros) списка Объекты и щелкните на кнопке Создать панели инструментов. Откроется окно Макрос (Macro), позволяющее ввести текст нового макроса с именем *Макрос1*, предложенным системой по умолчанию (рис. 9.1).

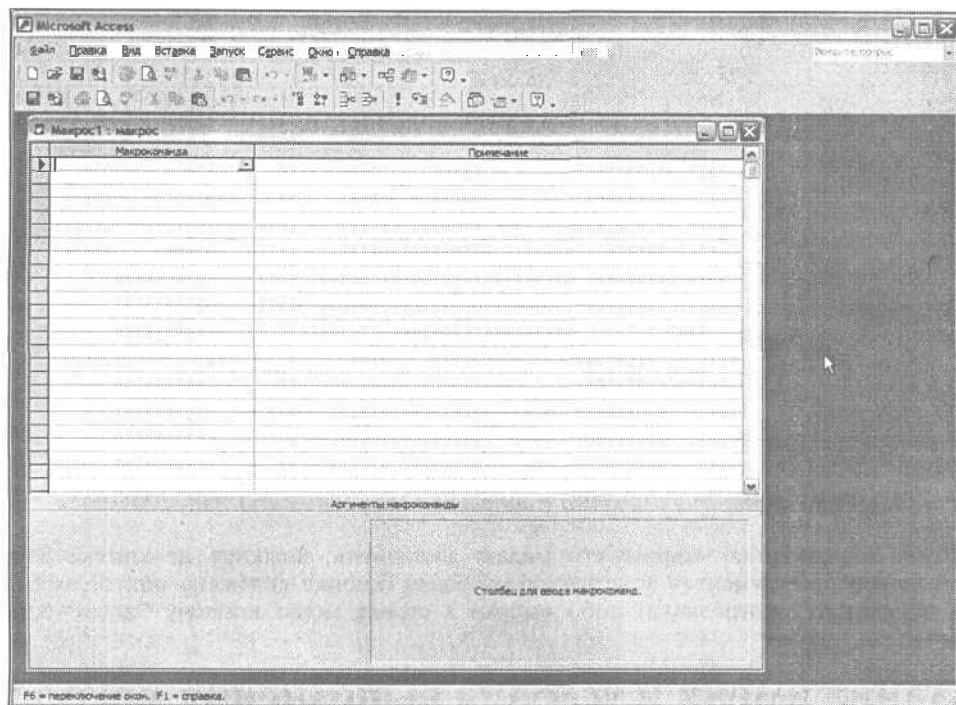


Рис. 9.1. Так выглядит окно макроса

Макрос определяется во многом так же, как структура таблицы базы данных. Левый столбец окна Макрос — Макрокоманда (Action) — позволяет выбрать из раскрывающегося списка требуемую макрокоманду, а правый — Примечание (Comment) — предназначен для ввода комментариев.

После того как макрокоманда выбрана, в левой нижней части окна появляются поля ввода *аргументов макрокоманды*. Они играют примерно ту же роль, что и параметры процедуры или функции. Аргументы снабжают макрокоманду данными, необходимыми для ее корректного выполнения.

В первой ячейке столбца Макрокоманда введите либо выберите с помощью раскрывающегося списка значение ЗапускМакроса (RunMacro). При этом откроется три поля ввода аргументов — Имя макроса (Macro Name), Число повторов (Repeat Count) и Условие повтора (Repeat Expression). Введя значения аргументов, вы сможете выполнить макрос, указанный в поле Имя макроса (рис. 9.2).



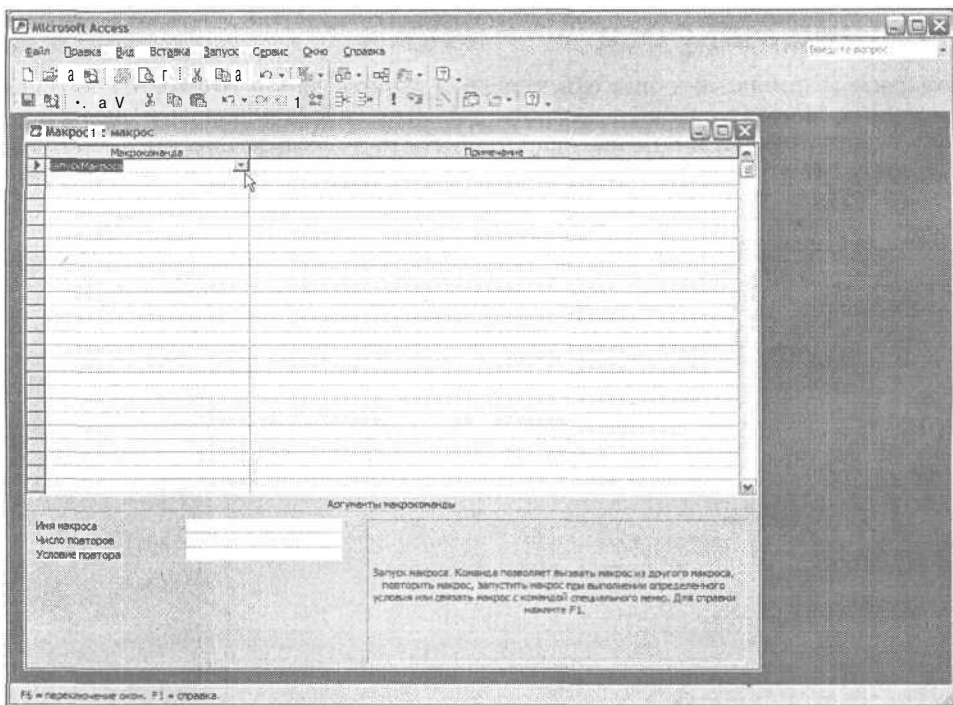


Рис. 9.2. Окно определения макроса с выбранной макрокомандой **ЗапускМакроса**

После определения макроса его можно выполнить, щелкнув на кнопке **Запуск** (Run) панели инструментов Конструктор макросов (кнопка снабжена пиктограммой в виде восклицательного знака) либо выбрав в строке меню команду **Запуск⇒Запуск** (Run⇒Run).

## Задание имени и условия выполнения макроса

Окно **Макрос** позволяет отобразить еще два столбца для задания дополнительных свойств макроса — **Имя макроса** (Macro Name) и **Условие** (Condition). Увидеть их на экране можно, выполнив последовательно две команды строки меню — **Вид⇒Имена макросов** (View⇒Macro Name) и **Вид⇒Условия** (View⇒Condition). Если необходимо, чтобы система всегда отображала эти столбцы, выберите команду **Сервис⇒Параметры** (Tools⇒Options), перейдите на вкладку **Вид** (View) диалогового окна **Параметры** (Options), обратитесь к группе элементов **Конструктор макросов** (Show in Macro), установите флажки **Столбец имен** (Name Column) и **Столбец условий** (Conditions column), а затем щелкните на кнопках **Применить** (Apply) и **ОК** (рис. 9.3).

Столбец **Условие** весьма полезен, так как позволяет задать логическое условие выполнения макроса. Если результат вычисления условия равен **True**, макрокоманда выполняется; в противном случае она действовать не будет.



Отдельные строки макрокда можно на время отключить, введя в полях столбца **Условие** литеральные значения **False**.

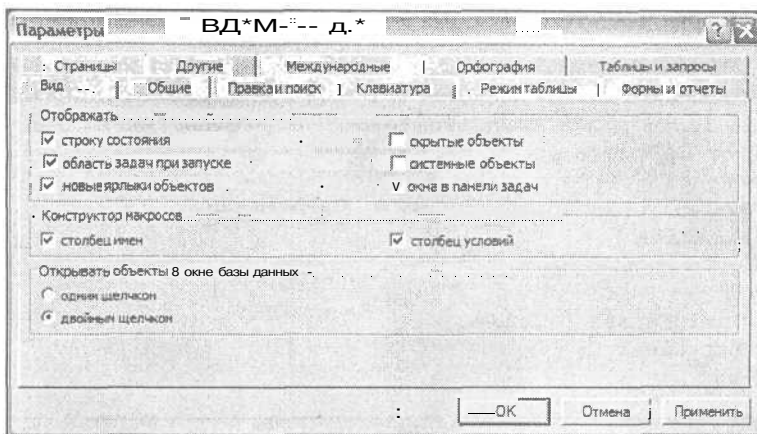


Рис. 9.3. Диалоговое окно *Параметры* позволяет изменить значения параметров работы *Access*, принятые по умолчанию

В ячейки столбца *Условие* могут быть помещены самые разнообразные логические выражения. Рис. 9.4 иллюстрирует использование команды отрицания результата вызова функции *TableExists*. Далее мы будем тестировать результат выполнения команды *CREATE TABLE* языка SQL. Эта команда рассматривалась выше, в разделе “Создание таблицы с помощью SQL”. Цель тестирования — проверить, существует ли таблица, которую мы предполагаем построить. Логическое выражение выглядит следующим образом:

```
Not TableExists ( "DOCUMENT" )
```

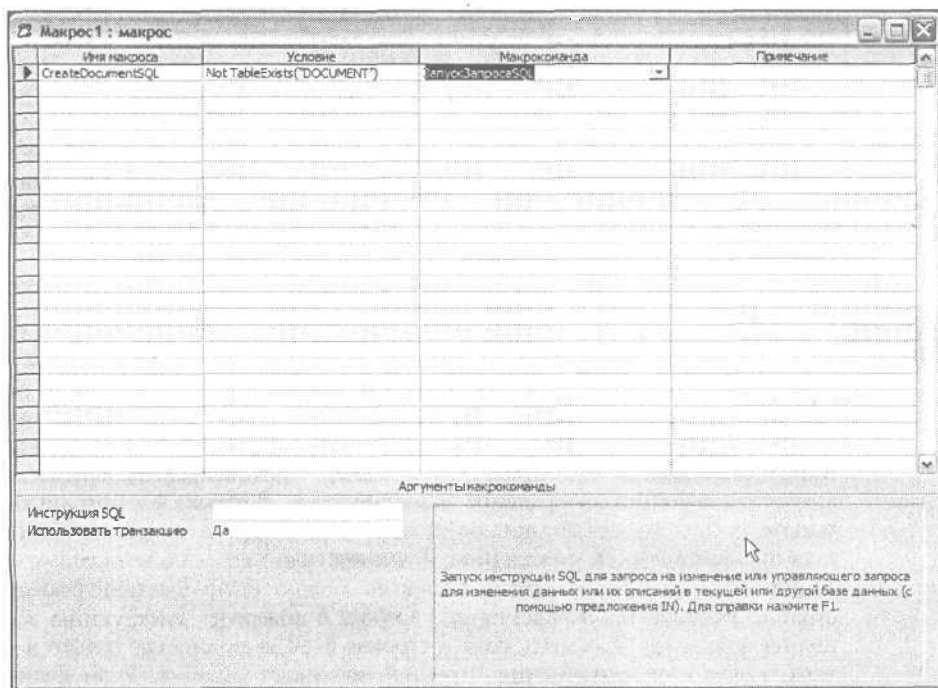


Рис. 9.4. Столбец *Условие* позволяет задать логическое выражение, управляющее процессом выполнения макрокоманды

### Листинг 9.1. Функция проверки наличия таблицы

```
1: Function TableExists(ByVal TableName As String) As Boolean
2:   On Error GoTo Except
3:   TableExists = True
4:   Call DoCmd.SelectObject(acTable, TableName)
5:   Exit Function
6:
7:   Except:
8:     TableExists = False
9:     Err.Clear
10:  End Function
```



Код листинга 9.1 будет работать независимо от того, существует ли открываемая таблица, если в редакторе Visual Basic включена опция Break On All Error Checked (ее можно установить на вкладке General диалогового окна Tools⇒Options)

Опция Break On All Error Checked очень удобна в процессе разработки. Безусловно, она не нужна при распространении базы данных конечным пользователям — ведь вы не хотите посещать своих пользователей каждый раз при возникновении ошибки в базе данных. Приложение должно остановиться, если нельзя обработать ошибочное условие, и нет смысла продолжать работу.



Щелчок правой кнопкой мыши в пределах столбца Условие приводит к появлению контекстного меню. При выборе элемента Построить (Build) открывается диалоговое окно Построитель выражений (Condition Builder), облегчающее задачу создания условных выражений и обеспечивающее доступ к различным объектам базы данных — таблицам, запросам, формам, отчетам, функциям, константам, операторам и общим выражениям. Инструменты окна Построитель выражений хорошо документированы в файлах оперативной справочной системы Access. Оно всегда в вашем распоряжении.

#### Анализ

Строка 1 листинга 9.1 содержит заголовок функции TableExists. В строке 2 определяется код обработки ошибок (подробнее см. главу “18-й час. Обработка ошибок во время выполнения программы”): компилятор получает инструкцию перехода на метку Except в случае возникновения любой ошибки выполнения кода функции. Оператор строки 3 — в предположении, что ошибок не будет (мы оптимисты!), — присваивает возвращаемому аргументу функции литеральное значение True. В строке 4 с помощью команды DoCmd.SelectObject осуществляется попытка открыть таблицу с заданным именем. (К сожалению, функция OpenTable не возвращает значения типа Boolean — иначе вызов ее можно было бы использовать в столбце Условие непосредственно.) Строка 6 содержит инструкцию завершения функции. Фрагмент кода в строках 8-10 выполняется только в случаях, когда при выполнении строки 4 возникает ошибка. Если функция возвращает значение False — это сигнал о том, что таблица не найдена. В строке 10 выполняется очистка содержимого объекта Err.

Объект Егг хранит информацию о характере ошибки и заполняется соответствующими данными при возникновении любой ошибки. Чтобы более подробно проанализировать причины ошибки, необходим дополнительный код (обратитесь к главе 18).

## Тестирование и отладка макроса

После создания макрос можно (и нужно) протестировать. Конечно, таких мощных инструментов отладки, какие имеются в интегрированной среде программирования Microsoft Visual Basic, нет, но некоторые (довольно эффективные) средства тестирования макросов все-таки существуют.

В ходе дальнейшего изложения мы будем опираться на пример макроса, описанный в предыдущем разделе. Чтобы создать макрос, выполните такие действия.

1. В окне базы данных выберите элемент Макросы списка Объекты.
2. Щелкните на кнопке Создать панели инструментов.
3. В ячейку столбца Условие введите выражение `Not TableExists ( "DOCUMENT" )`.
4. В ячейке столбца Макрокоманда выберите элемент `ЗапускЗапросаSQL (RunSQL)`.
5. В поле аргумента Инструкция SQL макрокоманды введите выражение SQL, содержащее команду построения таблицы DOCUMENT.
6. Сохраните модуль и переключитесь в режим макроса.
7. Щелкните на кнопке Запуск.
8. Появится запрос о сохранении макроса. Щелкните на кнопке Да (Yes).

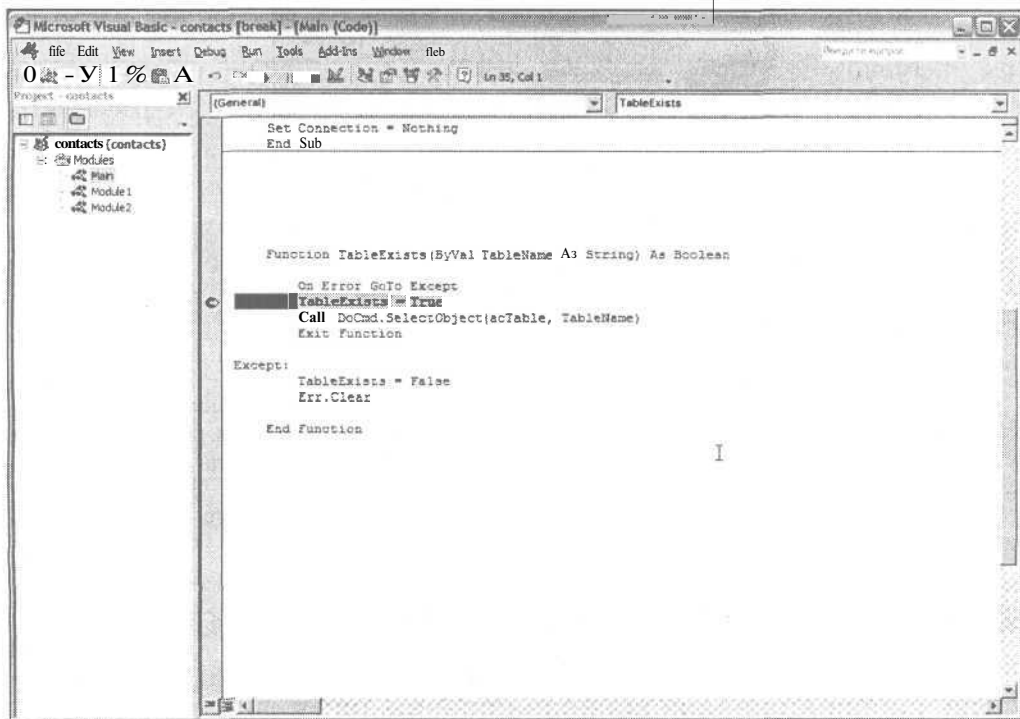


Рис. 9.5. При отладке макроса рекомендуется задавать точки останова в строках кода используемых функций

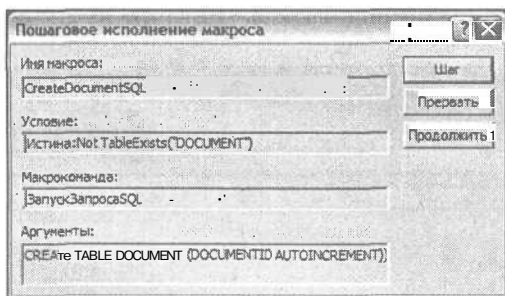
Выполнить макрос можно также с помощью команды меню **Запуск⇒Запуск**. Даже если таблица **DOCUMENT** существует (мы ведь ранее уже создавали ее), сообщение об ошибке выдано не будет, поскольку макрос просто не выполнится.

Чтобы осуществить пошаговое выполнение макроса, щелкните на кнопке **По шагам (Step)** — на панели инструментов Конструктор макросов она расположена справа от кнопки **Запуск** — либо воспользуйтесь командой меню **Запуск⇒По шагам (Run⇒Single Step)**. Кнопка **По шагам** действует в режиме переключателя — теперь при каждом щелчке на кнопке **Запуск** будет выполняться одна строка макроса.

Режим пошагового выполнения особенно полезен именно тогда, когда макрос состоит из нескольких строк. Если при тестировании макроса необходимо проверить и код функции или процедуры VBA, в соответствующих ее строках следует задать точки останова (контрольные точки). На рис. 9.5 показано окно редактора, в котором контрольной точкой отмечена строка **TableExists = True**.

Чтобы задать точку останова, откройте модуль, содержащий требуемую функцию. Щелкните в пределах нужной строки и нажмите клавишу <F9> (либо щелкните в левой колонке окна напротив строки). Строка будет подсвечена красным цветом и отмечена специальной пиктограммой. Чтобы снять контрольную точку, повторите операцию.

Редактор Microsoft Visual Basic позволяет открывать полезные дополнительные окна — **Watches**, **Locals** и **Immediate** (мы говорили о них ранее). Но в составе окна **Макрос** подобных инструментов нет. После выполнения очередной строки кода макроса отображается диалоговое окно **Пошаговое исполнение макроса** (рис. 9.6).



*Рис. 9.6. После выполнения строки кода макроса открывается диалоговое окно Пошаговое исполнение макроса, содержащее сведения о статусе макроккода*

Теперь, ознакомившись с основами создания и применения макросов, вы сможете использовать полученные знания для предварительного тестирования решений еще до их окончательной реализации в виде кода VBA либо для построения более мощных макросов, которым в проекте будет отведена самостоятельная роль. Остаток занятия мы посвятим рассмотрению ряда полезных примеров практического использования макросов.

## Макросы группы КопироватьОбъект

Макросы, основанные на команде **КопироватьОбъект**, (**CopyObject**) позволяют выполнять операции копирования различных объектов — таблиц, запросов, форм, отчетов, макросов, модулей или страниц доступа к данным Access — как внутри текущей базы данных, так и между разными базами данных.

В этом разделе рассказано, как создать макрос для копирования таблицы CONCORDANCE. Рассмотрите рис. 9.7 и выполните действия, указанные ниже.

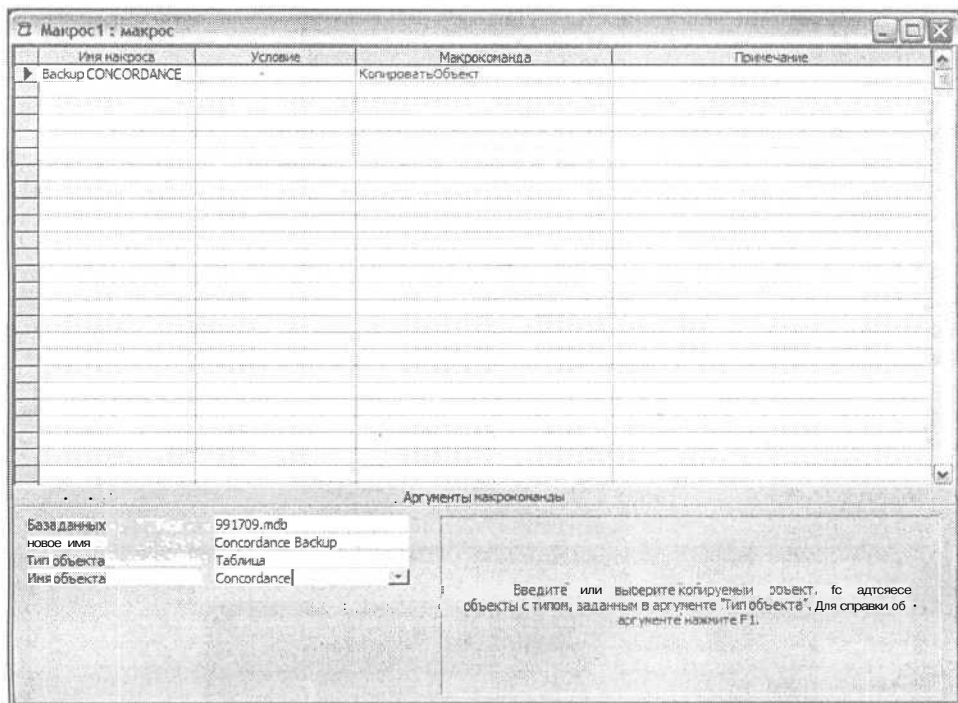


Рис. 9.7. Так создается макрос для копирования таблицы CONCORDANCE

1. В окне базы данных выберите элемент Макросы списка Объекты.
2. Щелкните на кнопке Создать панели инструментов.
3. В ячейке столбца Макрокоманда выберите элемент КопироватьОбъект. Ниже будут отображены поля ввода аргументов макрокоманды.
4. Поле База данных оставьте пустым — мы предполагаем, что копия таблицы CONCORDANCE должна содержаться в текущей базе данных.
5. В поле Новое имя введите имя таблицы-копии — CONCORDANCE\_BACKUP.
6. В раскрывающемся списке Тип объекта выберите элемент Таблица.
7. В поле Имя объекта введите CONCORDANCE либо выберите это наименование в раскрывающемся списке.

Не забудьте сохранить макрос и протестировать его. После однократного выполнения макроса список Таблицы окна базы данных пополнится новым объектом — CONCORDANCE\_BACKUP.

## Использование макросов УдалитьОбъект

Макрос типа УдалитьОбъект (DeleteObject) в сочетании с построенными ранее макросом копирования и функцией проверки существования таблиц (TableExists) — удобное средство организации полнофункциональной схемы резервного копирования таблиц. Чтобы реализовать эту идею в отношении таблицы

CONCORDANCE, создадим новый макрос на основе команды УдалитьОбъект и используем его совместно с макросом Concordance Backup и функцией TableExists.

Задача построения макроса типа УдалитьОбъект довольно проста — достаточно выполнить следующие действия.

1. В окне базы данных выберите элемент Макросы списка Объекты.
2. Щелкните на кнопке Создать панели инструментов.
3. В ячейке столбца Макрокоманда выберите элемент УдалитьОбъект.
4. В раскрывающемся списке Тип объекта выберите элемент Таблица и в поле Имя объекта введите CONCORDANCE\_BACKUP либо выберите это наименование в раскрывающемся списке.

Запустите макрос на выполнение, чтобы проверить корректность его работы, и сохраните ПОД именем Delete Concordance.

Если вы попытаетесь исполнить макрос повторно, Access откроет окно сообщения об ошибке (рис. 9.8).

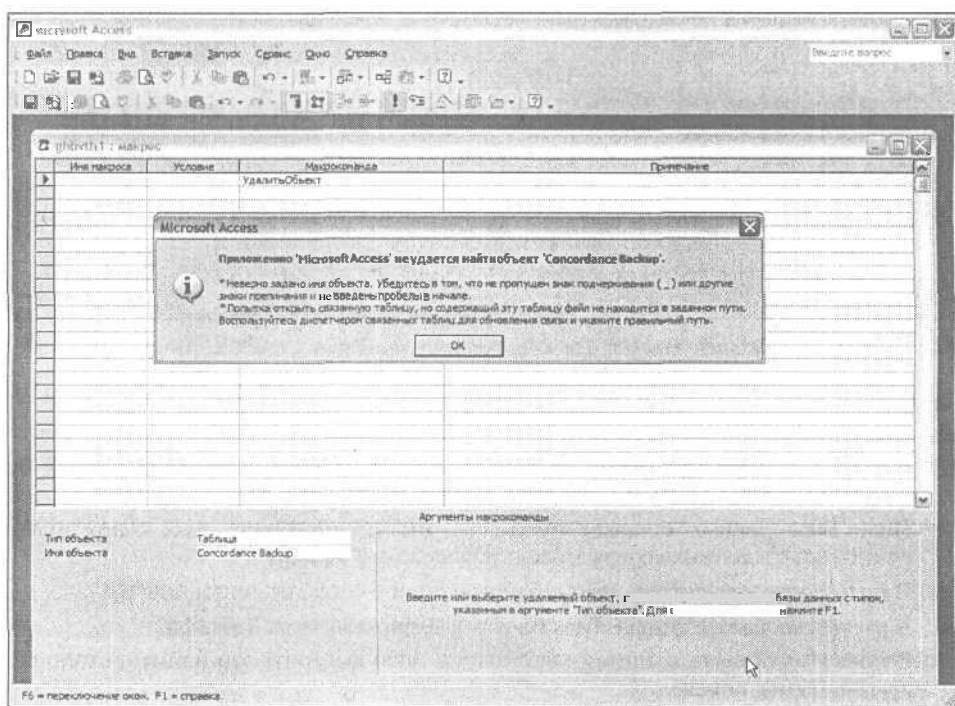


Рис. 9.8. Нельзя удалить таблицу, которой уже нет!

Чтобы объединить решения в общую схему, потребуется создать новый макрос, скопировав в него строки существующих. Технология повторного использования вполне применима и при работе с макросами. В новых макросах можно частично использовать код старых, но при этом необходимо проводить тестирование. А если строить новый объект на основе существующих без их редактирования, тестирование можно не проводить.

Макрос, который мы собираемся построить, должен вначале проверять факт существования объекта (в нашем примере — таблицы CONCORDANCE\_BACKUP) и только за-

тем можно предпринимать попытку его удаления. Если объект существует, поочередно должны быть выполнены макросы Delete Concordance и Concordance Backup.

Чтобы построить макрос создания резервной копии, используйте новый макрос и команду **ЗапускМакроса** (RunMacro). В новый макрос будут включены оба созданных ранее макроса. Реализация решения в соответствии с указанной схемой предполагает, что ранее созданные макросы не должны изменяться. Данный подход является продуктивным — создание нового объекта на основе существующих без какого-либо их редактирования.

Чтобы построить новый макрос, Backup, выполните следующие действия.

1. В окне базы данных выберите элемент Макросы списка Объекты.
2. Щелкните на кнопке Создать панели инструментов.
3. В ячейке столбца Макрокоманда выберите элемент ЗапускМакроса.
4. Введите логическое выражение TableExists ( "CONCORDANCE\_BACKUP" ) в ячейку столбца Условие.
5. В раскрывающемся списке аргумента Имя макроса выберите наименование макроса Delete Concordance.
6. Добавьте новую строку макроса и в ячейке Макрокоманда вновь выберите элемент ЗапускМакроса.
7. В раскрывающемся списке аргумента Имя макроса для второй строки выберите наименование макроса Backup Concordance.
8. Сохраните созданный макрос под именем Backup.

Поскольку все объекты, которые использовались нами при построении макроса Backup, ранее уже проверялись, мы можем быть в достаточной степени уверены, что и теперь они нас не подведут. Агрегация макросов упрощает код.

## Ключи от города

Макрокоманду **ВыполнитьКоманду** (RunCommand) можно сравнить с ключами от волшебного города. С ее помощью вам удастся справиться почти со всеми аспектами поведения Access. Макрос типа **ВыполнитьКоманду** требует задания единственного аргумента, который необходимо выбрать в раскрывающемся списке Команда. Список Команда содержит несколько десятков предопределенных инструкций Access.

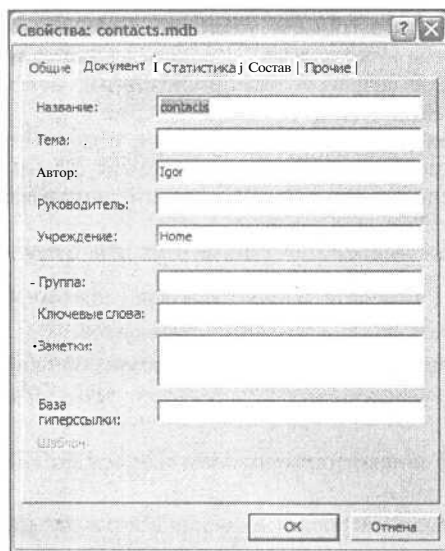
Возможности макросов типа **ВыполнитьКоманду** настолько обширны, что их описание составит отдельную книгу. Здесь уместно вновь напомнить вам о гигантском потенциале оперативной справочной системы Access. Стучитесь — и вам откроют!

Макросы **ВыполнитьКоманду** настолько же мощны, насколько и просты. Достаточно при создании нового макроса выбрать в раскрывающемся списке ячейки Макрокоманда значение **ВыполнитьКоманду**, а ниже, в списке Команда (Command), указать требуемую инструкцию Access. На рис. 9.9 представлен результат выполнения макроса типа **ВыполнитьКоманду** в том случае, когда в качестве аргумента выбрано значение DatabaseProperties.



В Access 2002 можно удалить информацию о пользователе из файла базы данных (т.е. одно из ее свойств). Для этого обратитесь к диалоговому окну Сервис⇒Параметры (Tools⇒Options) и на вкладке Общие (General) установите флажок Удалить личные сведения из файла.





*Рис. 9.9. С помощью макроса легко отобразить диалоговое окно, содержащее информацию о текущей базе данных Access*

## Импорт данных

Импорт информации из внешних источников — это весьма распространенная операция. Рано или поздно вам наверняка потребуется скопировать в таблицу Access содержимое электронной таблицы, воспользоваться текстовой информацией из разных источников либо перенести данные из одной базы Access в другую.

## Преобразование базы данных

Чтобы осуществить преобразование базы данных с помощью макроса, вначале необходимо создать соответствующий новый макрос. Для параметра Макрокоманда надлежит выбрать значение **ПреобразоватьБазуДанных** (TransferDatabase). В этом случае группа интерфейсных элементов Аргументы макрокоманды окна Макрос будет содержать следующие поля ввода: Тип преобразования (Transfer Type), Тип базы данных (Database Type), Имя базы данных (Database Name), Тип объекта (Object Type), Источник (Source), Получатель (Destination) и Только структура (Structure Only).

Параметр Тип преобразования допускает выбор одного из трех значений — Импорт (Import), Экспорт (Export) и Связь (Link). Элемент Импорт соответствует ситуации, когда требуется сохранить информацию из внешнего источника в текущей базе данных. Выбор значения Экспорт предполагает выполнение обратной операции, т.е. перенос данных из текущей базы Access во внешний файл. Наконец, режим Связь позволяет создать логическую ссылку на внешний источник данных. Связывание данных дает возможность избежать операции копирования и гарантирует, что при динамическом обновлении источника информация будет самой новой.

## Аргумент Тип базы данных

В качестве значения параметра Тип базы данных может быть выбрано символическое наименование любой базы данных, для которой имеется соответствующий драйвер, корректно установленный в компьютерной системе. Названия всех таких драйверов перечислены в раскрывающемся списке. Выбор последнего элемента списка, База данных ODBC (ODBS Database), требует задания соответствующего имени источника данных ODBC в поле аргумента Имя базы данных.



**ODBC** (сокращение от *Open DataBase Connectivity*) — протокол, обеспечивающий взаимодействие между компьютерными приложениями и различными системами управления базами данных. ODBC поддерживается большинством производителей программного обеспечения.

## Аргумент Имя базы данных

В качестве значения аргумента Имя базы данных необходимо задать имя физического файла данных либо название источника данных ODBC, взаимодействие с которым предполагается осуществлять.

## Аргумент Тип объекта

Параметр Тип объекта описывает разновидность объекта данных. Множество допустимых значений — это Таблица (Table), Запрос (Query), Форма (Form), Отчет (Report), Макрос (Macro), Модуль (Module). Выбранный тип объекта должен соответствовать как существу источника, так и природе получателя данных.

## Аргументы Источники Получатель

Параметр Источник задает наименование объекта-источника данных, а Получатель — название, которое будет присвоено объекту-копии или связывающей ссылке.

## Аргумент Только структура

Аргумент Только структура допускает выбор одного из двух значений — Да (Yes) или Нет (No). Если выбрано значение Нет, операции экспорта/импорта будет подвержена как структура объекта данных источника, так и его содержимое. В противном случае будет скопирована только структура.



Режим Только структура применим лишь в случае, если в качестве источника данных выбран объект Таблица.

С помощью макроса на основе команды ПреобразоватьБазуДанных легко решить задачу создания резервной копии таблицы, рассмотренную ранее.

## Преобразование текста

Макрос типа ПреобразоватьТекст позволяет обрабатывать (импортировать, экспортировать или связывать) текстовые данные упорядоченной структуры. (Если данные представлены в виде набора форматов, предварительно необходимо разнести однородные данные по нескольким источникам с помощью дополнительной программы.)

Чтобы воспользоваться макросом на основе команды ПреобразоватьТекст (TransferText), следует выполнить две важные операции. Во-первых, создайте новый макрос, выбрав макрокоманду ПреобразоватьТекст и задав нужные аргументы. Во-вторых, определите спецификацию преобразования (но это не так просто).

Множество аргументов макрокоманды ПреобразоватьТекст состоит из следующих элементов: Тип преобразования, Название спецификации (Specification Name), Имя таблицы (Table Name), Имя файла (File Name), С именами полей (Has Field Names), Имя таблицы HTML (HTML Table Name) и Кодовая страница (Code Page).

## Аргумент Тип преобразования

Множество значений параметра Тип преобразования охватывает различные варианты экспорта, импорта и связывания данных. Чтобы, например, осуществить импорт текстовых данных с разделителями полей, достаточно выбрать значение Импорт (разделители) (Import Delimiter), при этом режиму экспорта в формат HTML должен соответствовать Экспорт (HTML).

## Определение спецификации преобразования

В качестве значения параметра Название спецификации выступает содержимое определенного поля одной из скрытых системных таблиц Access.

Освоение процедуры определения спецификации мы начнем с создания текстового файла *PhoneBook.Txt*, содержащего строки данных с разделителями. Формат каждой строки имеет следующий вид:

Имя, Адрес, Город, Область, ПочтовыйИндекс

Для определения спецификации этих данных вполне достаточно.

Создание спецификации преобразования — составная часть процесса импорта, экспорта или связывания данных. Мы рассмотрим эту процедуру на примере задачи импорта данных из файла *PhoneBook.Txt*.

Чтобы определить спецификацию для макроса преобразования текстовых данных, выполните следующие действия.

1. Создайте новую базу данных.
2. В окне базы данных выберите элемент Таблицы (Tables) из списка Объекты.
3. Щелкните на кнопке Создать панели инструментов.
4. В списке режимов диалогового окна Новая таблица (New Table) выберите элемент Импорт таблиц (Import).
5. В раскрывающемся списке Тип файлов (Files of Types) диалогового окна Импорт (Import) укажите элемент Текстовые файлы (Text Files).
6. Проследуйте к папке, содержащей файл *PhoneBook.Txt*, выберите его и щелкните на кнопке Импорт.
7. В окне мастера Импорт текста будет автоматически выбрана опция С разделителями. Чтобы увидеть, как создается спецификация, щелкните на кнопке Дополнительно (Advanced).
8. Откроется диалоговое окно Спецификация импорта (Import Specification) (рис. 9.10). В нижней части окна отображается список распознанных системой столбцов (в нашем примере их пять) с временными наименованиями вида Поле1, Поле2, ...; замените названия более конкретными (Name, Address, City, Region, Postal\_Code).
9. Сохраните спецификацию: щелкните на кнопке Сохранить как (Save As), в поле Название спецификации диалогового окна Сохранение спецификации импорта и экспорта оставьте без изменений название, предложенное системой по умолчанию (*PhoneBook — спецификация импорта*) и щелкните на кнопке ОК. Закройте диалоговое окно Спецификация импорта щелчком на кнопке ОК.
10. Щелкните на кнопке Далее (Next) окна мастера Импорт текста. Установите флажок опции Первая строка содержит имена полей (First Row Contains Field Names).
11. Щелкните на кнопке Далее.
12. Третье диалоговое окно мастера Импорт текста позволяет указать, где необходимо сохранить данные — в новой или существующей таблице. Примите вариант, предложенный по умолчанию, — В новой таблице (New Table).
13. Щелкните на кнопке Далее.

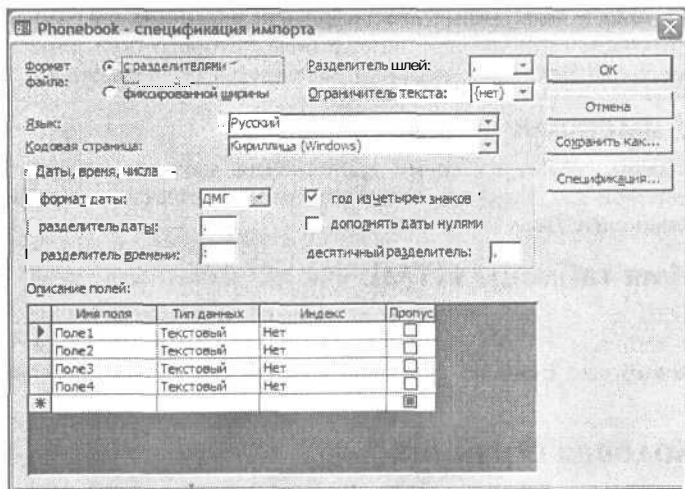


Рис. Р. 10. Так выглядит диалоговое окно Спецификация импорта

14. Четвертое окно мастера Импорт текста дает возможность задать наименование и тип каждого поля, а также признак принадлежности поля индексу. Выберите поле Region и укажите в раскрывающемся списке Индекс (Index) элемент Да (Допускаются совпадения) (Yes (Duplicates OK)).
15. Щелкните на кнопке Далее.
16. Оставьте выбранной опцию Автоматически создать ключ.
17. Щелкните на кнопке Далее.
18. Не изменяя предложенного системой имени таблицы — приемника данных (PhoneBook), щелкните на кнопке Готово (Finish), чтобы завершить процедуру.

Создание и сохранение спецификации преобразования было осуществлено нами при выполнении пп.8 и 9 инструкции. Откройте таблицу PhoneBook и взгляните на ее содержимое; если оно вас устраивает, значит, спецификация верна.

Теперь перейдите к окну создаваемого макроса и с помощью раскрывающегося списка Название спецификации выберите имя только что построенной спецификации преобразования текста.

## Аргумент Имя таблицы

В поле Имя таблицы необходимо указать уникальное название таблицы. Чтобы избежать ошибок, связанных с использованием существующих имен, можно воспользоваться макросом на основе команды УдалитьОбъект, рассмотренной в разделе "Использование макросов УдалитьОбъект".

## Аргумент Имя файла

Параметр Имя файла обязателен. Но одного имени файла, недостаточно. Как же проверить, действительно ли заданный файл существует? Чтобы ответить на этот вопрос, придется, вероятно, прибегнуть к дополнительным средствам. Вот один из способов обеспечения возможности динамического задания имени файла с данными, которые необходимо импортировать. В качестве значения аргумента Имя файла введите выражение

```
=InputBox( "Введите имя файла для импорта:", "Имя файла",  
"PhoneBook.Txt" )
```

Не забудьте в начале выражения ввести символ равенства (=). Теперь при выполнении макроса откроется диалоговое окно, в поле которого пользователь сможет ввести имя требуемого файла (по умолчанию предлагается PhoneBook.Txt).

## Аргумент С именами полей

ЕСЛИ необходимо, чтобы во время выполнения макроса первая строка данных трактовалась системой как перечень названий полей, в качестве значения аргумента С именами полей выберите Да.

## Аргумент Имя таблицы HTML

Access способна взаимодействовать и с данными в формате HTML. При заполнении поля Имя таблицы HTML убедитесь, что в качестве значения параметра Тип преобразования выбрано одно из следующих — Импорт (HTML), Экспорт (HTML) или Связь (HTML).

## Аргумент Кодовая страница

Список допустимых значений параметра Кодовая страница весьма обширный. Ваш выбор зависит от того, в какой кодовой странице набраны импортируемые/экспортируемые текстовые данные.

# Замена макрокоманд инструкциями на языке VBA

В ходе выполнения функции или процедуры, написанной на языке VBA, может быть имитирован вызов любой из макрокоманд Access. С этой целью используется объект класса DoCmd. Каждому типу макрокоманд, доступных в окне создания макроса, соответствует определенный метод объекта DoCmd. Так, например, при необходимости преобразования текста с помощью кода VBA следует применить следующую конструкцию:

```
DoCmd.TransferText [ТипПреобразования]  
[, НазваниеСпецификации], ИмяТаблицы, ИмяФайла  
[, СИменамиПолей] [, ИмяТаблицы HTML] [, Кодовая страница]
```

Аргументы ИмяТаблицы и ИмяФайла обязательны. Параметру ТипПреобразования по умолчанию присваивается значение Импорт (разделители). Выполняя операцию импорта с разделителями, нет необходимости использовать особую спецификацию преобразования. Значение аргумента СИменамиПолей, предлагаемое по умолчанию, равно False.

Ниже приведен листинг 9.2, содержащий описание процедуры, которая решает рассмотренную выше задачу импорта текстовых данных с разделителями.

### Листинг 9.2. Пример процедуры импорта текстовых данных с разделителями

```
1: Sub ImportTextDelimited()  
2:   Const DefaultFileName = "c:\temp\book\phonebook.txt"  
3:   Const ImportSpecification = "Название спецификации"  
4:   Dim FileName As String  
5:   FileName = InputBox( "Введите имя файла для импорта:", _  
       "Имя файла", DefaultFileName)  
6:   DoCmd.TransferText acImportDelim, ImportSpecification, _  
7:       "PHONE_BOOK", FileName  
8: End Sub
```

В строке 4 объявляется переменная `FileName`, которая предназначена для хранения имени импортируемого файла. Это имя вводится пользователем в диалоговом окне, открываемом при выполнении строки 5. Строка 6 содержит вызов макрокоманды импорта, использующей определенную ранее спецификацию.

## Макросы: быть или не быть

С точки зрения профессионального программиста, ответ на вопрос, использовать макросы или код VBA, однозначен — конечно, код. Макросы необходимы тем, кто не любит или не умеет программировать. Макросы можно также рассматривать как удобное средство проведения экспериментов с различными командами (такими как `TransferText`) и их аргументами. Макросы в чистом виде редко оказываются способными обеспечить полноценное и надежное решение — так или иначе их следует дополнять программными средствами анализа ошибок и динамического изменения данных.

Существенным преимуществом макросов является то, что их легко записать, а затем преобразовать в VBA-код и сохранить в модуле. Для преобразования макроса в код выполните следующие действия.

1. Выберите Макросы из списка Объекты.
2. Щелкните правой кнопкой мыши на макросе, который нужно преобразовать и выберите Сохранить как (Save As).
3. В диалоговом окне Сохранение (Save As) задайте имя и в списке Как (As) отметьте Модуль (Module). Щелкните на кнопке ОК.
4. В диалоговом окне Преобразования макроса (Convert) оставьте параметры, заданные по умолчанию.

Щелкните на кнопке Преобразовать (Convert) — будет создан модуль и откроется редактор Visual Basic. В листинге 9.3 приведен пример макроса Backup Concordance, преобразованный в VBA-код.

Листинг 9.3. Макрос Backup Concordance, преобразованный в VBA-код

```
1: Option Compare Database
2: Option Explicit
3: \ _____ - _____
4: Backup_Concordance
5:
6: \-----
7: Function Backup_Concordance ()
8:   On Error GoTo Backup_Concordance_Err
9:
10:   DoCmd.CopyObject "991709.mdb", "Concordance Backup", acTable,
      "Concordance"
11:
12:
13: Backup_Concordance_Exit:
14:   Exit Function
15:
16: Backup_Concordance_Err:
17:   MsgBox Error$
18:   Resume Backup_Concordance_Exit
19:
20: End Function
```

Преобразование макроса в код — обычное явление. Наиболее важен код строки 10, содержащий команду `CopyObject`. Все остальное — процедуры обработки ошибок, отображающее содержимое объекта `Error`. Код макросов, генерируемый VBA, рекомендуется использовать для обучения, обычно он грешит многословием. Написание кода дает вам больший простор для творчества. Начните с создания макросов. Если с их помощью не удалось решить поставленную задачу или обеспечить достаточную защиту от ошибок, преобразуйте макрос с VBA-код и внесите необходимые изменения.

## Резюме

В ходе этого занятия вы ознакомились со средствами окна проектирования макросов, научились выбирать макрокоманды и задавать значения их аргументов. Макросы — достаточно полезный и мощный инструмент. Они находят наилучшее применение в качестве средства обучения основам программирования и быстрого тестирования предварительных решений.

Запомните: прежде, чем передать заказчику готовое решение, желательно преобразовать все макросы в код с помощью методов объекта `DoCmd`. Код на языке VBA более управляем и прозрачен. Он позволяет применять средства анализа ошибок и динамического изменения данных.

Макросы — это полезное вспомогательное орудие. Но вы ведь планируете научиться программировать по-настоящему, не так ли? Что ж, тогда вперед!

## Вопросы и ответы

**Вопрос.** В каких случаях есть смысл обращаться именно к макросам?

**Ответ.** Макросы — это неплохой выбор, если вам необходимо решить небольшую задачу, не требующую написания отдельной программы на языке VBA.

**Вопрос.** Существуют ли операции, которые можно выполнить с помощью макросов, но нельзя — посредством программного кода?

**Ответ.** Нет. Макросы полезны и достаточно мощны, но программа на языке VBA способна на большее. Технология макропрограммирования довольно стара, и не исключено, что со временем она просто исчезнет с небосклона Microsoft Office.

**Вопрос.** Существуют ли другие объекты, подобные `DoCmd`, которые целесообразно применять в качестве альтернативы макросам?

**Ответ.** Безусловно. VBA — это объектно-ориентированный язык программирования, и в его арсенале имеется множество любопытных и мощных классов. Вы наверняка получите немало удовольствия от исследования возможностей объектов `Application` и `CurrentDb`.

**Вопрос.** Каким образом можно изучить все те инструменты и средства, которые Access предлагает мне как программисту?

**Ответ.** Сложный вопрос. Средства программирования для Access весьма обширны и многообразны, поэтому о скорой победе говорить не приходится. Книги, подобные нашей, практика, общение с коллегами, файлы оперативной справочной системы, информационные хранилища Internet — вот ваши лучшие советчики на трудном пути профессионального взросления.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Какую команду следует применять для импорта данных из таблицы Excel?
2. Как создается спецификация импорта данных?
3. Какая команда использует спецификацию импорта данных?
4. Как называется объект, применяемый для программирования макрокоманд в коде?
5. Могут ли объекты, подобные DoCmd, применяться в программах, ориентированных на использование с другими приложениями Office?

## Упражнения

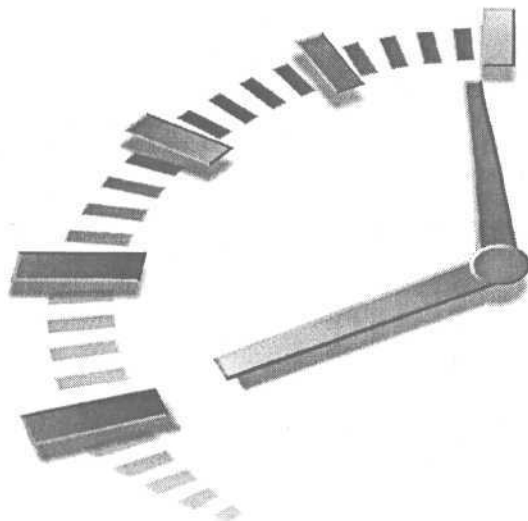
1. Создайте макрос для резервного копирования всех таблиц базы данных.
2. Напишите фрагмент кода, позволяющего пользователю ввести наименование файла, данные из которого предполагается импортировать.
3. Напишите код, аналогичный описанному выше макросу создания резервной копии.





## 10-й час

### Как использовать готовые решения



Задача создания программы для среды Access предполагает использование языка Microsoft Visual Basic for Applications (VBA). VBA поддерживается всеми приложениями из состава Microsoft Office XP. Если ранее вы уже работали с языком программирования Visual Basic, проблема освоения VBA не станет для вас серьезным препятствием. Наоборот, если вы уже знакомы с VBA (надеюсь, теперь можно об этом говорить), вам легко будет совладать и с Visual Basic.

Первым прототипом VBA послужил язык программирования PDP-11 B.A.S.I.C., разработанный Биллом Гейтсом (Bill Gates) и Полем Алленом (Paul Allen) и перенесенный в начале 70-х на компьютерную платформу MIPS Altair. Затем были ROM BASIC, GW-BASIC, BASIC for DOS, первая версия Visual Basic for DOS и Visual Basic for Windows. Сейчас существует уже шестая версия VB for Windows. VBA можно считать наследником и преемником всех упомянутых реализаций и версий языка BASIC.

Возможно, благодаря длительной и неоднозначной истории своего развития VBA обладает теперь мощными, выразительными и простыми для изучения средствами, особенно в сравнении с такими сложными языками программирования, как Ассемблер и C++. Встроенные библиотеки VBA содержат достаточно большой объем кода, который всегда доступен при реализации практических проблем — к счастью, вам не придется заниматься решением многих частных задач, поскольку все это уже сделано.

Основные темы занятия.

- Функции для работы со строками.
- Форматирование данных.
- Операции файлового ввода-вывода.

## Функции для работы со строками

В определенном смысле можно говорить, что все данные, с которыми работает программа, хранятся в виде наборов символов. Часто, например, оказывается полезным преобразование чисел в *строки* символов. Обработка символьных последовательностей (строк) одна из наиболее распространенных операций.

Данные, вводимые пользователем в интерактивном режиме, обычно представлены в виде строк. В ваше распоряжение предоставлена весьма обширная библиотека предопределенных функций и процедур, предназначенных для обработки строк. Впрочем, несмотря на столь впечатляющие (поверьте мне) возможности, конкретные задачи могут потребовать от вас реализации новых дополнительных средств.

В этом разделе будет рассказано, как пользоваться существующими функциями обработки строк и на их основе создавать новые.

## Взаимное преобразование строк и чисел

Синтаксис

Функции `Str` и `Val` предназначены для преобразования чисел в строки и строк в числа соответственно. Так выглядит синтаксис функции `Str`:

`Str( Число )`

Аргумент `Число` — это любое числовое литеральное значение или выражение. В качестве параметра функция `Str` принимает число, а возвращает строку цифр и символов, составляющих это число.

Синтаксис

Функция `Val` выполняет обратную операцию. `Val` получает строку числовых символов, а возвращает числовое значение. Синтаксис функции `Val` приведен ниже.

`Val( Строка )`

Аргумент `Строка` представляет собой последовательность символов, допустимых в записи числа. Вот несколько примеров:

```
"1234"  
"-54321"  
"6.999"  
"-0.78"  
"10e34"
```

Строчная `e` или прописная `E` служат для представления чисел в экспоненциальной нотации. Все другие символы считаются недопустимыми и игнорируются. Так, например, результатом вычисления выражения `Val( "1313 Etc" )` будет число 1313. Листинг 10.1 содержит несколько примеров использования функций `str` и `Val`.

Листинг 10.1. Примеры использования функций `Str` и `Val`

```
1 Sub StrAndVal( )  
2   Dim L As Long  
3   Dim D As Double  
4   D = Val( "10e34" )  
5   Debug.Print D  
6   L = Val( "3i" )  
7   Debug.Print L  
8   Dim S As String  
9   Dim T As String  
10: S = Str( D )  
11: Debug.Print S  
12: T = Str( L )  
13: Debug.Print T  
14: Debug.Print TypeName( T )  
15: Debug.Print T  
16: End Sub
```

Рассмотренные примеры преобразований очевидны. Команда в строке 14 с помощью функции `TypeName` осуществляет в окне Immediate вывод строки, содержащей наименование типа переменной `T`. (Напомним, что команда `Debug.Print` отображает содержимое своего аргумента-выражения в окне Immediate.) Результат вычисления процедуры `StrAndVal` представлен на рис. 10.1. Обратите внимание: вместо ожидаемого итога выполнения строки 4 (`10e34`) на экране отображается равноценное, но упрощенное число `1e+35`, а команда строки 7 выводит число 3, поскольку символ `i` при преобразовании отбрасывается.

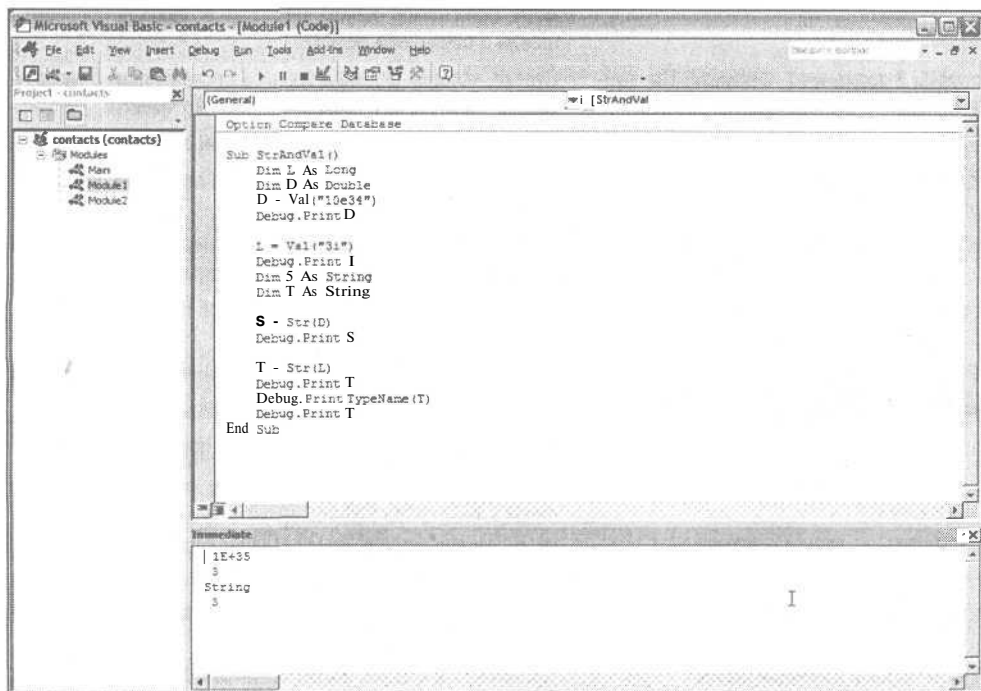


Рис. 10.1. Так выглядят результаты преобразований с помощью функций `Val` и `Str` в окне Immediate

## Строковые функции двойного назначения

В табл. 10.1 перечислены функции, которые реализованы в двух версиях. Если имя функции завершается символом доллара (\$), она возвращает значение типа `String`; если нет — результатом будет величина `Variant`.

Таблица 10.1. Строковые функции двойного назначения

Функция	Возвращаемое значение и описание
<code>Chr\$( КодСимвола )</code>	Символ
<code>ChrB\$( КодСимвола )</code>	Байт
<code>Command\$</code>	Строка, содержащая параметры командной строки

Функция	Возвращаемое значение и описание
CurDir\$(( Диск ))	Наименование текущего каталога на указанном диске. Параметр, указывающий букву дисковода, необязателен
Date\$	Строка, содержащая текущую дату
Dir\$( МаскаПоиска [, АтрибутыФайла] )	Наименование первого найденного файла, удовлетворяющего заданным критериям поиска
Error\$( НомерОшибки )	Сообщение об ошибке, соответствующее указанному номеру. Если номер ошибки не задан, возвращается сообщение о последней ошибке
Format\$( Выражение [, Формат [, Первый-ДеньНедели [, ПерваяНеделяГода]] )	Строка, которая форматируется в соответствии с инструкциями, заданными в необязательных аргументах
Hex\$( Число)	Заданное число в шестнадцатеричной системе счисления
Input\$( Число, [#]НомерФайла )	Строка, содержащая заданное число символов, которая считывается из указанного файла
InputB\$( ЧислоБайтов, [#]НомерФайла )	Строка символов, содержащая заданное число байтов, которая считывается из указанного файла
Lcase\$( Строка )	Строка, алфавитные символы которой преобразованы в нижний регистр
Left\$( Строка, ЧислоСимволов )	Левая часть строки, содержащая указанное число символов
LeftB\$( Строка, ЧислоБайтов )	Левая часть строки, содержащая указанное число байтов
Ltrim\$( Строка )	Строка, в которой удалены ведущие пробелы
Mid\$( Строка, НачальнаяПозиция [, ЧислоСимволов] )	Часть строки, содержащая указанное число символов, начиная с заданной позиции
MidB\$( Строка, НачальнаяПозиция [, ЧислоБайтов] )	Часть строки, содержащая указанное число байтов, начиная с заданной позиции
Oct\$( Число )	Строка, содержащая восьмеричное представление заданного десятичного числа
Right\$( Строка, ЧислоСимволов )	Правая часть строки, содержащая указанное число символов
RightB\$( Строка, ЧислоБайтов )	Правая часть строки, содержащая указанное число байтов
Rtrim\$( Строка )	Строка, в которой удалены хвостовые пробелы
Space\$( ЧислоПробелов )	Строка пробелов заданной длины
Str\$( Число )	Строка, содержащая символьное представление заданного числа
String\$( ЧислоСимволов, Символ )	Строка указанных символов заданной длины
Time\$	Строка, содержащая текущее время
Trim\$( Строка )	Строка, в которой удалены ведущие и хвостовые пробелы
Ucase\$( Строка )	Строка, алфавитные символы которой преобразованы в верхний регистр

Например, вызов Chr ( 65 ) даст в итоге значение Variant. Если же обратиться к функции Chr\$ ( 65 ), будет возвращена строка (String), хотя оба результата равны символу А.

Значения типа Variant более гибкие и управляемые, но их использование связано с издержками в виде дополнительного расхода памяти и процессорного времени, поэтому применять их следует осмысленно. Если определенный тип значения известен заранее, оговаривайте его явно.

## Функции преобразования типов данных

Функций преобразования типов данных настолько много, что подробно описать их в этой книге не удастся. Исчерпывающую информацию по данному вопросу можно найти в справочной системе. Существует множество функций, преобразующих строки в числовые выражения определенных типов данных. Все они имеют один вид: в функцию передается единственный аргумент — строка или числовое выражение, а возвращаемый результат представляет собой числовое значение нужного типа данных.

Ниже приведены некоторые из подобных функций.

- CBool преобразует выражение в значение типа Boolean.
- CByte преобразует выражение в значение типа Byte.
- CCur преобразует выражение в значение типа Currency.
- CDate преобразует выражение в значение типа Date.
- CDBl преобразует выражение в значение типа Double.
- CDec преобразует выражение в значение типа Decimal.
- CInt преобразует выражение в значение типа integer.
- CLng преобразует выражение в значение типа Long.
- CSng преобразует выражение в значение типа single.
- CStr преобразует выражение в значение типа String.
- CVar преобразует выражение в значение типа Variant.



Нельзя объявить переменную типа данных Decimal. Для хранения данных этого типа необходимо объявить переменную типа Variant.

Чтобы преобразовать строку "False" в переменную типа Boolean, используйте функцию CBool.

```
Dim B As Boolean  
B = CBool("False")
```

Преобразовать строку в значение типа Decimal можно, объявив переменную типа Variant и использовав функцию CDec.

```
Dim S As String  
S=1234567890123456789  
Dim D As Variant  
D = CDec(S)  
D = D * 10000  
MsgBox D
```

Функции преобразования активно применяются программистами, поскольку часто возникает необходимость предоставлять данные в виде строки (чтобы пользователь мог их отредактировать), а затем обновленные данные вновь преобразовывать в исходный тип для выполнения вычислений.

# Поиск строк



В составе VBA имеется функция `InStr`, предназначенная для поиска последовательностей символов внутри заданной строки. Подобная задача возникает на практике достаточно часто. Синтаксис обращения к функции `InStr` таков:

```
InStr( [НачальнаяПозиция,] ГдеИскать, ЧтоИскать [, _  
ОпцияСравнения] )
```

**НачальнаяПозиция** — необязательный аргумент, указывающий номер символа, с которого следует начинать поиск. Этот параметр полезен в том случае, когда искомая последовательность символов может в строке повторяться. Аргумент **ГдеИскать** задает строку, в которой будет выполняться поиск, а **ЧтоИскать** — искомую подстроку. В качестве параметра **ОпцияСравнения** может задаваться одно из значений предопределенного перечислимого типа — `vbUseCompareOptions`, `vbBinaryCompare`, `vbTextCompare` или `vbDatabaseCompare`.

Значение `vbUseCompareOptions` указывает функции на необходимость использовать опцию сравнения, установленную командой `Option Compare`. Режим `vbBinaryCompare` позволяет осуществлять поиск с учетом регистра символов, а `vbTextCompare` — без него.

Если функция `InStr` возвращает нуль, это значит, что искомая последовательность символов не найдена. Любое значение возврата, большее нуля, служит сигналом успешного завершения поиска и указывает на положение начального символа найденной последовательности в исходной строке. Зная позицию подстроки, нетрудно написать цикл для поиска всех ее вхождений.

Листинг 10.2 демонстрирует отличия между режимами поиска с учетом и без учета регистра символов, а листинг 10.3 содержит пример процедуры, выполняющей поиск всех вхождений указанной символьной последовательности в заданную "большую" строку.

## Листинг 10.2. Поиск строк с учетом и без учета регистра алфавитных символов

```
1: Sub Search( )  
2:   MsgBox InStr( 1, "Здравствуй, мир!", "Мир", vbBinaryCompare )  
3:   MsgBox InStr( 1, "Здравствуй, мир!", "Мир", vbTextCompare )  
4: End Sub
```

### Анализ

Процедура `MsgBox`, которая вызывается в строке 2, отображает окно сообщения, содержащее результат 0: поиск слова "Мир" в строке "Здравствуй, мир" завершился неудачей, поскольку с учетом регистра символов (опция `vbBinaryCompare`) слова "Мир" и "мир" считаются различными. Напротив, следующее окно сообщения высветит результат 13: без учета регистра (режим `vbTextCompare`) слова становятся равноправными.

## Листинг 10.3. Поиск всех вхождений подстроки

```
1: Sub FindAll( )  
2:   Const SEARCH_ME = "На дворе — трава, на траве — дрова"  
3:   Const SEARCH_FOR = "трав"  
4:   P = 0  
5:   Do While (1=1)
```

```

6:      P = InStr( P + 1, SEARCH_ME, SEARCH_FOR, vbTextCompare )
7:      If (P = 0) Then Exit Do
8:      MsgBox "" & SEARCH_FOR & "" найдено на позиции=" & P
9:  Loop
10: End Sub

```



В порядке напоминания отметим: строка 5 содержит заголовок бесконечного цикла, а в строке 7 осуществляется принудительный выход из него в случае отрицательного результата поиска (т.е. цикл прерывается).

#### Анализ

В цикле производится поиск без учета регистра алфавитных символов (в качестве опции сравнения выбрано значение `vbTextCompare`). Строка "трав" будет найдена дважды — на позициях 10 и 20, и эти числа будут отображены в окнах сообщений, открываемых в строке 8.

## Динамическое заполнение строк

Функции `String` и `Space` позволяют динамически заполнять строковые переменные и при необходимости очищать их. Функция `string` заносит в переменную заданный символ и повторяет операцию указанное количество раз. `Space` можно считать частным случаем функции `String` — она присваивает переменной строку пробелов определенной длины.

Если необходимо заполнить переменную набором одинаковых произвольных символов, применяйте функцию `String`. Чтобы присвоить переменной строку пробелов длины `n`, можно воспользоваться одним из вариантов — `String( n, " " )` либо `Space ( n )` — с одинаковым результатом.

## Форматирование данных

Синтаксис

Как самые простые, так и наиболее сложные операции форматирования могут быть легко выполнены с использованием функции `Format` и набора специальных символов форматирования — например @, &, <, или >. Синтаксис функции `Format` таков:

```
Format( Строка, СтрокаФормата )
```

Функция `Format` требует задания переменной или литерального значения типа `string` и строки, описывающей формат преобразования. Строка формата может содержать любые литеральные символы в сочетании со специальными знаками. В табл. 10.2 перечислены наиболее употребительные специальные символы форматирования и приведено их описание.

Символы @ и & играют роль шаблонов для представления любого единственного символа. Если строка формата содержит знак @ или &, он заменяется символом преобразуемой строки, расположенным в той же позиции. Действие команды < аналогично применению функции `Lcase$`, а команда > равносильна функции `Ucase$`. Листинг 10.4 представляет несколько простых примеров форматирования строк; их легко проверить с помощью средств окна `Immediate`.



Таблица 10.2. Специальные символы форматирования

Символ	Описание
@	Шаблон единственного символа
&	Шаблон единственного символа
<	Преобразование к нижнему регистру
>	Преобразование к верхнему регистру
!	Преобразование слева направо

Листинг 10.4. Примеры форматирования строк

```

1: Sub FormatDemo ( )
2:   MsgBox Ucase$ ( "test" ) = Format ( "test", ">" )
3:   MsgBox Lcase$ ( "TEST" ) = Format ( "TEST", "<" )
4:   MsgBox Format ( "5175551212", "(@@@) @@@-@@@@" )
5: End Sub

```

**Анализ**

При выполнении строки 2 откроется окно сообщения, содержащее слово True, поскольку левая и правая части логического выражения сравнения аналогичны. Окно, вызываемое в строке 3, также будет отображать слово True. Результат выполнения строки 4 — выражение "(517) 555-1212". Безусловно, это самые простые примеры. Хитроумные комбинации лите- ральных символов и специальных знаков форматирования могут привести к весьма любопытным результатам. Ваше творчество ограничено только способностью к воображению и, естественно, требованиями конкретной задачи. Единственное общее полезное правило состоит в том, что заниматься форматированием имеет смысл после того, как сделано самое главное — программа правильно решает задачу.

## Функции Date и Time

Существует две разновидности функций Date и Time. Date, например, возвращает текущую дату в виде значения типа Date, а функция Date\$ служит для представления даты в виде символьной строки. Функции Time и Time\$ возвращают значения текущего времени. Величины типа Date хранятся в виде чисел двойной точности.

Если в окне Immediate редактора Microsoft Visual Basic ввести команду Print Date, то результатом будет строка, содержащая значение текущей системной даты (например, введя эту команду 28 июня 2001 г., вы получите 28.06.2001). Команда Print Time выводит на экран текущее значение времени — скажем, 15:55:51. Команда Print Now позволяет увидеть одновременно и дату, и время (28.06.2001 15:55:51).

Дата и время сохраняются в виде единого числа типа Double: дате соответствует целая часть числа, а времени — дробная. Чтобы просмотреть внутреннее представление даты/времени, введите в окне Immediate команду Print Now / 1, под которой подразумевается неявное преобразование величины в число двойной точности. Дате 28.06.2001 и времени 15:55:51 соответствует число 37019.0441435185. (Целая часть равна количеству дней, прошедших с 1 января 1900 года.)

Следующий код позволяет выделить значение даты и значение времени из внутреннего представления даты и времени.

```
Dim DatePart As Double
Dim TimePart As Double
DatePart = Fix(Now)
TimePart = Now - Fix(Now)
```

В 22.22, 7 марта 2001 года значение DatePart составляет 36957, а значение TimePart — 0.933206. Полное значение — это число дней, прошедших с 1 января 1900 года.

Отрицательное значение даты свидетельствует о том, что используется дата до 1 января 1900 года. Время представляется в виде дробной части от 24 часов. Дробная часть, равная 0, соответствует полуночи, а 0,5 — полудню. 0,75 — это три четверти суток, или 18.00.

Дата и время, вероятно, второй по значимости (после строк) тип данных, используемый в приложениях баз данных. И вам наверняка придется его применять — и в форматах даты/времени, и, реже, — в виде внутренних числовых представлений.

## Операции файлового ввода-вывода

Вся информация хранится в файлах двоичного и текстового форматов. Пакетные и INI-файлы — это простейшие примеры файлов текстового формата. Всем вам знакомы также файлы формата HTML, документы текстовых процессоров, электронных таблиц и графических редакторов. VBA предлагает ряд полезных средств управления файловыми данными.

Для выполнения операций файлового ввода-вывода в программах на языке VBA применяются команды и функции Open, Close, FreeFile, Line Input, Input, Print, Write, Get и Put. Open осуществляет открытие файла, обеспечивая заданный уровень доступа — для чтения, записи или для чтения/записи. Close выполняет закрытие файла. Для обращения к открытым файлам применяются специальные дескрипторы-номера. Номеру открытого файла соответствует определенный ресурс операционной системы, поэтому во избежание его истощения каждый открытый файл после завершения использования должен обязательно закрываться. Функция FreeFile гарантирует, что выбранный номер файла не применяется другим процессом. Line Input обеспечивает возможность считывания строки текстового файла. Команды Write и Print позволяют записывать информацию в текстовый файл, причем Write включает очередную порцию данных в двойные кавычки, а Print — нет. Get и Put предназначены, соответственно, для выполнения операций чтения и записи двоичных данных.

Мы рассмотрим названные команды и функции по группам: вначале Open и Close совместно с FreeFile, затем Line Input, Input, Print, Write И, наконец, Get и Put.

## Команды Open и Close

Команда Open описывается довольно сложным синтаксическим выражением — о ней речь пойдет ниже. Но зато Close довольно проста:

```
Close #НомерФайла
```

НомерФайла — это переменная для хранения числа двойной точности, которое возвращается функцией FreeFile, получающей свободный ресурс от операционной системы.



Одно из правил хорошего тона в программировании гласит: следует использовать ресурсы в том же контексте, в котором они объявлены (открыты). Скажем, открыв файл внутри некоторой функции, вы должны закрыть его в этой же функции. Такой подход уменьшает вероятность повреждения файла, если тот случайно останется открытым.

Если файл все-таки должен оставаться открытым во время выполнения целого ряда функций, удачным решением будет расширение контекста посредством включения операций открытия/закрытия в "пределы" одного класса. В этом случае перед удалением объекта класса последний сам должен "позаботиться" о закрытии всех файлов (подробнее см. главу "21-й час. Основы программирования классов").

Команда открытия файлов, `Open`, более сложна. С ее помощью можно открывать как текстовые, так и двоичные файлы, причем в нескольких режимах — для чтения, записи и чтения/записи. Различие между процессами обработки текстовых и двоичных данных состоит в том, что строки текста отделяются одна от другой парой символов — *возврат каретки - перевод строки*. В двоичном режиме объем блока данных диктуется вашими потребностями и может быть произвольным. Текстовый файл легко открывается и отображается с помощью любой программы-редактора, но двоичный файл в режиме просмотра будет выглядеть, как "каша" из беспорядочно расположенных символов. Приведем синтаксис команды `Open`:

```
Open ИмяФайла [For Режим] [Access Доступ] [Блокировка] As
[#]НомерФайла [Len=ДлинаЗаписи]
```

**ИмяФайла** — это символьная переменная или константа, содержащая имя файла (и, возможно, путь к нему). Аргумент **Режим** указывает, будет ли файл открыт для записи, чтения или одновременно записи и чтения, а также предписывает, каким образом трактовать данные — как текстовые или двоичные. В качестве значения **Режим** может применяться одно из служебных слов — `Append`, `Binary`, `Input`, `Output` или `Random`. `Append` означает, что файл открывается в режиме записи и указатель положения устанавливается в конец файла. Слово `Binary` достаточно красноречиво — файл открывается как бинарный (двоичный). `input` соответствует режиму чтения текстового файла, `Output` предполагает возможность записи в текстовый файл, а `Random` позволяет совмещать операции чтения и записи текста без закрытия файла и его повторного открытия. Если параметр **Режим** не задан, по умолчанию выбирается значение `Random`. Аргумент **Доступ** задается только в том случае, если режим отмечен служебным словом `Binary`, и допускает значения `Read`, `Write` или `Read Write`, которые не нуждаются в дополнительных пояснениях. Необязательный параметр **Блокировка** позволяет указать один из признаков блокировки для обеспечения возможности работы с файлом в многопользовательском режиме — `Shared`, `Lock Read`, `Lock Write` и `Lock Read Write`.

**НомерФайла** — это целое число в интервале от 1 до 511. Наиболее удачная тактика связана с вызовом функции `FreeFile`, которая запрашивает у операционной системы свободный ресурс и возвращает его в виде числа. После этого число-номер следует сразу же использовать в команде открытия файла. Если на протяжении определенного времени номер файла остается невостребованным, многопоточная операционная система `Windows` может передать ресурс другому процессу. Параметр **ДлинаЗаписи** используется совместно с режимом `Binary` и указывает величину порции считываемых

или записываемых данных. Если вы будете работать только с одним символом, укажите `Len=1`. Довольно полезной оказывается конструкция `Len=Size(ТипДанных)`, позволяющая определить верную величину записи в конкретной ситуации. Листинг 10.5 предлагает ряд примеров открытия и закрытия текстовых и бинарных файлов.

#### Листинг 10.5. Примеры открытия/закрытия текстовых и бинарных файлов

```
1: Type Email
2:   Name As String
3:   Email As String
4: End Type
5:
6: Sub OpenAndClose( )
7:   Dim Handle As Double
8:   Handle = FreeFile
9:   Open "test.txt" For Output As #Handle
10:  Close #Handle
11:
12:  Handle = FreeFile
13:  Open "test.bin" For Binary Access Write As #Handle
14:  Dim Mail As Email
15:
16:  Do While (1 = 1)
17:    Mail.Name = InputBox ( "Введите имя (Q=Выход):", _
                           "Добавление имени", "" )
18:    If (Mail.Name = "Q") Then Exit Do
19:    Mail.Email = InputBox ( "Введите адрес email:", _
                            "Добавление адреса", "" )
20:    Put #1, , Mail
21:  Loop
22:  Close #Handle
23: End Sub
```

#### Анализ

В строках 1–4 определяется новый тип, `Email`. Подробнее о пользовательских типах см. главу “11-й час. От сложного к простому: создание собственных типов данных”. В строке 7 объявляется переменная `Handle` типа `Double`, которая будет использоваться в качестве номера открытого файла. Строка 8 содержит обращение к функции `FreeFile`, получающей от операционной системы свободный файловый ресурс. В строке 9 выполняется команда открытия файла `test.txt` в режиме записи текста, а в строке 10 файл закрывается. Хотя никакой практической ценности подобный код не представляет, он служит для демонстрации приемов открытия и закрытия текстовых файлов.

Строка 12 содержит повторный вызов функции `FreeFile` и оператор присваивания возвращенного ею результата переменной `Handle`. В строке 13 выполняется операция открытия файла `test.bin` в режиме записи двоичных данных. В строке 14 находится объявление объекта `Mail` ранее построенного пользовательского типа `Email`. Строки 16–21 задают бесконечный цикл вида `Do While ... Loop`, предназначенный для интерактивного (с помощью функции `InputBox`) ввода информации

об именах и адресах электронной почты пользователей. Строка 20 демонстрирует пример использования команды *Put* для записи полученных данных в двоичный файл. Команда *Put* более подробно рассмотрена в следующем разделе. Цикл прерывается при вводе символа *Q* в качестве имени пользователя. В строке 22 (мы не забыли!) файл закрывается.

Если попытаться открыть двоичный файл с помощью обычного текстового редактора, помимо значимых данных, мы, вероятно, увидим нагромождение служебных символов. Картина становится более прозрачной, если обратиться к средствам старой программы *debug.exe*.

Двоичные файлы полезны для хранения блоков структурированной информации, которые, скажем, описываются определенными пользователем типами данных, а текстовые файлы удобны в качестве хранилища произвольных текстов. Например, глава, которую вы читаете, записана в виде текстового документа *Word*, а вот адресную книгу, содержащую сведения об именах абонентов, номерах телефонов и т.п., вероятно, в каких-то случаях целесообразно сохранять в виде бинарного файла определенной структуры.

## Чтение и запись текстовых данных

Команды *Line Input*, *Input*, *Print* и *Write* предназначены для чтения и записи текстовой информации. Выбор той или иной команды диктуется условиями конкретной задачи и структурой используемых данных. Команда *Line Input* применяется для построчного чтения, *input* — для чтения текста с разделителями, *Print* — для вывода порций текста, а *Write* — для записи фрагментов данных в двойных кавычках с разделителями. Ниже приведены синтаксические формулы, описывающие каждую из названных команд.

```
Line Input #НомерФайла, ПеременнаяТипаString
Input #НомерФайла, Переменная1 [, Переменная2, Переменная3, ...]
Print #НомерФайла, Переменная1 [, Переменная2, Переменная3, ...]
Write #НомерФайла, Переменная1 [, Переменная2, Переменная3, ...]
```

Команда *Line Input* требует задания номера открытого файла и символьной переменной для хранения считанной из файла строки данных. *Input* считывает из файла, который задан с помощью номера, поля данных, ограниченные разделителями, и присваивает их переменным из указанного списка. Номер файла, используемый командами *Line Input* и *input*, — это числовая переменная двойной точности, полученная от функции *FreeFile*. *Input* позволяет непосредственно считывать и присваивать переменным как строки, так и данные других типов. Команды *Print* и *Write* также пользуются номером файла, возвращаемым функцией *FreeFile*, и предоставляют возможность записи одного или нескольких значений, ограниченных разделителями, в текстовый файл. Результат работы команды *Write* (мы применим ее в строке 7 листинга 10.6) — это строка, которая состоит из полей, заключенных в двойные кавычки и разделенных запятыми:

```
"Robert Golieb", "RobertGolieb@hotmail.com"
```

Те же значения, записанные с помощью команды *Print*, в файле будут выглядеть так:

```
Robert Golieb RobertGolieb@hotmail.com
```

Обратите внимание на отсутствие каких бы то ни было разделителей, кроме пробелов.

Применяя команды `Write` и `Input`, вы сможете легко создать простую базу данных в виде текстового файла, а также приложение для ее просмотра. Несмотря на то, что такое решение трудно назвать эффективным, в некоторых случаях оно может оказаться оправданным. Подобная информация может поступать из различных источников — Web-страниц, файлов конфигурации или баз данных старых форматов. Листинг 10.6 демонстрирует пример функции для создания и просмотра базы данных, содержащей сведения об именах пользователей и их электронных адресах.

#### Листинг 10.6. Пример использования команд `Input` и `Write`

```
1: Sub UsingInputAndWrite( )
2:
3:   Dim Handle As Double
4:   Handle = FreeFile
5:   Open "CommaDelimited.Txt" For Output As #Handle
6:
7:   Write #Handle, "Robert Golieb", "RobertGolieb@hotmail.com"
8:
9:   Close #Handle
10:
11:  Handle = FreeFile
12:  Open "CommaDelimited.Txt" For Input As #Handle
13:
14:  Dim Name, Email As String
15:
16:  Input #1, Name, Email
17:
18:  MsgBox "Корреспондент: " & Name & ", " & Email
19:
20:  Close #Handle
21: End Sub
```



Ключ к созданию высококлассного кода — последовательность в действиях. В качестве номера открытого файла мы, например, неоднократно использовали переменную `Handle` типа `Double`, получающую значение от функции `FreeFile`, — нам не приходилось всякий раз выдумывать новые имена или, хуже того, пользоваться литеральными значениями. Чем большее количество элементов кода подвергается разумной унификации, тем быстрее достигается качественный результат.



Открытые файлы — это ресурсы операционной системы. Поскольку они ограничены, программный код должен "заботиться" о том, чтобы все открытые файлы в конце концов обязательно закрывались. Если в промежутке между операциями открытия и закрытия файла происходит ошибка, файл может остаться открытым, что вполне может привести к повреждению данных. В главе "18-й час. Обработка ошибок во время выполнения программы" речь пойдет о создании кода, позволяющего избегать подобных потенциальных опасностей.

В строке 3 листинга 10.6 содержится объявление переменной `Handle` типа `Double`, предназначенной для хранения номера открытого файла. Этот файл запрашивается у операционной системы с помощью функции `FreeFile`, которая вызывается в строке 4. Полученный ресурс необходимо сразу же использовать, чтобы многопоточная операционная система не передала его другой программе или иному процессу той же программы. В строке 5 выполняется открытие текстового файла для записи, а строка 7 содержит инструкцию сохранения в файле двух литеральных символьных значений. В строке 9 файл закрывается. В строке 11 с помощью функции `FreeFile` вновь запрашивается свободный номер файла — обратите внимание, мы не стали полагаться на ранее используемый ресурс, а затребовали новый. Строка 12 содержит команду открытия того же текстового файла, но на сей раз в режиме чтения. В строке 14 объявляются две текстовые переменные — ниже, в строке 16, они используются для хранения результатов чтения данных из файла. В строке 18 полученная информация выводится на экран с помощью процедуры `MsgBox`, и в строке 20 файл закрывается.

Конечно, рассмотренный пример является достаточно условным. Чтобы придать ему практическую ценность, необходимо вынести операции записи и чтения данных в отдельные функции и создать приемлемый интерфейс пользователя. О том, как строить понятные пользовательские интерфейсы, будет рассказано в главе "19-й час. Создание экранных форм".

## Чтение и запись двоичных данных

Двоичные (бинарные) файлы находят самое широкое применение и позволяют сохранять данные в виде записей. Указав размер каждой записи, вы сможете последовательно сохранять и считывать их с помощью команд `Put` и `Get`. Строка 23 листинга 10.5 содержит пример использования инструкции `Put`. Команда `Get` действует тем же образом, но применяется для работы с файлами, открытыми в режиме `Read` или `Read Write`. Синтаксис команд `Put` и `Get` описан ниже.



```
Put #Номер файла, [Номер записи], Переменная
Get #НомерФайла, [НомерЗаписи], Переменная
```

Команда `Put` требует задания номера файла (предварительно открытого с помощью `Open`), необязательного номера записи и идентификатора переменной. В качестве переменной может выступать объект стандартного или пользовательского типа данных — такого, как, например `Email` (мы определяли этот тип в тексте листинга 10.5). Аргумент `НомерЗаписи` задается в том случае, если вам известен индекс записи, которую необходимо сохранить. Если команда `Put` выполняется без указания номера записи, данные сохраняются последовательно. Если вы ссылаетесь на номера записей, то должны принять дополнительные меры по обработке возможных ошибок, чтобы избежать опасности выхода за маркер конца файла. Команда `Get` считывает двоичные данные в простые переменные или объекты более сложных типов. В отношении использования аргумента `НомерЗаписи` действуют те же правила, которые были рассмотрены выше. Прежде чем приступить к явному заданию номера записи для сохранения или считывания, обратитесь к главе 18. Команда `Get` предъявляет те же требования к параметру `Переменная`,

что и Put. Единственное различие команд состоит в том, что для использования Get файл должен быть открыт в режиме Read или Read Write.

С помощью рассмотренной в этом разделе пары команд вы можете создавать собственные схемы хранения и считывания данных. Впрочем, настоятельно рекомендуем этого не делать. Применяйте средства прямого доступа к файловым данным только в тех случаях, когда нет других альтернатив. В большинстве ситуаций достаточно обратиться к современным мощным, но простым инструментам управления базами данных — и вы быстрее достигнете цели.

## Функции интерактивного ввода данных

Если задача, над которой вы трудитесь, предполагает активное взаимодействие с пользователем, целесообразно прибегнуть к средствам создания форм. Вопросы проектирования и построения форм подробно освещены в главе 19. Но в некоторых случаях, когда программе необходимо выдавать незамысловатые сообщения и оперировать простыми элементами данных, введенных пользователем, имеет смысл применять функции MsgBox (для отображения сообщений) и InputBox (для ввода информации).

Синтаксис вызова функции MsgBox таков:

**Синтаксис**

```
MsgBox( Сообщение [, Кнопки] [, Заголовок] [, ФайлСправки,
НомерТемы] )
```

Вызов функции MsgBox приводит к открытию модального окна. Термин *модальный* означает, что пока такое окно открыто, все другие окна приложения остаются недоступными для пользователя. В качестве аргумента Сообщение может использоваться переменная, именованная константа или литеральное значение символьного типа, содержащее собственно текст сообщения. Параметр Кнопки — одно из значений предопределенного перечислимого типа, например vbOKOnly или vbOKCancel. Полный список допустимых значений параметра Кнопки легко найти в оперативной справочной системе, задав ключевое слово поиска *MsgBox*. Необязательная пара аргументов ФайлСправки и НомерТемы содержит имя HLP-файла справки и номер темы — число, которое было назначено определенной теме справки при создании файла. Если оба параметра заданы верно, нажатие клавиши <F1> после открытия окна сообщения приведет к загрузке приложения справки и отображению указанной темы.

**Синтаксис**

Функция InputBox служит для получения информации от пользователя. Мы неоднократно обращались к ней ранее, хотя не владели достаточной информацией о ее возможностях. Ознакомьтесь с полным описанием синтаксиса функции InputBox, приведенным ниже.

```
InputBox( Сообщение [, Заголовок] [, ЗначениеПоУмолчанию]
[, Абсцисса] [, Ордината] [, ФайлСправки, НомерТемы] )
```

Функция InputBox открывает модальное окно, содержащее две кнопки: OK и Cancel. После щелчка на кнопке OK программа получает строку данных, введенных пользователем в поле диалогового окна. Щелчок на кнопке Cancel приводит к возврату строки нулевой длины. Символьный аргумент Сообщение уведомляет пользователя о том, какого рода данные требуются программе. Необязательный параметр Заголовок содержит строку, которая будет отображаться в заголовке диалогового окна. Аргумент ЗначениеПоУмолчанию позволяет задать символьную строку, кото-



рую пользователь сможет сразу принять (**щелкнув** на кнопке ОК) или отредактировать. Факультативные параметры Абсцисса и Ордината дают возможность задать декартовы координаты левого верхнего угла диалогового окна на экране. Если эти параметры не указаны, окно располагается в центре экрана. О назначении аргументов ФайлСправки и НомерТемы говорилось ранее, когда речь шла о синтаксисе вызова функции MsgBox.

Функция MsgBox применяется для вывода окна, **содержащего** сообщение об ошибке, и о возникновении особых условий. Она также регламентирует дальнейшие действия, например:

```
MsgBox( "Ошибка деления на ноль — введенные данные неверны!",  
vbExclamation )
```

Функция InputBox ожидает реакции пользователя в виде введенного символьного значения:

```
Dim UserName As String  
UserName = InputBox( "Введите имя пользователя", "Имя", "Guest")
```

При выполнении указанного кода открывается диалоговое окно, озаглавленное строкой Имя, и пользователю предлагается возможность отредактировать строку Guest, заданную по умолчанию. После щелчка на кнопке ОК переменной UserName будет присвоено введенное значение; щелчок на кнопке Cancel приводит к очистке содержимого переменной UserName.

## Еще раз об оперативной справочной системе

В недрах библиотек VBA таятся сотни и сотни функций, процедур, команд и объектов со своими методами и свойствами. Чтобы найти и использовать их, необходимо сформулировать задачу, сведя ее описание к емкому существительному или глаголу. После этого следует обратиться за помощью к файлам оперативной справочной системы Microsoft Access и Visual Basic. Откройте окно Справка Microsoft Visual Basic, перейдите на вкладку Указатель (Index) и введите одно из ключевых слов.

До сих пор мы имели дело преимущественно с функциями и процедурами. Термины *метод* и *свойство* имеют отношение к проблематике объектно-ориентированного программирования. Методы — это функции и процедуры, а свойства — элементы данных, служащие частью определенного класса. Более подробные сведения об использовании классов и создании объектов вы узнаете из главы "21-й час. Основы программирования классов".

## Резюме

В арсенале программиста, выбравшего язык VBA, — сотни предопределенных функций и процедур. В вашем распоряжении также достаточно большой объем кода, когда-либо написанного на языках BASIC и Visual Basic. В ходе этого занятия были рассмотрены наиболее употребительные функции обработки строк, интерактивного ввода данных и выполнения файловых операций. Файлы оперативной справки VBA хранят много дополнительных сведений. Коммерческие продукты, разрабатываемые сторонними производителями, — еще один неисчерпаемый источник решений и технологий.

Теперь вы знаете, как манипулировать наиболее распространенными типами данных, обрабатывать информацию, вводимую пользователем, и сохранять ее в текстовых и двоичных файлах. Буквально еще несколько лет назад подобная технология работы была единственным доступным способом решения задач. Приобретенные навыки позволят вам справляться • со многими задачами — от самых простых до достаточно сложных.

На следующих занятиях вы научитесь применять полученные знания для решения задач управления базами данных, проектирования пользовательских интерфейсов и создания сложных типов данных. Все приемы программирования, которые будут рассмотрены позже, основываются на фундаменте, заложенном нашими совместными усилиями. Раздел вопросов и ответов прольет дополнительный свет на ряд тем, рассмотренных на этом и предыдущих занятиях.

## Вопросы и ответы

**Вопрос.** Каким образом можно решить задачу поиска и замены строк, чтобы ее реализация была схожа с той, которая предлагается, скажем, в Microsoft Word?

**Ответ.** Конечно, подобные функции вы вполне смогли бы реализовать самостоятельно. Впрочем, имеется достаточно образцов готового кода, и, вероятно, существуют решения аналогичных задач. Немного побродив по просторам Web, вы наверняка найдете все то, что ищете.

**Вопрос.** Можно ли, решая задачи в среде Access, использовать инструменты Word или Excel для управления данными?

**Ответ.** Да. Word, Excel и другие приложения из состава Microsoft Office XP обладают интерфейсами OLE Automation. Это означает, что из среды прикладных программ Access вы можете обращаться к этим приложениям и их службам. Глава "24-й час. Управление информацией о контактах Outlook" содержит примеры, касающиеся вопросов взаимодействия программ Access 2002 и Outlook 2002.

**Вопрос.** Имеет ли смысл постоянное использование текстовых или двоичных файлов для хранения данных?

**Ответ.** По большому счету, нет. Подчас это бывает удобно или необходимо, но вообще, любой код, предназначенный для манипуляций с данными, должен предполагать использование табличных структур. С этой точки зрения система Access весьма удобна, и если вы сделаете выбор в ее пользу, то многое приобретете и ничего не потеряете.

## Задания

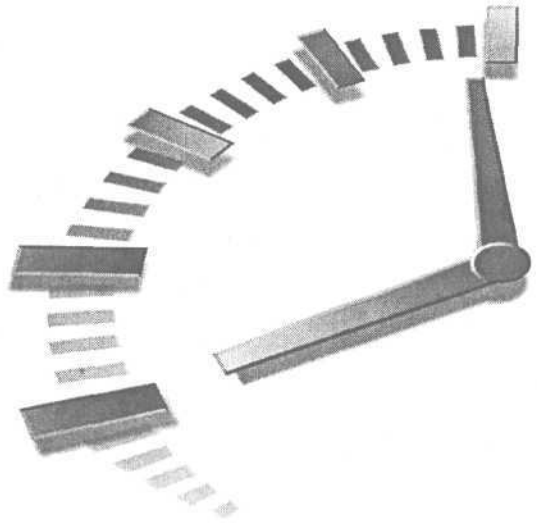
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Как называется функция, предназначенная для поиска подстрок?
2. В чем принципиальное отличие модальных окон от остальных?
3. Как должна выглядеть команда открытия текстового файла в режиме чтения?
4. Какую функцию можно использовать для интерактивного ввода данных?

## Упражнения

1. Напишите выражение для форматирования 9-значного почтового индекса.
2. Постройте процедуру, добавляющую указанную строку в заданный файл.
3. Создайте подпрограмму поиска заданного имени в двоичном файле, предполагая, что при сохранении данных использовался тип Email и файл содержит более одной записи.



## **Часть IV**

**Определение типов данных.**

**Использование массивов и коллекций**

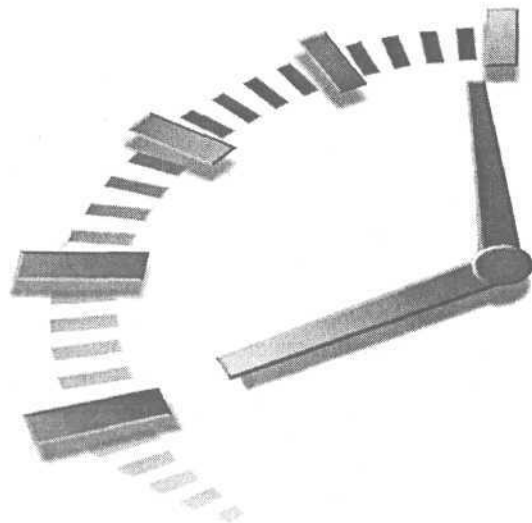
### **Темы занятий**

11-й час. От сложного к простому: создание собственных типов данных

12-й час. Управление данными переменного объема

13-й час. Коллекции данных





## 11-й час

# От сложного к простому: создание собственных типов данных

Пройдя долгий путь десяти уроков, вы заслужили право на отдых. Эта глава намного короче остальных по объему, но она важна по содержанию, поскольку служит прологом к дальнейшему изложению, посвященному непростым доктринам и темам программирования. В данной главе речь пойдет о типах данных, создаваемых пользователем-программистом.

За служебным словом *Tуре* (в других языках программирования употребляется термин *Record*) кроется концепция агрегации данных, имеющая глубокие исторические корни. Еще на заре информатизации первобытные программисты совершили открытие: во многих ситуациях удобнее воспринимать набор разрозненных элементов данных в виде единой целостной сущности. Хотя первая волна восторгов по поводу объектно-ориентированного подхода к программированию с шумом прокатилась еще в начале 90-х, эта технология до сих пор находится в центре внимания "компьютерного" сообщества. И ее реализация в рамках VBA — не исключение.

Типы данных, создаваемые пользователем, — первый уровень объектно-ориентированного подхода. При рассмотрении реальной задачи зачастую бывает весьма удобно создать новый тип данных, состоящий из нескольких других, более простых, и позволяющий облегчить восприятие проблемы, ее формализацию и решение. Пользовательский тип данных — это шаг к пониманию и использованию полноценных классов и их объектов.

В этой главе мы рассмотрим важные темы, связанные с агрегацией данных. Основные темы занятия.

- Как строить новые типы данных на основе существующих.
- Создание экземпляров пользовательских типов данных.
- Какие данные может содержать тип, определяемый пользователем.
- Перечислимые типы.

# О понятии агрегации

Агрегацию можно трактовать как способ, придуманный человеком для преодоления барьера сложности восприятия окружающего мира. Все мы, усвоив тысячелетний опыт наших предшественников, научились "втискивать" небывало сложные понятия в относительно простые образы.

Что вы можете сказать о любви? Да, это чертовски сложная штука! Говорите, что хотите — о полете бабочки над цветущим лугом, родных глазах со слезинками счастья, грубых биохимических реакций в организме, закоулках подсознания, в которых, согласно Фрейдю, скрываются низменные страсти, о сонетах Шекспира и лирике Байрона — все равно любовь остается абстракцией, а абстракции чудовищно сложны. Впрочем, вы и я способны совместить все мыслимые аспекты этого состояния человеческой души в одном представлении и назвать его простым словом — *любовь*. Теперь все понятно (нет — конечно, далеко не все, а может быть, непонятно ничего вовсе).

Технические идеи также бывают сложными. Я говорю "VBA", а подразумеваю "язык программирования для Windows, обладающий продолжительной историей развития и имеющий какое-то отношение к парню по имени Билл Гейтс". Все это труднопроизносимое многословие укладывается в три буквы — *V-B-A*. Дополнительное преимущество агрегации (пусть даже два человека по-разному истолковывают некое "агрегированное" понятие) состоит в том, что она обеспечивает отправную точку, которая окажется несомненно полезной в ходе дальнейших обсуждений и уточнений. Теперь даже сложная вещь может быть понята и адекватно воспринята, если растолковать подробно в письменном или устном виде.

Типы данных — это та начальная ступень, которую программисты, искавшие способы агрегации простых элементов данных в более сложные информационные единицы, просто не могли обойти. Языки — естественные и искусственные — и сейчас продолжают развиваться. Не секрет, что компьютер породил даже целый пласт новой жаргонной лексики.

Типы данных, определяемые пользователем, — это совсем не страшно. Так что давайте, не мудрствуя лукаво, приступим к делу.

## Объявление новых типов



В своей основной обязательной форме объявление типа требует двух строк кода. Первая строка содержит служебное слово `Type`, за которым следует уникальное название типа, а вторая состоит из пары слов — `End Type`. Синтаксис объявления типа, не содержащего членов, выглядит так:

```
Type ИмяТипа  
End Type
```



В типе, определенном пользователем, должен содержаться хотя бы один элемент.

Все слова в определении типа вводятся буквально, без кавычек и разделителей. Тип необходимо наделить уникальным именем. Правила именования типов обычны; в качестве имени может использоваться любая последовательность допустимых символов. Имя должно быть внятным и красноречивым, отражающим суть и назначение

данных. Объявление типа подчиняется и другим правилам. Все объявления типов должны располагаться в начале модуля, вне каких бы то ни было функций или процедур, и содержать, по меньшей мере, по одному члену. Размещение объявлений типов внутри функций (процедур) не допускается.

#### Новый термин

*Член типа, определенного пользователем, — это переменная, служащая частью конструкции типа и указываемая в пределах между первой и последней строками объявления типа.*

Если тип объявлен верно, он становится частью всплывающего в окне модуля списка доступных типов (рис. 11.1). Листинг 11.1 демонстрирует пример объявления простого типа, содержащего единственный член — целочисленную переменную.

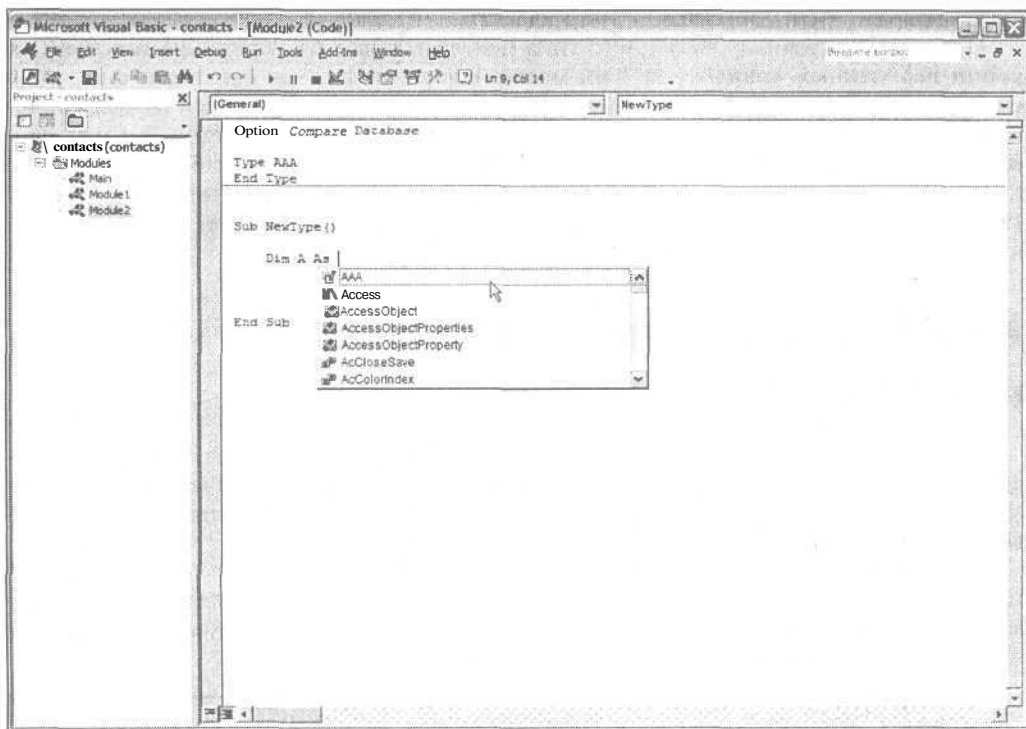


Рис. 11.1. Окно модуля с раскрытым списком доступных типов данных

#### Листинг 11.1. Пример объявления простого пользовательского типа данных

```
1: Type AAA
2:   AnInt As Integer
3: End Type
4:
5: Sub NewType ( )
6:   Dim A As AAA
7:   A.AnInt = 5
8: End Sub
```



Строки 1-3 листинга 11.1 содержат объявление нового типа, AAA. Строка 2 тела объявления определяет целочисленную переменную-член `AnInt`. Строки 5-8 иллюстрируют приемы использования созданного типа: в строке 6 содержится объявление объекта A нового типа, а в строке 7 — оператор присваивания значения его члену (переменной `AnInt`).

Обратите внимание на использование в строке 7 символа точки. Запись вида `ИмяЭкземпляраТипа.ИмяЧлена` трактуется компилятором следующим образом: "Необходимо взять экземпляр `ИмяЭкземпляраТипа` и обратиться к его члену `ИмяЧлена`". Конструкцию `Dim A As AAA` называют *объявлением экземпляра (объекта)* типа или класса. Если представить человечество как "класс", каждого из нас можно назвать "экземпляром" (или "объектом") этого класса.

Отдельный экземпляр созданного типа получает в свое распоряжение собственные копии переменных-членов типа. Тип можно воспринимать как оригинал, с которого снимается необходимое число слепков. Поэтому при каждом объявлении экземпляра пользовательского типа данных в памяти создается новый полный комплект переменных, служащих частью этого типа. Если строку 6 листинга 11.1 переписать, введя `Dim A As AAA, B As AAA`, это будет означать создание двух объектов типа AAA — A и B. Присваивание переменной `AnInt` объекта A значения 5 не оказывает никакого влияния на содержимое одноименной переменной объекта B.

## Какие данные может содержать пользовательский тип

В составе типа, определяемого пользователем, могут находиться самые разнообразные данные — строки, целые и вещественные числа, переменные для хранения значений валюты, даты и даже экземпляры других нестандартных типов — одним словом, объекты всех встроенных и созданных типов данных.

В пользовательский тип данных, однако, не могут включаться объекты, помеченные служебными словами `Const`, `Static` и `Global`, а также функции и подпрограммы. Напротив, последние могут быть членами *классов* VBA, хотя другие языки позволяют размещать определения функций даже в типах данных.



Некоторые языки программирования, например C, допускают определение функций в составе типов данных. Аналогом директивы `Type` в языке C служит слово `struct`. Это полезно знать, поскольку сама операционная система Windows написана на языке C, а функции Windows Application Programming Interface (API) в состоянии существенно расширить возможности вашего VBA-кода. Многие из них предполагают наличие в прикладной программе определений соответствующих типов данных.

Листинг 11.2 содержит примеры использования в составе создаваемого типа объявлений самых разных переменных и иллюстрирует приемы работы с ними. Точное количество и состав набора членов определяемого типа данных обуславливаются исключительно потребностями конкретной задачи.

## Листинг 11.2. Пример использования сложного типа данных

```
1: Type LotsaData
2:   AString As String
3:   Anint As Integer
4:   ADouble As Double
5:   SomeMoney As Currency
6:   TheDate As Date
7:   TheTime As Date
8:   TripleA As AAA
9:   RecordSet As ADODB.Recordset
10:End Type
11:
12: Sub Initialize( )
13:   Dim Data As LotsaData
14:   Data.AString = "Что-то вот эдакое!"
15:   Data.AnInt = 13
16:   Data.ADouble = 340000000000000#
17:   Data.SomeMoney = 5000
18:   MsgBox Data.SomeMoney
19:   Data.TheDate = Date
20:   Data.TheTime = Time
21:   Data.TripleA.AnInt = 16
22:   Set Data.RecordSet = Nothing
23: End Sub
```

### Анализ

Строки 1–10 листинга 11.2 содержат объявление типа данных, названного `LotsaData`. В строке 2 объявляется символьная переменная, в строке 3 — целочисленная, в строке 4 — двойной точности; строка 5 содержит определение члена типа `Currency`, строки 6 и 7 — переменные типа `Date`; в строке 8 объявляется экземпляр типа `AAA`, а строка 9 содержит объявление объекта стандартного класса `ADODB.Recordset`.

В строках 12–23, где находится код процедуры `Initialize`, создается объект типа `LotsaData` и выполняется инициализация его членов. После создания экземпляра типа (строка 13) разрешается ссылаться на его члены с помощью оператора точки (`.`). Чтобы получить доступ к членам вложенных объектов составных типов, достаточно перечислить цепочку имен, отделяя их символом точки (как сделано в строке 21). Конструкция `Data.TripleA.AnInt = 16` означает следующее: "присвоить литеральное значение 16 переменной `Anint` из состава такого объекта `TripleA` типа `AAA`, который, в свою очередь, служит членом экземпляра `Data` типа `MoreComplexType`".

### Новый термин

**Идиома** в программировании — это способ выражения законченной мысли с помощью средств языка. Например, пользовательский тип данных — это идиома.

### Новый термин

**Класс** — это тип данных, экземпляры которого могут не только что-то "знать", но и "уметь". "Знание" заключено в содержимом членов-объектов данных, а "умение" — в наборе членов-функций и процедур. Воспринимайте классы как результат развития концепции пользовательских типов, полученный за счет добавления в состав типа функций и процедур.

*Объект* — это экземпляр класса. (В ходе изложения мы иногда, отступая от канонов строгости, употребляем это слово также для ссылки на элементы данных других типов.)

Для обеспечения поддержки парадигмы объектно-ориентированного программирования в язык и среду VBA на протяжении нескольких последних лет были внесены существенные изменения. RecordSet, член типа MoreComplexType, определенного в тексте листинга 11.2, — это объект класса ADODB.Recordset. Разработчики языка VBA и участники проекта по его модернизации, связанной с добавлением объектных возможностей, решили ввести средства, позволяющие различать присваивание *обычное* и *объектное*. Вот почему в строке 22 используется служебное слово Set. Более подробные сведения о классах и их объектах приведены в главе "21-й час. Основы программирования классов".

## Создание экземпляров пользовательских типов данных

Объявления переменных (экземпляров) пользовательских типов данных подчиняются общим правилам. Такие конструкции могут находиться как внутри функций или подпрограмм, так и за их пределами. Помните: все переменные, объявленные в теле любой функции или процедуры, называются *локальными*, а за ее пределами — *глобальными*. Признак "глобальности" объекта можно явно подчеркнуть заданием служебного слова Global, а внутри функций (подпрограмм) допускается использовать квалификатор Static. Листинг 11.3 содержит ряд примеров объявления и применения локальных, глобальных и статических экземпляров пользовательских типов данных.

Листинг 11.3. Примеры использования локальных, глобальных и статических объектов данных

```
1: Type Typel
2:   NoData As Integer
3: End Type
4:
5: Dim T1 As Typel
6: Global T2 As Typel
7:
8: Sub OtherTypeVars( )
9:   Dim T3 As Typel
10:  Static T4 As Typel
11: End Sub
```

### Анализ

В строках 1-3 листинга 11.3 находится объявление типа Typel, содержащего один член — целочисленную переменную. В строке 5 создается экземпляр T1 нового типа; хотя объявление лишено явного указания служебного слова Global. Переменная T1 трактуется компилятором как глобальная внутри текущего модуля, и к ней разрешается обращаться из любой строки этого модуля, хотя из других модулей, которые могут существовать в программе, она не "видна". Уровень относительной доступности элементов данных, функций и процедур задается контекстом их определения. Строка 6 содержит явное объявление глобального экземп-

ляра типа `Type1`. Если конструкция объявления элемента данных включает слово `Global`, этот элемент становится доступным из любой строки программы. Язык VBA не предусматривает специальных служебных слов для объявления локальных переменных — свойство "локальности" подразумевается неявно. В строках 8–11 определяется процедура, в теле которой создается еще два экземпляра типа `Type1`. Переменная `T3` локальна, поэтому существует только в период выполнения процедуры `OtherTypeVars`. Строка 10 содержит инструкцию создания статической переменной `T4` типа `Type1`. Переменные, снабженные служебным словом `static`, доступны только в контексте объявления — внутри функции или подпрограммы, но сохраняют свои значения от одного успешного вызова функции (процедуры) к другому.

## Перечислимые типы

Перечислимые типы во многом схожи с константами, но между ними существуют и серьезные различия. *Константа* — это именованное неизменное значение, представляющее единственный элемент данных. *Перечислимый тип* (перечисление) — это набор имен, которым присваиваются целочисленные значения: либо неявно (в этом случае значение, отвечающее определенному имени-члену типа, зависит от порядка следования имен), либо непосредственно. Взгляните на синтаксис объявления перечисления и сопоставьте его с конструкцией объявления пользовательского типа.



```
Enum ИмяПеречислимогоТипа
    Элемент1 [= ЦелочисленноеЗначение]
    [Элемент2 [= ЦелочисленноеЗначение]]
    ***
    [ЭлементN [= ЦелочисленноеЗначение]]
End Enum
```

Конструкция открывается строкой, содержащей служебное слово `Enum` и имя перечислимого типа, и завершается парой слов `End Enum`. Число элементов-членов типа произвольно. Каждому элементу или только некоторым из них могут быть поставлены в соответствие целочисленные значения.

Перечислимые типы находят применение в тех ситуациях, когда удобно определить набор имен, которым отвечают строго определенные целые числа. Несмотря на целочисленный характер значений элементов-членов перечислимого типа, их нельзя приравнивать обычным переменным типа `Integer` или сопоставлять с такими переменными — подобные операции допустимы только в отношении объектов того же перечислимого типа. Листинг 11.4 иллюстрирует пример использования перечислимого типа.

### Листинг 11.4. Пример использования перечислимого типа

```
1: Enum EmploymentStatus
2:     esInterviewed = 1
3:     esHired = 2
4:     esTerminated = 3
5:     esLeaveOfAbsence = 4
6: End Enum
7:
8: Sub SetEmploymentStatusWithEnum (ByVal CurrentStatus As _
    EmploymentStatus)
```

```

9:
10: If (CurrentStatus = esHired) Then
11:     ' Что-то нужно сделать
12: End If
13:
14: End Sub
15:
16: Sub SetEmploymentStatusWithInteger (ByVal CurrentStatus As Integer)
17:
18: If (CurrentStatus >= 1 And CurrentStatus <= 4) Then
19:     If (CurrentStatus = 1) Then
20:         ' Нужно сделать то же самое
21:     End If
22: End If
23: End Sub

```

### Анализ

Строки 1–6 листинга 11.4 содержат объявление перечислимого типа `EmploymentStatus`, предусматривающего четыре возможных элемента-значения: `esInterviewed`, `esHired`, `esTerminated` и `esLeaveOfAbsence`. В строках 8–14 находится текст процедуры, использующей объект созданного перечислимого типа, а ниже (в строках 16–23) приведена аналогичная по функциональным возможностям процедура, применяющая обычные значения типа `Integer`.

Обратите внимание, насколько изящнее первая версия. По причине того, что переменным перечислимого типа могут присваиваться только заранее оговоренные значения, необходимость в дополнительных проверках (аналогичных приведенной в строке 18) отпадает. Напротив, в варианте, не использующем переменные перечислимого типа, такие проверки обязательны, иначе о надежности кода говорить не стоит. Естественно, программа потребует добавления дополнительных строк.



Используйте именованные константы вместо литералов и перечислимые типы вместо наборов постоянных целочисленных значений.

Кроме того, обратите внимание, насколько более прозрачна и доходчива первая версия условной конструкции (`If (CurrentStatus = esHired) Then`) в сравнении со второй (`If (CurrentStatus = 1) Then`): `esHired` — конкретное имя, а `1` — просто некоторое абстрактное число. Перечислимый тип — это идиома, придающая коду ясность и четкость. Иными словами, всякий раз, когда возникает необходимость в использовании ограниченного набора целочисленных значений, удобно применять перечислимые типы; в противном случае придется написать код большего объема и менее высокого качества.

## А теперь повторим

Пользовательский тип — удобное средство построения структуры данных, которая не укладывается в схему стандартных простых или составных типов. Например, если задача требует хранения нескольких элементов информации, относящейся к абоненту электронной почты, целесообразно объявить новый тип и включить в него все атрибуты данных, характеризующие абонента — имя, фамилию, почтовый адрес и т.п.

В функцию или процедуру всегда **проще** передать одну переменную сложного типа, нежели дюжину значений простых типов; таким образом, типы, определяемые пользователем, — это еще и средство упрощения интерфейса функций и подпрограмм.

## Резюме

На этом занятии мы изучили конструкции объявлений пользовательских и перечислимых типов данных, способы создания экземпляров новых типов и их практического применения. Типы, определяемые пользователем, — это средство агрегации данных, а перечисления — эффективный инструмент представления ограниченных наборов целочисленных значений.

Рассмотренные темы — первый шаг на пути к постижению таинств объектно-ориентированного программирования. Впереди — крутой подъем, обойти который не удастся. Вам придется обращаться к объектам чуть ли не в каждой программе, поэтому лучше заранее установить с ними доверительные отношения.

Итак, вы изучили вопросы объявления новых типов и перечислений, создания локальных, глобальных и статических экземпляров типов и ознакомились с присущими им ограничениями. Разделы "Вопросы и ответы" и "Задания", приведенные ниже, позволят вам убедиться в прочности своих знаний.

## Вопросы и ответы

**Вопрос.** Создав новый тип данных, могу ли я считать, что уже знаю приемы объектно-ориентированного программирования?

**Ответ.** Нет. Традиционные структуры данных, какими бы сложными они ни были, — это еще не объекты. Объектно-ориентированное программирование основывается на концепции нового типа данных — *класса* (подробнее см. главу 21).

**Вопрос.** Могут ли типы быть вложенными?

**Ответ.** Да. В объявлении нового типа допускается использовать переменные существующих типов. Единственный критерий истины — практическая целесообразность.

**Вопрос.** Разрешается ли включать в объявление типа процедуры или функции?

**Ответ.** Нет. Подобную возможность предоставляет конструкция определения класса.

**Вопрос.** Следует ли безоговорочно отказываться от констант в пользу перечислимых типов?

**Ответ.** Создавайте и используйте перечислимые типы только в тех случаях, когда имеете дело с наборами целочисленных значений. Если константы никак не связаны по смыслу, определяйте их с помощью служебного слова `Const`.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

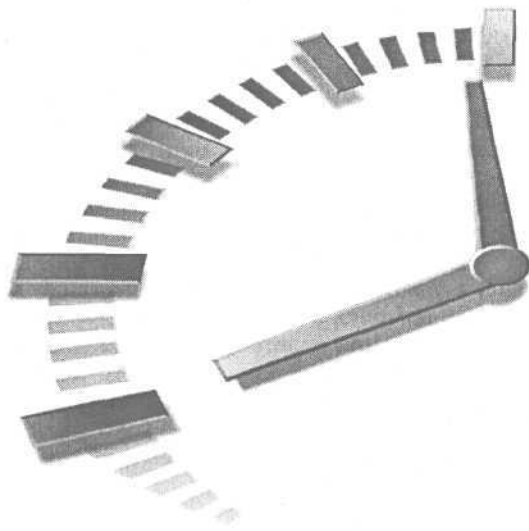
## Тесты

1. Может ли пользовательский тип данных содержать переменную перечислимого типа?
2. Для каких целей применяются пользовательские и перечислимые типы и в чем их различие?

3. Могут ли в объявлениях типов содержаться функции или процедуры?
4. Допускаются ли в определениях типов члены-константы?
5. Позволяется ли включать в объявление типа ссылочную переменную?

## Упражнения

1. Создайте объявление типа для хранения данных об имени человека, его адресе и номере телефона.
2. Определите тип, одним из членов которого будет экземпляр типа, созданного в предыдущем упражнении. Что может послужить мотивом подобных действий? Почему бы просто не исправить объявление исходного типа?
3. Объявите перечислимый тип, представляющий все сорта мороженого.



# 12-й час

## Управление данными переменного объема

Ввод данных в память компьютера и их обработка — одна из основных функций, выполняемых программами. Массивы (главный объект нашего внимания в ходе текущего занятия) выступают базовой и наиболее часто используемой структурой данных.

*Массив* определяется в виде набора, состоящего из нуля или более элементов данных одного типа. Подобная конструкция обладает и сильными, и слабыми сторонами. Несмотря на ряд ограничений, массивы — при умелом управлении ими — способны стать чрезвычайно полезным, простым и мощным инструментом программирования.

Основные темы занятия.

- Определение массивов.
- Динамическое изменение размеров массивов.
- Функции, позволяющие избежать ошибок при использовании массивов.
- Сортировка массивов данных.

## Знакомство с массивами

Массивы — одна из традиционных и наиболее "древних" структур данных. Массивы, позволяющие хранить в памяти компьютера наборы однородных данных, находят самое широкое применение. Они, например, способны с успехом заменить конструкцию вида *Select Case* (о ней речь пойдет в главе "21-й час. Основы программирования классов").

Если данные уже представлены в виде таблицы или коллекции, применять массивы, вероятно, нецелесообразно. Однако, если стоит задача создания временного хранилища данных в памяти и их динамической обработки, в такой ситуации массивы сослужат хорошую службу. /

Массив — это непрерывный блок адресов памяти, обозначаемый одним именем и трактуемый в виде единственной переменной. Механизм построения массивов по-



звolyет говорить о памяти как о множестве "ячеек", каждая из которых способна хранить определенное значение. Все значения, присвоенные элементам массива, относятся к одному типу данных. Впрочем, если речь идет о типе Variant, природа отдельных элементов массива может быть совершенно различна.

Массивы бывают и многомерными. Способность компьютеров управлять массивами данных намного превосходит пределы житейского восприятия. Мир, который нас окружает, трехмерный. Даже если вы и не физики, то все же нетрудно мысленно представить еще одно, четвертое, измерение — время. А что дальше? Но компьютерная программа способна оперировать массивами произвольной размерности и объема — лишь бы хватило памяти. Конечно, все это, большей частью, теория. На практике, к нашему всеобщему удовольствию, возможностей одно- и двухмерных массивов, как правило, вполне достаточно.

Для хранения целого числа необходимо 32 бита, или 4 байта. Целочисленный массив из 10 элементов, таким образом, займет в памяти, около 40 байт. Массивы легче воспринимать в виде ячеек памяти (рис. 12.1). Каждый элемент массива адресуется посредством индекса, который может принимать значения в пределах от наименьшего до наибольшего допустимого номера включительно.

34	56	66	31	78					
----	----	----	----	----	--	--	--	--	--

Рис. 12.1. Для размещения массива используется непрерывный фрагмент компьютерной памяти. В каждой ячейке содержится 16 бит, или 2 байта



В основе операций над индексами массивов лежит предположение о том, что массиву отвечает так называемый базовый адрес памяти. Адрес очередного элемента массива в памяти можно рассчитать по формуле  $\text{БазовыйАдрес} + (\text{Индекс} * \text{РазмерТипаЭлемента})$

К счастью, компилятор VBA выполняет все вычисления самостоятельно, и заботиться о них не нужно. Только в языках ассемблера и С подобные операции все еще находят широкое применение.

## Объявление массивов



Самый сложный вопрос, связанный с использованием массивов в программе на языке VBA, заключается в определении верного исходного размера массива и типа его элементов. Синтаксис объявления массива лишь незначительно отличается от конструкции объявления переменной:

```
Dim ИмяМассива ( [ИсходныйРазмер] ) As ТипДанных
```

В выражении объявления могут использоваться служебные слова ReDim (вместо Dim) и Static (дополнительно). О них речь пойдет чуть позже. Основная конструкция объявления массивов связана с использованием слова Dim, за которым следует имя массива, сопровождаемое целым числом в круглых скобках. Если число в скобках не задано, массиву изначально память не отводится. Завершается выражение объявления как обычно — служебным словом As и обозначением типа данных, в качестве которого допустимо использовать любой стандартный или пользовательский тип.



При попытке изменить длину массива фиксированного размера, т.е. массива, объявленного с помощью служебного слова `Dim` с указанием размера, произойдет ошибка. Решить эту проблему можно, объявив массив с помощью `ReDim` и оставив скобки пустыми (а лучше не задавайте размер массива при его объявлении).

## Массивы фиксированного размера

Количество элементов массива *фиксированного размера* задается в конструкции его объявления. На протяжении всего сеанса работы программы число элементов такого массива остается постоянным. Объявление массива фиксированного размера отвечает синтаксическому выражению, приведенному выше, за тем исключением, что значение `ИсходныйРазмер` должно быть указано обязательно. Текст листинга 12.1 содержит пример объявления и использования массива фиксированного размера.

### Листинг 12.1. Пример использования массива фиксированного размера

```
1: Sub FixedArray( )
2:   Dim Ints(100) As Integer
3:
4:   ' Инициализация элементов массива
5:   For I = 1 To 100
6:     Ints(I) = I
7:   Next I
8:
9:   For I = 100 to 1 Step -1
10:    Debug.Print Ints(I)
11:  Next I
12: End Sub
```

#### Анализ

Строка 2 содержит выражение объявления массива `Ints`, предназначенного для хранения 100 целых чисел. Строки 5–7 демонстрируют конструкцию цикла для итеративного прохождения по всем элементам массива. В практике программирования вам часто придется выполнять начальную инициализацию элементов массивов определенными значениями. Если инициализация не производится, каждому элементу массива по умолчанию присваивается некоторое значение, зависящее от типа: например, числовые элементы будут содержать значение 0, а символьные — `null` (пустая строка). В строках 9–11 расположен цикл, позволяющий отобразить содержимое всех элементов массива, начиная с последнего и заканчивая первым, в окне Immediate редактора Microsoft Visual Basic.

Мы не случайно остановимся на обоих направлениях перемещения по элементам массива — прямом и обратном. Это важно, поскольку во многих случаях (как, например, при сортировке данных, о чем речь пойдет в одной из последних разделов этой главы) потребуются самые разнообразные способы обработки массивов данных.

## Динамические массивы

*Динамическим* называется массив, число элементов которого может изменяться во время выполнения программы — в отличие от массива фиксированного размера, определяемого на этапе кодирования. Если для объявления динамического массива исполь-

зуется служебное слово `Dim`, значение в круглых скобках опускается. Существует и другой вариант объявления, предусматривающий обязательное задание размера массива:

```
ReDim ИмяМассива С ИсходныйРазмер ) As ТипДанных
```

Роль признака "изменяемости" объема массива играет теперь служебное слово `ReDim`. Все остальное аналогично — идентификатор, число-размер в круглых скобках и обозначение типа данных.

При необходимости изменения размера массива после инициализации его элементов следует предусмотреть средства создания резервной копии данных. На первый взгляд, последовательность действий может быть такой: объявить временный массив, скопировать значения массива, подвергающегося изменениям, во временный, выполнить модификацию массива и вернуть значения обратно. Это, однако, не лучший метод — VBA позволяет использовать другой, в котором выражение повторного объявления содержит служебное слово `Preserve`. Листинг 12.2 содержит пример, иллюстрирующий процедуру изменения размера массива с использованием инструкции `Preserve`.

#### Листинг 12.2. Пример объявления динамического массива и изменения его размера

```
1: Sub DynamicArray( )
2:   ReDim Strings(5) As String
3:   ReDim Strings(10) As String
4:
5:   Strings(1) = "Greetings"
6:   Strings(2) = "Wilkommen"
7:   Strings(3) = "Bienvenido"
8:   Strings(4) = "Привет"
9:   Strings(5) = "Здоровенькі були"
10:
11:  ReDim Preserve Strings(30) As String
12:
13:  Dim Elem As Variant
14:  For Each Elem In Strings
15:    Debug.Print Elem
16:  Next
17: End Sub
```

#### Анализ

Строки 2 и 3 листинга 12.2 содержат объявления одного и того же массива символьных переменных. Второе объявление просто удлиняет массив до десяти элементов; служебное слово `Preserve` в данном случае необязательно, поскольку массив пуст. В строках 5–9 выполняется инициализация пяти первых элементов: обратите внимание на использование индекса в круглых скобках, который указывает на номер текущего элемента массива. Строка 11 содержит инструкцию повторного объявления, предусматривающего увеличение размера массива до 30 элементов и гарантирующего посредством слова `Preserve` сохранность данных. Цикл, который расположен в строках 13–16, позволяет удостовериться, что содержимое массива не изменилось. Опасность потери значений подстерегает вас в случае отсутствия директивы `Preserve` либо уменьшения количества элементов вместо его увеличения. В последнем случае даже `Preserve` не сможет помочь, ибо нельзя сохранить то, чего заведомо нет.

# Статические массивы

Все статические переменные объявляются внутри процедур или функций. То же справедливо и в отношении *статических* массивов — т.е. массивов, объявленных с использованием служебного слова `Static`. Статические объекты данных **сохраняют** свои значения даже в том случае, когда выполнение функции (подпрограммы) завершилось. Они могут продолжать работу при очередном обращении к той же функции или подпрограмме.

Конструкция объявления статического массива отличается присутствием служебного слова `Static`, например:

```
Static Dim Array1(10) As String
```

Если изменить слово `Dim` на `ReDim`, статический массив станет еще и динамическим. Статические массивы находят разное применение (во второй части этой главы приводится ряд **конкретных** примеров).

## Задание точки отсчета индекса массива

В некоторых языках программирования, например `C`, элементы массивов отсчитываются, начиная с нуля. Другими словами, допустимые значения индекса массива из десяти элементов, определенного в программе на языке `C`, заключены в интервале от 0 до 9. `VBA` позволяет явно указать точку отсчета индексов массивов. Этой цели служит команда `Option Base`:

```
Option Base 0 | 1
```

Команда `Option Base` должна располагаться в верхней части программного модуля. По умолчанию в качестве базы отсчета принимается значение 0. В этом случае первый элемент любого массива будет адресоваться нулевым значением индекса, а *N*-й — значением *N-1*. Если вам удобно отсчитывать элементы массива с нуля, положитесь на стандартное "поведение" системы; в противном случае явно укажите конструкцию `Option Base 1`. Главное, будьте последовательны в своих решениях.

## Использование массивов для хранения данных

Вполне возможно, что через некоторое время с программным кодом, который вы — к восторгу современников — удосужились написать, придется работать и вашим преемникам. Вероятно, этим счастливицам не понравится, что в тексте кода вы (не очень предусмотрительно) использовали явные обращения к нулевому или первому элементу массива. Наилучший способ исключения подобной ситуации состоит в использовании специальных средств языка, позволяющих избежать зависимости от выбранной точки отсчета индексов.

Существует такая альтернатива: применять циклы вида `For Each ... Next` либо обратиться к функциям `Lbound` и `Ubound`. Рекомендуется остановить выбор на указанных функциях — если, разумеется, ваш код вообще предусматривает циклические действия над элементами массивов. Синтаксические формулы обращения к функциям `Lbound` и `Ubound` таковы:

```
Lbound( ИмяМассива [, Измерение] )  
Ubound( ИмяМассива [, Измерение] )
```

Синтаксис

Обе функции требуют наличия, как минимум, одного аргумента — имени массива. Второй (необязательный) параметр Измерение, указывает номер измерения массива, для которого необходимо вычислить значение верхней или нижней границы изменения индекса. Если аргумент Измерение не задан, по умолчанию предполагается, что речь идет о первом измерении массива.



Значение, установленное командой Option Base, не оказывает влияния на порядок отсчета номеров измерений массива — они всегда перечисляются, начиная с единицы.

Далее рассмотрим два массива — двухмерный и трехмерный:

```
Dim Array1(5, 20) As String
Dim Array2(5, 5, 10) As Integer
```

Первый массив, Array1, двухмерный. Такие массивы называют *матрицами*. Второй, Array2, — это трехмерная матрица. (В принципе, матрицей можно назвать даже *n*-мерный массив.) Массив Array1 содержит 100 индексированных "ячеек" для хранения символьных значений. Если вызвать функцию Ubound, передав ей единственный параметр Array1, она возвратит значение 5. Результатом обращения Ubound( Array1, 2 ) будет число 20.

Функции Lbound и Ubound гарантируют, что востребованное программой значение индекса массива находится в допустимом диапазоне. Функции удобно применять в качестве границ изменения переменной цикла — убедитесь в этом, прочитав текст листинга 12.3.

### Листинг 12.3. Пример объявления многомерного массива и использования функций Lbound и Ubound

```
1: Sub DisplayBounds( )
2:
3:   Dim I As Integer
4:   Dim Array1(5, 10) As Integer
5:   Dim Array2(100) As Integer
6:
7:   Debug.Print Ubound( Array1, 2 )
8:   Debug.Print Ubound( Array2 )
9:
10:  For I = Lbound( Array2 ) To Ubound( Array2 )
11:    Array2(I) = I: Debug.Print array2(I)
12:  Next I
13:
14: End Sub
```

#### Анализ

Строка 4 листинга 12.3 демонстрирует объявление массива, содержащего 5 индексированных "ячеек" в первом измерении и 10 — во втором. Таким образом, массив Array1 способен хранить 50 целых чисел. В строке 5 объявлен одномерный массив Array2, предусматривающий возможность хранения 100 целочисленных значений. Строка 7 иллюстрирует применение функции Ubound для получения объема второго измерения массива Array1, а результатом вычисления выражения, расположенного в строке 8, будет "длина" единственного (первого) измерения массива Array2. Строки 10–12 содержат пример цикла For ... Next, в котором границы изменения индекса задаются с помощью функций Lbound и Ubound.

Использование функций `Lbound` и `Ubound` можно отнести к образцам хорошего стиля программирования, гарантирующего надежность и переносимость кода в условиях динамического изменения размеров массивов. Даже если со временем понадобится исправить в программе конструкции объявлений массивов, циклы обновлять не придется.

Приведенный ниже листинг 12.4 иллюстрирует приемы циклической обработки элементов многомерного массива. Следует отметить, что вполне допустимы операции обращения к *любому* измерению массива, и, как правило, они связаны со значительными издержками в отношении скорости работы программы, поскольку циклы по необходимости становятся вложенными. С увеличением числа измерений массива вычислительная сложность алгоритмов циклической обработки возрастает по экспоненциальному закону. Показатели быстродействия ухудшаются и при значительном росте количества элементов в каждом измерении; в этом случае также существенно повышаются требования к объему свободной памяти.

Предположим, что на обработку одного элемента массива затрачивается одна секунда (разумеется, это условно). Если массив содержит два измерения по 100 элементов в каждом, на выполнение операций со всеми элементами потребуется 10000 секунд. Добавление в массив еще одного измерения той же "протяженности" приведет к возрастанию времени обработки на два порядка, т.е. до 1000000 секунд. (Мы уже не упоминаем о затратах оперативной памяти, необходимой для хранения такого числа объектов данных.) Решив объявить многомерный массив, примите к сведению аргументы, изложенные выше.



В примере предыдущего абзаца единицы времени взяты условно. Кроме того, существуют алгоритмы, обрабатывающие данные значительно эффективнее, чем вложенные циклы.

#### Листинг 12.4. Пример процедуры обработки многомерного массива

```
1: Option Base 1
2: Sub TwoDimensions( )
3:   Dim Two_D_Array(10, 10) As String
4:   Dim I As Integer, J As Integer
5:   For I = Lbound( Two_D_Array, 1 ) To Ubound( Two_D_Array, 1 )
6:     For J = Lbound( Two_D_Array, 2 ) To Ubound( Two_D_Array, 2 )
7:       Two_D_Array(I, J) = I & "*" & J
8:       Debug.Print Two_D_Array(I, J)
9:     Next J
10:  Next I
11: End Sub
```

#### Анализ

Строка 3 листинга 12.4 содержит объявление двухмерной матрицы размерами 10x10. В строке 4 содержатся определения целочисленных переменных `I` и `J`, которые будут использоваться в качестве индексов. Буквы `I`, `J` и `K` традиционно применяются с подобной целью в математике, поэтому они уместны и в тексте программы. (Следует напомнить: будьте последовательны в своих действиях. Если вы использовали букву `I` в качестве индекса первого измерения массива в одном месте программы, поступайте так и впредь. Созидательную энергию лучше направить в более полезное русло.) Строка 5 содержит заголовок внешнего цикла `For ... Next`, а строка 6 – внутреннего. В обоих случаях для определения границ интервалов изменения индексов применяются функции `Lbound` и `Ubound`. Строка 6 иллюстрирует способ обращения к элементу двухмерного массива. Выражения индексов в круглых скобках разделяются символом запятой.

Массивы с числом измерений, превышающим 1 или 2, используются относительно редко и преимущественно в приложениях математического или технического характера.

# Функции для управления массивами

Синтаксис

В VBA существует еще две нужные функции — `Array` и `Erase`. Функция `Array` возвращает массив, который построен на основании значений элементов, переданных ей в виде списка аргументов. Синтаксис функции `Array` таков:

```
Array( [Аргумент1 [, Аргумент2 [, ... [, АргументN] ... ] ] )
```

`Array` возвращает массив, содержащий значения, переданные в виде параметров. Если выражение вызова функции не содержит аргументов, возвращается пустой массив.

Функция `Erase` присваивает каждому элементу массива пустое значение, равноценное `null` и зависящее от типа массива: для `String` это пустая строка, для чисел — значение 0, для `Variant` — `Empty`, а для массивов объектов — `Nothing`. После выполнения операции `Erase` массивы всех типов, кроме `Variant`, по-прежнему допускают индексацию. Листинг 12.5 содержит примеры использования функций `Array` и `Erase`.



Операции сравнения содержимого элементов массивов различных типов выполняются по-разному. Функция `Erase`, примененная к массиву типа `Variant`, приводит к полной очистке массива и присваиванию ему единого значения `Empty`. При очистке массива объектов каждый элемент получает значение `Nothing`; элементы массива строк становятся равными `""`, а элементы числового массива — 0. Конечно, все это не совсем привлекательно (укоризненный взгляд в сторону Microsoft), но сейчас положение дел именно таково. Поэтому, собираясь выяснять, пуст тот или иной массив, не забывайте об указанных выше особенностях.

## Листинг 12.5. Примеры использования функций `Array` и `Erase`

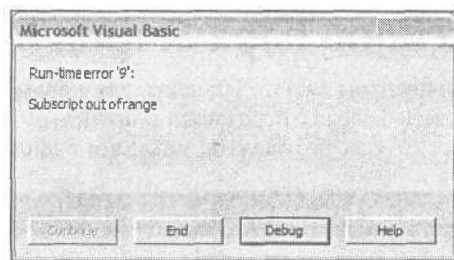
```
1: Sub ArrayFunctions( )
2:   Dim A( )
3:   A = Array( 1, 2, 3, 4, 5 )
4:   Dim I As Integer
5:   For I = Lbound( A ) To Ubound( A )
6:     Debug.Print A(I)
7:   Next I
8:   Erase A
9: End Sub
```

### Анализ

Строка 2 листинга 12.5 содержит объявление пустого массива `A` типа `Variant`. В строке 3 массив заполняется значениями, указанными при вызове функции `Array`: 1, 2, 3, 4 и 5. Строки 4–7 выглядят знакомым образом: здесь объявляется целочисленная переменная цикла и выполняется циклическое отображение содержимого каждого элемента массива в окне `Immediate`. Строка 8 содержит инструкцию очистки массива `Erase`, в результате выполнения которой массиву присваивается значение `Empty`.



Если вы попытаетесь применить функции `Lbound` или `Ubound` по отношению к массиву типа `Variant`, подвергнутому операции `Erase`, откроется окно сообщения об ошибке, показанное на рис. 12.2. Но для работы с очищенными массивами всех других типов, включая пользовательские, указанные функции могут применяться.



*Рис. /22. Окно сообщения о выходе значения индекса массива за допустимые пределы*

## Проверка переменных с помощью функции **IsArray**

Функция `IsArray` в качестве аргумента принимает переменную любого типа и возвращает булево значение, определяющее, является ли переменная массивом. В функцию `IsArray` можно передавать и переменные, и литеральные значения. Данная функция оказывается особенно полезной при работе с данными типа `Variant`, когда реальный тип данных не очевиден. Пример использования функции `IsArray` приведен в листинге 12.6.

### Листинг 12.6. Использование функции `IsArray` для проверки, является ли переменная массивом

```
1: Dim A
2: MsgBox IsArray(A) 'Отображает значение False
3:
4: Dim B ()
5: MsgBox IsArray(B) 'Отображает значение True
6:
7: Dim C As Variant
8: C = Array("A", "B", "C")
9: MsgBox IsArray(C) 'Отображает значение True
10:
11: Dim D As Integer
12: MsgBox IsArray(D) 'Отображает значение False
13:
14: Dim Y As ADODB.Recordset
15: MsgBox IsArray(Y) 'Отображает значение False
16:
17: Dim F(10) As String
18: MsgBox IsArray(F) 'Отображает значение True
19:
20: MsgBox IsArray(Array(1, 2, 3)) 'Отображает значение True
```

Настоятельно советуем избегать создания такого двусмысленного кода, в котором требовалось бы использовать функции типа `IsArray`. Однако вы можете попасть в ситуацию, когда без данной функции не обойтись. В этом случае еще раз пересмотрите код — возможно, его удастся переработать так, чтобы он стал более ясным.



# Передача массивов параметров в процедуры

Последний аргумент процедуры можно объявить как параметр `ParamArray` (массив значений типа `Variant`), что позволит передавать в процедуру набор значений, которые будут преобразованы в массив. Как это сделать, показано в листинге 12.7.

## Листинг 12.7. Использование параметра типа `ParamArray`

```
1: Sub CallTest()  
2:   Call TestParamArray("Paul Kimmel", "(517) 555-1234", _  
3:     "pkimmel@softconcepts.com")  
4: End Sub  
5:  
6: Sub TestParamArray(ContactName As String, _  
7:   ParamArray ContactInfo() As Variant)  
8:   Dim S As String  
9:   S = ContactName  
10:  Dim I As Integer  
11:  For I = LBound(ContactInfo) To UBound(ContactInfo)  
12:    S = S & ContactInfo(I)  
13:  Next I  
14:  
15:  MsgBox "Информация о пользователе: " & S  
16: End Sub
```

### Анализ

Тестирование процедуры начните с запуска `CallTest`. Эта процедура вызывает подпрограмму `TestParamArray` с тремя аргументами. Как видно из текста листинга, процедура `TestParamArray` имеет всего два аргумента: `ContactName` и `ParamArray`. При тестировании аргументу `ContactName` присваивается значение "Paul Kimmel", а два других аргумента помещаются в массив `ParamArray`. В результате работы программы в окне сообщений появится вся известная информация о пользователе, независимо от того, сколько аргументов передается в подпрограмму.



Программистам, использующим язык Ассемблер, наверное, будет легче изучить массивы параметров, поскольку технология их применения сходна с тем, как данные помещают в стековый фрейм перед вызовом функции и получают из него при вызове функции. Всего несколько лет назад программисты вручную управляли стековым фреймом, теперь его обработка производится компиляторами.

`ParamArray` необходим по двум причинам. Язык программирования C поддерживает переменное число аргументов функции, поэтому Visual Basic, вероятно, наследует эту возможность. Такая техника работает, поскольку передаваемые в функцию данные помещаются в стековый фрейм, который на самом деле является массивом данных. Несмотря на то, что стековый фрейм управляется компьютером, в компилятор автоматически добавляется код обработки стекового фрейма, поэтому массив аргументов может пригодиться. От разработчиков компиляторов потребуются больше усилий для того, чтобы преобразовать данные из стека в специальную переменную, которая обеспечивает доступ к стековому фрейму как к массиву.

# Функции, возвращающие массивы

VBA не дает возможности непосредственного возврата массива из функции. Впрочем, один способ решения подобной задачи все-таки существует — он основан на "модели поведения" функции `Array`, которая, по существу, возвращает значение типа `Variant`.



Поскольку возвращаемое значение имеет тип `Variant`, вероятно, придется использовать функцию `IsArray`.

Тип `Variant` представляет собой "обобщение" самых разнообразных типов данных. В программу, содержащую обращения к элементам данных типа `Variant`, компилятор добавляет специальные фрагменты кода, которые позволяют определить (на основе избранных вами способов инициализации переменных) действительные типы значений, содержащихся в объектах `Variant`.

Если в ходе решения поставленной задачи потребуется функция, возвращающая массив, обратитесь к типу `Variant`. В листинге 12.8 демонстрируется соответствующий пример.

## Листинг 12.8. Пример построения функции, возвращающей массив типа `Variant`

```
1: Function GetMyArray( ) As Variant
2:   Static Data (100) As Integer
3:   GetMyArray = Data
4: End Function
5:
6: Sub TestReturnArray( )
7:   Dim MyArray As Variant
8:   MyArray = GetMyArray( )
9:   If (IsArray(MyArray)) Then
10:     MsgBox Ubound(MyArray)
11:   End If
12: End Sub
```

### Анализ

Строки 1–4 листинга 12.8 содержат определение функции `GetMyArray`, которая объявляет и возвращает статический массив. Локальные переменные, которые должны быть использованы вне контекста их объявления, следует определять как статические, — в строке 2 мы так и поступили, чтобы гарантировать корректность обращения к данным после выхода из функции, в которой они объявлены. В строках 6–12 приведен пример подпрограммы, которая демонстрирует использование массива, возвращенного функцией `GetMyArray`.

В данном случае можно говорить еще об одном ходе: зная, что данные нам понадобятся, мы объявили их не заблаговременно, а именно тогда, когда это оказалось необходимым — т.е. внутри функции `GetMyArray`. На самом деле процедура размещения и инициализации массива — ни что иное, как конструктор. *Конструктор* — термин объектно-ориентированного прогаммирования, и в данной части книги используется нечасто. Конструктор — это специальный метод класса, который выполняет размещение и инициализацию класса (подробнее см. главу 21).

# Что еще следует знать о массивах

Массивы просты в использовании, но не достаточно надежны. Чтобы ладить с ними, вы должны ясно осознавать и четко выполнять несколько несложных правил.

Прежде чем обратиться к элементу массива по индексу, вы должны гарантировать "попадание" последнего в допустимый интервал. Функции Lbound и Dbound — вот ваши надежные помощники.

При использовании глобальных массивов создайте отдельные функции для выполнения операции изменения их размеров и проверки правильности индексов. Хотя подобные функции окажутся небольшими по объему, ваша программа приобретет ясность и управляемость — ведь один фрагмент кода исправить гораздо легче, чем несколько, верно?

## Сортировка данных

Массивы — простой и удобный инструмент хранения и логической организации данных. Одна из наиболее распространенных операций, выполняемых над элементами массивов, — это *сортировка*. Существует целый ряд алгоритмов и методов сортировки. В их числе достаточно назвать метод "пузырька", сортировку посредством выбора и "быструю сортировку".

Проблемам сортировки посвящено достаточное количество статей, монографий и учебников. "Классикой жанра" считается *Numerical Recipes in C: The Art of Scientific Computing* Уильяма Пресса (William H. Press) (издательство Cambridge University Press, 1993). Названная работа содержит исчерпывающий свод подробных описаний самых разнообразных алгоритмов и их реализаций на языке программирования C. Существуют и публикации, которые специально ориентированы на читателя, нуждающегося в готовых решениях на языке Visual Basic. Например, обратившись к книге *Ready-To-Run Visual Basic Algorithms* Рода (Rod) и Кеннета (Kenneth) Стефензов (Stephens) (издательство John Wiley & Sons, 1998), вы сможете пользоваться приведенной информацией непосредственно, не прибегая к исправлению текста функций или его трансляции с одного языка программирования на другой. (В числе лучших книг следует упомянуть всемирно известную монографию Дональда Кнута (Donald E. Knuth) *Искусство программирования*, т.3. *Сортировка и поиск*. — М.: Изд. дом "Вильямс", 2000. — Прим.перев.).

На страницах нашей книги представлено три самых популярных, достаточно простых и чрезвычайно эффективных алгоритма сортировки. При нынешних скоростях процессоров, подбирающихся к гигагерцовым отметкам, даже старенький метод "пузырька" демонстрирует приличные результаты на объемах данных, исчисляемых десятками тысяч элементов. Впрочем, с увеличением размеров массивов различия в производительности алгоритмов "пузырька" и выбора, с одной стороны, и метода "быстрой сортировки", с другой, существенно возрастают.

## Сортировка методом „пузырька“

Алгоритм "пузырька" (Bubble Sort) — неплохой выбор в том случае, если объем массива не превышает 10000 элементов и данные уже в достаточной мере отсортированы. Если говорить кратко, метод состоит в сопоставлении содержимого каждой пары элементов массива и обмене их местами в случае успешного результата сравнения. При выполнении сортировки по возрастанию значение элемента с индексом  $i$  сопоставляется с содержимым следующего элемента —  $i+1$ . Если первая величина превосходит вторую, элементы меняются местами. Чтобы изменить порядок сортировки (т.е. получить массив, упорядоченный по убыванию), достаточно вместо оператора сравнения "больше" ( $>$ ) использовать оператор "меньше" ( $<$ ). Пример процедуры сортировки, реализующей метод "пузырька", приведен в листинге 12.9.

## Листинг 12.9. Пример сортировки массива с помощью метода "пузырька"

```
1: Sub Swap(ByRef Data() As Long, ByVal I As Long, ByVal J As Long)
2:   Dim Temp As Long
3:   Temp = Data(I)
4:   Data(I) = Data(J)
5:   Data(J) = Temp
6:   Debug.Print "Меняем местами " & Data(I) & " и " & Data(J)
7: End Sub
8:
9: Sub BubbleSort (ByRef Data() As Long)
10:  Dim I As Long, J As Long
11:  For I = Lbound( Data ) To Ubound( Data ) - 1
12:    For J = I + 1 To Ubound( Data )
13:      If (Data(I) > Data(J)) Then
14:        Call Swap ( Data, I, J )
15:      End If
16:    Next J
17:  Next I
18: End Sub
19:
20:
21: Sub FillArrayAndSort()
22:
23:   Const Size = 10
24:   Dim Data(Size) As Long
25:   Dim I As Long
26:   Randomize Time
27:   For I = LBound(Data) to UBound(Data)
28:     Data(I) = Rnd * Size
29:   Next I
30:
31:   Debug.Print "Начало работы: " & Time
32:   Call BubleSort(Data)
33:   Debug.Print "Конец работы: " & Time
34:
35: End Sub
```

### Анализ

Для выполнения сортировки по методу "пузырька" необходимо иметь два целочисленных индекса --  $I$  и  $J$ . Строка 11 содержит заголовок внешнего цикла, а строка 12 -- внутреннего. Чтобы сопоставить значение каждого  $i$ -го элемента массива с содержимым  $i+1$ -го элемента, необходимы именно два цикла. Обратите внимание на начальное значение переменной внутреннего цикла --  $J = I + 1$  (строка 12). Строка 13 выполняет сравнение значений элементов  $Data(I)$  и  $Data(J)$ . В случае удачи (когда значение предыдущего элемента окажется большим очередного последующего) элементы меняются местами -- эта операция выполняется с помощью процедуры `Swap`. Чтобы протестировать подпрограмму сортировки, выполните процедуру `FillArrayAndSort`.



Следует отметить, что метод "пузырька" не заслуживает высокой оценки. Его частое использование объясняется исключительной простотой. Столкнувшись с задачей сортировки, вы должны непременно уделить время изучению характеристики данных и выбору алгоритма, который в наибольшей мере удовлетворяет конкретным условиям и требованиям быстрой работы программы.

Если необходимо отсортировать данные других типов, придется внести в текст процедуры незначительные изменения, связанные с типом передаваемого массива. Все остальное останется в силе. Следует также обратить ваше внимание на значение верхней границы внешнего цикла, `Ubound(Data) - 1`. Последний элемент массива не должен охватываться внешним циклом, поскольку он проверяется внутренним.

Процедура сортировки методом "пузырька" в изложенной выше редакции, примененная к массиву из 10000 целых чисел, выполняется на компьютере Intel Pentium 800 с 256 Мбайт оперативной памяти приблизительно 51 секунду. Возможно, это мало о чем говорит, но подобная информация вам пригодится, так как такая скорость работы вполне может устроить. В терминах теории вычислительной сложности алгоритмов степень эффективности метода "пузырька" оценивается функцией  $O(n^2)$ . Другими словами, время решения задачи — это квадратичная функция от ее "размерности" ( $n$  в данной ситуации равно числу элементов массива), поскольку в худшем случае программе предстоит выполнить  $n^2$  операций сравнения. При размерности задачи, равной 10000, количество операций сравнения ограничено величиной 100000000. Ясно, что при дальнейшем росте  $n$  вам вряд ли поможет даже самый производительный компьютер.

## Сортировка посредством выбора

Для алгоритма *сортировки посредством выбора* (Selection Sort) справедлива та же оценка эффективности, зависящая от размерности задачи, что и для метода "пузырька", —  $O(n^2)$ . Отличие метода сортировки посредством выбора состоит в том, что перестановка элементов массива выполняется только в случае, когда найдена точная позиция текущего элемента относительно остальных. Метод "пузырька" же предполагает перемещение элементов при любом успешном результате теста сравнения.

Листинг 12.10 содержит текст процедуры, реализующей алгоритм сортировки посредством выбора. Как и в предыдущем примере, представлены неплохие результаты при числе элементов, не превосходящем 10000, но с ростом размерности задачи показатели производительности заметно ухудшаются.

Листинг 12.10. Пример сортировки массива посредством выбора

```
1: Sub SelectionSort(ByRef Data( ) As Long)
2:   Dim I As Integer, J As Integer, SwapIndex As Integer
3:   For I = Lbound(Data) To Ubound(Data) - 1
4:     SwapIndex = I
5:     For J = I + 1 To Ubound( Data )
6:       If (Data(SwapIndex) > Data(J)) Then
7:         SwapIndex = J
8:       End If
9:     Next J
10:    Call Swap(Data, I, SwapIndex)
11:  Next I
12:End Sub
```

### Анализ

На том же компьютере на сортировку 10000 числовых элементов с помощью алгоритма выбора было затрачено около 19 секунд — наличие повышение эффективности работы примерно на 40%. Рост производительности существенно зависит от числа реально выполняемых перемещений элементов, что, в свою очередь, определяется характеристиками относительной упорядоченности исходного множества данных. Алгоритмы отличаются незначительно. Обратите внимание на дополнительную переменную `SwapIndex`, объявленную в строке 2. В строке 4 она получает текущее значение `I`. В строке 7 вместо "слепого" перемещения элементов запоминается текущее значение индекса `J`, для которого тест сравнения дал **положительный** результат. На очередном шаге внутреннего цикла с `J`-м элементом будет

сопоставляться уже не  $I$ -й, как прежде, а тот, номер которого хранится в `SwapIndex`. Операция перемещения вынесена за пределы внутреннего цикла и поэтому выполняется только один раз на каждом шаге внешнего цикла, т.е. всего  $n$  раз вместо потенциальных  $n^2$  раз в худшем случае.

## Быстрая сортировка

Степень эффективности алгоритма *быстрой сортировки* (Quick Sort) оценивается функцией  $O(n * \ln(n))$ , где  $\ln(n)$  — натуральный логарифм от числа  $n$ . Кривая натурального логарифма с увеличением значения аргумента растет гораздо медленнее, чем, скажем, парабола квадратичной функции.

Алгоритм быстрой сортировки основан на подходе, обозначаемом в комбинаторной математике красочным классическим изречением *разделяй-и-властвуй* (divide-and-conquer). Сортируемый массив делится на части, которые анализируются и при необходимости вновь подвергаются делению. При значениях данных, близких к случайным, исходное множество делится приблизительно пополам. Если степень упорядоченности исходных данных более высока, количество операций деления множества на подмножества увеличивается, что приводит к ухудшению характеристик быстродействия алгоритма. Листинг 12.11 демонстрирует пример реализации алгоритма быстрой сортировки с помощью рекурсивной процедуры Quicksort.

Новый термин

*Рекурсивной* называется процедура или функция, которая в процессе выполнения ссылается сама на себя.

### Листинг 12.11. Пример реализации алгоритма "быстрой сортировки"

```

1: Sub QuickSort( ByRef Data( ) As Long, ByVal Left As Long,
      ByVal Right As Long )
2:
3:   Dim I As Long, J As Long
4:   Dim Elem As Long
5:
6:   If (Right > Left) Then
7:
8:     Elem = Data( Right )
9:     I = Left - 1
10:    J = Right
11:
12:    Do While (True)
13:      Do
14:        I = I + 1
15:        Loop While (Data(I) < Elem)
16:
17:      Do
18:        J = J - 1
19:        If (J < LBound(Data)) Then GoTo Break
20:        Loop While (J >= LBound(Data) And Data(J) > Elem)
21: Break:
22:
23:     If (I >= J) Then Exit Do
24:     Call Swap( Data, I, J )
25:     Loop
26:
27:     Call Swap( Data, I, Right )
28:     Call QuickSort( Data, Left, I - 1 )
29:     Call QuickSort( Data, I + 1, Right )
30: End If
31: End Sub

```

```

32:Sub FillArrayAndSort( )
33:
34:  Const Size = 10 '
35:  Dim Data(Size) As Long
36:  Dim I As Long
37:  Randomize Time
38:  For I = Lbound( Data ) To Ubound( Data )
39:      Data(I) = Rnd * Size
40:  Next I
41:
42:  Call Dump(Data)
43:
44:  Debug.Print "Начало: " & Time
45:  Call QuickSort( Data, Lbound( Data ), Ubound( Data ) )
46:  Debug.Print "Конец: " & Time
47:  Call Dump( Data )
48:
49:End Sub
50:
51:
52:Sub Dump(ByRef Data( ) As Integer)
53:  Dim Elem As Variant
54:  For Each Elem In Data
55:      Debug.Print Elem
56:  Next Elem
57:End Sub
58:
59:Sub Swap( ByRef Data( ) As Long, ByVal I As Long, ByVal J As Long )
60:  Dim Temp As Long
61:  Temp = Data(I)
62:  Data(I) = Data(J)
63:  Data(J) = Temp
64:  Debug.Print "Меняем местами " & Data(I) & " и " & Data(J)
65:End Sub

```

## Анализ

Процедура Quicksort, текст которой размещен в строках 1–30 листинга 12.11, демонстрирует высокие показатели производительности даже на больших объемах данных — так, целочисленный массив размером 100000 элементов, упорядоченных случайным образом, был отсортирован на том же компьютере менее чем за 1 секунду, на сортировку 1 миллиона чисел ушло приблизительно 10 секунд, а сортировка массива с 10 миллионами элементов потребовала 2 минуты и 4 секунды.

Функция Dump используется для вывода содержимого сортируемого массива в окне Immediate. Впрочем, ею не рекомендуется пользоваться, если количество элементов массива превышает несколько сотен.

В строках 59–65 размещен текст процедуры Swap, обращения к которой содержатся в листингах 12.9, 12.10 и 12.11. Цель наших действий проста: создать копию значения одного элемента массива, а затем поменять содержимое двух элементов местами.

Процедура FillArrayAndSort создает массив, заполняет его случайными значениями, вызывает процедуру сортировки и выводит на экран справочные данные о времени, затраченном на решение задачи. С ее помощью можно легко протестировать рассмотренные выше процедуры сортировки методом "пузырька" и выбора, если заменить выражение в строке 49 соответствующим вызовом.

Строки 1–30 содержат процедуру Quicksort — это характерный случай, когда размер кода подпроцедуры может превышать пределы нескольких строк. (В принципе, процедуру Quicksort нетрудно и сократить, если вынести цикл, охватывающий строки 12–24, в отдельный именованный блок.) Интерфейс подпроцедуры (строка 1) описывается тремя па-

раметрами — именем массива и текущими значениями его нижней и верхней границ. Поскольку реализация алгоритма предполагает разбиение исходного множества на два подмножества с последующей рекурсией, аргументам присвоены имена `Left` (индекс левой границы рассматриваемой части массива) и `Right` (индекс правой границы). В строке 3 объявлены целочисленные индексные переменные `I` и `J`. Строка 4 содержит выражение объявления переменной, тип которой совпадает с типом сортируемого массива; она используется в нескольких местах кода для хранения содержимого "граничного" элемента, определяющего порядок разбиения множества на подмножества.

В строке 6 первым делом выполняется проверка того, действительно ли значение правой границы (`Right`) превышает величину левой (`Left`). В строке 8 сохраняется граничное значение, определяющее разбиение массива на части. Далее индексу `I` присваивается значение `Left - 1`, а переменной `J` — величина, хранимая в `Right`. Бесконечный цикл, охватывающий строки 12–24, используется для нахождения точек деления левой (строки 13–15) и правой (строки 17–20) половин текущего подмножества. Если значение `I` достигает величины `J` или превышает ее, выполнение цикла завершается; в противном случае элементы массива, адресуемые индексными значениями `I` и `J`, меняются местами. Процесс повторяется до тех пор, пока не выполнится условие `I >= J`. Далее в строке 26 вновь вызывается процедура `Swap`, а затем следуют рекурсивные обращения к процедуре `Quicksort` для дальнейшей обработки левой и правой половин текущего подмножества данных.

Существуют еще более сложные алгоритмы сортировки, учитывающие особенности конкретных типов и разновидностей данных. Однако их исчерпывающий анализ выходит за рамки предмета нашего обсуждения. При необходимости более подробную информацию вы сможете найти в книге Роберта Сэджевика (*Robert Sedgwick*) *Algorithms in C++* и в ряде других публикаций, упомянутых выше. Учтите, что во многих случаях вам придется переводить тексты примеров на язык VBA. К счастью, бизнес-приложения, для создания которых и предназначен VBA, отнюдь не исчерпываются задачами сортировки данных. Кроме того, проявив определенную настойчивость, вы наверняка сможете отыскать на Web-сайтах множество готовых решений.

## Резюме

Массивы — это мощные и эффективные структуры данных, способные сохранить миллионы единиц информации и обеспечить удобные средства управления ею. Элементами массивов могут служить значения стандартных и пользовательских типов данных, а также объекты классов.

Главная черта массивов — простота их использования. Массивы настолько доступны, что могут применяться программистами любого уровня квалификации. Впрочем, существуют и отрицательные стороны. В ходе занятия вы усвоили, что обязательными условиями успешного применения массивов служат действия, связанные с объявлением индексных переменных и контролем за допустимыми границами их изменения. Если подобные операции станут неотъемлемой частью создаваемых вами процедур, работа с массивами окажется значительно более эффективной и надежной.

Соединение нескольких идей и приемов воедино — это, пожалуй, одна из главных концепций, лежащих в основе объектно-ориентированного подхода к программированию. Решив обратиться к массивам, доверяйте собственному опыту и интуиции, но не ограничивайте свое творческое воображение только рамками массивов. Следующее занятие посвящено изучению более современной разновидности массивов — коллекций. А пока обратитесь к приведенным ниже разделам "Вопросы и ответы" и "Задания", чтобы отшлифовать полученные знания.



# Вопросы и ответы

**Вопрос.** Можно ли создать массив для хранения элементов различных типов данных?

**Ответ.** Да. Объявите массив с указанием типа Variant, и вы сможете сохранять в нем данные различных типов.

**Вопрос.** Необходимо ли при завершении работы с массивом использовать функцию Erase?

**Ответ.** Делать это вовсе не обязательно. Но если вы поступаете именно так, будьте последовательны в своих действиях.

**Вопрос.** Насколько велик объем памяти, доступный для объявления массива внутри функции?

**Ответ.** Еще несколько лет назад, когда не существовало мощных 32-разрядных операционных систем, таких как Windows 2000, допустимый размер массива ограничивался объемом стека — специальной области оперативной памяти. Но теперь программисту позволено объявлять внутри функции или процедуры весьма широкие массивы.

**Вопрос.** Существуют ли более эффективные алгоритмы сортировки, нежели рассмотренные методы "пузырька", выбора или "быстрой сортировки"?

**Ответ.** Алгоритм "быстрой сортировки" теоретически обладает наилучшей оценкой поведения, однако с учетом свойств конкретных наборов сортируемых данных могут быть построены и более эффективные алгоритмы. Если время работы вашей программы имеет серьезное значение, необходимо проанализировать данные и по возможности оптимизировать алгоритм.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Как называется функция, позволяющая построить и вернуть массив?
2. Какая функция используется для инициализации элементов массива значением, равноценным null?
3. Действует ли функция Erase одинаково в отношении массивов данных различных типов?
4. Каким образом можно динамически, т.е. в ходе выполнения программы, изменить размер массива?
5. Какой из алгоритмов сортировки более эффективен — метод "пузырька", выбора или "быстрой сортировки"?

## Упражнения

1. Напишите выражение сравнения строк для использования в процедуре сортировки.
2. Исправьте текст процедуры Dump таким образом, чтобы вывод данных осуществлялся в файл. Почему лучше выносить подобный код в отдельный именованный блок (функцию или процедуру), а не располагать его непосредственно в том месте программы, где он необходим?
3. Внесите изменения в текст процедуры BubbleSort, чтобы осуществить сортировку данных в порядке их убывания. (Имейте в виду: в листинге 12.9 предполагается вариант сортировки по возрастанию.)



## 13-й час

# Коллекции данных

На протяжении многих лет университетские курсы лекций, посвященные структурам данных, сводились к изложению приемов управления динамическими массивами. Динамический массив — чрезвычайно простая и доступная структура данных (за информацией обращайтесь к предыдущей главе, "12-й час. Управление данными переменного объема"). В начале 90-х была разработана концепция объектно-ориентированного программирования. С тех пор во многих системах программирования, включая и VBA, арсенал пополнился средствами создания классов и объектов. Для нас с вами это означает возможность многократного использования однажды созданного кода, собственного или заимствованного.

*Коллекции* — одно из достижений объектно-ориентированного подхода. Коллекция представляет собой разновидность динамического массива, оформленного в виде стандартного класса VBA. Сегодня коллекции способны послужить более удачным вариантом выбора во всех ситуациях, где прежде применялись массивы. В образцах кода, написанного ранее, вы все еще сможете встретить примеры обращения к массивам. Но в современных приложениях чаще употребляются коллекции.

Основные темы занятия.

- Знакомство с коллекциями.
- Объявление объектов коллекций.
- Применение стандартных средств управления коллекциями.
- Примеры использования коллекций.

## Знакомство с коллекциями

Коллекция — это индексированная структура данных, обладающая встроенными возможностями добавления, удаления элементов и их итеративной обработки. Элементами коллекции могут служить единицы данных тех же типов, которые допустимы при использовании массивов. Коллекция способна содержать данные стандартных и пользовательских типов, а также объекты классов. Коллекции могут быть как однородными, так и неоднородными — т.е. охватывать элементы данных различных типов.

Дополнительные возможности, предоставляемые коллекциями, — это встроенные средства изменения размеров и динамической обработки ошибок.

Словом, коллекции — одна из самых мощных и универсальных структур данных, имеющих в распоряжении программиста, применяющего VBA.

## Использование коллекций

Коллекции находят самое широкое применение — в стандартных визуальных компонентах (например, раскрывающиеся списки), в объектах класса `ADODB` для представления полей, в `ADOX` для хранения информации о таблицах и ключах базы данных и т.д. Вам непременно потребуется изучить приемы обращения с коллекциями, чтобы научиться работать с визуальными компонентами и объектами ADO.

На страницах этой книги, мы уже неявно обращались к коллекциям, когда использовали объекты класса `ADODB.Recordset`. Таким образом, коллекции — это настолько важная тема, без изучения которой всем нам просто не обойтись.

## Термины объектно-ориентированного программирования

В составе библиотек VBA имеется код, определяющий существо коллекции. Он носит название класса `Collection`. Воспринимайте класс как формализованное описание того, какие данные и какие действия могут поддерживаться и выполняться фрагментом (объектом) кода.

Тип данных `Integer`, например, "осведомлен" о возможностях выполнения операций с целыми числами. Класс же способен содержать в своем составе переменные различных типов и "знать", как обращаться с каждой из них. Но класс, кроме того, "умеет" что-то делать. Созидательная способность заключена в наборе процедур и функций, объявленных в составе класса.

Именованный блок кода, содержащий формализованные описания собственных "знаний" и "умений", называют *классом*. Создаваемый программистом экземпляр (переменная) класса носит название *объекта*.

Определенные в составе класса переменные, процедуры и функции называют общим термином *атрибуты*. Существует две разновидности атрибутов. Атрибуты данных — это *свойства*. Свойства — это те же переменные, за исключением того, что при обращении к свойству необходимо обязательно указывать наименование объекта (экземпляра), которому оно принадлежит. Вторая разновидность атрибутов — *методы* — это объявленные в составе класса процедуры и функции. Отличие методов от обычных процедур или функций заключается в том, что они существуют только в составе класса и полностью ему принадлежат.

Процесс и результат объявления атрибутов класса называется *инкапсуляцией*. Вы можете, например, услышать такое выражение: "Класс инкапсулирует объявленные в его составе свойства и методы". До появления объектной концепции информатика следовала парадигме *структурного программирования*. Табл. 13.1 представляет удобную сопоставительную схему терминов структурного и объектного программирования.

Новый термин

*Парадигма* — это, в контексте нашего разговора, набор общих приемов или стиль программирования. Например, объектно-ориентированное программирование — это парадигма (концепция), в соответствии с которой при декомпозиции (разделении) задачи на более мелкие подзадачи во главу угла ставится объект, а не процесс.

Таблица 13.1. Сопоставительная схема терминов структурного и объектного программирования

Термин структурного подхода	Термин объектного подхода	Значение
Переменная	Свойство	Элемент данных
Функция или подпрограмма	Метод	Фрагмент кода

Новые термины введены не для того, чтобы запутать вас. Они несут в себе как прежнюю, так и новую смысловую нагрузку. Например, термин *переменная* служит для обозначения хранимой порции данных, но в слове *свойство*, помимо прямого толкования, появляется дополнительное значение, подчеркивающее принадлежность данных объекту класса.

Не следует углубляться в проблемы абстрактной семантики — указанные термины нужно просто отчетливо понимать и различать. Это поможет в работе, да и общение с коллегами значительно упростится, если вы будете говорить с ними на одном языке.

Класс *Collection* содержит четыре атрибута. Свойство *Count* предназначено для хранения количества элементов коллекции. Метод *Add* позволяет добавлять в коллекцию новые элементы, *Remove* служит для удаления элементов, а *Item* дает возможность ссылаться на элементы по значению индекса.

В ходе этого занятия будут подробно рассмотрены все названные атрибуты класса *Collection*.

## Создание объектов коллекций

Мы не погрешим против истины, если в каком-то смысле отнесем классы к типам данных, объявляемым пользователем. Коллекции — это тип данных, разработанный специалистами Microsoft и включенный в состав VBA. Класс коллекций содержит четыре атрибута, перечисленных в предыдущем разделе.

Средства поддержки парадигмы объектно-ориентированного программирования были включены в окружение VBA в последние годы. Конструкции и приемы определения объектов классов в VBA несколько отличаются от тех, которые применяются по отношению к стандартным и пользовательским типам данных, рассмотренным ранее. (Сейчас, рассматривая способы объявления объектов, мы имеем в виду класс коллекций, хотя все сказанное далее справедливо и для других классов.) Синтаксис объявления коллекции таков:

```
Dim МояКоллекция As Collection
```

Нетрудно заметить, что внешне ничего не изменилось. Замените идентификатор *МояКоллекция* более точным именем — и процесс объявления будет завершен. Впрочем, существует одна важная особенность — непосредственно после объявления объекта соответствующая ему переменная содержит значение *Nothing* (*ничего*), равноценное *null* или *Empty*, которыми инициализируются стандартные типы данных. Поэтому сразу воспользоваться объектом вы не сможете.

Чтобы объект стал полноценным, соответствующей ему переменной необходимо присвоить значение существующего объекта либо создать новый, выделив память с помощью директивы *New*. При создании нового объекта возможны два варианта действий. Команду *New* допускается использовать непосредственно в строке объявления:

```
Dim МояКоллекция As New Collection
```

Существует также альтернативное решение: вначале просто объявите объект, а затем выделите память для него посредством команды New и операции присваивания:

```
Dim МояКоллекция As Collection  
Set МояКоллекция = New Collection
```

Обратите внимание на наличие служебного слова Set. Операции присваивания, аргументами которых служат объекты классов, требуют обязательного использования слова Set, которое служит характерным признаком действий над объектами. Если вы забудете указать Set, откроется окно сообщения об ошибке (рис. 13.1).

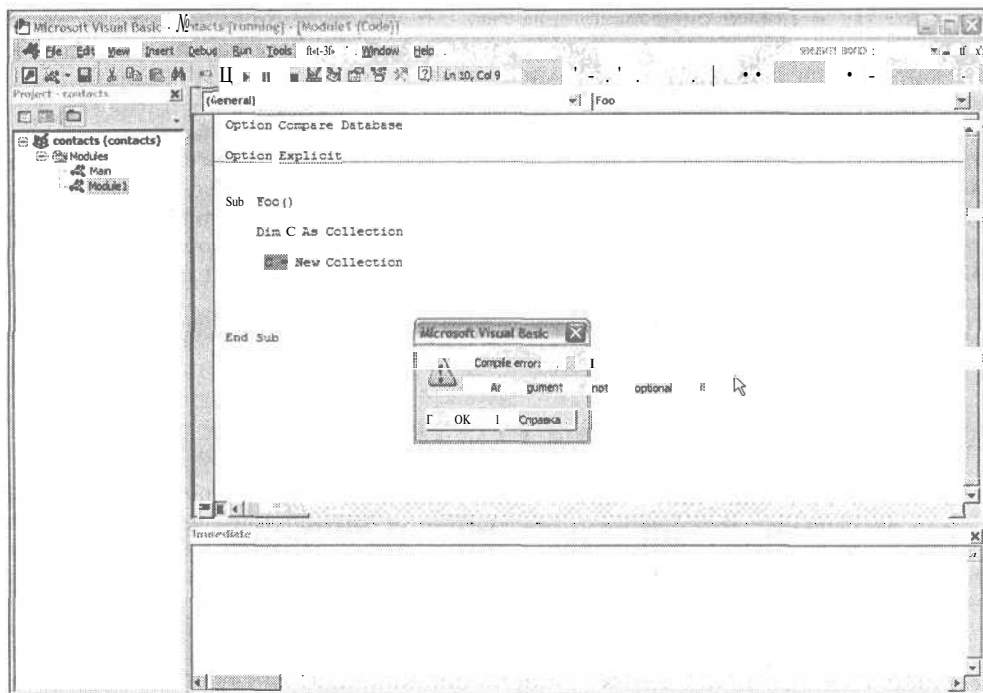


Рис. 13.1. Забыв ввести служебное слово Set в операторе присваивания объектов, вы получите сообщение об ошибке

После завершения работы с объектом следует очистить выделенную для него память. Подобная операция выполняется таким образом:

```
Set МояКоллекция = Nothing
```

Служебное слово New появилось в языках C и C++ для резервирования фрагмента общей области динамически распределяемой памяти (называемой термином *куча* — Heap). Воспринимайте *Heap*-память как часть оперативной памяти компьютера, которую операционная система Windows способна выделить вашей прикладной программе. Объекты занимают больше памяти, нежели переменные простых типов. Способ создания объекта посредством двух строк кода — объявления с последующим использованием команды New — удобен в тех ситуациях, если память надлежит выделить лишь в тот момент, когда объект действительно необходим.

Представьте себе программу, которая в одночасье израсходовала всю доступную ей память — некоторым приложениям во время их работы требуются многие миллионы байтов. Если вся память исчерпана, непрофессионально написанные программы просто "рушатся".

Но не будем о грустном. Сейчас вам достаточно запомнить, что объект коллекции (и любой другой) может быть реально создан либо одновременно с его объявлением, либо позже, с помощью отдельной команды.

## Управление коллекциями данных

Коллекция — это "умный" динамический массив, который наделен способностями автоматического пополнения. Свойство Count дает ответ на вопрос, сколько элементов содержится в коллекции. Методы Add и Remove позволяют накапливать данные в коллекции и удалять их. Доступ к отдельным элементам коллекции осуществляется с помощью метода Item или цикла For Each . . Next.



Действует общепринятое соглашение, согласно которому объекты коллекций называют существительными во множественном числе, производными от наименований типов элементов. Например, коллекцию элементов типа String уместно назвать *Strings*.

## Добавление элементов в коллекцию

Синтаксис

Для пополнения коллекции применяется метод Add. Синтаксис его вызова таков:

```
Call ИмяОбъектаКоллекции.Add( Элемент [, КлючевоеИмя, [До |  
, После]] )
```

ИмяОбъектаКоллекции — это, как вы догадываетесь, имя объекта коллекции, а Add — название метода класса Collection. Идентификатор объекта отделен от наименования метода оператором точки. (Служебное слово Call не является обязательной частью конструкции вызова процедуры. Если слово Call все-таки употреблено, список аргументов процедуры — в том числе пустой — должен быть заключен в круглые скобки. Если же слово Call отсутствует, скобки не применяются.) Основная форма вызова метода Add предполагает задание единственного аргумента Элемент. В качестве значения параметра допустим литерал, переменная, выражение или наименование объекта. Например, если указать на месте аргумента элемент выражение  $A = 5$ , в коллекцию будет занесено значение типа Boolean, равное True при значении переменной A, равном 5, и False — в противном случае. Если Элемент — единственный аргумент в списке параметров процедуры Add, заданный элемент займет в коллекции свободное место.

Параметр КлючевоеИмя позволяет дать элементу, добавляемому в коллекцию, дополнительный псевдоним. Элемент, наделенный псевдонимом, допускает обращение как по индексу, так и по имени. (Подробности приведены ниже, в разделе "Метод Item и свойство Count".) Следующие по порядку необязательные параметры — это До и После (в определенный момент может быть задан только один из них). Если указан аргумент До, новый элемент будет размещен в коллекции непосредственно *перед* элементом, индекс или псевдоним которого задан параметром До. Аргумент После позволяет определить номер или имя элемента, *за* которым следует разместить новый. В конструкции вызова метода Add аргумент После отличается от До наличием дополнительной предшествующей запятой. Листинг 13.1 демонстрирует несколько примеров использования метода Add класса Collection.

### Листинг 13.1. Примеры использования метода Add класса Collection

```
1: Sub CollectionDemoAdd( )
2:     Dim Strings As New Collection
3:     ' Вызов со скобками
4:     Call Strings.Add( "1" )
5:     ' Вызов без скобок
6:     Strings.Add "2"
7:     Call Strings.Add( "3", "три", , 2 )
8:     Call Strings.Add( "4", "четыре", "три" )
9:     MsgBox Strings.Item( "три" )
10:    MsgBox Strings.Item( 1 )
11:    Dim I As Integer
12:    For I = 1 To Strings.Count
13:        Debug.Print Strings.Item( I )
14:    Next I
15:    Set Strings = Nothing
16: End Sub
```



При вызове функции и присваивании возвращенного ею значения наличие скобок обязательно. Рекомендуется планомерно использовать скобки и при вызове функций, и при обращении к процедурам, хотя в последнем случае они необязательны.

#### Анализ

В строке 2 листинга 13.1 выполняются одновременное объявление нового объекта `strings` класса `Collection` и выделение памяти для него посредством команды `New`. Строка 4 содержит самый простой вариант вызова метода `Add`: процедуре передается единственный аргумент — значение первого элемента коллекции — `"1"`. В строке 6 выполняется аналогичная операция, только без использования круглых скобок: в коллекцию добавляется второй элемент, `"2"`. Вызов процедуры `Add` в строке 7 включает уже три аргумента — содержимое очередного элемента (`"3"`), его псевдоним (`"Три"`) и параметр 2, указывающий на номер элемента, *после* которого следует вставить новый (речь идет именно о параметре `После`, поскольку указана дополнительная запятая). Строка 8 предполагает вставку элемента `"4"`, снабженного именем `"Четыре"`, непосредственно перед элементом `"Три"`. Элементы `"3"` и `"4"` допускают обращение как с помощью значений индекса, так и посредством указания их псевдонимов. Строки 9 и 10 демонстрируют два варианта обращения к одному и тому же элементу, `"3"`, а строки 11–14 представляют конструкцию цикла, в котором содержимое каждого элемента коллекции `Strings` последовательно отображается в окне `Immediate` (рис. 13.2).

Строка 15 определяет директиву очистки фрагмента памяти, занятого объектом коллекции (хотя, отметим, делать это необязательно, поскольку компилятор VBA выполняет подобные функции автоматически). Строка 16 завершает процедуру.



Языки программирования Visual Basic for Applications и Java не требуют обязательного выполнения операций по освобождению фрагментов памяти, отведенных объектам. Компилятор решает подобные задачи, называемые сборкой мусора, автоматически. Впрочем, явное применение команды очистки памяти служит признаком завершения использования конкретного объекта и существенно уточняет и проясняет ваши намерения. Рекомендуем вам следовать таким принципам.

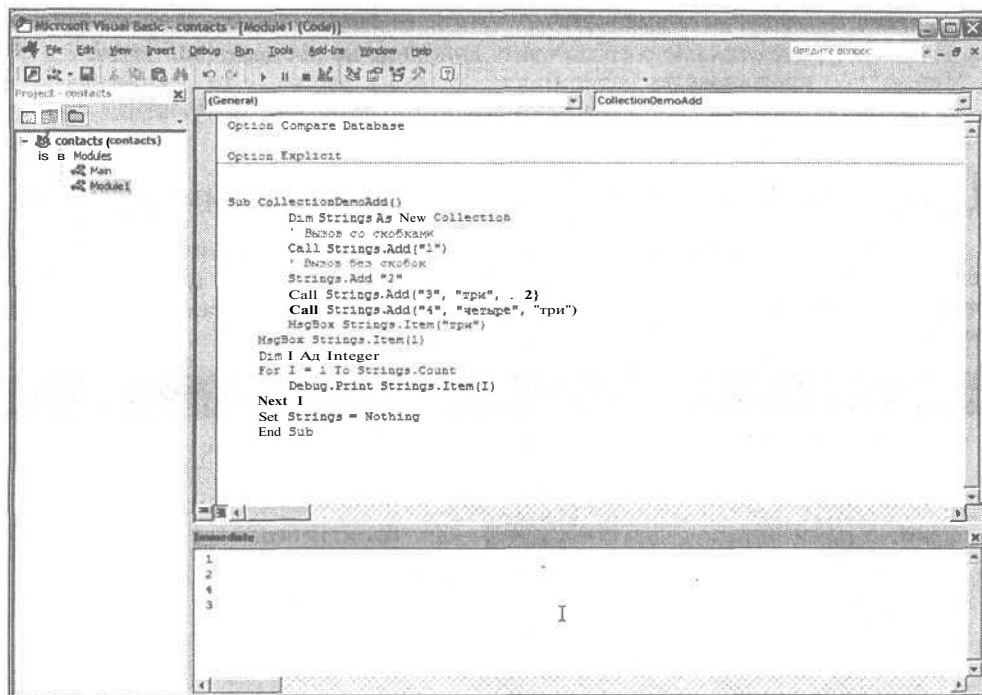


Рис. 13.2. Так выглядит содержимое коллекции, построенной с помощью процедуры листинга 13.1

## Удаление данных из коллекции

Синтаксис

Методу Remove передается единственный аргумент — значение индекса. В качестве индекса может задаваться целое число в интервале от 1 до значения, содержащегося в Count, либо строка, указывающая ключевое имя (псевдоним) элемента. Синтаксис вызова метода Remove таков:

```
Call ИмяОбъектаКоллекции.Remove ( Индекс )
```

(Обращаясь к процедуре, указывайте служебное слово Call и круглые скобки. Во всех примерах последующих глав это правило соблюдается.) Назначение имени ИмяОбъектаКоллекции пояснять не нужно. За ним, после оператора точки, следует название метода (Remove) и выражение индекса — целочисленное либо символическое. Если задано целое число, оно не должно выходить за границы интервала изменения индекса, а строка не может не совпасть с одним из ключевых имен элементов коллекции. (Пример использования метода Remove вы найдете ниже, в листинге 13.2.)

## Метод Item и свойство Count

Свойство Count содержит значение количества элементов коллекции.

На предыдущем занятии, изучая динамические массивы, предпринимались явные действия для установления размера массива и его увеличения. Аналогичные средства присутствуют и в классе коллекций. Подобное свойство объектов называют *сокрытием*



данных или сократим функциональности. Теперь можно не заботиться о мелких деталях реализации, а полностью сосредоточить внимание на поставленной задаче.

Метод `Item` предоставляет возможность доступа к элементу коллекции по заданному значению индекса или ключевого имени. О том, как пользоваться методом `Item`, вы узнаете, изучив текст листинга 13.2.

## Циклическая обработка элементов коллекции

С циклической конструкцией `For Each ... Next` вы ознакомились в главе "5-й час. Программирование управляющих структур". Коллекции — идеальный объект применения циклов `For Each`. Листинг 13.2 содержит примеры использования различных способов обработки данных в коллекции.

Листинг 13.2. Подробный пример использования коллекций

```
1 Sub CollectionDemo( )
2     Dim Strings As New Collection
3     Call Strings.Add( "Но в столовой" )
4     Call Strings.Add( "у кошки" )
5     Call Strings.Add( "даже" )
6     Call Strings.Add( "хлеба" )
7     Call Strings.Add( "ни" )
8     Call Strings.Add( "крошки!" )
9     Dim Str As Variant
10:    Dim Text As String
11:    For Each Str In Strings
12:        Text = Text & " " & Str
13:    Next Str
14:    Debug.Print Text
15:    Dim I As Integer
16:    For I = Strings.Count To 1 Step -1
17:        Debug.Print "Удаляем: " & Strings.Item(I)
18:        Call Strings.Remove(I)
19:    Next I
20:    Set Strings = Nothing
21:End Sub
```

### Анализ

I Строка 2 содержит конструкцию объявления коллекции `Strings` и выделения памяти для нее. Важно помнить, что элементами коллекции могут служить данные практически любых типов. В строках 3-8 выполняются операции добавления в коллекцию элементов типа `string`. В строке 9 объявляется переменная `str` типа `Variant`, а в строке 10 — символьная переменная `Text`. Цикл `For Each` требует использования переменной типа `Variant`. `Str` употребляется в роли приемника данных из очередного элемента коллекции, а строка `Text` в результате получает "сумму" всех элементов (фразу из рассказа Мыши, персонажа повести Льюиса Кэрролла *Приключения Алисы в стране чудес*) и затем (см. строку 14) отображается в окне `Immediate`.

Строки 16-19 содержат цикл, на каждом шаге которого удаляется очередной элемент, начиная с последнего. Перед удалением элемента его содержимое отображается в окне `Immediate`. Команда в строке 20 выполняет очистку памяти.

Если необходимо удалить все элементы коллекции, используйте обратный цикл. (При удалении очередного элемента значение `Count` уменьшается на единицу.) В противном случае, используя `Count` в качестве верхней границы диапазона изменения ин-

декса, вы в какой-то момент получите сообщение об ошибке выхода индекса за допустимые пределы (см. рис. 12.2 в главе 12). К сожалению, для удаления элементов коллекции цикл `For Each ... Next` применять нельзя.

## Где еще применяются коллекции

Наилучший способ распознавания коллекций в числе других объектов: поиск названий — имен существительных во множественном числе. Часто (об этом уже говорилось) коллекции называют именно так. Вот почему желательно и удобно следовать общепринятым соглашениям и правилам игры.

Встретив в тексте программы, отображаемой в окне редактора Microsoft Visual Basic, идентификатор, напоминающий имя коллекции, поместите в его пределах курсор и нажмите клавишу `<F1>`. Содержимое окна оперативной справочной системы поможет вам убедиться в справедливости своих предположений.

Коллекции находят широкое применение. Визуальные компоненты (например, раскрывающиеся или комбинированные списки) используют коллекции для хранения текстовых элементов. Объекты DAO и ADO содержат коллекции, хранящие информацию о таблицах, полях, ключах, столбцах и индексах баз данных. Существует и множество других примеров. Коллекции настолько же просты, насколько и мощны. Теперь, ознакомившись с коллекциями, вы сможете легко справиться с объектами любых других классов.

## Резюме

Коллекция — это, по существу, динамический массив (см. главу 12). Коллекция представляет собой стандартный класс VBA, используемый для хранения элементов простых или пользовательских типов данных, причем в любых сочетаниях.

Объекты намного мощнее, нежели фрагменты обычного кода. Удачно спроектированный класс предоставляет возможность сокрытия данных и функциональных средств, так что пользователь получает в свое распоряжение только ту часть интерфейса, которая действительно необходима. Если, работая над новым программным проектом, вы решите применить массивы, рекомендуем изменить намерения в пользу коллекций. Коллекции более просты в управлении, они способны динамически изменять свои размеры, не требуя от вас дополнительных строк кода.

Изучение коллекций — это удачный старт на пути к классам и объектам. О классах речь пойдет на следующих занятиях. А сейчас настоятельно советуем не пропустить разделы "Вопросы и ответы" и "Задания", чтобы еще более усорершенствовать полученные знания.

## Вопросы и ответы

**Вопрос.** Чем вы рекомендуете воспользоваться при создании нового программного проекта — массивами или коллекциями?

**Ответ.** Безусловно, более предпочтительны коллекции, хотя и массивы применяются довольно часто, особенно в старых программах.

**Вопрос.** Возможно ли расширить набор атрибутов класса `Collection`?

**Ответ.** Да. В теории и практике объектно-ориентированного программирования подобная возможность носит название *наследования*. Подробнее см. главу "21-й час. Основы программирования классов".

**Вопрос.** Обязательно ли, завершив работу с объектом класса, выполнять процедуру очистки памяти с помощью присваивания значения `Nothing`?

**Ответ.** Я сам это делаю, хотя среди ваших коллег наверняка найдутся те, кто считает такие действия излишними и возлагает ответственность на стандартные средства сборки мусора, которые имеются в составе VBA. Важно придерживаться единого стиля работы.

**Вопрос.** Все ли объекты коллекций работают одинаково?

**Ответ.** Да. Если вы встретите переменную, представляющую собой объект коллекции VBA, можете быть уверены в доступности всех базовых атрибутов, которые мы изучили, — `Count`, `Item`, `Add` и `Remove`. Однако в новых классах коллекций, построенных с использованием механизмов наследования, могут содержаться дополнительные свойства и методы.

## Задания

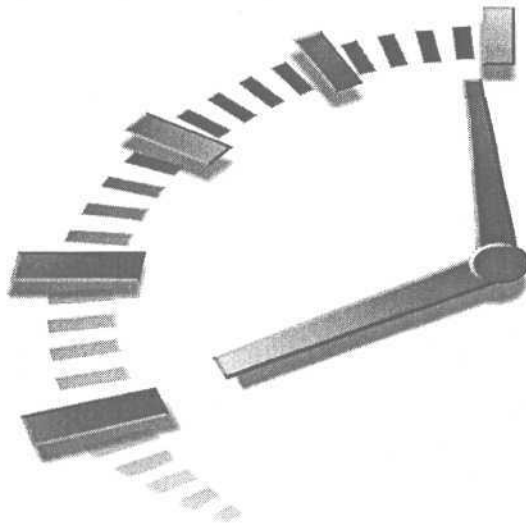
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Чем отличается метод от обычной функции (процедуры)?
2. Как называется метод, позволяющий добавлять в коллекцию новые элементы?
3. Каково назначение конструкции присваивания переменной-объекту значения `Nothing`?
4. Обязательно ли перед операцией очистки памяти, отведенной объекту коллекции, удалять все его элементы?
5. Назовите способы циклического прохождения по элементам коллекции.

## Упражнения

1. Создайте коллекцию и добавьте в нее 10 целочисленных значений.
2. Напишите процедуру, которая позволяет отобразить содержимое коллекции, построенной при выполнении предыдущего упражнения, в окне `Immediate`.
3. Исправьте текст процедуры сортировки по методу "пузырька" (см. главу 12) с учетом возможности обработки элементов коллекции.



# Часть V

## Программирование и базы данных Access

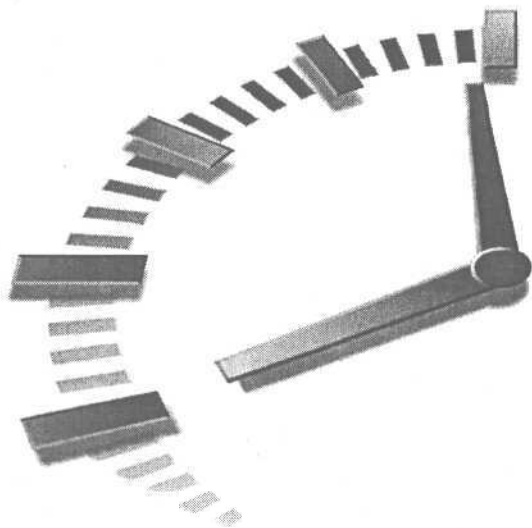
### Темы занятий

14-й час. Стил ь программирования: "Что такое хорошо, и что такое плохо"

15-й час. ADO DB — ваш верный помощник

16-й час. Применение языка SQL





## 14-й час

### Стиль

## программирования: „Что такое хорошо, и что такое плохо”

Поговорим о каше. Разве вы не понимаете, какое отношение этот славный продукт питания имеет к программированию? Тогда это просто везение, если вам не доводилось иметь дела с программным кодом, напоминающим беспорядочное месиво. Да, существует и такая неприглядная сторона профессиональной жизни. Собственно говоря, пенять не на что, поскольку компьютерная индустрия до сих пор не выработала четких критериев оценки стилистических качеств программного кода и не предложила единых правил игры.

Программисты до сих пор расходятся во мнениях, какой код следует считать хорошим, а какой — плохим. Автор этой книги написал первую строчку своей первой программы еще в 1978 году, но стал профессионально заниматься программированием только в 1987 году. В течение этих двенадцати лет он видел разное: от безобразных опусов, нацарапанных новичками-студентами, до безукоризненных, потрясающих, наилучших в мире программных творений. Как ни парадоксально, и те, и другие оказались в равной степени поучительными.

Задача написания удобного для чтения, стилистически грамотного кода одинаково важна как для программиста-любителя, в одиночку корпящего над выстраданной проблемой, так и для профессионала, работающего в коллективе единомышленников над сложным и долговременным проектом. Создавая код, вы становитесь автором, но возвращаясь к своей программе позже, чтобы внести необходимые исправления или воспользоваться какими-либо готовыми решениями, вы становитесь ее пользователем. Как показывает опыт, написанный код всего через несколько недель может показаться автору совершеннейшей ерундой. Поэтому материал, который рассматривается на этом занятии, окажется полезным даже в том случае, если ваш собственный продукт (как вы считаете) никогда не увидит ни одна живая душа. В какой-то момент он все-таки может кому-нибудь понадобиться — вам самому, вашему начальнику, коллеге или ученику, которому захочется узнать, как не следует писать программы.

Ниже рассказывается о приемах и соглашениях, которые помогут вам продлить жизнь программ и облегчить свою собственную. Считаю, что хорошо сделанная работа дает возможность поскорее приступить к новому и еще более интересному проекту, который — не будем лицемерить — наверняка отзовется хрустом свежих (честно заработанных!) купюр в кармане.

Основные темы занятия.

- Соглашения об именовании объектов программы.
- Отступы и комментарии — средства обеспечения удобочитаемости текста.
- Как упростить код и обеспечить его повторное использование.
- Советы по тестированию и отладке кода.

## Соглашения об именах

Степень удобства восприятия программного текста во многом зависит от качества принятой системы именования объектов. Если говорить об объеме работы по вводу текста, то усилия, требуемые для набора комментариев, с одной стороны, и строк кода — с другой, примерно равнозначны. В большинстве случаев код, в котором соблюдены правила именования, почти не нуждается в дополнительных примечаниях. Если же текст программы запутан и неоднозначен, комментарии, конечно же, необходимы — а это удвоит объем вашей работы. Недоработанный код — это настоящий кошмар: дальнейшее сопровождение такой программы потребует от вас огромных затрат времени, сил и интеллектуальной энергии. Подобных примеров немало, и они, увы, множатся — но это не значит, что вы, уважаемый читатель, должны им подражать.

Существует два подхода к вопросу о том, что представляет собой правильная система именования объектов программы. Обе точки зрения привлекают внимание определенных кругов приверженцев. Одна из школ отстаивает мнение о полезности системы, основанной на так называемой *префиксной нотации*, или, как ее часто называют, *венгерской нотации*. Такая концепция была предложена в начале 80-х Шарлем Симони (Charles Simonyi). Венгерская нотация предполагает разработку и использование системы префиксов, в сокращенной форме описывающих принадлежность переменных, объектов, функций и процедур определенному типу. Например, буква *i* может быть поставлена в соответствие типу `integer`, и тогда названию каждого объекта кода и данных, который имеет отношение к целочисленному типу, должен предшествовать символ *i*. Так, например, переменную типа `Integer`, предназначенную для хранения данных о возрасте человека, следовало бы назвать `iAge`. Можно привести и другие примеры с подробными пояснениями.

### Новый термин

*Слабо типизированным* называется язык программирования, компилятор или интерпретатор которого не выполняет строгих проверок соответствия значений аргументов типам переменных. Примером может служить такая ситуация: при объявлении целочисленной переменной компилятор не препятствует присваиванию ей значений других числовых типов. К слабо типизированным относится, например, язык C.

### Новый термин

*Строго типизированный* язык программирования не допускает неточностей в отношении трактовки типов переменных. Все значения должны обязательно соответствовать оговоренным типам переменных. Примером строго типизированного языка служит C++. VBA также можно отнести к строго типизированным языкам, хотя и не в такой степени "строгим", как большинство реализаций C++.

И в книге, и в повседневной практике я не придерживаюсь префиксной нотации — более того, мне вообще не нравится какая-либо нотация. Доводы против употребления венгерской нотации можно выдвинуть следующие.

- Многие из причин, обусловивших возникновение префиксной нотации, сегодня утратили свою актуальность. Нотация была впервые применена в слабо типизированном языке С, который не относится к объектно-ориентированным. Но большинство современных инструментальных сред программирования строго типизированы.
- Многие языки, разработанные после возникновения префиксной нотации, реализуют объектную концепцию, позволяя создавать необозримое многообразие новых классов. Получается, что для каждого класса придется внести соответствующий префикс. Как тогда вести разрастающийся список префиксов и четко следовать ему?
- Даже для стандартных классов не существует единого унифицированного списка префиксов, поэтому каждый тип можно называть по-разному, что отнюдь не решает проблему.



Если вы решили использовать венгерскую нотацию, создайте собственный (а лучше — коллективный) список префиксов и неукоснительно соблюдайте все правила.

Для обозначения переменных, классов, объектов, функций и процедур, вместо префиксной нотации можно применять понятные однозначные словосочетания — это второй общий способ обеспечить удобочитаемость программного текста. Полные слова и признаки контекста передают смысл сказанного достаточно точно, поэтому дополнительные пояснения чаще всего не требуются. Далее приведены простые правила именования, которых следует придерживаться.

- Целые, неусеченные, слова объясняют назначение объектов программы достаточно четко.
- Для именования функций и подпрограмм часто удобно применять словосочетания из существительных и глаголов.
- Следует избегать использования нестандартных сокращений.
- Если аббревиатуры необходимы, их надлежит унифицировать.



Создание списка аббревиатур, стандартных для предметной области, в которой вы работаете, приводит пример одного из возможных способов использования сети intranet. Можно создать intranet на своем компьютере и сделать его доступным для всех пользователей локальной сети. Публикация информации о проекте в intranet уменьшает объем оборота бумажных документов, гарантирует постоянный доступ и обеспечивает хранилище достоверной информации.

Конечно, легче запомнить и применять представленный перечень аргументов, нежели список из десятков или сотен префиксов. Если вам важно обозначить в наименовании переменной признак принадлежности ее определенному типу, укажите более ли менее полное название типа. Для этого потребуются несколько раз нажать клавиши. Например, если переменная обозначает главную форму приложения, ее можно назвать, скажем, `FormMain`. Только и всего.

## Использование целых слов при именвании

Размышляя над именем процедуры или функции, назовите ее понятным словосочетанием, и дополнительные комментарии станут излишними. Кроме того, чем более лаконично содержимое функции, тем менее вероятны недоразумения. Просмотрите текст листинга 14.1.





Советуем, по возможности, создавать отдельные функции даже в том случае, если они окажутся такими простыми, как приведенная выше `GetTotalSale`. Именованный блок кода гораздо более красноречив, чем те же строки, употребленные непосредственно в окружении многих других.

#### Листинг 14.1. Пример выбора удачных имен программных объектов

```
Function GetTotalSale( ByVal Sale As Double, Optional SalesTax As
Double = 0.0825 ) As Double
    GetTotalSale = Sale * (1 + SalesTax)
End Function
```

#### Анализ

Словосочетание `GetTotalSale` точно указывает на предназначение функции — расчет полной стоимости продажи. Смысл параметров `Sale` и `SalesTax` также достаточно ясен. Поскольку функция коротка и обладает удачным именем, она не нуждается в дополнительных комментариях, а наименования ее переменных — в префиксах.

Нетрудно представить, как тот же код может выглядеть в исполнении других программистов. Например, глагол *Get* легко заменяется *Calc*, а слово *Tax* превращается в *Tx*. Однако такие сокращения способны легко привести к разночтениям и непониманию (скажем, *Tx* — стандартная аббревиатура названия штата Техас).

## Использование словосочетаний при именовании подпрограмм и функций

Рекомендуется перед именами функций и подпрограмм употреблять глаголы, указывающие на определенное действие, а затем добавлять существительное, обозначающее предмет или объект, на который оказывается влияние. Существительное и глагол, соединенные вместе, в обычной речи хорошо передают смысл высказывания, и программный код не будет исключением. Если в теле функции или процедуры выполняются только те действия, о которых свидетельствует ее название, больше ничего и не требуется.

*Get* (взять, получить), часть наименования функции листинга 14.1, — это глагол, *Total* (полный) — прилагательное, а *Sale* (стоимость продажи) — существительное. Соединенные вместе, эти слова ясно выражают назначение функции.

#### Новый термин

**Процедура** — общее название функций и подпрограмм. В данном издании этот термин используется в случае, когда излагаемый материал относится и к функциям, и к подпрограммам.



Появление в именах программных объектов сокращенных слов можно объяснить некоторыми традициями, которые перешли к нам, так сказать, по наследству — прежние версии компиляторов и интерпретаторов зачастую просто не поддерживали возможностей задания длинных имен. Следуя привычке, многие программисты по сей день употребляют в своем коде аббревиатуры. Язык VBA в этом смысле ничем вас не ограничивает — используйте идентификаторы такой длины, какую считаете целесообразной.

Как правило, если описательное наименование процедуры очень длинное, значит в ней выполняется слишком много действий. Подумайте о том, чтобы разбить ее на несколько процедур.

Так что, намереваясь создать процедуру или функцию, придумайте для нее подходящее имя, состоящее, возможно, из глагола, описывающего выполняемое действие, и существительного, указывающего на предмет ваших забот.

## Избегайте нестандартных аббревиатур

Во многих отраслях промышленности и бизнеса приняты соглашения, касающиеся употребления сокращений. Та или иная аббревиатура в контексте подобного соглашения становится вполне понятной, но стоит упустить контекст из виду, как сразу возникают недоразумения. В качестве общего правила порекомендуем следующее: старайтесь избегать нестандартных сокращений и акронимов. Если аббревиатуру, которую вы намереваетесь ввести, нельзя найти в обычном словаре, используйте полный вариант словосочетания.

Допустим, все потенциальные пользователи вашей программы осведомлены об используемых сокращениях — что ж, пожалуйста, действуйте! Впрочем, список аббревиатур и их значений создать все равно придется. Автору этой книги например, приходилось участвовать в проектах, находивших применение в различных отраслях. Естественно, запоминать сотни различных жаргонных сокращений, принятых в разных организациях, — дело совершенно ненужное и неблагодарное. Если уж без сокращений никак не обойтись, пользуйтесь четким списком, согласованным с коллегами и заказчиками.

## Правила выравнивания текста программы

*Отступы* (не содержащие видимых символов фрагменты начала строк кода) — весьма полезное и важное средство обеспечения хорошего восприятия программного текста. Следуйте правилам выравнивания текста, и ваш код приобретет ясность и выразительность. Сопоставьте два листинга (14.2 и 14.3), которые содержат тексты одной и той же функции. В первом отступы используются беспорядочно и случайно, а во втором позиции начала строк четко выровнены по вертикали.

Листинг 14.2. Пример неверного использования отступов

```
1: Function GetIQText( ByVal IQ As Integer ) As String
2:   If IQ < 79 Then
3:     GetIQText =
4:       "Коэффициент ниже 79 свидетельствует о серьезном интеллектуальном отставании"
5:       ElseIf IQ >=79 And IQ < 90 Then
6:         GetIQText = "Низкий или средний уровень интеллекта"
7:         ElseIf IQ >= 90 And IQ < 105 Then
8:           GetIQText = "Нормальный уровень интеллекта"
9:         Else: GetIQText = "Интеллект выше нормального"
10:        End If
11:      End Function
```

### Анализ

Увы, мне приходилось иметь дело с десятками тысяч строк кода, схожего с приведенным выше. Если откровенно, я не хочу обвинять кого-либо в том, что подобное "творчество" носило преднамеренный характер. Скорее всего, текст стал таким после целого ряда бессистемных исправлений. В общем, большинству программистов понятно, что это нехорошо. Сколько раз приходилось слышать фразу: "Да, я знаю, но сейчас у меня просто нет времени — подшлифую как-нибудь потом". Потрясающе, ведь и говорящему, и внимающему заранее известно, что это "потом" не наступит никогда.

Процедура листинга 14.2 выполняет анализ коэффициента интеллекта (IQ) и возвращает строку текста, отвечающую значению IQ. Задача весьма проста, но код чрезвычайно труден для чтения (не так ли?), поскольку отступы введены удивительно неграциозно. (Текст настолько безобразен, что я испытывал затруднения при его наборе.) А теперь взгляните на листинг 14.3.

#### Листинг 14.3. Пример удачного использования отступов

```

1:  Function GetIQText( ByVal IQ As Integer ) As String
2:
3:      If IQ < 79 Then
4:          GetIQText = _
              "Коэффициент ниже 79 свидетельствует о _
              серьезном интеллектуальном отставании"
5:      ElseIf IQ >= 79 And IQ < 90 Then
6:          GetIQText = "Низкий или средний уровень интеллекта"
7:      ElseIf IQ >= 90 And IQ < 105 Then
8:          GetIQText = "Нормальный уровень интеллекта"
9:      Else
10:         GetIQText = "Интеллект выше нормального"
11:      End If
12:
13:  End Function

```

#### Анализ

Листинг 14.3 содержит тот же код, что и листинг 14.2, но на сей раз текст просто радует глаз, и все благодаря какой-то, казалось бы, мелочи — правильному выравниванию текста. Обратите внимание на отступы и пустые линии, выделяющие тело функции на фоне ее начальной и завершающей строк. Служебные слова условной конструкции `If` выровнены по одной позиции; выражения, соответствующие каждому из условий, сдвинуты вправо еще на один символ табуляции. Отступы одновременно упрощают восприятие строк кода и определяют уровни их вложенности: теперь понятно, что конструкция `If` принадлежит функции `GetIQText`, а каждое из выражений присваивания отвечает определенному условию.

Наконец, листинг 14.4 представляет вашему вниманию лучшую (по мнению автора) реализацию той же функции.

#### Листинг 14.4. Пример использования выражения `Select Case`

```

1:  Function GetIQText ( ByVal IQ As Integer ) As String
2:
3:      Const RETARDED = _
          "Коэффициент ниже 79 свидетельствует о _
          серьезном интеллектуальном отставании"
4:      Const LOW = "Низкий или средний уровень интеллекта"
5:      Const NORMAL = "Нормальный уровень интеллекта"
6:      Const ABOVE_NORMAL = "Интеллект выше нормального"
7:
8:      Select Case IQ
9:          Case Is < 79
10:             GetIQText = RETARDED
11:          Case Is < 90
12:             GetIQText = LOW
13:          Case Is < 105
14:             GetIQText = NORMAL
15:          Case Else
16:             GetIQText = ABOVE_NORMAL

```

17: End Select  
18:  
19: End Function

#### Анализ

Листинг 14.4 содержит третий (вероятно, самый удачный) вариант функции анализа уровня интеллекта. (Возможно, после знакомства с улучшенным кодом незаметно повысится и ваш собственный IQ.) Текстовые константы вынесены в отдельный блок, им присвоены конкретные наименования. Строки конструкции Select Case подвергнуты тщательному выравниванию с помощью отступов и выглядят безупречно.

## Как упростить код

Существует целый ряд стратегий, позволяющих облегчить работу с программным кодом. Хорошо, если эти простые правила, перечисленные ниже, со временем станут настолько привычными, что вы сможете руководствоваться ими, не затрудняя себя лишними раздумьями.

- Избегайте использования вложенных условных конструкций.
- Чаще используйте процедуры и функции.
- Создавайте короткие и уместные процедуры.

Применяйте указанные правила на практике, и код заметно упростится. Вы сможете без труда вносить в него необходимые исправления и повторно использовать готовые фрагменты в новых проектах.

## „Нет” — вложенным условным конструкциям

К сожалению, существует практика создания конструкций Select Case и If ... Then, вложенных одна в другую. В подобных случаях код приобретает такой вид:

```
If Условие1 Then
    If Условие2 Then
        End If
Else
    If Условие3 Then
    ElseIf Условие4 Then
        If Условие5 Then
            End If
        Else
            End If
    End If
End If
```

Это еще не самый плохой вариант — все может быть гораздо хуже. Стоит забыть об использовании отступов и добавить побольше строк, описывающих последовательность действий для каждого из условий, — и код максимально будет напоминать подгоревшую молочную кашу. И вновь, повторюсь, никого не следует подозревать в злых намерениях — обычные небрежность, халатность и легкомыслие. Одна и та же программа может выглядеть и вопиюще скверно, и просто здорово — все зависит от профессионализма, последовательности в применении названных правил, от элементарной аккуратности.

#### Новый термин

Разложение на элементарные операции подобно разложению на множители в математике. В программировании это означает разбиение повторяющегося кода на процедуры или классы, которые помещаются в программу только один раз. Например, если один и тот же набор из 10 строк повторяется в программе несколько раз, переместите его в процедуру и замените повторяющиеся строки вызовом этой процедуры.

Приведенный фрагмент легко переписать следующим образом:

```
If Условие1 Then
    Call Процедура1
Else
    Call Процедура2
End If
```

Процедура1 реализует цепочку действий, выполняемых в том случае, если Условие1 становится истинным, а Процедура2 соответствует обратной ситуации. Созданные процедуры также, в свою очередь, могут быть упрощены за счет построения дополнительных процедур и функций. Код, таким образом, становится понятным и совершенно прозрачным.

## „Да” — процедурам и функциям

Вынесение строк кода в отдельные блоки — процедуры и функции — упрощает задачу сопровождения программы и повторного использования ее фрагментов. Рассмотрим конструкцию Select Case. Если вы разместите строки, обрабатывающие каждый *случай* (case), в том же блоке, где размещено само выражение Select Case, возможность повторного использования такого кода окажется ограниченной узким конкретным контекстом. Взгляните на следующий фрагмент:

```
Select Case Условие
    Case Значение1:
    Case Значение2:
    Case Else
End Select
```

Если вы введете строки для обработки всех случаев здесь же, непосредственно в теле выражения Select Case, о возможности повторного применения кода придется просто забыть. Но если каждому случаю будет поставлена в соответствие отдельная процедура, вы сможете гораздо легче использовать их в текущем или новых проектах. Разумеется, нельзя воспринимать подобный совет как догму — это всего лишь предостережение. Если код Select Case настолько же прост, как тот, который приведен в листинге 14.4, дополнительные процедуры не потребуются. Но если вы видите, что условная конструкция имеет тенденцию к усложнению, исследуйте возможность расчленения ее на части с помощью процедур и функций.

## Создавайте короткие и уместные процедуры

Ранее уже говорилось о том, что удобно, называя процедуры, использовать наименования, состоящие из глагола, который описывает выполняемую операцию, и существительного, указывающего на объект воздействия. Из этого правила вытекает следующее требование: процедура должна выполнять только те операции, о которых свидетельствует ее название.

Часто, например, возникает ситуация, когда необходимо снабдить программу дополнительными средствами обработки ошибок. Далее будет рассмотрена функция, которая открывает файл и возвращает номер открытого файла. Если задача требует, чтобы файл существовал, целесообразно написать не одну, а две функции, одна из которых вызывает другую. Изучите текст листинга 14.5.

### Листинг 14.5. Пример построения иерархии функций

```
1: Function DoOpenFile( ByVal FileName As String ) As Double
2:     DoOpenFile = FreeFile
3:     Open FileName For Random As #DoOpenFile
4: End Function
5:
```

```

6: Function FileExists(ByVal FileName As String ) As Boolean
7:   FileExists If (Len( Dir( FileName ) ) > 0)
8: End Function
9:
10:Function OpenFile( ByVal FileName As String ) As Double
11:  If (FileExists(FileName)) Then
12:    OpenFile = DoOpenFile(FileName)
13:  Else
14:    ' процедура обработки ошибок
15:  End If
16:End Function

```

### Анализ

Первая из функций, DoOpenFile, текст которой расположен в строках 1-4, непосредственно решает задачу, открывая файл и возвращая его номер в виде числа двойной точности. Вторая функция, FileExists, находящаяся в строках 6-8, выполняет проверку существования файла с заданным именем. Внешняя функция, OpenFile, выполняет операции по проверке существования файла и в случае успеха вызывает функцию DoOpenFile. Обращаться следует, разумеется, к функции OpenFile. Результат получается простым и удобным.

Таким образом, создаются внутренние и внешние функции. Вызываемая функция обеспечивает обработку ошибок, а внутренняя функция, с префиксом Do, непосредственно выполняет операцию.

## Как комментировать код

Работая над черновиком программы, вы наверняка выдвигаете определенные предположения о ее назначении. Уделите пару минут, чтобы зафиксировать их в виде примечаний к коду. Гораздо проще комментировать текст прогаммы во время ее написания, нежели позже, когда придется восстанавливать в памяти ход своих рассуждений. Вот несколько простых советов по поводу написания комментариев.

- Используйте полноценные предложения.
- Комментируйте длинные фрагменты кода.
- Сопровождайте примечаниями неоднозначные участки кода и фрагменты, заимствованные из других источников.

## Используйте полноценные предложения

Этот совет, считаем, вполне понятен. Не ленитесь выражать свою мысль в виде законченных предложений на родном языке. Они помогут в ходе дальнейшей работы над прогаммой и пригодятся при написании руководств.



Хорошо известна прогамма JavaDoc.exe, которая автоматически генерирует документацию к прогамме, читая ее исходный текст на языке Java. Вполне возможно, вам удастся найти подобное средство, ориентированное на применение в среде VBA.

## Комментируйте длинные фрагменты кода

ЕСЛИ процедура или функция содержит более трех-пяти строк, подумайте о написании нескольких предложений, которые поясняют ваш замысел, раскрывают назначение кода и оговаривают особые условия его использования.

## Сопровождайте примечаниями неоднозначные участки кода

Если код предусматривает выполнение особо точных операций либо заимствован из внешнего источника, не лишайте себя возможности ввести пару-тройку примечаний. При комментировании чужого кода обязательно укажите полное наименование источника, из которого он получен, наименование файла или документа, номер версии, фамилию автора, дату опубликования, номера страниц и т.п. Вполне вероятно, что со временем вам потребуется вновь обратиться к тому же источнику за очередной редакцией кода, разъяснениями или помощью в преодолении нештатных ситуаций.

## О возможностях повторного использования кода

Индустрия, основанная на принципах повторного применения программного кода, располагает миллиардами долларов. Имя ей — **объектно-ориентированное** программирование. Собственно говоря, одна из предпосылок появления этой отрасли была связана с поисками ответа на вопрос, как добиться возможностей многократного использования ранее созданного программного кода. (Подробнее вопросы объектно-ориентированного программирования освещены в главе "21-й час. Основы программирования классов".) Впрочем, решить подобную задачу можно не только в рамках объектной парадигмы. И ниже рассказано, как этого достичь.

Код, который следует применить, написан — т.е. вам уже не нужно этим заниматься. Он наверняка прошел стадию отладки и тестирования — стало быть, если вы не собираетесь его исправлять, повторная отладка не потребуется. Поскольку код ранее уже кем-то использовался (может быть, даже вами), не исключено, что он способен полностью решить вашу конкретную проблему. Наконец, если вставить подобный код в некую "обрамляющую" его процедуру, вам потребуется протестировать только эту процедуру. Листинг 14.5 иллюстрирует сказанное. Если вы позаимствовали готовую функцию, подобную DoOpenFile, а затем облекли ее в "оболочку", схожую с OpenFile, достаточно будет проверить, насколько правильно работает новая, внешняя, функция.



Важно как можно раньше утвердить систему именования процедур и функций и последовательно ее придерживаться. Объяснения просты: если вы решили изменить название функции, придется пролистать текст программы и отредактировать все ссылки на эту функцию. Исправленный код, разумеется, нуждается в тестировании. Впрочем, существует иной способ достижения цели. Представьте, у вас есть процедура. Если перед ее именем ввести префикс (скажем, *Do*), а затем создать новую, "обрамляющую", процедуру, дав ей имя прежней, весь код останется в неприкосновенности, и тестировать доведется только вновь созданную процедуру.

Чем более лаконична процедура или функция, чем более четкими и понятными именами и комментариями она снабжена, тем выше вероятность ее повторного использования. Словом, если вы прислушаетесь к советам, которые прозвучали в ходе этого занятия, вам удастся дать многим программным творениям — своим и чужим — вторую жизнь, тем самым облегчив собственную.

# Советы по тестированию и отладке кода

В главах "17-й час. Отладка кода" и "18-й час. Обработка ошибок во время выполнения программы" содержатся подробные сведения по вопросам тестирования и отладки программного кода, поэтому рассматривать в данный момент нецелесообразно. Дадим лишь некоторые советы-предостережения.

Исправляя код, обязательно подвергайте его повторному тестированию даже в том случае, если изменения кажутся незначительными; в противном случае вы рискуете собственной профессиональной репутацией. При отсутствии специальных средств контроля версий программного продукта ведите журнал, в котором в хронологическом порядке перечисляются выявленные ошибки и внесенные исправления. Минуты, потраченные сейчас, впоследствии обернутся часами сэкономленного времени и массой сохраненных нервных клеток.

## Еще раз о работе с данными

Решить проблему удобочитаемости программного кода могут также применяемые вами способы работы с данными. Вот несколько кратких советов по этому поводу.

- Используйте именованные константы вместо литеральных значений.
- Проясняйте назначение и смысл параметров процедур и функций с помощью служебных слов `ByVal`, `ByRef` и `Optional`.
- Ограничьте применение глобальных переменных.

Если вам приходится использовать литеральные символьные или числовые константы, определите их явно с помощью служебного слова `Const` и четких, понятных наименований. Характерный пример приведен в тексте листинга 14.4.

Чтобы гарантировать неизменность значения параметра, переданного функции, используйте служебное слово `ByVal`. Если, в соответствии с вашим замыслом, значение аргумента может изменяться внутри функции, назначьте переменной квалификатор `ByRef`. Наконец, если аргумент в большинстве случаев принимает некоторое заранее известное значение, обозначьте его как `Optional`. Пример использования служебного слова `Optional` приведем в листинге 14.1.

## Резюме

Как вы пишете собственные программы — дело личное. Единого универсального способа оформления программного кода, увы, не существует, но какой-то вам все равно придется избрать и последовательно использовать. Приняв определенную стратегию действий, вы сможете освободить время, направив свои знания и энергию на достижение более высоких целей.

На этом занятии вы узнали о том, как с помощью упрощения и уменьшения размера процедур и функций добиться возможности повторного использования собственного и заимствованного кода. Мы обсудили схемы именования программных объектов: процедуры и функции, например, удобно называть словосочетаниями, состоящими из глагола и существительного. Расчленение многоуровневых управляющих структур на составные части — еще один способ обеспечения удобочитаемости и управляемости кода. Столкнувшись с задачей изменения имени существующей функции, вы можете "облечь" ее рамками новой, избежав при этом необходимости внесения многочисленных исправлений и проведения исчерпывающего повторного тестирования.



Если приемы, о которых было рассказано, войдут в ваш арсенал, то профессиональная деятельность станет гораздо более продуктивной, а результаты приобретут новое качество. Минутку, не надо бросаться в погоню за качеством прямо сейчас — уделите время изучению разделов "Вопросы и ответы" и "Задания", которые с нетерпением ожидают вашего внимания.

## Вопросы и ответы

**Вопрос.** Существует ли единый рецепт "правильного" программирования?

**Ответ.** К сожалению, нет. Вот почему программирование в чем-то сродни искусству. Впрочем, есть достойные образцы, стратегии и приемы, которым можно и нужно следовать. Первое и главное — программа должна работать верно.

**Вопрос.** Необходимо ли пытаться реализовать все ваши советы сразу, на первом же этапе работы над программой?

**Ответ.** Нет. Если настойчиво их применять, они станут привычными и естественными. Кроме того, я настоятельно рекомендую по возможности просматривать код программы несколько раз.

**Вопрос.** Существуют ли другие приемы или стандарты, которые стоит изучить?

**Ответ.** Лучшие стратегии работы формируются на практике — творите, выдумывайте, пробуйте!

## Задания

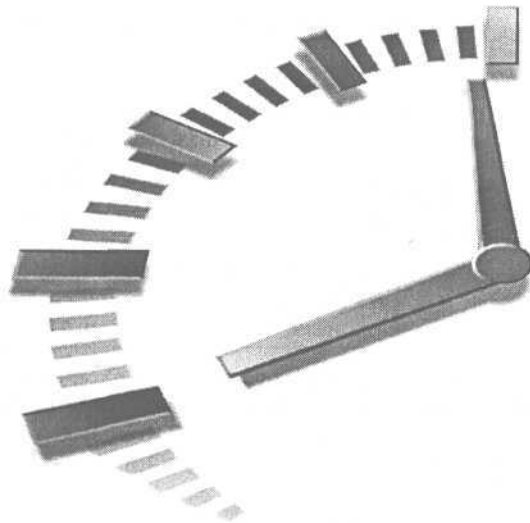
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Назовите удобный способ именования процедур и функций.
2. В чем заключаются положительные стороны идеи повторного применения программного кода?
3. Смысл префиксной, или венгерской, нотации состоит в разработке системы сокращенных названий типов, используемых для именования переменных. Верно ли это?
4. Термин *процедура* используется для обозначения и подпрограмм, и функций. Верно ли это?
5. Назовите важное эмпирическое правило, которое обуславливает приемлемый размер кода процедуры или функции.

## Упражнения

1. Задайте имя функции, предназначенной для вычисления полной и частичной сумм.
2. Предложите имена для пары функций, одна из которых просто закрывает файл, а другая выполняет проверку корректности выполнения этой операции.
3. Задайте названия переменной, предназначенной для хранения номера открытого файла, хранения номера ошибки открытия файла, а также для процедуры, вычисляющей сумму процента при известных сумме капитала и процентной ставке.



# 15-й час

## ADODB — ваш верный помощник

Технологии управления базами данных развиваются стремительно. История их развития весьма поучительна — ее можно было бы обсудить за чашечкой кофе, — но достаточно сказать одно: базы данных достигли того уровня возможностей, на котором задачи программирования решаются намного проще, нежели еще несколько лет назад.

Одним из серьезных промежуточных достижений на долгом пути совершенствования технологий стало создание (усилиями специалистов Microsoft) стандарта доступа к базам данных, получившего название *Open DataBase Connectivity*, ODBC. ODBC определяет единые правила работы с базами данных самых различных типов. Участь программистов еще более облегчилась с появлением нового протокола взаимодействия с базами данных — *ADO*. Это также несомненная заслуга Microsoft. Поскольку ADO — самый свежий и мощный стандарт, большую часть занятия мы посвятим именно ему.

Доступ к библиотеке ADODB в среде Microsoft Visual Basic осуществляется из диалогового окна **Tools⇒References**. Класс ADODB содержит объекты классов для представления таблиц, запросов, записей, столбцов, полей и многих других объектов данных, обеспечивая самые разные возможности их обработки. Теперь вам удастся, отбросив в сторону все лишнее, сосредоточить внимание на особенностях и способах решения самой задачи. Многие трудоемкие и рутинные обязанности ADODB принимает на себя.

Основные темы занятия.

- Соединение с базой данных и использование наборов данных.
- Операции добавления, обновления и удаления записей.
- Сохранение информации из баз данных в коллекциях.
- Атрибуты объектов ADODB, предназначенные для управления базами данных.
- Использование информационных ресурсов и поиск готовых решений.
- Использование методов `AddItem` и `RemoveItem` для управления элементами коллекций.

# Соединение с базой данных

Задача программирования для Access в немалой степени упрощается за счет того, что программный код хранится в самой базе данных. Первым делом вы должны научиться создавать соединение с базой данных.



В диалоговом окне **Tools**⇒**References** редактора Visual Basic содержится список всех доступных библиотек ActiveX. Это библиотеки скомпилированных программ, которые можно вызывать из собственного VBA-кода. ADODB доступны в проекте по ссылке `msado26.tlb`. Название текущей версии ADO — Microsoft ActiveX Data Objects 2.7. Library. ADO — это название протокола, а ADODB — название объекта в библиотеке.

В составе ADODB имеется класс Connection. Среда Microsoft Visual Basic по умолчанию настроена на использование протокола ADO, поэтому вы сможете обращаться к подклассам ADODB без дополнительных усилий. Чтобы обеспечить соединение с базой данных, выполните следующие действия.

1. Определите процедуру.
2. Объявите в ней переменную-объект класса ADODB.Connection.
3. Задайте информацию о провайдере OLE DB (соответствующая структура подробно рассмотрена ниже).
4. Откройте соединение.
5. По завершении работы освободите память, выделенную объекту Connection.

## Новый термин

Данные о *провайдере* (provider) *OLE DB* (очередной протокол, разработанный Microsoft и призванный заменить ODBC) — это новая версия *источника данных*. Поскольку OLE DB, помимо традиционных средств, предлагает дополнительные, более мощные, словосочетание *источник данных* было решено заменить термином *провайдер*. Говоря привычным языком, провайдер — это источник данных.

Способы выполнения всех названных операций иллюстрируются в тексте листинга 15.1.

## Листинг 15.1. Пример использования объекта ADODB.Connection

```
1 Sub DemoADODB ( )
2   Const Provider = "Provider=Microsoft.Jet.OLEDB.4.0;"
3   Const DataSource = "Data Source=C:\Data\Hour15.mdb"
4   Dim Connection As New ADODB.Connection
5   On Error GoTo Finally
6       Call Connection.Open(Provider & DataSource)
7       Connection.Close
8   Finally:
9       If (Err.Number <> 0) Then
10:           MsgBox Err.Description
11:       End If
12:       Set Connection = Nothing
13:End Sub
```



Оформляйте длинные символьные литералы в виде именованных констант, вынесенных за пределы кода, в котором они используются — текст программы станет гораздо более удобным для чтения.

В строках 2 и 3 определяются именованные константы, содержащие описание параметров соединения — сведения о провайдере и путь к файлу базы данных. Строка с информацией о провайдере выглядит следующим образом:

```
Provider=Microsoft.Jet.OLEDB.4.0
```

Далее указывается наименование базы данных и путь к ней. В нашем конкретном случае выражение таково:

```
Data Source=C:\Data\Hour15.mdb
```



Применение метода `Open` класса `ADODB.Connection` — это, вероятно, самая непростая часть операции создания соединения с базой данных, поскольку строка, описывающая сведения о провайдере, выглядит для непосвященных как нечто неподвластное сознанию. Однако вам не стоит опасаться. Смело копируйте и используйте ее в своих приложениях.

(Разумеется, вы должны заменить путь к базе данных, указанный в примере, собственным, реальным.) Обратите внимание, что строка соединения — это единое целое, включающее в себя блоки информации о провайдере и пути к базе данных, разделенные символом точки с запятой.

Свойства и методы класса принято называть обобщенным термином *интерфейс*. Исчерпывающие сведения по этим вопросам вы найдете в главе "21-й час. Основы программирования классов".

Строка 4 листинга 15.1 иллюстрирует прием единовременного объявления и создания объекта класса `ADODB.Connection`. Помните о том, что `Connection` — это неотъемлемая часть интерфейса класса `ADODB`, поэтому для ссылки на нее необходимо указывать наименование "внешнего" класса, `ADODB`, отделяя его оператором точки (`.`). Поскольку `ADODB.Connection` — это, в свою очередь, также класс, в конструкции создания его объекта следует использовать служебное слово `New`. Строка 5 определяет выражение обработчика ошибок (подробнее см. главу "18-й час. Обработка ошибок во время выполнения программы"), который при возникновении ошибки обеспечивает автоматический переход к строке кода, содержащей указанную метку.

В строке 6 осуществляется обращение к процедуре-методу открытия базы данных. Ниже вы можете вставить строки, выполняющие определенные операции с данными. По завершении работы необходимо закрыть соединение таким образом, как это сделано в строке 7.

В строках 9—11 выполняется анализ допущенной ошибки и отображается содержимое одного из свойств объекта `Err`. (Подробнее о классе `Err` см. главу 18.) Инструкция строки 12 освобождает память, выделенную объекту `Connection`.

## Аргументы процедуры открытия базы данных

Метод `Open` класса `ADODB.Connection` требует задания аргумента, содержащего строку соединения. В строке соединения объектом ADO распознаются следующие параметры, представленные в формате `Аргумент=Значение` и разделенные символом ТОЧКИ С запятой: `Provider`, `File Name`, `Remote Provider` и `Remote Server` (их описание приведено в табл. 15.1). Строка соединения может содержать и другие аргументы (например, имя и пароль пользователя), но они не обрабатываются ADO и должны следовать правилам, которые регламентируются конкретным провайдером.

Таблица 15.1. Параметры соединения, поддерживаемые объектом ADO

Наименование	Описание
Provider	Наименование провайдера
File Name	Файл провайдера, содержащий данные
Remote Provider	Наименование провайдера, используемого в клиентских приложениях
Remote Server	Наименование приложения удаленного сервера, используемого клиентскими программами

Параметры, перечисленные в табл. 15.1, обрабатываются ADO, а все остальные (скажем, имя и пароль пользователя) драйвером OLE DB. Листинг 15.2 демонстрирует код, аналогичный предыдущему. Соединение обеспечивается не с помощью провайдера OLE DB, а средствами протокола ODBC.

Листинг 15.2. Открытие базы данных с использованием ODBC

```

1: Sub ProviderWithODBCAlias ( )
2:
3:   Const ConnectionString = "DSN=Chapter15;UID=;PWD="
4:   Dim RecordSet As New ADODB.Recordset
5:   Dim Connection As New ADODB.Connection
6:
7:   On Error GoTo Finally
8:
9:   Call Connection.Open(ConnectionString)
10:
11:  Connection.Close
12:
13:  Finally:
14:    If (Err.Number <> 0) Then
15:      MsgBox Err.Description
16:    End If
17:
18:    Set RecordSet = Nothing
19:    Set Connection = Nothing
20:End Sub
21:

```

#### Анализ

Единственное принципиальное различие между двумя рассмотренными процедурами состоит в том, что в последней для доступа к данным применяются средства ODBC. Стандарт ODBC, верой и правдой служивший на протяжении многих лет и активно используемый поныне, требует построения так называемых источников данных (об этом рассказано в следующем разделе).

Но с ростом популярности Web и соответствующих средств программирования возникают задачи обработки данных, которые поступают из источников новых типов. Спрос рождает предложение, и поэтому Microsoft разработала расширенный стандарт средств соединения с базами данных, назвав его *OLE DB*. Наиболее полная информация об OLE DB находится по Internet-адресу <http://www.microsoft.com/data/oledb/default.htm>.

# Создание источника данных ODBC

Источник данных ODBC (стандарта средств соединения с базами данных, разработанного Microsoft) — это объект реестра Windows, обуславливающий способы обращения к конкретной базе данных из прикладной программы. Источник данных ODBC предполагает использование определенной динамической библиотеки, которая разработана либо поставщиком системы управления базами данных, либо сторонней фирмой, и обеспечивает возможности взаимодействия с базой данных в рамках стандарта ODBC.

Чтобы создать источник данных ODBC, выполните следующие действия.

1. Щелкните на кнопке Пуск (Start) панели задач Windows.
2. Выберите команду **Настройка**⇒**Панель управления** (Settings⇒Control Panel).
3. В окне **Панель управления Windows** (рис. 15.1) щелкните на пиктограмме **Источники данных ODBC** (Data Sources (ODBS)), чтобы загрузить приложение **Администратор источников данных ODBC** (ODBC Administrator).

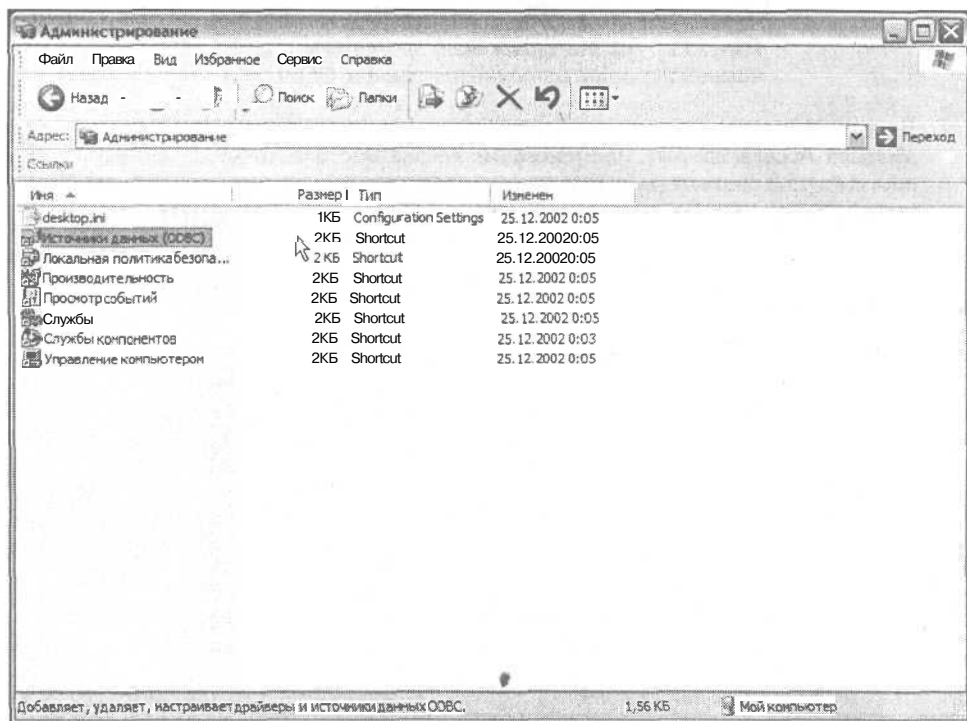


Рис. 15.1. Чтобы создать или отредактировать источник данных ODBC, щелкните на пиктограмме **Источники данных ODBC** окна **Панель управления Windows**

4. Перейдите на вкладку **Пользовательский DSN** (User DSN) окна **Администратор источников данных ODBC** и щелкните на кнопке **Добавить** (Add) (рис. 15.2).
5. В списке диалогового окна **Создание нового источника данных** (Create New Data Source) выберите элемент **Microsoft Access Driver (\*.mdb)**.
6. Щелкните на кнопке **Готово** (Finish).

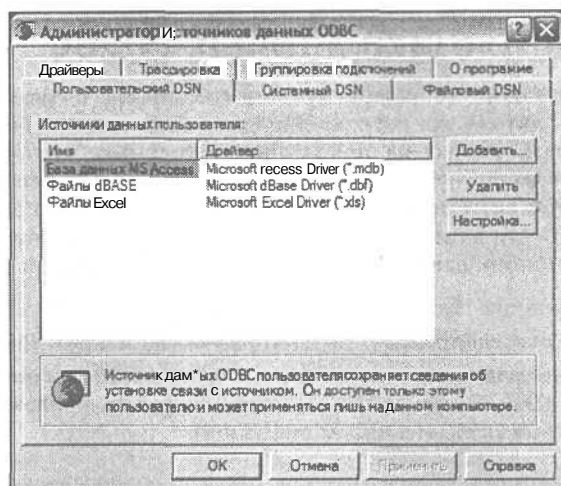


Рис. 15.2. Так выглядит окно программы  
Администратор источников данных ODBC

7. В поле Имя источника данных диалогового окна Установка драйвера ODBC для Microsoft Access введите наименование создаваемого источника данных (примеру листинга 15.2 соответствует имя Chapter15).
8. Щелкните на кнопке Выбрать (Select) группы опций База данных (Database).
9. В диалоговом окне Выбор базы данных (рис. 15.3) (Select Database) перейдите к нужному файлу базы данных (в тексте листинга 15.1 указано C:\Data\Hour15.mdb — эту строку легко заметить).

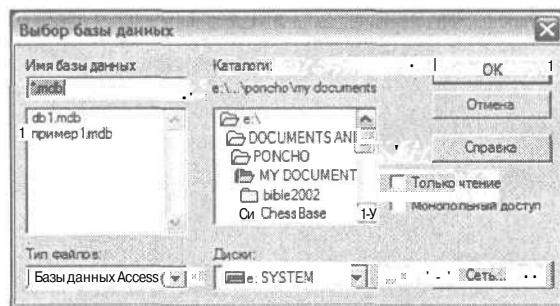


Рис. 15.3. Выберите нужный файл базы данных Access

10. Щелкните на кнопке ОК диалогового окна Установка драйвера ODBC для Microsoft Access.
11. Щелкните на кнопках Применить (Apply) и ОК окна Администратор источников данных ODBC, чтобы сохранить созданный источник данных.

Необходимо отметить, что предложенная выше последовательность действий по созданию источника данных в значительной степени зависит от выбранного драйвера базы данных. В одном случае процедура более сложна и продолжительна, в другом — проще и короче. Как правило, каждый ODBC-драйвер укомплектован системой оперативной справки, поясняющей особенности создания источников данных на его основе, — читайте и экспериментируйте.

# Провайдер: что это такое?

На первый взгляд, набор *источников данных* может показаться похожим на мусорную свалку. Не судите слишком строго. Если Microsoft проявит последовательность и настойчивость, вскоре все мы будем обращаться исключительно к услугам *провайдеров*.

При необходимости расширить толкование определенного термина часто создают еще один, который, охватывая прежний, вносит новый смысл. Термин *источник данных*, используемый в контексте стандарта ODBC, означает ссылку на базу данных. Термином *провайдер*, введенным в обиход с появлением OLE DB, обозначаются как традиционные базы данных — скажем, формата MS SQL Server, Access, Oracle, DBF и т.д., — так и хранилища информации новых видов.

Стандарт OLE DB призван обеспечить возможность работы с источниками данных самых разных типов — от "старинных" баз данных, которые обслуживаются программами, написанными на языке COBOL, до современных серверов Oracle и сайтов Web.

## Задание имени и пароля пользователя

ЕСЛИ база данных защищена паролем, в строке соединения могут быть заданы имя пользователя и его пароль. Напомним, что в строку соединения, наряду с описанием провайдера, допустимо вводить и другие параметры.

Имя пользователя и пароль указываются в формате *Аргумент=Значение*. Строка 2 листинга 15.2 содержит пример задания пустых имени и пароля пользователя. Эти параметры могут задаваться либо непосредственно в строке соединения, либо позже, в качестве второго и третьего аргументов процедуры `ADODB.Connection.Open`.



Если имя и/или пароль пользователя задаются и в строке соединения, и при вызове метода `Open`, значения аргументов процедуры `Open` имеют более высокий приоритет.

## Выбор режима соединения

Метод `ADODB.Connection.Open` способен принимать до четырех параметров. Первый из них (строка соединения) — обязателен, остальные — нет. Второй и третий параметры — соответственно, имя и пароль пользователя; в качестве четвертого может быть задано одно из значений встроенного перечислимого типа, обозначающее режим соединения.

Доступны два режима соединения — `adConnectUnspecified` и `adAsyncConnect`. Первый, предлагаемый по умолчанию, обеспечивает соединение с базой данных в синхронном режиме. Под *синхронным* понимается такой вызов процедуры или функции (в данном случае — метода ADO), при котором работа программы приостанавливается до тех пор, пока выполнение метода не будет завершено.

### Новый термин

*Поток*, или *нить* (thread) — это подпрограмма, выполняемая внутри приложения. В однопоточном приложении в определенный момент времени работает единственный внутренний процесс — в отличие от многопоточного, в котором одновременно выполняется несколько процессов-нитей.



Говоря о потоке, представим для простоты, что он состоит из единственной функции. В однопоточном, синхронном, приложении обращение к такой функции означает, что в момент ее работы остальной код программы не выполняется. Но в многопоточном приложении после запуска функции-нити программа продолжает свое выполнение, не дожидаясь завершения работы функции. Приемы программирования одно- и многопоточных приложений существенно различаются.



Другой режим соединения, *асинхронный*, обозначается аргументом `adAsyncConnect`. В этом случае обращение к функциям ADO выполняется в пределах отдельного потока, и программа получает возможность продолжения работы еще до завершения выполнения кода созданного потока.

Если блок кода вызывается в синхронном режиме, возврат из него означает, что выполнение блока завершилось. Но в асинхронном режиме основной код и вызванный блок "живут" и выполняются независимо. Работа в асинхронном режиме требует большего внимания, поскольку необходимо контролировать состояние процессов одновременно в разных частях кода.

## Управление наборами данных

Объект класса `Connection` позволяет программе обращаться к провайдеру. Но основная работа происходит на уровне наборов данных. Если `Connection` — это объект, предоставляющий программе общий "канал" связи с данными, то объект класса `Recordset` позволяет обращаться к конкретным подмножествам источника данных. Отметим, что в большинстве случаев (за исключением некоторых "глобальных" операций, связанных, например, с резервным копированием базы данных в целом) программа в определенный момент времени имеет дело только с некоторой частью данных.

Объект `Recordset` дает возможность одновременного обращения к одной строке данных. Традиционный термин *запись* служит для обозначения одной строки данных из одной таблицы. Объекты `Recordset`, применяемые для работы с системами реляционных баз данных, к примеру Access 2002, способны содержать информацию более чем из одной таблицы. В этом смысле уместно употреблять новый термин — *строка*. Поэтому далее будем говорить, что объект `Recordset` позволяет манипулировать строками данных.

Прежде чем научиться пользоваться объектом `Recordset`, вам необходимо узнать, как его объявлять, открывать и закрывать. Этим вопросам посвящены следующие разделы.

## Открытие и закрытие наборов данных

Объект `Recordset` — составная часть класса `ADODB`. Поэтому для обращения к нему необходимо ссылаться на класс `ADODB` (к слову, объект с таким же названием имеется и в составе класса `DAO`).



`ADODB.Recordset` — это, в свою очередь, класс. Для создания объектов класса необходимо использовать директиву `New`, а при выполнении операций присваивания — служебное слово `Set`. Синтаксис выражения объявления и создания объекта `ADODB.Recordset` таков:

```
Dim ИмяОбъекта As New ADODB.Recordset
```

В качестве имени объекта укажите любую последовательность допустимых символов. Некоторые программисты обозначают объекты наборов данных сокращением `RS`. Вам могут встретиться также образцы кода, в которых идентификаторы снабжены префиксами `ado`, указывающим нам то, что речь идет об объектах класса `ADO`, а не `DAO`. Автор книги предпочитает обычное словосочетание `RecordSet` или некоторые его вариации. Пожалуйста, поступайте так, как считаете нужным, но будьте последовательны в своих решениях и действиях — тогда код станет более четким и простым для восприятия.

Чтобы получить возможность использования объекта `Recordset`, одной инструкции объявления и создания недостаточно — объект необходимо открыть. Синтаксис обращения к методу `Open` объекта `ADODB.Recordset` имеет следующий вид:

```
Call ИмяОбъекта.Открыть( [Источник], [ИмяОбъектаСоединения],
[ТипКурсора], [ТипБлокировки], [Опции] )
```

Словосочетание `ИмяОбъекта` означает, что здесь необходимо указать название объявленного и созданного ранее объекта класса `ADODB.Recordset`, а далее, после оператора точки (`.`), ввести наименование метода — `Open`. Все аргументы процедуры не обязательны — впрочем, это не означает, что их вовсе не нужно задавать; просто соответствующие значения могут быть установлены предварительно, до обращения к `Open`, с помощью свойств класса `ADODB.Recordset`. Самую полную информацию вы сможете найти в оперативной справочной системе.

Завершив работу с набором данных, вы должны закрыть его. Чтобы закрыть набор данных, следует использовать метод (как вы догадались?) `Close`, который действительно прост — он не требует аргументов. Листинг 15.3 содержит расширенный вариант процедуры листинга 15.1, включающий обращения к объекту `ADODB.Recordset`.

### Листинг 15.3. Пример использования объекта `ADODB.Recordset`

```
1 Sub DemoADODB( )
2   Const Provider = "Provider=Microsoft.Jet.OLEDB.4.0;"
3   Const DataSource = "Data Source=C:\Data\Hour15.mdb"
4   Dim RecordSet As New ADODB.Recordset
5   Dim Connection As New ADODB.Connection
6   On Error GoTo Finally
7       Call Connection.Open(Provider & DataSource)
8       Call RecordSet.Open( "MUSIC", Connection, _
9           adOpenDynamic, AdLockOptimistic )
10      RecordSet.Close
11      Connection.Close
12:Finally:
13:  If (Err.Number <> 0) Then
14:      MsgBox Err.Description
15:  End If
16:  Set RecordSet = Nothing
17:  Set Connection = Nothing
18:End Sub
```

#### Анализ

Тексты листингов 15.1 и 15.3 совпадают — за исключением тех строк, в которых содержатся инструкции объявления-создания и использования объекта класса `ADODB.Recordset`. В строке 4 создается набор данных. Строка 8 демонстрирует прием открытия объекта `RecordSet`. (В этом примере подразумевается, что объект набора данных ссылается на таблицу Access под названием `MUSIC`.) Между строками 9 и 10 можно поместить код, предназначенный для обработки записей таблицы. Строка 10 закрывает набор записей, а строка 11 — объект соединения. В строках 16 и 17 выполняются инструкции освобождения памяти, отведенной объектам `RecordSet` и `Connection`.

Далее остановимся на строках 8 и 9, содержащих вызов метода открытия набора записей. Первый аргумент процедуры `Open` — это наименование источника данных, таблицы `MUSIC`, а второй — имя объекта соединения, `Connection`. Третий параметр определяет тип курсора. Запись `adOpenDynamic` означает, что данные могут свободно добавляться, модифицироваться и удаляться, причем все изменения, в это же время вносимые в таблицу

другими пользователями, будут динамически отслеживаться. Четвертый аргумент, `adLockOptimistic`, определяет тип блокировки записей таблицы. "Оптимистический" уровень блокировки означает, что запись блокируется вами (другим пользователям будет отказано в доступе, предполагающем возможность внесения изменений), причем только в тот момент, когда вызывается метод `Update`. Иными словами, гарантируется, что в промежутке между началом внесения изменений и их сохранением содержимое записи не может быть изменено другими пользователями базы данных.

## Разновидности наборов данных

Первый аргумент, Источник, передаваемый процедуре `Open`, определяет вид набора данных. В качестве источника могут использоваться объект класса `Command`, литерал, представляющий строку выражения на языке SQL, таблица, запрос, хранящая процедура или наименование сохраненного набора данных.

### Объекты `Command`

`Command` — это объект в составе класса `ADO`, способный содержать код, возвращающий набор строк данных или выполняющий их модификацию. Объект класса `Command` — один из возможных источников данных, указываемых на месте первого параметра метода `Open` объекта `Recordset`.

### Хранимые процедуры

В качестве первого аргумента метода `Open` может задаваться имя такого объекта хранимой процедуры, который содержит код на языке SQL. Подробнее об использовании хранимых процедур Access и соответствующих им объектов VBA см. главу "16-й час. Применение языка SQL".

### Литеральные выражения SQL

Методу `Open` может передаваться также символьная строка, в которой находится выражение, написанное на языке SQL. В этом случае вы, автор, получаете самые широкие и гибкие возможности в определении конкретных задач, которые должна выполнить процедура `Open`. О конструкциях языка SQL и способах их практического использования речь пойдет в главе 16.

Наиболее полная информация о программировании на SQL изложена в таких книгах, как, например, *Sams Teach Yourself Microsoft Access in 21 Days* и *Sams Teach Yourself Microsoft Access 2002 in 24 Hours*. Рекомендуем также посетить виртуальный книжный магазин издательства *Sams*, который находится по адресу <http://www.mcp.com> — вам наверняка удастся найти книги, более подробно освещающие вопросы программирования на языке SQL.

Строки 8–9 листинга 15.3 можно переписать с использованием литерального выражения SQL в качестве источника данных следующим образом:

```
Call RecordSet.Open( "SELECT * FROM MUSIC", Connection, adOpenDynamic, _  
AdLockOptimistic )
```

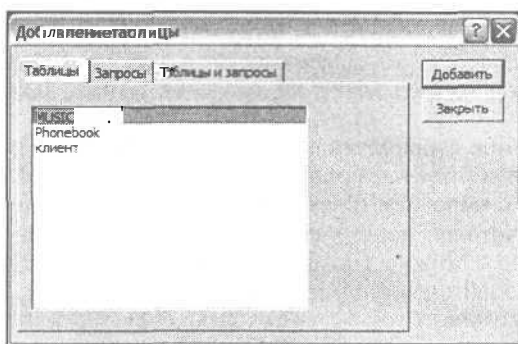
Обратите внимание, что теперь таблица `MUSIC` представлена запросом на языке SQL. Более предпочтительным будет выглядеть вариант предварительного объявления именованной константы, содержащей строку выражения SQL; тогда в качестве аргумента процедуре `Open` достаточно передать название этой константы.

### Таблицы и запросы

Наиболее употребительная разновидность источников данных, указываемых при вызове метода `Open`, — это таблицы и запросы. Запрос, написанный на языке SQL, может ссылаться на данные, физически разнесенные по нескольким таблицам.

## 15-й час. ADODB - ваш верный помощник

- 261



*Рис. 15.5. Выбор элемента, отмеченного звездочкой, означает, что в набор данных будут включены все столбцы таблицы*

Запустите процедуру DemoADODB на выполнение. Вы увидите, что она работает точно так же, как и прежде. Поведение системы не зависит от сложности передаваемого выражения SQL. Иногда имеет смысл создать именованный SQL-запрос в среде Access, как мы сделали только что, чтобы получить возможность его редактирования, при этом не изменяя код VBA. В других ситуациях максимальная гибкость достигается при объявлении именованной символьной константы, содержащей выражение на языке SQL, непосредственно в теле процедуры или функции. Все способы приведут вас к цели — выберите самый удобный и эффективный.

## Добавление и изменение данных

Наиболее популярные операции обработки наборов данных связаны с добавлением новых строк и редактированием существующих. С технической точки зрения они схожи. Прежде всего, необходимо открыть объект Connection. Затем следует создать объект Recordset и открыть его в режиме записи, т.е. передать процедуре Recordset.Open в качестве аргумента ТипКурсора одно из предопределенных значений — adOpenKeySet либо adOpenDynamic.

Чтобы добавить запись в открытый набор данных, достаточно обратиться к методу ИмяОбъекта.AddNew, где ИмяОбъекта — название корректно созданного и открытого объекта класса ADODB.Recordset. Далее следует присвоить полям новой записи соответствующие значения и вызвать метод ИмяОбъекта.Update, позволяющий сохранить запись в базе данных.

Задача редактирования существующих данных так же проста. Вместо вызова метода AddNew напишите код, отыскивающий запись, которую необходимо изменить. Поиск записи может быть выполнен посредством циклической конструкции формата Do . . . Loop, в условии которой проверяется признак достижения конца набора данных (с помощью метода EOF). Можно также привлечь выражение SQL, возвращающее необходимую запись. Любой способ приемлем — правда, второй, с применением SQL, вероятно, более эффективен.

Листинг 15.4 демонстрирует пример добавления новой записи в таблицу MUSIC. (Если таблица вами еще не создана, выполните п.1 инструкции, приведенной в предыдущем разделе, "Таблицы и запросы".)

### Листинг 15.4. Пример добавления новой записи в набор данных

```
1: Sub DemoADODB ( )
2:   Const Provider = "Provider=Microsoft.Jet.OLEDB.4.0;"
3:   Const DataSource = "Data Source=C:\Data\Hour15.mdb"
4:   Const ConnectString = Provider & DataSource
5:
6:   Dim RecordSet As New ADODB.Recordset
7:   Dim Connection As New ADODB.Connection
8:
9:   On Error GoTo Finally
10:
11:   Call Connection.Open(ConnectString)
12:   Call RecordSet.Open( "MUSIC", Connection, adOpenKeyset,
    AdLockOptimistic )
13:
14:   RecordSet.AddNew
15:
16:   RecordSet( "FIRST_NAME" ).Value = "Natalie "
17:   RecordSet( "LAST_NAME" ).Value = "Merchant"
18:   RecordSet( "TITLE" ).Value = "TIGERLILLY"
```

```

19: RecordSet( "FORMAT" ).Value = "CD"
20: RecordSet( "PUBLISHER" ).Value = "Elektra"
21:
22: RecordSet.Update
23: RecordSet.Close
24: Connection.Close
25:
26: Finally:
27:   If (Err.Number <> 0) Then
28:     MsgBox Err.Description
29:     Err.Clear
30:   End If
31: Set RecordSet = Nothing
32: Set Connection = Nothing
33: End Sub

```

## Анализ

Принципиальные различия листингов 15.3 и 15.4 состоят в том, что последний содержит строки 14–22. В строке 14 выполняется инструкция добавления в таблицу новой записи. В строках 16–20 изменяются значения полей новой записи — с этой целью мы используем формат обращения к свойству `Fields` объекта `RecordSet`, допускаемый по умолчанию. Метод `Update`, который вызывается в строке 22, сохраняет внесенные изменения в базе данных.



Если вы не указываете свойство, но ваш код неявно предполагает обращение к нему, компилятор трактует это как ссылку на свойство, предлагаемое по умолчанию. Строки 16–20 листинга 15.4 демонстрируют именно этот прием: выражения `RecordSet(ИмяСтолбца).Value` и `RecordSet.Fields(ИмяСтолбца).Value` равноценны.

Чтобы отредактировать текст существующей записи таблицы, достаточно заменить строку 14, содержащую обращение к методу `AddNew`, конструкцией поиска требуемой записи. Весь оставшийся код (строки 16–33) в изменениях не нуждается.

Наименования полей строки и присваиваемые им значения могут быть переданы методу `AddNew` в качестве параметров. Аргументом может быть как имя одного поля, так и массив полей (первый аргумент), и массив значений, как в следующем фрагменте кода.

```

Dim Fields As Variant
Dim Values As Variant
Fields = Array("FIRST_NAME", "LAST_NAME", "TITLE", _
"FORMAT", "PUBLISHER")
Values = Array("Faith", "Hill", "Breathe", "CD", "Warner Bros.")
Call Recordset.AddNew(Fields, Values)

```

В этом фрагменте кода объявлены и инициализированы два массива, которые передаются в метод `AddNew`. Однако при тестировании легче найти ошибку, если значения полей присваиваются в коде по очереди.

## Редактирование полей

Объект класса `Recordset` содержит свойство-коллекцию `Fields`. В одной из предыдущих глав отмечалось, что *свойство* — это элемент данных, принадлежащий классу. Свойством может быть и порция данных простого типа, и объект какого-либо класса. Свойство `Fields` представляет собой объект класса `Collection`.

Коллекция `Fields` — это свойство объекта `Recordset`, предлагаемое по умолчанию. Вот почему выражения `RecordSet (ИмяСтолбца)` и `RecordSet.Fields (ИмяСтолбца)` равноправны. Кроме того, вам следует знать о существовании класса `Field`, объекты которого входят в состав коллекции `Fields`.

Основными тремя свойствами объекта `Field` являются `OriginalValue`, `Value` и `Name`. Свойство `Value` — это переменная типа `Variant`; ее используют для доступа к значению, хранящемуся в поле. Свойство `OriginalValue` (типа `Variant`) позволяет получить или восстановить исходное значение поля. Переменная `Name` дает возможность сослаться на поле по имени столбца таблицы.

Фрагмент кода, подобный приведенному в строке 17 листинга 15.4 — `RecordSet ("ARTIST")`, — означает ссылку на объект типа `Field`. Далее, разумеется, вы можете воспользоваться всеми атрибутами этого объекта. Так, например, часть 17-й строки — `.Value` — представляет собой обращение к свойству `Value` указанного объекта `Field`. Строки 16–20 могут быть переписаны в более строгой форме:

```
16: RecordSet.Fields ( "FIRST_NAME" ).Value = "Natalie"
17: RecordSet.Fields ( "LAST_NAME" ).Value = "Merchant"
18: RecordSet.Fields ( "TITLE" ).Value = "TIGERLILLY"
19: RecordSet.Fields ( "FORMAT" ).Value = "CD"
20: RecordSet.Fields ( "PUBLISHER" ).Value = "Elektra"
```

Это наглядный пример обращения к вложенным объектам через интерфейс объектов более высокого уровня. Согласитесь, код достаточно прост и нагляден.

## Удаление строк из набора данных

Синтаксис

Существует два способа удаления строк из набора данных. Один из них основан на использовании команды `DELETE` языка `SQL`. Другой, более прямолинейный и грубый, предполагает написание `VBA`-кода, который обеспечивает переход к каждой удаляемой строке набора данных, а также вызывает метод `Delete` объекта класса `Recordset`.

Синтаксис обращения к методу `Delete` объекта `ADODB.Recordset` таков:

```
Call ИмяОбъекта.Delete ( [ГруппаЗаписей] )
```

При отсутствии необязательного параметра `ГруппаЗаписей` будет удалена только текущая запись набора данных. Самые распространенные допустимые значения параметра `ГруппаЗаписей` перечислены в табл. 15.2.

**Таблица 15.2. Допустимые значения аргумента метода `Recordset.Delete`**

Значение перечислимого типа	Описание
<code>adAffectCurrent</code>	Действие по умолчанию — удаляется текущая запись
<code>adAffectGroup</code>	Удаляется группа записей, определенная свойством <b>Filter</b>
<code>adAffectAll</code>	Удаляются все записи (операция равносильна команде <code>DELETE FROM ИмяТаблицы</code> языка <code>SQL</code> )
<code>adAffectAllChapters</code>	Удаляются все записи раздела

Метод `Delete` объекта `Recordset` почти настолько же дееспособен, как и команда `DELETE` языка `SQL`. Варианту вызова `Call ИмяОбъекта.Delete ( adAffectAll )` соответствует команда `DELETE FROM ИмяТаблицы`. Конечно, конструкция `DELETE` языка `SQL` более гибкая — она позволяет, например, использовать условное предложение `WHERE` для выполнения фильтрации записей по любым критериям либо вложенную команду `SELECT`. Подробнее о применении `SQL` см. главу 16.

# Поиск записей

Обычно операции над данными подразумевают задание группы или подмножества записей. Создавая отчет, вводя команды модификации или удаления данных, вы, как правило, предполагаете воздействовать на определенную часть информации. Поэтому для начала рекомендуем изучить приемы сужения набора данных.

Существует два основных способа поиска записей, к которым впоследствии будет применена конкретная команда: вы можете создать и применить запрос Access, построить символьную строку, содержащую выражение на языке SQL, открыть объект набора данных (скажем, таблицу) и использовать циклическую VBA-процедуру, позволяющую отобрать нужные записи.

Команда SQL выполняется сервером базы данных (т.е. чаще всего другим — мощным — компьютером), поэтому производительность прикладной программы в этом случае окажется, вероятно, более высокой. Впрочем, код на языке VBA нередко демонстрирует завидную гибкость. Поэтому наилучший выбор — разумное сочетание обоих подходов.

Предварительное создание запроса значительно упрощает код. Если же вся работа по поиску записей возлагается на VBA-программу, код существенно вырастает в размерах и усложняется. Хотя во внешнем коде, как правило, приходится заботиться во вторую очередь, при прочих равных условиях они способны сыграть немалую роль. Неряшливый и запутанный код исправлять и сопровождать, разумеется, значительно труднее и накладнее.

Ниже мы рассмотрим оба способа поиска строк данных. Главное — определить подмножество записей, подлежащих обработке; все остальные операции более просты.

## Поиск в таблице

При поиске записей в таблице ее имя используется в качестве источника данных. Решение, целиком реализованное средствами VBA, является более гибким. В листинге 15.5 представлены способы поиска записей, удовлетворяющих заданным критериям.

### Листинг 15.5. Пример поиска записей в наборе данных средствами процедуры на языке VBA

```
1: Sub Test ()
2:   Call FindRecords("LAST_NAME", "Merchant")
3: End Sub
4:
5: Function OpenConnection( ) As ADODB.Connection
6:   Const Provider = "Provider=Microsoft.Jet.OLEDB.4.0;"
7:   Const DataSource =
8:     "Data Source=C:\Data\Hour15.mdb"
9:
10:
11:   Set OpenConnection = New ADODB.Connection
12:   Call OpenConnection.Open( Provider & DataSource, "", "" )
13:
14: End Function
15:
16: Sub FindRecords( ByVal SearchFieldName As String, _
17:   ByVal FindValue As String )
18:   Dim Connection As ADODB.Connection
19:   On Error GoTo Finally
20:   Set Connection = OpenConnection
```



```

21:
22: Dim RecordSet As New ADODB.Recordset
23:
24: Call RecordSet.Open( "MUSIC", Connection, _
25:     adOpenDynamic, adLockOptimistic )
26: RecordSet.MoveFirst
27: Do While Not RecordSet.EOF
28:     If (RecordSet(SearchFieldName) = FindValue) Then
29:         Debug.Print RecordSet( "TITLE" )
30:     End If
31:
32:     RecordSet.MoveNext
33: Loop
34:
35: RecordSet.Close
36: Connection.Close
37:
38: Finally:
39: If (Err.Number <> 0) Then
40:     Call MsgBox( Err.Description )
41:     Err.Clear
42: End If
43: Set RecordSet = Nothing
44: Set Connection = Nothing
45: End Sub

```



В строке 41 листинга 15.5 вызывается метод `Clear` объекта `Err`. Используется этот метод не всегда. Объект `Err` очищается при достижении конца процедуры обработки ошибки, выполнении команд `Exit Function`, `Exit Sub`, `Exit Property`, одной из команд `Resume` или следующего оператора `On Error`.

Можно использовать команду `Err.Clear`, чтобы не держать в памяти информацию о том, когда очищается данный объект.

#### Анализ

Строки 1–3 содержат тестовую процедуру, единственное назначение которой состоит в вызове функции `FindRecord`, реализующей основной алгоритм поиска. Подобные простые процедуры весьма удобны при последовательной отладке программы. Когда процесс тестирования завершен, такие процедуры можно просто удалить. Существуют специальные директивы компилятора, предназначенные для удаления вспомогательного кода (см. главу “17-й час. Отладка кода”). В строках 5–14 представлена отдельная функция, выполняющая задачу создания объекта соединения и его открытия. Обратите внимание, что функция возвращает значение типа `ADODB.Connection`.

Строки 16–45 демонстрируют текст процедуры поиска. В строке 18 объявляется переменная типа `Connection` — обратите внимание, что в данном случае служебное слово `New` не задается, поскольку задача непосредственного создания объекта соединения с выделением ему необходимого фрагмента памяти возложена на функцию `OpenConnection`. В строке 19 устанавливается обработчик, в случае возникновения ошибки передающий управление на метку `FINALLY`. Впрочем, код, следующий за меткой, выполняется всегда — не только в случае ошибки. Строка 20 устанавливает объект соединения — напомним, в операциях присваивания объектов используется служебное слово `Set`.

Строка 24 выполняет открытие набора данных, задаваемого таблицей MUSIC. Команда строки 26 перемещает указатель к первой записи набора. Строки 27-33 содержат конструкцию цикла, позволяющего итеративно просмотреть все записи таблицы. Суть алгоритма поиска заключена в условном выражении строки 28. Если искомая запись найдена, часть ее содержимого отображается в окне Immediate. Обратим особое внимание на использование методов MoveFirst (строка 26) и MoveNext (строка 32). Без этих двух строк код цикл либо никогда не выполнится, либо, начавшись, не будет завершен.

В строках 35-45 находятся команды закрытия объектов, очистки памяти и отображения информации об ошибке, если таковая произошла.

## Поиск с использованием запроса

Преимущество использования запроса состоит в том, что он способен возвращать наборы данных, содержащие информацию сразу из нескольких таблиц. С помощью запроса легко выполняется предварительная фильтрация полей и записей набора данных. Чтобы продемонстрировать применение запроса, внесем в код листинга 15.5 небольшие изменения. Представим запрос с помощью именованной символьной константы.

Листинг 15.6 содержит только ту часть текста листинга 15.5, которая подверглась изменениям.

### Листинг 15.6. Фрагмент листинга 15.5, измененный в связи с использованием запроса

```
22: Dim SQL As String
23: SQL = "SELECT * FROM MUSIC WHERE " & SearchFieldName & _
24:      "='" & FindValue & "'"
25: Dim RecordSet As New ADODB.Recordset
26: Call RecordSet.Open( SQL, Connection, adOpenDynamic, _
27:      adLockOptimistic )
28: RecordSet.MoveFirst
29: Do While Not RecordSet.EOF
30:     Debug.Print RecordSet( SearchFieldName )
31:     RecordSet.MoveNext
32: Loop
```

#### Анализ

Обратите внимание на появление новой символьной переменной SQL. Запрос, представленный в строке 23, "собирается" из нескольких литеральных констант, соединенных с содержимым аргументов SearchFieldName (наименование поля таблицы MUSIC, по которому осуществляется поиск) и FindValue (искомое значение, хранящееся в этом поле). В строке 26 процедуры Open в качестве параметра источника данных передается уже не имя таблицы, а название переменной, содержащей текст запроса на языке SQL.

Поскольку запрос сам по себе предусматривает фильтрацию записей посредством предложения WHERE, необходимость использования условного выражения в теле цикла отпадает. Предложение WHERE команды SELECT языка SQL способно содержать любое число аргументов в формате `ИмяПоля Оператор Значение`, где в качестве оператора используются операторы сравнения (скажем, =), а тип значения должен совпадать с типом поля (символьные значения заключаются в одинарные кавычки). Аргументы-предикаты "склеиваются" посредством логических операторов AND, OR и других, при необходимости их заключают в круглые скобки. Синтаксис предложения WHERE в упрощенной форме описывается следующим образом.

... WHERE ИмяПоля Оператор Значение [AND | OR ИмяПоля Оператор Значение ...]

Предложение WHERE может включать в себя также вложенные запросы. Более подробно вопросы программирования на языке SQL приведены в главе "16-й час. Применение языка SQL".

## Свойство Filter

Объект класса Recordset содержит полезное свойство — Filter. Переменная Filter принадлежит типу variant и выполняет роль, несколько схожую с назначением предложения WHERE конструкции SQL. Общий способ использования свойства Filter заключается в присваивании ему символьной строки, которая содержит одну или несколько записей-предикатов, оформленных в виде ИмяПоля Оператор Значение. Предикаты соединяются логическими операторами и при необходимости заключаются в круглые скобки. Если после строки 20 листинга 15.5 включить строку `RecordSet.Filter = SearchFieldName & "=" & FindValue & ""`,

то в дальнейшем условное выражение If больше не понадобится.

Итак, вы ознакомились с несколькими вариантами организации поиска записей в наборе данных. Какой из них выбрать в том или ином случае — решать вам. Все они помогут вам достичь цели; весь вопрос в том — насколько эффективно.

## Сохранение данных в коллекциях

Коллекции применяются во многих ситуациях. Класс Recordset, например, содержит коллекцию Fields, а многие визуальные компоненты (например списки) пользуются коллекциями для отображения данных на экране. Часто возникает необходимость в извлечении элемента коллекции, представляющей информацию в базе данных, и копировании его в коллекцию визуального компонента.

Единого способа взаимного копирования элементов коллекций не существует; впрочем, эта задача достаточно проста, и ниже предлагается способ ее решения. Вначале определим общую последовательность действий, необходимых для копирования элементов коллекций.

- Выбрать коллекцию, которая будет выступать в роли источника данных.
- Определить или создать экземпляр коллекции, принимающей данные.
- Построить цикл, на каждом шаге которого очередной элемент коллекции-источника будет считываться, а затем добавляться в коллекцию-приемник с помощью стандартного метода Add.

В листинге 15.7 представлен текст примера.

### Листинг 15.7. Пример копирования элементов одной коллекции в другую

```
24: Dim Target As New Collection
25: RecordSet.MoveFirst
26: Do While Not RecordSet.EOF
27:     Call Target.Add(RecordSet("FIRST_NAME").Value) & _
28:         " " & RecordSet("LAST_NAME").Value
29:     RecordSet.MoveNext
30: Loop
31:
32: Dim Artist As Variant
33: For Each Artist In Target
34:     Debug.Print Artist
35: Next Artist
36: Set Target = Nothing
37: RecordSet.Close
```

Приведенный код необходимо рассматривать в контексте примера листинга 15.5 (достаточно вставить его в листинг 15.5 после строки 24, в которой выполняется открытие набора записей RecordSet. Target (см. строку 25)). Это произвольная коллекция. После ее определения выполняется цикл, на каждом шаге которого считывается очередная запись набора данных RecordSet и значение объекта Field, связанного с полем ARTIST, заносится в новый элемент коллекции Target. Для определения факта достижения последней записи RecordSet используется метод EOF. В строке 28 в коллекции сохраняется поле, а не его значение.

Строки 33–35 содержат цикл, позволяющий отобразить в окне Immediate содержимое каждого элемента коллекции Target. Далее выполняется оставшаяся часть кода листинга 15.5.

Если придется работать с коллекциями других видов, достаточно незначительно изменить рассмотренный код. Самое существенное отличие примера от реальной программы состоит в нецелесообразности открывать/создавать обе коллекции в пределах одной и той же процедуры (функции) — коллекция Target, например, может принадлежать визуальному компоненту, размещенному на одной из экранных форм приложения.

## Использование AddItem и RemoveItem

В Microsoft Access 2002 в элементы управления список и поле со списком было добавлено два метода — AddItem и RemoveItem.



Метод AddItem получает два аргумента и добавляет элемент в коллекцию, представляющую собой список или поле со списком. Метод RemoveItem получает в качестве аргумента индекс и удаляет из коллекции элемент с этим индексом. Синтаксис вызова методов следующий:

Объект.AddItem(Строка, Индекс)  
Объект.RemoveItem(Индекс)

При работе с именем ComboNamesList в поле со списком необходимо сначала инициализировать список, используя массив (или другой источник данных) посредством кода, аналогичного приведенному в листинге 15.8. Если вы намерены добавлять элементы в список вручную, а не из источника данных, в диалоговом окне Properties установите свойство Row Source Type на Value List. Свойство RowSourceType можно изменить также в коде программы (см. приведенный ниже листинг).

### Листинг 15.8. Использование метода AddItem для добавления элементов списка

```
1: Private Sub Form_Load()  
2:   Dim Data As Variant  
3:   Data = Array("Иван Иванов", "Федор Федоров",  
4:     "Наталья Петрова", "Сидор Ильин")  
5:   ComboNamesList.RowSourceType = "Value List"  
6:   Dim I As Integer  
7:   For I = LBound(Data) To UBound(Data)  
8:     Call ComboNamesList.AddItem(Data(I), I)  
9:   Next I  
10:End Sub
```

В листинге 15.8 в качестве источника данных используется массив. Чтобы заполнить список вручную, необходимо задать свойство `RowSourceType` равным `Value List` (см. строку 5). В цикле (строки 7-9) выполняется копирование данных из массива в список. Второй аргумент — это индекс элемента в списке. Чтобы удалить элемент из списка или поля со списком, используйте метод `RemoveItem`. Помните, что при удалении происходит перенумерация оставшихся элементов списка, поэтому для выполнения этой операции не удастся использовать цикл `For Next`. Ниже приведен пример использования метода `RemoveItem`.

```
Do While (ComboNamesList.ListCount>0)
    Call ComboNamesList.RemoveItem(0)
Loop
```

В этом цикле элементы с нулевым индексом будут удаляться до тех пор, пока не исчезнут со списка.

А можно заполнить список данными из источника данных. Для этого задайте свойство `RowSourceType` равным `Table/Query` и укажите источник данных. Добавить в форму список или поле со списком вам поможет мастер, подробнее о котором рассказано в главе "19-й час. Создание экранных форм".

## Использование информационных ресурсов и поиск готовых решений

За последние годы выстроены горы программного кода и плещется море вспомогательных данных — ни одна книга не в состоянии вместить в себя всю эту информацию. Издатели состязаются в попытках выпуска самых разнообразных и объемных пособий, посвященных одной и той же теме. В этом кратком разделе будут названы те информационные источники, которые окажут вам эффективную помощь в освоении и использовании средств класса ADO.

Настоящее издание, разумеется, поможет взять удачный старт, но дорога, которую вам предстоит осилить, достаточно длинна. Вот в чем парадокс — тем, кто любит и умеет профаммировать, возможно, попросту не нужны тысячи страниц технической документации, но для новичка те же тысячи страниц могут стать непреодолимым препятствием.

Наилучший совет таков. С одной стороны, следует осваивать материал по мере необходимости, в процессе работы над конкретной задачей. Но если вы хотите добиться успеха, вам никуда не уйти от серьезной работы по изучению самых передовых и эффективных алгоритмов, приемов работы и образцов профаммного кода. Вы сможете начать профаммировать для Access сразу же по завершении чтения этой книги; но насколько далеко вы продвинетесь в профессиональном смысле — зависит от ваших дальнейших действий, желаний и настойчивости.

### Новый термин

*Объектная модель* — это иерархическая диаграмма классов, которые реализованы в той или иной инструментальной оболочке и используются для создания программ. Термин *программа* служит для обозначения исполняемого модуля, динамической библиотеки или компонента.

В сети World Wide Web вы найдете полезную информацию по этому вопросу, например, по адресу <http://www.microsoft.com/data/ado>.

## Резюме

Основная причина, побуждающая приобщиться к технологиям профаммирования на языке VBA для системы Access 2002, может быть связана с необходимостью решения задач обработки информации из баз данных. В ходе этого занятия вы изучили

приемы использования средств ActiveX Data Objects, реализованных в виде классов ADODB и ADOX, для выполнения наиболее распространенных операций над содержимым объектов базы данных.

Большая часть программного кода, решающего задачи обработки данных, связана с перемещением информации из базы данных в память (в форме, удобной для обработки компьютером или пользователем), а затем обратно в базу данных. Создание объектов соединений и наборов данных, а также управление ими — наиболее часто выполняемые операции, причем большая часть кода связана именно с обслуживанием наборов данных.

Класс ADODB содержит в своем составе большое число объектов разных классов, каждый из которых обладает собственным набором атрибутов — свойств и методов. Свойствам и методам присущи определенные особенности. Запомнить все их тонкости совершенно нелегко даже после сотен часов практической работы. К тому моменту, когда вы будете читать эти строки, библиотеки VBA наверняка пополнятся новыми классами либо Microsoft выдумает нечто решительно новое. Вот почему настоятельно рекомендуется в своей деятельности полагаться на качественные, прошедшие проверку временем решения других разработчиков, настойчиво изучать их опыт и почаще обращаться за помощью к оперативной справочной системе и другим качественным информационным источникам.

Приведенные ниже разделы "Вопросы и ответы" и "Задания" дадут вам еще одну возможность проверить, насколько прочно усвоен материал, пройденный на этом занятии.

## Вопросы и ответы

**Вопрос.** Каков, по вашему мнению, наилучший способ изучения языка программирования SQL?

**Ответ.** Издательство *Sams* (и *Издательский дом "Вильямс"*. — *Прим. ред.*) предлагают широкий выбор книг по программированию на SQL и смежным темам. Специальную литературу можно приобрести также у поставщиков систем управления базами данных. Единственное, что необходимо оговорить, — не все диалекты SQL взаимозаменяемы.

**Вопрос.** Какие положительные моменты имеет объектная модель?

**Ответ.** Объектная модель представляет собой наглядную диаграмму, иллюстрирующую состав классов и их взаимосвязи.

**Вопрос.** Каким образом отсортировать записи в наборе данных?

**Ответ.** В вашем распоряжении имеются предложение `ORDER BY` конструкции `SELECT` языка SQL, свойство `Order` объекта `ADODB.Recordset`, а также свойство `SortOrder` объекта `ADOX.Column`.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Назовите синоним термина *провайдер*.
2. С помощью какого предложения SQL могут быть отфильтрованы данные в запросе?

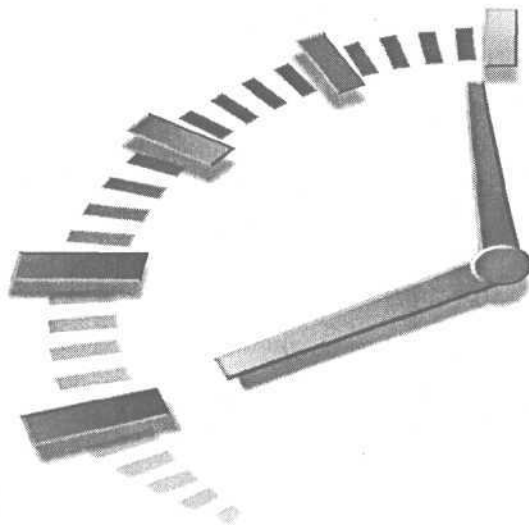
3. Какое свойство ограничивает данные, доступные в записи?
4. Назовите имя свойства для представления данных, хранящихся в поле таблицы. Объекту какого класса оно принадлежит?
5. Как называется коллекция в составе объекта `Recordset`, предназначенная для хранения данных?

## Упражнения

1. Исправьте текст запроса в листинге 15.6, чтобы предусмотреть возможность сортировки записей в порядке возрастания по значению поля, название которого задано переменной `SearchFieldName`. (Совет: воспользуйтесь предложением `ORDER BY`.)
2. Поясните, каким образом код листинга 15.7 можно еще более упростить и унифицировать.
3. Продемонстрируйте, как связать элементы произвольной коллекции с записями набора данных. (Совет: воспользуйтесь ключевым полем таблицы.)

# 16-й час

## Применение языка SQL



Программирование для Access связано, в основном, с использованием языка Visual Basic for Applications (VBA). Язык Structured Query Language (SQL — язык структурированных запросов; аббревиатура, произносится примерно как “сикуэл”) — это вовсе не то же самое, что VBA. VBA и SQL — два различных языка программирования: VBA универсален и ориентирован на самое широкое применение, а SQL используется исключительно в задачах управления базами данных.

Ни одна из книг, посвященных программированию для Access, не будет полной без более или менее подробного рассмотрения приемов программирования с помощью SQL. На прошлом занятии вы изучили способы использования средств класса ADODB. Следует знать о том, что методы ADODB при обращении к серверу базы данных выполняют неявное преобразование фрагментов написанного вами кода VBA в выражения на языке SQL. Все происходит именно так: и VBA, и другие инструменты программирования взаимодействуют с системами управления базами данных посредством команд SQL. Особенности подобных процессов зависят от конкретной среды разработки; впрочем, важно другое — почему используется именно SQL?

SQL — это мощный язык программирования, ориентированный на применение в базах данных. Несмотря на чрезвычайную гибкость VBA, подчас целесообразно обращаться к мощным средствам SQL напрямую. Приведем наглядный пример. Программируя в Access, для хранения информации вы можете использовать, помимо инструментов Access, и средства иных серверов баз данных — скажем, MS SQL Server. SQL Server и другие “большие” системы управления базами данных обычно размещаются на отдельных компьютерах, гораздо более мощных и производительных по сравнению с персональными станциями пользователей. Разумеется, мощный сервер способен решить ту же задачу значительно быстрее.

С помощью SQL вы можете заставить программу выполняться на сетевом сервере (а не на станции пользователя), т.е. с большей скоростью. Это же справедливо и в отношении баз данных Access.

Удобство использования — еще один немаловажный фактор. Язык SQL был создан специально для работы с базами данных. Поэтому часто проще написать одно выражение SQL вместо целого блока VBA-кода, выполняющего те же действия.



Различные диалекты и реализации SQL далеко не равнозначны. Хотя многие из них целиком или частично поддерживают стандарты SQL, разработанные ANSI (American National Standards Institute — Американским институтом национальных стандартов, неправительственной организацией, которая создает промышленные стандарты, не обязательные в применении. — *Прим. перев.*), существуют и более принципиальные отклонения. На этом занятии вы научитесь создавать код SQL, поддерживаемый Access 2002. Перенести его в среду другой системы управления базами данных сопряжено с внесением в текст программы некоторых изменений. Если, тестируя такой код, вы столкнетесь с проблемами, обратитесь к фирменной документации, поставляемой в комплекте с программным обеспечением сервера базы данных.

Итак, этот урок посвящен вопросам программирования на SQL в среде Access 2002.

Основные темы занятия.

- Различные формы команды SELECT.
- Изменение содержимого базы данных с помощью команд UPDATE, INSERT и DELETE.
- Объекты ADODB и хранимые процедуры Access.

## Использование выражения SELECT

Команда SELECT — одно из самых популярных, многогранных и гибких выражений языка SQL. SELECT возвращает набор данных, отобранных на основании указанных критериев фильтрации и упорядоченных в соответствии с заданными условиями. Выражение SELECT способно принимать как простую, так и чрезвычайно сложную форму, позволяющую реализовать точную и исчерпывающую логику отбора данных. В пределах команды SELECT может одновременно выполняться несколько операций — фильтрация, группировка, сортировка данных и вычисления. Ниже мы рассмотрим некоторые формы команды разной степени сложности. И начнем мы с нескольких достаточно простых примеров.

## Простые формы SELECT

Команда SELECT в своей самой простой форме состоит из служебного слова SELECT. За ним следуют список необходимых полей и предложение FROM, которое указывает на таблицу, содержащую названные поля:

SELECT СписокПолей FROM ИмяТаблицы



В соответствии с общепринятым соглашением служебные слова SQL вводятся в верхнем регистре (как это сделано в тексте синтаксической формулы, приведенной выше). Стоит соблюдать это правило или нет — решать вам; главное — будьте последовательны в своих действиях.

Интересно, что в практике виртуального общения в Internet слова, набранные прописными буквами, служат для обозначения повышенной "громкости" речи. Впрочем, SQL был придуман задолго до появления электронной почты, программы NetMeeting и переговоров в режиме chat.

SELECT и FROM — это служебные слова SQL: SELECT обозначает название команды, а FROM — предложение, представляющее источник данных. В качестве параметра СписокПолей допускается применять либо символ звездочки (\*), указывающий на то, что в результат выборки должны быть включены все поля таблицы, либо перечень наименований полей, разделенных символом запятой. Вы можете отобрать любое число полей таблицы.



Далее мы будем обращаться к той же базе данных, которая использовалась нами на прошлом занятии. Предполагается, что база содержит две таблицы. Первая носит название MUSIC, состоит из следующих полей: ID AutoNumber, ARTIST Text(50), TITLE Text(50), FORMAT Text(50), PUBLISHER Text(50) — и предназначена для хранения сведений о музыкальных альбомах. Структура второй таблицы, TRACKS, содержащей информацию об отдельных композициях альбомов, описывается таким перечнем полей: ID AutoNumber, MUSIC\_ID Number, TRACK\_TITLE Text(50) и TRACK\_LENGTH Date/Time.

Например, чтобы получить полное содержимое таблицы MUSIC (т.е. все ее записи со всеми полями), достаточно выполнить команду

```
SELECT * FROM Music
```

Чтобы протестировать указанную команду, вы должны предварительно создать таблицу с перечисленными выше полями и заполнить ее информацией о любимых звукозаписях. Затем выполните следующие действия.

1. Откройте базу данных, содержащую таблицу MUSIC.
2. В окне базы данных выберите элемент Запросы (Queries) списка Объекты (Objects).
3. Щелкните на кнопке Создать (New) панели инструментов.
4. В диалоговом окне Новый запрос (New Query) выберите режим Конструктор (Design View) и щелкните на кнопке OK.
5. Откроется диалоговое окно Добавление таблицы (Show Table). Щелкните на кнопке Заккрыть (Close). На экране останется открытым окно Запрос на выборку (Query Designer).
6. Выберите в строке меню главного окна Access команду Вид⇒Режим SQL (View⇒SQL View).
7. Введите текст команды **SELECT \* FROM Music**.
8. Чтобы выполнить команду SQL, выберите в строке меню Запрос⇒Запуск (Query⇒Run).

Далее в ходе этого занятия вы сможете, если не будет дополнительных указаний, использовать данную инструкцию для тестирования всех примеров кода на языке SQL. В некоторых случаях будут демонстрироваться и другие способы создания SQL-кода, не требующие ввода текста в окне базы данных.



Не забывайте почаще сохранять результаты своей работы.

Работая над текстом запроса, можно сохранить его в любой момент — достаточно выбрать команду меню Файл⇒Сохранить (File⇒Save) (или Файл⇒Сохранить как (File⇒Save As), если операция выполняется впервые).

## Фильтрация данных с помощью предложения WHERE

Предложение WHERE команды SELECT используется в тех случаях, когда необходимо ограничить множество записей, возвращаемых запросом. Предложение WHERE, помимо выражения SELECT, употребляется и в других конструкциях SQL. Сведения, которые приведены в этом разделе, в равной степени применимы к командам SELECT, UPDATE И DELETE.

В конструкции SELECT предложение WHERE занимает место после FROM. Синтаксис WHERE в контексте выражения SELECT таков:

```
SELECT СписокПолей
      FROM ИмяТаблицы
      WHERE ИмяПоля Оператор Значение [OR | AND ИмяПоля Опе-
ратор Значение ...]
```

Предложение WHERE располагается после предложения FROM. За служебным словом WHERE следует набор предикатов в формате ИмяПоля Оператор Значение. Количество предикатов (выражений сравнения) не ограничено. В качестве аргумента Оператор используется любой из обычных операторов сравнения. ИмяПоля — это наименование одного из полей таблиц, имеющих отношение к запросу, а Значение — величина соответствующего типа (символьные литералы заключаются в одинарные кавычки).

Предикату может предшествовать служебное слово NOT, обозначающее унарный оператор отрицания. Предикаты "склеиваются" посредством логических операторов конъюнкции (AND) и дизъюнкции (OR) (реже — других). Листинг 16.1 содержит несколько примеров запросов, каждый из которых легко протестировать в окне Запрос на выборку. Все они предназначены для получения данных из таблицы MUSIC.

#### Листинг 16.1. Примеры использования команды SELECT

```
1: SELECT * FROM Music
2: SELECT * FROM Music WHERE Artist = 'Elvis Presley'
3: SELECT * FROM Music WHERE Publisher = 'Elektra' AND _
   Artist <> 'Natalie Merchant'
4: SELECT * FROM Music WHERE Artist LIKE 'Natalie*'
5: SELECT * FROM Music WHERE Artist <> 'Elvis Presley' AND _
   Artist LIKE 'Elvis*'
6: SELECT * FROM Music WHERE NOT (Artist LIKE 'Natalie*')
```

#### Анализ

Каждая строка листинга 16.1 — это отдельный запрос. Строка 1 возвратит все записи таблицы MUSIC. Запрос строки 2 предполагает отбор только тех записей, которые имеют отношение к Элвису Пресли (Elvis Presley). В строке 3 выполняется поиск строк данных о музыкальных альбомах (выпущенных звукозаписывающей компанией Elektra) всех исполнителей, кроме Натали Мэтчент (Natalie Merchant). Строка 5 содержит пример операции поиска по маске. Символ звездочки (\*) в тексте искомого аргумента обозначает любую последовательность символов (в том числе и нулевой длины). Так, запрос строки 4 возвратит информацию о звукозаписях всех исполнительниц по имени Natalie (может быть, даже Наташи Королевой). Следующая команда предполагает поиск записей всех Элвисов, кроме знаменитого Элвиса Пресли (вероятно, собранный урожай сведений окажется не слишком богатым, если только вы не ярый поклонник Элвиса Костелло (Elvis Costello)). Наконец, запрос последней, 6-й, строки возвратит все записи таблицы MUSIC, кроме тех, в которых упоминается любая из Natalie.

## Операторы, применяемые в предложении WHERE

SQL — довольно выразительный язык программирования. Реализация SQL, поддерживаемая Access 2002, совместима со стандартом ANSI-89. Чтобы изменить версию ANSI-89 на ANSI-92, выберите команду Сервис⇒Параметры (Tools⇒Options), перей-

дите на вкладку Таблицы и запросы (Tables/Queries) и установите флажок Текущая база данных (This Database). Чтобы реализация SQL была совместима с ANSI-92 по умолчанию, на вкладке Другие (Advanced) диалогового окна Сервис⇄Параметры из списка Формат файла по умолчанию (Default File Format) выберите Access 2002. Выберите данную опцию, если планируете преобразовать базу данных в формат SQL-сервер. При необходимости всю информацию, которая касается стандартов SQL, разработанных ANSI, вы сможете найти по адресу <http://www.ansi.org>.

В пределах предложения WHERE, помимо привычных логических операторов и операторов сравнения, допускается использование также других конструкций, обозначаемых служебными словами BETWEEN и IN. Оператор BETWEEN позволяет задать верхнюю и нижнюю границы диапазона изменения Значений поля, а оператор IN дает возможность указать список искомых значений.

## Оператор BETWEEN

Оператор BETWEEN, сопровождаемый служебным словом AND, используется в контексте предложения WHERE для задания границ интервала изменения величины аргумента (граничные значения включаются в интервал). Листинг 16.2 содержит короткий пример.

Листинг 16.2. Пример использования оператора BETWEEN

```
SELECT * FROM Music
WHERE Artist BETWEEN 'Elvis Presley' AND 'Rolling Stones'
```

### Анализ

Единственная команда SELECT возвратит набор данных, содержащих сведения об альбомах исполнителей, имена которых — в алфавитном порядке — попадают в интервал, простирающийся от Elvis Presley до Rolling Stones.

## Оператор IN



Оператор IN позволяет задать набор искомых значений поля таблицы. Синтаксис конструкции IN в контексте команды SELECT описывается следующей формулой:

```
SELECT СписокПолей
FROM ИмяТаблицы
WHERE ИмяПоля IN (Значение1, Значение2, ...)
```

Легко привести пример использования оператора IN в команде поиска данных в таблице MUSIC:

```
SELECT * FROM Music WHERE Publisher IN ('Polygram', 'Elektra', 'Capitol')
```

### Анализ

Указанная команда SELECT вернет все строки данных, которые имеют отношение к звукозаписям, выпущенным компаниями Polygram, Elektra и Capitol.

## Логические операторы

Логические операторы конъюнкции (AND), дизъюнкции (OR) и отрицания (NOT) действуют в SQL точно так же, как и в VBA. Операторы AND и OR бинарны, т.е. требуют наличия двух операндов, а NOT — унарный оператор, воздействующий на единственный аргумент. Примеры использования операторов AND и NOT приведены в листинге 16.1.

Предложение WHERE может содержать любое число предикатов, соединенных логическими операторами и сгруппированных с помощью круглых скобок. Операторы SQL (что естественно) обладают определенным приоритетом выполнения. Однако запомнить значения приоритетов достаточно сложно. Изучите выражения SQL, приведенные ниже.

```
SELECT * FROM Music
    WHERE Publisher = 'Capitol' AND Last_Name = 'Cocker'
OR Last_Name = 'Merchant'
SELECT * FROM Music
    WHERE Publisher = 'Capitol' AND (Last_Name = 'Cocker' OR
Last_Name = 'Merchant')
```



В булевой логике оператор AND аналогичен умножению, а OR — сложению. Поэтому приоритеты выполнения операторов таковы: первым вычисляется AND, затем OR (если, конечно, не расставить скобки).

#### Анализ

Оба выражения содержат почти одинаковый текст, но возвращают совершенно разные результаты. В первом случае вначале вычисляется логическое выражение AND, соединяющее два предиката (`Publisher = 'Capitol' AND Last_Name = 'Cocker'`), а затем — оператор OR. Поэтому будут найдены записи первого артиста (Joe Cocker) в издательстве Capitol и записи Natalie Merchant во всех издательствах. Второй оператор сначала выполняет действие в скобках, поэтому будут найдены записи обоих артистов в одном издательстве — Capitol.



Намереваясь использовать в предложении WHERE несколько предикатов, уточните свой замысел с помощью скобок, а не полагайтесь на якобы твердое знание приоритетов выполнения операций.

## Предложение WHERE и вложенные команды SELECT

В качестве аргумента Значение предложения WHERE может выступать как значение простого типа (иллюстрируемое выше), так и результат выполнения вложенной команды SELECT. Подобно другим языкам программирования (например, VBA), SQL в каждой конкретной реализации имеет собственный синтаксический анализатор, способный разобрать написанный вами код буквально по "косточкам".

Вложенная команда SELECT может быть настолько же сложной, как и любая другая. Первой всегда выполняется наиболее глубоко "спрятанная" команда SELECT. Хотя с формальной точки зрения вы вольны конструировать выражения любой степени сложности, однако, выходя за "разумные" рамки одного уровня вложенности, вы обрекаете сервер базы данных на серьезные испытания.

Вложенная конструкция SELECT выступает в роли правого операнда предиката в предложении WHERE. Приведем соответствующий пример:

```
SELECT * FROM Music
    WHERE Title IN
        (SELECT Track_Title FROM Tracks)
```

Чтобы понять смысл этого запроса, рассмотрение нужно начать с внутренней команды SELECT, которая предполагает выбор значений поля TRACK\_TITLE всех записей таблицы TRACKS. Внешний запрос подразумевает возврат всех полей таких записей

таблицы MUSIC, в поле TITLE которых содержится любое из значений, полученных в результате выполнения вложенного запроса.

Необходимо высказать одно предостережение. Легко упустить из виду, что на самом деле смысл данных столбца TRACK\_TITLE таблицы TRACKS не отвечает назначению столбца TITLE таблицы MUSIC, т.е., образно говоря, предпринимается попытка сопоставить "шило" с "мылом". Выражение, корректное синтаксически, далеко не всегда верно с логической точки зрения.

Исправим оплошность, построив более простое выражение, в котором значение поля первичного ключа ID таблицы MUSIC сравнивается с величиной, хранящейся в поле внешнего ключа MUSIC\_ID таблицы TRACKS. В этом случае команда SQL примет следующий вид:

```
SELECT * FROM Music, Tracks
WHERE Music.Id = Tracks.Music_Id AND (Music.Title LIKE_
                                     Tracks.Track_Title)
```

#### Новый термин

**Первичный ключ (Primary Key)** — это один или несколько столбцов таблицы, значения которых однозначно определяют уникальную запись таблицы. Первичный ключ одновременно служит и главным индексом таблицы.

#### Новый термин

**Внешний ключ (Foreign Key)** устанавливает правило, в соответствии с которым в качестве значений одного или нескольких полей одной таблицы могут использоваться только такие комбинации величин, которые существуют в первичном ключе другой таблицы.

Понятия первичного и внешнего ключей служат (извините за каламбур) “ключами” к осмыслению основ теории реляционных баз данных.



Более подробную информацию о вложенных запросах вы сможете получить, адресовав Помощнику Microsoft Office словосочетание поиска *Подчиненные запросы*.

Если две или несколько таблиц в рамках предложения WHERE связаны операторами сравнения однородных полей, принято говорить об их *неявном объединении* (implicit join). Механизмы объединения таблиц более подробно рассмотрены ниже, в одноименном разделе этой главы.

Мы отнюдь не утверждаем, что вложенные запросы заведомо неэффективны и вредоносны. Подчас они могут оказаться как раз тем инструментом, который необходим и целесообразен. Однако знайте, что подчиненные запросы — это не то средство, к которому можно обращаться без особой нужды.

## Сортировка данных

Конструкция выражения SELECT позволяет упорядочить возвращаемые наборы данных по убыванию или возрастанию значений полей. Этой цели служит предложение ORDER BY, после которого указываются одно или несколько выражений вида ИмяПоля [ПорядокСортировки], разделенных символом запятой. Аргументы ИмяПоля указывают на поля таблиц, перечисленных в предложении FROM, а в качестве необязательного параметра ПорядокСортировки допустимо использовать одно из служебных слов — ASC или DESC. ASC означает, что записи будут упорядочены по возрастанию величин, хранящихся в указанном поле, а DESC предполагает обратный порядок сортировки — по убыванию значений поля. В результате выполнения следующего запроса набор данных будет отсортирован в порядке убывания значений поля ARTIST таблицы MUSIC, а записи с совпадающими значениями в поле ARTIST — по возрастанию содержимого поля TITLE.

```
SELECT * FROM Music
ORDER BY Artist DESC, Title ASC
```

По умолчанию, т.е. в случае отсутствия одного из компонентов (ASC или DESC), предлагается порядок сортировки по возрастанию.

## Группировка столбцов

Предложение GROUP BY применяется для группировки данных в столбцах. К нему необходимо обращаться при использовании так называемых агрегирующих функций языка SQL, например SUM.

Предложение GROUP BY, подобно ORDER BY, в своей общей форме требует задания списка наименований полей, разделенных запятыми.

Группируя данные по определенным столбцам возвращаемого набора, следует включить в группу либо все столбцы набора данных, либо те из них, которые не использованы в качестве аргументов агрегирующих функций.

Предложение GROUP BY применяется в тех случаях, когда необходимо получить только одну строку из группы строк, в определенных столбцах которых хранятся идентичные значения. Например, таблица TRACKS содержит внешний ключ MUSIC\_ID, ссылающийся на первичный ключ ID таблицы MUSIC. Выполнив запрос

```
SELECT MUSIC_ID FROM TRACKS,
```

мы получим столько значений, сколько композиций описано в таблице TRACKS, причем числа будут повторяться. Чтобы избежать повторений одинаковых величин (т.е. единожды сослаться на каждый музыкальный альбом), необходимо использовать команду

```
SELECT Music_Id FROM Tracks GROUP BY Music_Id
```

В результате мы получим набор уникальных значений. Представим следующий запрос — он вычисляет количество композиций каждого музыкального альбома (теперь без GROUP BY просто не обойтись):

```
SELECT Music_Id, COUNT( Music_Id )
FROM Tracks
GROUP BY Music_Id
```

Указанная команда возвратит набор записей, перечисляющих уникальные коды музыкальных альбомов наряду с числом композиций в каждом из них. Функция COUNT вычисляет количество записей с одинаковыми значениями MUSIC\_ID, а предложение GROUP BY Music\_Id гарантирует, что будет возвращена только одна запись для каждой группы совпадающих значений MUSIC\_ID.

## Использование предложения HAVING

SQL не разрешает ссылаться на агрегирующие функции в контексте предложения WHERE. Но иногда необходимо гарантировать соответствие возвращаемого набора данных определенным условиям. Предложение HAVING подобно WHERE, в частности, оно помогает ограничить объем множества данных, получаемых в результате выполнения SELECT.

HAVING позволяет включать любое число предикатов, объединенных посредством булевых логических операторов.

Листинг 16.3. демонстрирует пример использования предложения HAVING и оправданного применения вложенного запроса.

Листинг 16.3. Пример использования предложения HAVING и вложенного запроса

```
1: SELECT * FROM Music WHERE Id =
2:     (SELECT Music_Id FROM Tracks
```

```
3:      GROUP BY Music_Id
4:      HAVING CDATE(SUM(Track_Length)) > CDATE("0:6:0")
```

### Анализ

Строка 1 содержит заголовок внешнего запроса, а текст подчиненного запроса расположен в строках 2–4. Подзапрос группирует записи таблицы TRACKS в соответствии со значениями поля MUSIC\_ID. Предложение HAVING в строке 4 осуществляет сравнение суммы продолжительности звучания всех композиций каждого альбома с константой, равной шести минутам.

В результате выполнения всего запроса будет возвращен набор записей таблицы MUSIC, для каждой из которых существует внешний ключ из таблицы TRACKS и удовлетворяется условие подчиненного запроса. Строка 4 демонстрирует пример употребления встроенной SQL-функции CDATE, выполняющей преобразование числа или строки в значение типа DATETIME. В нашем случае с помощью функции CDATE осуществляется сопоставление суммы временных интервалов (длительностей), выраженных в секундах, с литеральной константой, равной шести минутам. Примеры использования некоторых встроенных функций SQL рассмотрены ниже, в одноименном разделе этой главы.

## Объединение таблиц

Одна из основных целей, которая была поставлена и достигнута разработчиками модели реляционной базы данных, связана с необходимостью исключения неоправданных потерь пространства для хранения данных. В прежние времена программисты, которые создавали базы данных, должны были заранее фиксировать число полей таблицы, требуемых для хранения элементов информации определенных типов — скажем, телефонных номеров. В связи с этим проблем возникало немало: программист предусмотрел, например, два таких поля, а в некоторых случаях следовало бы иметь три.

Что делал несчастный с подобной "плоской" базой данных, в которой все данные о некотором объекте должны были содержаться в пределах единой монолитной записи? Да, он вынужден был изменять структуру таблицы, добавляя в нее недостающие поля. Через какое-то время все мучения повторялись. А что происходило с теми записями, в которых по-прежнему хранились два телефонных номера, а не три или четыре? Добавленные поля оставались пустыми, невостребованными, т.е. место в таблице (и на диске) безвозвратно терялось.

Те, кто старался предвидеть ход событий, сознательно проектировали таблицы, содержащие множество "дыр". Приверженцы другой крайности, пытавшиеся учесть каждый лишний байт, вынуждены были постоянно "упражняться" с процедурами реструктуризации базы данных и все время исправлять тексты прикладных программ.

Мрачная картина, не правда ли? Но тут на помощь пришли реляционные базы данных (от англоязычного термина Relational Databases — **RDBMS**. — *Прим. перев.*). Реляционная модель позволяет распределять порции информации, имеющей отношение к определенному объекту, по нескольким таблицам вместо одной, монолитной и нерасчленимой. Каждая таблица предназначена для хранения определенного подмножества данных.

Например, база данных о покупателях может состоять, скажем, из двух таблиц, одна из которых содержит некоторые "статические" поля (имя, почтовый адрес и пр.), а другая предназначена для хранения переменного числа записей, относящихся к каждому покупателю (например, информации о номерах кредитных карточек). Теперь набор кредитных карточек каждого покупателя будет описываться именно таким количеством записей таблицы, какое нужно, — не больше и не меньше.

Реляционные базы данных предполагают наличие механизма объединения хранящейся в нескольких таблицах информации об объекте в целостную "картинку". Прежде для решения подобной задачи достаточно было переместиться к конкретной записи единственной таблицы, — конечно, такой способ намного проще, но он сопряжен с потерями, о которых мы говорили выше.



Число таких таблиц реляционной базы данных, которые содержат однородные порции определенной информации (имеющей отношение к объектам одного типа), не ограничено — их может быть две, три и более. Объединение выполняется с помощью дополнительных *ключевых* столбцов и, безусловно, более трудоемко в практической реализации по сравнению с методами обработки "плоских" таблиц — но выигрыш, тем не менее, очевиден.

Процесс и результат сбора таких данных об определенном объекте, которые хранятся в нескольких таблицах, принято называть *объединением* таблиц. В разделе "Предложение WHERE и вложенные команды SELECT" речь шла об одном из вариантов этого механизма — так называемом неявном объединении, когда несколько однородных полей данных из нескольких таблиц сопоставляются в контексте предложения WHERE. В этом случае возвращаемый набор данных будет содержать все записи с найденными соответствиями. Разумеется, это полезно. Однако ситуация может усложниться, как в примере, рассмотренном выше (в нем речь шла о таблицах, содержащих сведения о покупателях и их кредитных карточках), в случае, если у каких-либо покупателей просто нет кредитных карт. Просмотрите такой гипотетический запрос:

```
SELECT * FROM Customer, Credit_Cards
WHERE Customer.Id = Credit_Cards.Customer_Id
```

(Выражение `Customer.Id` означает ссылку на поле ID таблицы CUSTOMER. Это пример использования общепотребительного — полного — синтаксиса обращения к полю таблицы.) Как известно, `CREDIT_CARDS.CUSTOMER_ID` — внешний ключ, ссылающийся на поле `CUSTOMER.ID`. Если таблица `CREDIT_CARDS` не содержит сведений о кредитных карточках какого-либо покупателя, предложение WHERE не "сработает" и в возвращенном наборе данных этот покупатель вообще не будет упомянут. Теперь представьте последствия подобного поведения программы, обрабатывающей счета за услуги, которые фирма предоставила потребителям: "незамеченные" счастливицы торжествуют, вы, автор шедевра, уволены, у шефа инфаркт, компания — на грани банкротства. ("А в остальном ... все хорошо!")

Это как раз тот случай, когда вас выручит выражение на основе служебного слова JOIN. Оператор JOIN работает с двумя аргументами-таблицами: первую называют *левой*, а вторую — *правой*. Существует три разновидности конструкций JOIN — INNER JOIN, LEFT JOIN и RIGHT JOIN. Каждая из них служит определенной цели, и о них будет рассказано более подробно.

## Оператор INNER JOIN

Конструкция INNER JOIN равнозначна условию эквивалентности, используемому в предложении WHERE. Оператор INNER JOIN позволяет вернуть все записи, для которых выполняется условие равенства содержимого столбцов двух объединяемых таблиц. Приведем соответствующий пример:

```
SELECT *
FROM Music INNER JOIN Tracks ON Music.Id = Tracks.Music_Id
```

Данная команда возвратит все записи таблиц MUSIC и TRACKS, для которых `MUSIC.ID = TRACKS.MUSIC.ID`. Она равносильна следующему выражению:

```
SELECT * FROM Music, Tracks
WHERE Music.Id = Tracks.Music_Id
```

## Оператор LEFT JOIN

Оператор LEFT JOIN применяется в тех случаях, когда следует вернуть все записи левой таблицы и только те строки правой, значения полей которых соответствуют данным левой таблицы. Поэтому в полях *правой* таблицы возвращенного множества данных допускаются значения null. Рассмотрим пример:

```
SELECT *
```

```
FROM Music LEFT JOIN Tracks ON Music.Id = Tracks.Music_Id
```

В результате выполнения указанного запроса будут возвращены все записи таблицы MUSIC — даже те, для которых в таблице TRACKS нет соответствий. В последнем случае поля результата, относящиеся к таблице TRACKS, окажутся пустыми.

## Оператор RIGHT JOIN

Конструкция RIGHT JOIN прямо противоположна по назначению оператору LEFT JOIN, рассмотренному выше. При использовании RIGHT JOIN возвращенный набор данных будет содержать все записи правой таблицы и только те строки левой, для которых в правой таблице имеются соответствия. Например:

```
SELECT *
```

```
FROM Music RIGHT JOIN Tracks ON Music.Id = Tracks.Music_Id
```

Результат выполнения запроса будет содержать все записи таблицы TRACKS, включая те, для которых отсутствуют ключевые значения в таблице MUSIC.

## Объединение запросов

Два запроса SQL, возвращающие одинаковое число полей совместимых типов, разрешается объединять с помощью служебного слова UNION. Запросы не зависят от соседних, но выполняются вместе, как одна команда SQL, давая в результате единый набор данных. По умолчанию оператор UNION устраняет из возвращенного множества данных повторяющиеся строки. Чтобы в результат включались все записи, после оператора UNION необходимо добавить служебное слово ALL.

Никто не запрещает вам использовать конструкции UNION, действующие как единственное выражение SELECT с оператором OR, хотя это не самый удачный выбор. Лучший вариант применения связан с необходимостью сочетания в одном возвращенном наборе данных, близких по природе, но расположенных в разных таблицах. Приведенный ниже пример возвращает значения столбца ID из таблиц MUSIC и TRACKS.

```
SELECT Id FROM Music
```

```
UNION
```

```
SELECT Id FROM Tracks
```

В результате будут получены все значения столбцов ID таблиц MUSIC и TRACKS. Конечно, этот пример нельзя назвать полезным с практической точки зрения — он просто демонстрирует технику работы. Но если предположить, что у вашего друга есть собственная фонотека (и своя таблица Access такой же структуры MUSIC) и вы хотели бы составить общий каталог записей, тогда UNION — как раз то, что нужно.

## Переименование столбцов результата

Встречаются ситуации, когда бывает полезным объединить данные двух или более столбцов результата выборки в один столбец. Более детально рассмотрим столбец ARTIST таблицы MUSIC. Если необходимо отсортировать записи по фамилии исполнителя, без дополнительного кода сделать это будет довольно трудно. Невольно напрашивается вывод о необходимости разбиения столбца ARTIST на два — скажем, FIRST\_NAME и LAST\_NAME. Действительно, это достаточно гибкое решение. Но возникают подозрения, удастся ли в дальнейшем получать такие же результаты (т.е. с корректно отформатированным полным именем исполнителя), как и прежде.

Проблема решается с помощью средств переименования столбцов результата запроса. SQL позволяет объединять данные нескольких столбцов в один и присваивать ему новое имя посредством оператора AS. Предположим, что мы все-таки расчленили

столбец ARTIST на два, FIRST\_NAME и LAST\_NAME, предназначенных для хранения имен и фамилий. Тогда выражение запроса, возвращающего полные имена исполнителей, могло бы выглядеть так:

```
SELECT First_Name + ' ' + Last_Name AS Artist FROM Music
ORDER BY Last_Name
```

В новой редакции таблицы MUSIC уже нет столбца с именем ARTIST — вместо него созданы столбцы FIRST\_NAME и LAST\_NAME. Приведенный выше запрос "склеивает" значения имен и фамилий в единое целое под привычным именем — ARTIST. Поскольку фамилии и имена хранятся отдельно, стало возможным использовать предложение ORDER BY Last\_Name.



Оператор AS особо пригодится в тех случаях, когда результат отбора содержит вычисляемые значения или неудачно названные столбцы таблиц.

Разбиение столбца ARTIST на две составляющие — пример, полезный сам по себе. В этом случае вы получаете более гибкие возможности упорядочения и поиска данных.

## Добавление записей

Команда INSERT INTO позволяет добавлять записи в таблицу базы данных и допускает несколько способов применения. Все они рассмотрены ниже.

## Добавление данных в указанные поля

Наиболее употребительный вариант использования команды INSERT INTO предусматривает добавление записи в существующую таблицу с указанием списка полей. Ниже приведена синтаксическая формула подобного выражения:

```
INSERT INTO ИмяТаблицы (ИмяПоля1 [, ИмяПоля2, ...])
VALUES (Значение1 [, Значение2, ...])
```

В верхнем регистре набраны служебные слова SQL. После фразы INSERT INTO указывается имя таблицы, за которым следует список наименований полей, заключенный в круглые скобки. Список может содержать только те поля, в которые вы хотите занести значения (если поле помечено признаком обязательного заполнения, его имя должно присутствовать в списке. — *Прим. перев.*). Количество значений, перечисленных в круглых скобках после служебного слова VALUES, и их типы должны соответствовать содержимому списка полей.

Следующий пример иллюстрирует процедуру пополнения реестра музыкальной коллекции, рассматриваемой ранее, данными о новом приобретении — очередном компакт-диске Джонни Кэша (Johnny Cash).

```
INSERT INTO Music (First_Name, Last_Name, Title, Format, Publisher)
VALUES ('Johnny', 'Cash', 'JOHNNY CASH AT FOLSOM PRISON AND SAN_
QUENTIN', 'CD', 'Columbia')
```

После выполнения этой команды в таблицу MUSIC будет добавлена запись со следующими значениями полей: FIRST\_NAME = 'Johnny', LAST\_NAME = 'Cash', TITLE = 'JOHNNY CASH AT FOLSOM PRISON AND SAN QUENTIN', FORMAT = 'CD', PUBLISHER = 'Columbia'. Обратите внимание на то, что не указано имя поля первичного ключа ID и соответствующего ему значения, поскольку это поле снабжено признаком автоматического заполнения AutoNumber.

## Добавление полной строки данных

Еще один распространенный вариант команды INSERT предполагает задание полного списка значений полей новой записи. Порядок перечисления значений должен соответствовать перечню полей таблицы, используемому при ее создании командой CREATE TABLE. Синтаксическая формула в этом случае становится более краткой ввиду отсутствия списка наименований полей.

```
INSERT INTO ИмяТаблицы  
VALUES (Значение1 [, Значение2, ...])
```

Порядок следования значений, указанных после служебного слова VALUES, и их типы должны соответствовать структуре таблицы. В качестве символического обозначения величины, заносимой в поле AutoNumber, используется литерал 0; Access самостоятельно позаботится о том, чтобы вычислить и сохранить в поле AutoNumber нужное значение. Например:

```
INSERT INTO Music  
VALUES (0, 'Jewel', 'Kilcher', 'PIECES OF YOU', 'CD', 'Atlantic')
```



Я всегда записываю текст SQL именно так, разбивая длинные команды на отдельные строки, соответствующие основным предложениям. Так, заголовок команды INSERT INTO удобно расположить в одной строке, предложение VALUES — в следующей и т.д.

Фраза INSERT INTO Music означает, что данные будут добавлены в таблицу MUSIC. После имени таблицы список полей не приводится — вместо него сразу следуют слово VALUES и полный список значений.

## Вызов команды Insert с параметрами

Версия SQL, которая реализуется в Access, допускает формат команды INSERT, содержащий параметры и предполагающий динамический интерактивный ввод их значений. Параметры размещаются в списке VALUES и обозначаются именами, заключенными в квадратные скобки. Возможность задания параметров поддерживается в обоих рассмотренных выше вариантах INSERT — со списком полей и без такового.

Единственной синтаксической особенностью параметрической версии INSERT является наличие аргументов, взятых в квадратные скобки. Приведенные ниже примеры демонстрируют использование параметров в обоих вариантах INSERT — с заданием полной строки данных (листинг 16.4) и указанием списка имен полей (листинг 16.5).

Листинг 16.4. Пример параметрической команды INSERT с заданием полной строки данных

```
INSERT INTO Music  
VALUES (0, [Имя], [Фамилия], [Название], [Формат], [Компания] )
```

Листинг 16.5. Пример параметрической команды INSERT с указанием списка имен полей

```
INSERT INTO Music (First_Name, Last_Name, Title, Format, Publisher)  
VALUES ([Имя], [Фамилия], [Название], [Формат], [Компания])
```

### Анализ

Команда INSERT листинга 16.4 содержит пять параметров. При ее выполнении система последовательно откроет пять диалоговых окон с за-

просами на ввод соответствующих значений (окно, предлагающее ввести значение параметра Имя, показано на рис. 16.1). Обратите внимание: число параметров с учетом литерального значения 0, отвечающего полю ID, равно числу столбцов в таблице MUSIC.

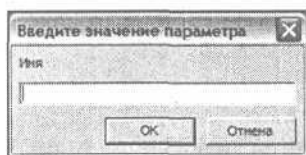


Рис. 16.1. Так выглядит диалоговое окно, в котором предлагается ввести значение параметра, указанного в команде INSERT

В команде INSERT листинга 16.5 приведены только те поля, которые подлежат интерактивному заполнению.

В пределах одной и той же команды INSERT могут задаваться как литеральные значения, так и параметры — в любых сочетаниях. Например, заведомо зная значения, которые следует сохранить в большинстве полей, вы можете указать параметры только для одного или двух — именно так чаще всего и происходит.

## Добавление записи с помощью SELECT

Весьма полезный способ применения команды INSERT связан с задачей копирования данных из одной таблицы в другую. В этом случае вместо предложения VALUES используется вложенный запрос на основе SELECT. Ниже приведена соответствующая синтаксическая формула.

```
INSERT INTO ИмяТаблицыПриемника  
    (ИмяПоля1 [, ИмяПоля2, ...])  
    SELECT ИмяТаблицыИсточника.ИмяПоля1 [,  
        ИмяТаблицыИсточника.ИмяПоля2, ...]  
FROM    ИмяТаблицыИсточника
```

Практически, все осталось на месте, кроме предложения VALUES, в котором указывается выражение SELECT. Напомним, что перечни полей SELECT и INSERT INTO должны совпадать по типам и количеству элементов.

Предположим, у вашего друга также есть коллекция музыкальных записей и собственная база данных с таблицей MUSIC, поля которой в точности совпадают с полями одноименной таблицы вашей базы данных. Как создать базу данных для объединенной коллекции? Используйте SELECT и INSERT INTO, как показано в листинге 16.6.

### Листинг 16.6. Пример команды INSERT INTO с вложенным запросом, выполняющим импорт полей из идентичной базы данных

```
INSERT INTO MUSIC (ID, ARTIST, TITLE, FORMAT, PUBLISHER)  
SELECT 0, ID, ARTIST, TITLE, FORMAT, PUBLISHER FROM MUSIC2
```

#### Анализ

Команда INSERT INTO не претерпела изменений до списка полей. Поскольку в качестве первичного ключа используется автоматически создаваемый счетчик, первый элемент команды SELECT равен 0. После литерального значения 0 каждое поле исходной таблицы (во вложенном запросе) помещается в список полей команды SELECT.

Команда INSERT (теперь вы согласитесь) демонстрирует немалую гибкость. Выражение SELECT само по себе "незаурядно", но в сочетании с INSERT способно творить чудеса.

# Обновление данных

Команда UPDATE применяется для одновременного изменения содержимого полей одной или нескольких записей. Основной вариант выражения позволяет обновить значения всех столбцов таблицы. Расширенная версия дает возможность сузить набор записей, подвергающихся воздействию, с помощью предложения WHERE. Ниже приведена общая синтаксическая формула команды UPDATE.

```
UPDATE ИмяТаблицы
SET ИмяПоля1 = Значение1 [, ИмяПоля2 = Значение2, ...]
[WHERE Предложение]
```

Служебные слова SQL в соответствии с общепринятым соглашением вводятся в верхнем регистре — так, как в данном случае набраны UPDATE, SET и WHERE. Предложение SET должно включать, по меньшей мере, один предикат вида ИмяПоля = Значение, и их количество не ограничено. Знакомое вам предложение WHERE, как всегда, необязательно. Листинг 16.7 демонстрирует несколько примеров употребления команды UPDATE для изменения данных в таблице MUSIC.

## Листинг 16.7. Примеры использования команды UPDATE

```
1: UPDATE Music SET Title = UCase( TITLE );
2: UPDATE Music SET First_Name = ICap( [First_Name] );
3: UPDATE Music SET Publisher = 'Columbia Records'
   WHERE Publisher = 'Columbia'
```

### Анализ

Каждая из строк листинга 16.7 содержит отдельную команду SQL. В строке 1 используется встроенная VBA функция UCase, переводящая содержимое поля TITLE всех записей таблицы MUSIC в верхний регистр. Строка 2 иллюстрирует применение пользовательской функции ICap, предназначенной для преобразования первого символа переданной строки (в данном случае — значения поля FIRST\_NAME) в верхний регистр. Текст функции ICap приведен в листинге 16.8. Строка 3 демонстрирует команду UPDATE, содержащую предложение WHERE и обновляющую только те записи таблицы MUSIC, в поле PUBLISHER которых хранится значение "Columbia".

## Листинг 16.8. Пример пользовательской функции, которую можно вызвать в команде SQL

```
1: Function ICap( ByVal FieldValue As String ) As String
2:   ICap = UCase(Left$(FieldValue, 1)) & LCase(Mid$(FieldValue, 2))
3: End Function
```

### Анализ

Функция ICap достаточно проста. Первая буква строки FieldValue, переданной в качестве аргумента, с помощью стандартной функции UCase переводится в верхний регистр, а остальные символы, посредством LCase, — в нижний. Как мы уже неоднократно говорили, функции и процедуры не должны быть пространными и сложными.



При более внимательном рассмотрении вы могли бы обратить внимание, что текст функции, приведенной в листинге 16.8, нуждается в дополнениях. Что случится, например, если переданная в виде параметра строка окажется пустой? Да-да, это наша с вами ошибка — выполнение функции будет

прервано. В данном случае необходимо принять меры, способные предотвратить аварийный сброс программы (подробнее об этом — в главе "18-й час. Обработка ошибок во время выполнения программы").

## Удаление данных

Оптимизация



Информацию, попавшую в базу данных, когда-либо наверняка придется удалить. Эта обязанность в языке SQL возложена на команду DELETE. Она довольно проста (впрочем, ничего удивительного — как говорится, "ломать — не строить"). Синтаксис выражения DELETE таков:

```
DELETE FROM ИмяТаблицы [WHERE Предложение]
```

Предполагая использовать команду DELETE в прикладной программе, будьте внимательны. Подумайте о возможности открытия окна предупреждающего сообщения (функция MsgBox придется как раз кстати), которое позволит пользователю подтвердить свои зловещие намерения или вовремя от них отказаться.



Изучением технологий программирования на SQL и управления базами данных нельзя заниматься между прочим, в часы беззаботного досуга. В понятие "хорошей программы" входит достаточно большое число различных составляющих. Характерный пример. Записи в таблице MUSIC отвечает несколько строк таблицы TRACKS — это понятно. Мы удалили эту запись. Что делать с соответствующей информацией таблицы TRACKS? (Другими словами, если нет сведений о музыкальном альбоме, зачем нужны эдакие "сиротские" строки о его композициях?)

При удалении записей одной таблицы зачастую следует позаботиться об изъятии логически взаимосвязанных записей из других таблиц. То же справедливо и в отношении операций изменения данных.

Во многих системах управления базами данных поддерживаются процедуры каскадного удаления и обновления информации. Имеется в виду, что система самостоятельно следит за логической целостностью данных. Для реализации возможностей каскадного выполнения операций обычно необходимы дополнительные действия со стороны администратора базы данных и авторов приложений. Если необходимая настройка не произведена, ответственность за целостность данных возлагается на код прикладной программы SQL или VBA.

Заметно упрощают работу также хорошие средства программирования и администрирования баз данных. Не бойтесь обращаться за помощью к литературе, информационным источникам в Internet и квалифицированным специалистам, способным подсказать решения в трудных ситуациях. Многие известные эксперты с пониманием откликнутся на ваши просьбы, если вы обратитесь к ним по электронной почте или с помощью средств ведения виртуальных дискуссий.

Нескольких простых примеров будет вполне достаточно. Введя команду DELETE FROM Music, вы заставите Access полностью очистить таблицу MUSIC — вряд ли это может входить в ваши планы. Зато команда DELETE с предложением WHERE применяется весьма часто.

# Вызов функций из команд SQL

В листинге 16.7, рассмотренном выше, были приведены примеры обращения и к стандартным функциям Access VBA (скажем, UCase), и к пользовательским функциям, определенным в прикладной программе (таким как ICap). Если создаваемый SQL-код проявляет тенденцию к усложнению, попробуйте воспользоваться существующими функциями или написать собственные, чтобы упростить выражение SQL. Язык SQL, несомненно, "прекрасен и могуч", но подчас вам придется сталкиваться с определенными ограничениями и сложностями при попытке решения некоторых задач (например, подобных рассмотренной ранее операции изменения регистра первого символа строки, хранящейся в поле таблицы).

SQL предлагает целый ряд встроенных функций — достаточно назвать COUNT, предназначенную для подсчета числа записей, или SUM — для суммирования числовых значений. Примеры использования некоторых функций ранее уже приводились в листингах этой главы (скажем, листинг 16.3 демонстрирует способы применения функций SUM и CDATE).

Правила обращения к функциям в SQL практически не отличаются от тех, которые приняты в VBA. Необходимо задать имя функции и передать ей верное количество значений параметров требуемых типов. Время, практика и готовность обращаться за помощью ко всем доступным информационным источникам — это все, что вам потребуется для овладения механизмами использования функций SQL.

## Хранимые процедуры

С появлением средств ActiveX Data Objects арсенал программиста, использующего Access, пополнился возможностями создания и применения хранимых процедур. *Хранимая процедура* в SQL равнозначна функции. Аппарат хранимых процедур в таких крупномасштабных системах управления базами данных, как, например, Oracle, существует с давних пор. (В Oracle имеется даже самостоятельный процедурный язык программирования под названием PL/SQL.)

Среди многочисленных достоинств, присущих хранимым процедурам, достаточно отметить два основных: хранимые процедуры выполняются на серверах баз данных и позволяют определять интерфейсы. Понятно, что при запуске процедур на сервере (который, как правило, намного более производитель по сравнению с персональными станциями) прикладная программа будет выполняться существенно быстрее. Этот вывод справедлив даже в том случае, если Access 2002 работает на том же компьютере, что и ваше приложение, поскольку хранимые процедуры выполняются непосредственно ядром Access. А именованные интерфейсы обеспечивают настолько же простые возможности обращения к хранимым процедурам, как и при использовании обычных функций. Вы передаете значения аргументов, затем определенные вами операции выполняются и возвращают требуемые результаты.

Процедуры хранятся в базе данных, в коллекции Procedures, входящей в состав объекта Catalog. Каждый элемент коллекции Procedures — это объект класса Procedure. Объект Procedure содержит атрибуты DateCreated, DateModified, Name и Command. Command — это объект данных, содержащий собственно код тела процедуры.

Хотя хранимые процедуры, на самом деле, физически располагаются в одной из системных таблиц базы данных, беспокоиться о подобных деталях сейчас вам вовсе не обязательно — доступ к процедурам вы получаете с помощью коллекции Procedures объекта Catalog. Прежде чем воспользоваться процедурой, ее необходимо создать и добавить в объект Catalog. Ниже приведен синтаксис определения хранимой процедуры.

```
PARAMETERS [Параметр1] Тип {, [Параметр2] Тип, ...};  
Текст SQL
```





После служебного слова PARAMETERS следует список пар Параметр Тип, определяющих интерфейс процедуры. Объем списка не ограничен. Квадратные скобки в данном случае служат составной частью конструкции, а не признаком необязательности синтаксического элемента, как прежде. (Здесь — во избежание недоразумений — необязательные элементы отмечены фигурными скобками.) Список параметров завершается символом точки с запятой, после которого набирается текст тела процедуры на языке SQL. Листинг 16.9 демонстрирует два примера хранимых процедур: первая возвращает набор записей таблицы MUSIC, соответствующих указанному значению поля PUBLISHER, а вторая отбирает записи той же таблицы, удовлетворяющие заданным значениям полей PUBLISHER и FORMAT.

#### Листинг 16.9. Примеры хранимых процедур

```

1: PARAMETERS [APublisher] TEXT;
2:   SELECT First_Name + ' ' + Last_Name As Artist, Title, _
      Format, Publisher
3:   FROM Music WHERE Publisher = [APublisher]
4:
5: PARAMETERS [APublisher] TEXT, [AFormat] TEXT;
6:   SELECT First_Name + ' ' + Last_Name As Artist, Title, _
      Format, Publisher
7:   FROM Music
8:   WHERE Publisher = [APublisher] AND Format = [AFormat]
```

#### Анализ

Строки 1-3 задают текст первой процедуры — далее будет показано, как в виде единой строки он присваивается переменной-свойству Command-Text объекта ADODB.Command. Процедура принимает текстовое значение, переданное в качестве параметра [APublisher], и отбирает все строки таблицы MUSIC, содержимое поля PUBLISHER которых равно значению параметра. Текст второй процедуры приведен в строках 5–8. Она выполняет схожую операцию, но работает уже с двумя аргументами ([APublisher] и [AFormat]), задающими искомые значения полей PUBLISHER и FORMAT.

## Добавление хранимой процедуры в каталог

Чтобы хранимая процедура стала доступной для использования, ее следует добавить в базу данных с помощью свойств и методов объектов ADODB.Command и ADOX.Catalog. Собственно, многие из конструкций и выражений VBA, необходимых для выполнения такой операции, вам уже знакомы. Листинг 16.10 содержит простой пример, иллюстрирующий все действия по созданию и сохранению первой из процедур листинга 16.9.

#### Листинг 16.10. Пример добавления хранимой процедуры в базу данных

```

1: Sub CreateStoredProcedure( )
2:   Dim Connection As ADODB.Connection
3:   Set Connection = CurrentProject.Connection
4:   Dim Command As New ADODB.Command
5:   Dim Catalog As New ADOX.Catalog
6:   Set Command.ActiveConnection = Connection
7:   Command.CommandText = "PARAMETERS [APublisher] TEXT;" & _
8:     "SELECT ARTIST, TITLE, FORMAT," & _
9:     "PUBLISHER FROM Music WHERE Publisher = [APublisher]"
10:   Set Catalog.ActiveConnection = Connection
```

```

11: Call Catalog.Procedures.AppendI "Artist By Publisher", Command )
12: Set Command = Nothing
13: Set Catalog = Nothing
14: Set Connection = Nothing
15: End Sub

```



Обратите внимание, что в команде SELECT фигурирует столбец ARTIST, а не два отдельных столбца — FIRST\_Name и LAST\_NAME.

### Анализ

Сначала необходимо добавить ссылку на Microsoft ADO Ext, 2.7 for DLL в диалоговом окне Tools⇒Options редактора Visual Basic. Чтобы решить задачу, следует открыть соединение с базой данных, в данном случае мы переместили фокус на CurrentProject.Connection — соединение с активной базой данных. В строке 4 создается объект Command, предназначенный для занесения в него текста хранимой процедуры. В качестве непосредственного исполнителя основных действий по сохранению процедуры в базе данных выступает объект класса Catalog — он создается в строке 5. В строках 7–10 последовательность символов, содержащая текст процедуры, присваивается атрибуту CommandText объекта Command. В строке 11 выполняется операция сохранения процедуры под именем "Artist By Publisher" в базе данных, а ниже осуществляются действия по очистке памяти.

Важно понимать, что все операции по созданию вспомогательных объектов приведены в тексте листинга 16.10 только в целях демонстрации — вам вовсе не обязательно заново создавать и открывать объекты классов Connection и Catalog в каждой процедуре. В принципе, сделать это достаточно один раз, в самом начале программы, а закрывать и удалять объекты целесообразно непосредственно перед ее завершением. Подробные сведения по таким вопросам изложены в главе "19-й час. Создание экранных форм".

## Выполнение хранимой процедуры

Теперь, когда хранимая процедура создана, вы можете обращаться к ней в любой момент по мере надобности. Чтобы выполнить процедуру, следует воспользоваться методом Execute класса Command. Если хранимая процедура возвращает некий набор данных (как в нашем случае), надлежит присвоить результат выполнения метода Execute переменной класса ADODB.Recordset. Если же хранимая процедура выполняет команды SQL (такие как INSERT, DELETE или UPDATE), не возвращающие значений или наборов данных, объект Recordset не нужен.

Листинг 16.11 демонстрирует действия, необходимые для выполнения хранимой процедуры.

### Листинг 16.11. Пример выполнения хранимой процедуры

```

1: Sub ExecuteProcedure( )
2:   Dim Connection As ADODB.Connection
3:   Set Connection = CurrentProject.Connection
4:
5:   Dim Catalog As New ADOX.Catalog
6:   Set Catalog.ActiveConnection = Connection
7:
8:   Dim Command As ADODB.Command
9:   Set Command = Catalog.Procedures("Artist By Publisher").Command

```

```

10:
11: Dim Publishers As ADODB.Recordset
12: Dim RecordsAffected As Long
13: Command.Parameters ("[APublisher]").Value = "Elektra"
14: Set Publishers = Command.Execute()
15:
16: Publishers.MoveFirst
17: Do While RecordSet.EOF = False
18:     Debug.Print Publishers( "Artist" )
19:     Publishers.MoveNext
20: Loop
21:
22: Publishers.Close
23: Set Publishers = Nothing
24: Set Command = Nothing
25: Set Catalog = Nothing
26: Set Connection = Nothing
27: End Sub

```

#### Анализ

Многие строки кода вам уже знакомы, поэтому остановимся лишь на главном. Вам следует выделить объекты соединения и каталога. Обратите внимание, что в строке 11 не создается новый объект Recordset — в нем нет необходимости, поскольку данный объект возвращается командой `Command.Execute` в строке 14. Поэтому дополнительные операции выделения памяти в строке 11 не нужны. В строке 13 определяется значение `Parameters` объекта `Command`, а в строке 14 вызывается метод `Execute`, возвращающий набор данных, которые являются результатом выполнения запроса хранимой процедуры.

Полученный в результате выполнения команды строки 14 набор данных, может изменяться, как и любой другой — мы неоднократно выполняли подобные операции ранее.

## Создание запросов в базах данных формата SQL Server

В проектах Access SQL можно использовать тремя способами: создавать соединения с SQL-базами данных, преобразовывать базы данных Access в формат SQL Server, устанавливая связь с SQL-таблицами, используя команду **Файл⇒Внешние данные⇒Связь** (**File⇒Get External Data⇒Link**). Выбор зависит от целей проекта и условий его разработки. Для повышения надежности, вероятно имеет смысл использовать формат SQL Server. А если для работы требуется всего лишь некоторые данные из другой базы данных, достаточно создать связь с таблицей или использовать объект `Connection`.

Если вы планируете устанавливать связь с таблицами SQL Server, рекомендуется создать объект источника данных. Либо же использовать OLE DB и ADO для непосредственного указания информации о провайдере (Provider). Эти операции были описаны в предыдущих главах.

После того как доступ к данным SQL Server обеспечен, работа с такой базой данных, по большому счету, не будет отличаться от работы с данными Access. Необходимо получить объект `Recordset`, определить запрос в зависимости от требуемой операции и открыть набор записей. С этого момента управление данными происходит так же, как и в Access — с помощью SQL или VBA-кода либо их комбинации.

Вам необходимо лишь помнить, что SQL Server — другой сервер баз данных, поэтому текст запросов может несколько отличаться в зависимости от используемой версии SQL Server. Разобраться с подобными тонкостями поможет хорошая книга по работе с применяемой версией сервера баз данных, эксперименты, и, конечно же, терпение.

## Резюме

Да, это был серьезный и продолжительный урок. Вам пришлось много потрудиться. Впрочем, в этой книге изложены лишь общие концепции программирования на SQL. Если вы нуждаетесь в более подробной информации, настоятельно рекомендуем обратиться к пособиям, специально посвященным SQL, — например, к книге Райана Стефенса (Ryan Stephens) *Sams Teach Yourself SQL in 24 Hours*. SQL — это мощный самостоятельный язык программирования, имеющий достаточно глубокие исторические традиции. Почти каждая система управления базами данных снабжена собственным диалектом SQL. Если, помимо Access, вы предполагаете работать с другими системами, будет лучше воспользоваться специальными руководствами и фирменной документацией.

В ходе этого занятия вы ознакомились с различными конструкциями одной из основных команд SQL — SELECT, включая возможности объединения таблиц (предложение JOIN) и запросов (UNION), а также построения подчиненных (вложенных) запросов. Мы кратко рассмотрели средства добавления (INSERT), обновления (UPDATE) и удаления (DELETE) данных. Вы научились пользоваться предложением WHERE (поддерживаемым почти во всех названных командах) для фильтрации данных.

Механизмы создания и использования хранимых процедур весьма серьезны и заслуживают особого внимания. Теперь, благодаря объектам ADO, доступ к ним получили и программисты, работающие с Access. Достаточно хорошо освоить базовые конструкции SQL — и хранимые процедуры станут вашим надежным помощником.

А теперь сделайте еще одно усилие и ознакомьтесь с оставшимися разделами этой главы.

## Вопросы и ответы

**Вопрос.** Действительно ли SQL, как и VBA, обладает собственным набором служебных слов и синтаксических правил?

**Ответ.** Да. Все языки (и языки программирования в том числе) обладают собственным словарем и грамматикой. Словарь перечисляет допустимые (значимые) слова, а грамматика диктует правила их употребления. Теперь вы знакомы уже с двумя языками программирования (речь не идет о более опытных читателях). Осталось лишь научиться с их помощью понятно выражать свои мысли.

**Вопрос.** Существуют ли какие-либо ограничения, связанные с объемом единого запроса на языке SQL?

**Ответ.** Нет. Вы можете создавать настолько сложные и широкие запросы, насколько это необходимо и целесообразно. Но SQL — это, как говорят, "монолитный" язык. Каждый запрос всегда содержит только одно выражение, пусть даже весьма обширное. Впрочем, и для SQL справедливо общее эмпирическое правило — чем проще, тем лучше.

**Вопрос.** Что вы можете сказать о стиле программирования на SQL?

**Ответ.** Правила "хорошего тона" в программировании едины. Не пытайтесь "втиснуть" решение всей задачи в рамки одного выражения SQL. Старайтесь представить ее в виде последовательности частных подзадач, решаемых с помощью простых

конструкций SQL и связанных командами на языке VBA. Этим вы существенно упростите обязанности по дальнейшему развитию и сопровождению программного проекта.

**Вопрос.** Где я могу найти информацию об определенной версии SQL?

**Ответ.** Посетите Web-сайт ANSI (<http://www.ansi.org>) либо обратитесь за литературой к поставщику той системы управления базами данных, которой пользуетесь.

## Задания

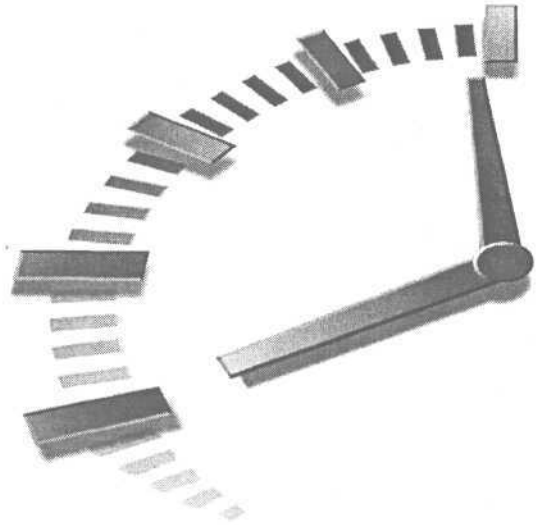
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Что такое подчиненный (вложенный) запрос?
2. С какой целью употребляется служебное слово ALL, используемое совместно с UNION?
3. Можно ли применять вложенный запрос в контексте команды DELETE? Если да, то зачем?
4. Ограничиваются ли возможности хранимых процедур использованием команды SELECT?

## Упражнения

1. Напишите предложение WHERE с предикатом IN, позволяющее отобрать все записи таблицы MUSIC, в поле PUBLISHER которых содержатся значения Elektra или Empire.
2. Исправьте команду UPDATE, приведенную в строке 2 листинга 16.6, таким образом, чтобы в верхний регистр были переведены первые символы имени и фамилии исполнителя.
3. Создайте хранимую процедуру, позволяющую удалить из таблицы MUSIC запись по заданному значению поля LAST\_NAME.



# Часть VI

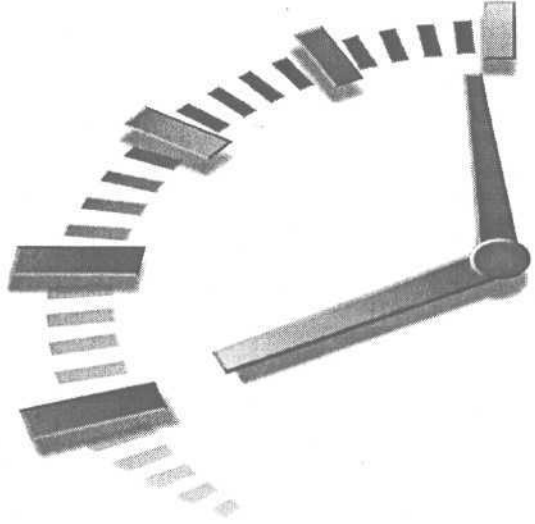
## Работа над ошибками

### Темы занятий

17-й час. Отладка кода

18-й час. Обработка ошибок во время выполнения программы





## 17-й час

### Отладка кода

Термином *bug*, широко распространенным в сообществе создателей программного обеспечения, обозначаются любые ошибки или недочеты, которые приводят к неверным результатам или аварийным сбоям. Производным прилагательным характеризуются профаммы, содержащие подобные ошибки, — и потому ненадежные и грозящие пользователям обернуться огорчениями и разочарованиями. На этом занятии речь пойдет о том, как уменьшить количество потенциальных ошибок в профамме, быстро их найти и легко устранить с помощью специальных приемов тестирования и отладки.

Следует отметить, что задача поиска ошибок в профамме нерешаема только в том случае, если сам **программный** код непоследователен, неряшлив и запутан. В этой главе излагаются советы о том, как написать код, не содержащий ошибок, — по крайней мере, таких ошибок, которые нельзя обнаружить.

Основные темы занятия.

- Пошаговое тестирование профаммного кода.
- Приемы эффективной отладки.
- Использование условных директив компилятора.

### Тестирование кода

В компьютерной науке широкую известность получила так называемая *проблема неразрешимости*. Если не вдаваться в математические детали, ее можно сформулировать так: в настоящий момент не существует строгого алгоритма, способного доказать правильность результатов выполнения кода профаммы при всех возможных условиях. Говоря проще, никто не может дать полных гарантий того, что конкретная профамма (независимо от степени ее сложности) не содержит никаких ошибок.

#### Новый термин

Термин *алгоритм* обычно используют для ссылки на процедуру или функцию. В некоторых случаях под алгоритмом понимают набор связанных функций.

Таким образом, факт безошибочности профаммного кода недоказуем. Отсюда следует, что всякий код может (и даже должен) содержать ошибки. (Самый красноречивый пример связан с нашумевшей проблемой неверной трактовки дат, следующих за 31 декабря 1999



года, старыми программами, многие из которых успешно работали в течение десятилетий.) Поскольку невозможно написать код, *полностью* свободный от ошибок, *относительно* "чистый" программный продукт — это тот идеал, к которому нужно стремиться. Но каков же должен быть опыт, чтобы не допускать ошибок, чтобы быстро их отыскивать в нужных случаях? Это как раз тот вопрос, на который вы найдете ответ в этой главе.

Предлагаем несколько советов, которые помогут определить тот объем работы, который следует выполнить, чтобы убедиться в надежности написанного кода. Итак, первый этап — это *промежуточное тестирование*.



В последнее время все большую популярность завоевывает JUnit, автоматизирующий тестирование для Java. Вероятно, сходные с JUnit тесты будут в ближайшее время созданы и для других языков программирования. Кроме того, все большее внимание уделяется реструктуризации задач. Это наиболее формальный подход к созданию программ, когда сходные части кода выделяются в отдельный блок — подпрограмму или функцию. Данному вопросу посвящена прекрасная книга Мартина Фаулера (Martin Fowler) *Refactoring: Improving the Design of Existing Code*, которая вышла в издательстве *Addison-Wesley*. Указанное издание предназначается, в основном, опытным программистам, тем не менее вы также можете почерпнуть немало идей по управлению кодом.

Почти все программисты, с которыми мне когда-либо приходилось встречаться, говорили, что они тестируют свой код. Но как и когда именно? Какое количество кода подвергается единовременному тестированию? Эти вопросы весьма важны.

Следует тестировать *маленькие* порции написанного кода. Если вы создали функцию, которая будет реализовывать некоторую часть общего замысла, тут же отдельно ее и протестируйте. Например, удачными объектами промежуточного тестирования служат функции сортировки, рассмотренные в главе "12-й час. Управление данными переменного объема". Каждая из них представляет собой законченное решение задачи сортировки. Поэтому имеет смысл написать небольшую программу (или, по меньшей мере, процедуру), предназначенную для проведения тестирования.



Для тестирования отдельных строк кода и пошагового тестирования отдельных подпрограмм используйте окно Immediate в редакторе Visual Basic. Это прекрасный способ первичного тестирования, но он не столь надежен и удобен, как создание в модуле отдельной подпрограммы тестирования.

Удобными объектами независимого промежуточного тестирования можно назвать функции, подпрограммы, классы и модули, входящие в состав проекта. Как правило, все они представляют собой некие самостоятельные кирпичики, из которых строится программа, поэтому вначале их *следует* протестировать отдельно и только потом — в совокупности.

## О важности промежуточного тестирования

Во время работы над программой ее автор не может не помнить о состоянии "ближайших" объектов кода и данных. Создавая функцию или процедуру, человек предвидит разнообразные ситуации, в которых она будет применяться.

Например, намереваясь изменить содержимое набора данных Recordset, программист, естественно, предполагает, что объект создан, открыт и указывает на требуемую запись. О подобных вопросах обычно задумываешься в ходе работы, и только в редких случаях — когда что-либо происходит не так — приходится вновь к ним возвращаться и восстанавливать в памяти забытые цепочки ассоциаций, предположений и т.д.

Вот почему тестировать фрагменты кода необходимо сразу же после их написания — ни в коем случае не позже. Иначе вы рискуете упустить из виду какие-нибудь ранее предусмотренные аспекты поведения программы.

## Что именно следует тестировать

Важно понимать, *что именно* должен проверять тестовый код. Естественно, первым делом стоит проверить основные **предположения**, но вообще код должен быть "прошупан" всеми мыслимыми случайными наборами входных данных. При создании тестовых процедур программисты обычно делают упор на проверке штатных — типичных и предусмотренных — ситуаций. Конечно, код должен работать с той информацией, характеристики которой вы заранее проанализировали. Ну, а что случится, если данные выйдут за пределы "разумных" рамок? Тестовая процедура должна проверить корректность ваших предположений и, возможно, выявить случаи, которые остались незамеченными. Процесс промежуточного тестирования заставляет вас воспринимать небольшой фрагмент кода как решение самостоятельной равноправной задачи.

Тестовый код имеет значение не только для вас, автора программы, но и для ваших коллег и последователей, которые в процессе изучения кода смогут понять, *что* вы учли, а что осталось за пределами внимания. Народная мудрость "Одна голова хорошо, а две лучше" еще раз доказывает свою истинность.

Следующий ряд вопросов, затрагиваемых в ходе промежуточного тестирования, связан с выявлением необоснованных **взаимозависимостей** различных фрагментов кода. Подобные факты служат признаком незрелости кода и свидетельствуют о непрофессионализме его автора. Например, если алгоритм быстрой сортировки (см. главу 12) реализован так, что он не работает в виде отдельной функции, следует пересмотреть такое решение полностью.



Еще одно неявное и необходимое следствие применения технологии промежуточного тестирования состоит в выработке навыков создания независимых фрагментов кода.

Следует стремиться полностью избавиться от взаимозависимостей функций и процедур. То же справедливо и в отношении классов; класс должен содержать все атрибуты, необходимые и достаточные для решения поставленной перед ним задачи.

## Как тестировать программный код для Access

Чтобы выполнить промежуточное тестирование программных решений в среде Access, целесообразно иметь вспомогательную базу данных. Пустая база — хороший отправной пункт, поскольку в этом случае принимаются во внимание ситуации, когда исходные данные отсутствуют. Ниже приведена общая последовательность действий, необходимых для тестирования кода Access.

1. Постройте новую вспомогательную базу данных той же структуры, что и основная.
2. Создайте новый модуль, если вы собираетесь тестировать процедуру (функцию), либо добавьте класс, подлежащий тестированию.
3. Напишите подпрограмму, которая вызывает тестируемый блок кода с передачей верного числа параметров требуемых типов.
4. Запустите программу на выполнение в режиме отладки.

Если ваш код прошел все стадии, перечисленные в приведенном ниже списке, значит, он в достаточно хорошей "форме".

- Код компилируется и запускается без ошибок — скажем, нет сообщений о том, что некий объект отсутствует.
- Задача решается верно.
- Код способен обработать самые серьезные ошибки времени выполнения.

Если программа, пройдя через все испытания, "выжила", вы должны решить, что делать дальше. При условии, когда экзаменуемый блок особенно важен для всего приложения в целом, будет нелишним обратиться к коллеге с просьбой взглянуть на код "со стороны", чтобы еще раз удостовериться, все ли ситуации предусмотрены и протестированы. В противном случае вам придется зафиксировать сделанное документально и перейти к работе над другими частями проекта. Выберите последний вариант действий также в том случае, если в вашем коллективе есть специальная группа, занимающаяся тестированием и проверкой качества выпускаемого программного обеспечения.

Если код "споткнулся" на каком-либо из трех названных этапов, вам пригодится информация, изложенная в следующих разделах.

## **Как избавиться от необоснованных взаимозависимостей блоков кода**

Причины неуспешной компиляции из-за отсутствия глобальной переменной, функции или неудовлетворительной реализации зависимости от какого-либо другого внешнего объекта кроются в особенностях стиля вашей работы.

Если код ссылается на внешний объект или переменную, которые в данный момент недоступны, изучите возможность добавления этого объекта в состав интерфейса тестируемой функции или класса — другими словами, передавайте его имя в виде параметра процедуры или функции либо включите в набор свойств класса. (Подробнее о классах и их атрибутах см. главу "21-й час. Основы программирования классов".) Общее правило таково: не прекращайте работу над кодом, содержащим необоснованные внешние ссылки и зависимости.

Если код ссылается на глобальную переменную, попробуйте включить ее в состав набора аргументов процедуры/функции. Тщательно проанализируйте, какие глобальные переменные имеет смысл преобразовать в локальные. Глобальные объекты — потенциальный источник ошибок и недоразумений.

Если тестируемый блок кода ссылается на процедуру или функцию, подумайте, действительно ли она необходима.

## **Что делать, если код работает неверно**

Программа не решает поставленную задачу в нескольких случаях. Если виной всему неверные исходные данные, попробуйте сформулировать условия, которым должны удовлетворять данные, и дополните код условными конструкциями, которые смогут гарантировать корректность обрабатываемой информации. Более подробные сведения по этим вопросам приведены ниже, в разделе "Верификация исходных условий".

Если код неверен сам по себе — что ж, разбирайтесь и исправляйте ошибку. Можно сказать одно — найти и устранить дефект гораздо легче в небольшом фрагменте кода, во время промежуточного тестирования, нежели в составе целой программы.

## **Как быть в случае возникновения непредусмотренной ситуации**

ЕСЛИ во время выполнения кода возникает непредвиденная ошибка, первым делом необходимо выявить ее источник. Причиной ошибки служит зависимость тестируемого блока от других объектов программы. В этом случае выполните рекомендации,

рассмотренные выше. Часто ошибку можно устранить с помощью программного обработчика. Иногда достаточно просто зафиксировать ее, а затем попытаться "обойти".

Возможность установить закономерность возникновения ошибки — это уже немало. В программах, предполагающих диалог с пользователем, нередко бывает достаточно "вывесить" на экран окно с подробным объяснением случившегося. Представьте себе программу, которая должна подключаться к сетевым ресурсам. Если соединения нет, программа работать не будет. Сообщите об этом пользователю и предложите способы выхода из создавшейся ситуации.

В тестовых процедурах могут применяться специальные конструкции и приемы, позволяющие выявить потенциальные ошибки. Об этом будет рассказано далее.

## Использование ловушек

Стратегии, о которых пойдет речь в этом и двух следующих разделах, разработаны давно. Заслуга принадлежит Дэйву Тилену (Dave Thielen). Достаточно сказать, что этот человек отвечал за разработку MS DOS 5. Если в то время вы уже занимались компьютерами, то, вероятно, помните, какой ужасной была предыдущая, четвертая, версия операционной системы DOS. Она просто пестрила ошибками. Всемогущий Билл Гейтс, возможно, сказал тогда: "Друзья мои! Либо в пятой версии вы исправите недочеты, либо всех уволю без выходного пособия".

Так вот, DOS 5 завоевала успех и признание миллионов пользователей. Позже Дэйв Тилен написал книгу *No Bugs!: Delivering Error-Free Code in C and C++* (издательство Addison-Wesley, 1992). Это пособие — одно из моих любимых, и тому есть две причины. Первая — небольшой объем, всего около 180 страниц (сравните со многими современными фолиантами, для которых 1000 страниц становится порочной нормой). А вторая — пожалуй, главная — заключается в том, что в книге излагаются полезные советы и ценные рекомендации. С момента выхода указанного пособия в свет и по сей день я с успехом применяю в процессе программирования на любом языке описанные в ней подходы. Поставщики инструментальных средств программирования сочли необходимым включить некоторые предложенные Дэйвом Тиленом технологии в состав своих продуктов. Скажем, VBA содержит штатные средства верификации условий.

Под *верификатором условий* понимается дополнительный код, который осуществляет контроль выполнения заранее заданных условий и приостанавливает работу программы, если последние не удовлетворяются. В процессе *трассировки* программа с помощью вспомогательных конструкций создает отчет — во внешнем файле или в окне (в таком, например, как Watches) — о том, что она делает. Скажем, Microsoft Visual C++ предлагает инструменты верификации условий и трассировки, а язык Object Pascal среды Delphi имеет собственные возможности верификации. Языки VB и VBA также снабжены средствами верификации условий и аналогом трассировки — функциями, реализованными в составе класса Debug. В текущем и двух следующих разделах эти мощные и эффективные технологии рассмотрены более подробно.

## Что такое ловушка

Слово *ловушка* (trap) достаточно понятно: вы попадаете в нее, и она захлопывается. В программировании ловушка представляет собой строку кода, позволяющую удостовериться в том, что некий блок выполнен. Ее цель — уведомить, что в процессе своего выполнения программа "прошла" через определенное место кода.

Использование ловушек — удобное средство, гарантирующее, что процессом тестирования будут охвачены все ветви сложного кода. Предположим, что при тестировании управляющей структуры `If ... Then ... Else ... End If` блок

кода, следующий за Else, никогда не выполнялся. Вправе ли вы надеяться, что в один не очень прекрасный момент он вас не подведет? Ответ однозначен — нет. С другой стороны, может оказаться и так, что раз этот код никогда не работает, он вовсе не нужен.

## Как применять ловушки

В VBA технология применения ловушек реализуется очень легко. Изучите текст листинга 17.1.

### Листинг 17.1. Пример использования ловушек

```
1 Sub Trap(ByVal FileName As String, ByVal TrapNumber As Long)
2     Debug.Print "Ловушка: " & FileName & "(" & TrapNumber & ")"
3     Stop
4 End Sub
5
6 Sub SomeCodeToTrap( )
7     Dim Condition As Boolean
8     If (Condition = True) Then
9         Call Trap( "Module1", 1 )
10:    Else
11:        Call Trap( "Module1", 2 )
12:    End If
13:End Sub
```

#### Анализ

Строки 1–4 содержат текст процедуры-ловушки, а строки 6–13 заняты процедурой, подвергаемой тестированию. Процедура Trap достаточно проста. Ее можно создать один раз, а затем при необходимости импортировать в нужные модули. В нашем случае Trap вызывается с двумя аргументами — именем модуля (FileName) и номером строки-ловушки (TrapNumber), который должен быть уникальным в пределах текущего модуля.

Когда ловушка "захлопывается", вызывается процедура Trap, которая выводит информацию об имени модуля и номере блока кода в окно отладки. Во время выполнения программы следите за содержимым окна Immediate и помечайте все пройденные строки-ловушки символом комментария. Не советуем полностью удалять эти строки из текста программы — их наличие в виде комментариев говорит о том, что соответствующие блоки кода были протестированы.

Строки 6–13 представляют собой пример кода, который должен пройти тестирование с помощью ловушек. Применяйте строки-ловушки в особо ответственных случаях — скажем, в важных условных конструкциях. Конечно, весь код проверять таким образом неразумно, но и полностью отказываться от подобной возможности тестирования, разумеется, не следует. Если ранее проверенный код подвергся изменениям, желательно убрать символ комментария и повторить тест вновь. Если при просмотре текста программы вы нашли строку-ловушку, не помеченную символом комментария, сразу же протестируйте эту часть кода. Если не удастся подобрать соответствующие условия теста, подумайте — может быть, такой блок кода вовсе и не нужен. Его удаление обеспечит полную гарантию избавления от потенциальных неприятностей (кто из классиков мрачно пошутил: "Наилучшее средство от головной боли — гильотина"?).

# Трассировка кода

Значение глагола *trace* (следить), от которого происходит название необходимого в работе режима, рассмотренного в этом разделе, также достаточно понятно. (Поверьте, сидеть в засаде или подглядывать в замочную скважину вам не придется.) Назначение функций трассировки состоит в документировании порядка выполнения кода и различных характеристик его состояния. Одна из типичных задач трассировки связана с подсчетом количества обращений к определенным фрагментам кода. Подобная информация может оказаться полезной в процессе выявления блоков кода, нуждающихся в оптимизации.

Приемы реализации процедур трассировки схожи с методами использования ловушек, рассмотренными в предыдущем разделе. Процедуре трассировки передается несколько аргументов — имя тестируемого модуля (скажем, `FileName`), номер блока кода, уникальный в пределах модуля (`TraceNumber`), и подробное сообщение, характеризующее состояние параметров программы (`TraceMessage`). По значению аргумента `TraceNumber` блок кода легко отыскать в тексте модуля с помощью команды меню `Edit⇒Find` окна редактора VBA. В качестве параметра `TraceMessage` можно передавать имя выполняемой в данный момент процедуры или функции, значение переменной или любой другой элемент данных, за состоянием которого необходимо проследить. Одним словом, трассировка помогает понять, что именно и в какой момент делает программа.



Только на протяжении последних пяти лет в составе инструментальных сред программирования появились мощные интегрированные отладчики, позволяющие указывать объекты наблюдения (*watches*), следить за содержимым стека вызовов (*call stack*) и задавать точки останова (*breakpoints*). Средства наблюдения и просмотра стека вызовов можно, по существу, приравнять к функциям трассировки, а механизмы обработки точек останова близки по смыслу процедурам использования ловушек и верификации условий.

Впрочем, одно вовсе не исключает другого — рассматриваемые технологии тестирования остаются весьма полезными в условиях, когда приложение отлаживается за пределами среды программирования. Средства задания точек останова (или контрольных точек) и другие названные инструменты, конечно, весьма удобны, но точка останова, скажем, — это объект окна среды, а не приложения как такового.

Нужно уметь и рационально использовать и штатные средства, предлагаемые интегрированным отладчиком, и те приемы, которые мы сейчас рассматриваем. В борьбе с ошибками все средства хороши — лишь бы они помогли добиться полного успеха.

Одним из преимуществ процедуры трассировки служит то, что она способна сохранять информацию в виде файлового отчета, а не только отображать ее в окне Immediate редактора VBA. Сопоставление полученного отчета с тем, что, по вашему мнению, должна была выполнить программа, — весьма поучительный способ достижения хороших результатов. Листинг 17.2 демонстрирует пример реализации процедуры трассировки.

## Листинг 17.2. Пример процедуры трассировки

```
1: Sub Trace(ByVal FileName As String, _  
2:   ByVal TraceMessage As String, ByVal TraceNumber As Long)  
3:   Dim Output As String  
4:   Output = "Трассировка: " & FileName & "(" & _  
5:     TraceNumber & ") в " & Now & vbCrLf & _
```

```

6:      TraceMessage & vbCrLf
7:      Debug.Print Output
8: End Sub

```

## Анализ

[ Листинг 17.2 содержит текст процедуры трассировки. Добавьте ее в тот же модуль, который собираетесь проверять, и вы получите удобное средство тестирования. Если необходимо применить готовую процедуру трассировки в других базах данных, воспользуйтесь командами экспорта (File⇒Export File) и импорта (File⇒Import File) модуля.

В строке 3 объявляется переменная для хранения строки данных, форматируемой ниже (в строках 4–5) в соответствии с шаблоном `ИмяФайла(НомерБлока)` в `ДатаВремя`. Предопределенная константа `vbCrLf` содержит управляющие символы возврата каретки и перевода строки, позволяющие разбить одну логическую строку данных на несколько физических, которые с помощью команды `Debug.Print` отображаются затем в окне `Immediate`.

Другая версия процедуры `Trace` могла бы осуществлять вывод информации в журнальный текстовый файл. (Я, например, в свое время использовал обе.) Чтобы реализовать подобный код, необходимо открыть (`Open`) текстовый файл в режиме `Append`, использовать команду `Write` для записи в него форматированной строки, а затем закрыть (`Close`) файл.

Собираясь "поохотиться за ошибками", держите эти орудия под рукой — во время отладки своих драгоценных программ вы сэкономите массу времени и нервов.

## Верификация исходных условий

Без начальных условий и предположений программирование вообще невозможно. Используя конструкции верификаторов, вы требуете выполнения оговоренного условия — иначе программа работать не будет. Верификаторы настолько важны и полезны, что соответствующие методы были включены в состав классов большинства современных систем программирования.

Класс `Debug` в `VBA` содержит метод `Assert`. Функция `Assert` предполагает задание единственного параметра типа `Boolean` (проверяемого условия) и приостанавливает выполнение программы, если условие не выполнено. Собственно, это как раз то, что нужно. `Assert` играет роль "блюстителя порядка". Если для корректной работы программы необходимо выполнить определенное условие, `Assert` аккуратно за этим проследит.

Существует, вероятно, столько же примеров использования верификаторов, сколько и образцов кода, нуждающегося в гарантиях выполнения определенных условий. Если вы намереваетесь обновить содержимое файла, то файл (это совершенно очевидно) должен существовать. Конечно, можно было бы написать, скажем, такую строку кода:

```
If (Len( Dir( FileName ) ) > 0) Then
```

Во время выполнения программы подобная конструкция, безусловно, в состоянии проверить факт существования файла, но она не сможет довести до сведения программиста (и пользователя) информацию о том, что файла не существует. Но если код несколько расширить (как показано ниже), он от этого только выиграет, увеличив свою функциональность:

```
Debug.Assert Len( Dir( FileName ) ) > 0
If (Len( Dir( FileName ) ) > 0) Then
```

Если условие верификации не удовлетворяется, программа приостанавливает выполнение на текущей строке кода. Причины ошибки могут быть различными — файл оказался удаленным другим процессом, аргументу `FileName` передано неверное значение и т.п. В любом случае верификатор даст знать, что оговоренное вами обязательное условие не выполнено.



Не используйте конструкции, подобные `If (Len(Dir(FileName)) > 0) Then`, непосредственно. Заключите эту строку в тело функции и дайте последней четкое название, скажем, `Function FileExists (ByVal FileName As String) As Boolean`. В этом случае код приобретет ясность и будет удобным для чтения.

Во время отладки вместо задания точки останова в строке `If ... Then` достаточно разместить выше конструкцию `Assert`. Позаботиться о последствиях понадобится только тогда, когда проверяемое условие вдруг не выполнится. Ясно, что процесс тестирования будет протекать значительно быстрее, если программа приостанавливается лишь в случае возникновения какой-либо непредвиденной ситуации.

Помните, что, пользуясь инструментом задания точек останова, вы будете ограничены рамками редактора VBA. А верификаторы — это неотъемлемая и самодостаточная часть кода. Впрочем, ищите "золотую" середину.

## Использование соглашений

ЕСЛИ над проектом трудится целая команда специалистов, зачастую применяется стиль программирования, основанный на использовании соглашений. *Соглашение* подразумевает, что код будет корректно работать только в том случае, если выполняются определенные условия. Если, скажем, вы написали функцию, осуществляющую запись данных в файл, вы вправе оговорить, что программист, который будет пользоваться этой функцией, обязан сам позаботиться о существовании файла, поскольку соответствующая проверка вами не предусмотрена.

Теперь вы можете сосредоточить внимание непосредственно на проблеме (пополнения файла данными) и значительно уменьшить объем кода за счет исключения дополнительных проверок. Да, но как обеспечить выполнение соглашения? Можно, конечно, написать комментарий и надеяться, что он будет принят к сведению. Но лучше, разумеется, воспользоваться конструкцией верификации. Если условие верификатора не выполнится, программист, допустивший оплошность и не проверивший факт существования файла, обязательно обратит внимание на верификатор и вспомнит о соглашении.

Соглашения — это неплохо, но лучше, если они будут подкреплены верификаторами (как говорится, "доверяй, но проверяй"). Собственно, сказанное справедливо и в том случае, если вы являетесь единоличным автором проекта, поскольку напоминает об условиях, необходимых для работы программы. Кроме того, использование соглашений незаменимо для групп программистов, не работающих в непосредственной близости, поскольку позволяет делегировать ответственность и избегать ненужных затрат.

## Повторное использование кода отладки

Вспомогательный код (как и любой другой код) нуждается в отладке. Однако, приступая к новому проекту или модулю, вы не должны заново создавать и отлаживать, например *Trace*, *Trap* и *Assert*. Напишите подобные функции один раз, отладьте их, экспортируйте модуль в файл (скажем, *Debug.bas*), а затем импортируйте по мере необходимости в каждый новый проект.

Теперь, имея привычный набор инструментов, вы сможете работать гораздо более эффективно.



# Использование директив компилятора

Ни в коем случае не следует избавляться от отладочного кода после завершения процедур тестирования. Важно помнить, что программный код, однажды написанный и распространенный, начинает жить своей собственной жизнью. Не исключено, что со временем вам вновь придется к нему обратиться, чтобы внести требуемые исправления и изменения.

Если вы написали множество замечательных отладочных процедур, а затем, накануне передачи приложения в эксплуатацию, решили их удалить (как казалось, за ненадобностью), в дальнейшем, при необходимости внесения изменений, придется все это восстанавливать. Как же быть? На помощь приходят директивы компилятора.

Директивам компилятора в тексте программы предшествует символ фунта (#). *Директивы компилятора* — это инструкции, указывающие, что делать с тем или иным блоком кода во время его компиляции. Наиболее частое применение в них находят константы и условные управляющие структуры. Обычно с помощью служебного слова `#Const` задается именованная константа, а затем строится управляющая структура вида `#If ... #End If`, охватывающая блок кода и проверяющая с помощью значения константы, следует включать блок в версию исполняемого файла программы или нет.



Конструкция `If ... End If` — это вовсе не то же самое, что `#If ... #End If`. Первая представляет собой полноправную часть кода, а вторая действует лишь во время его компиляции.

Вместо безвозвратного удаления отладочных процедур из окончательной версии исходного текста кода, целесообразно заключить их в пределы условных директив компиляции, поведение которых и результат компиляции будут зависеть от значения единственной константы. Листинг 17.3 демонстрирует приемы использования константы и условной директивы компилятора, указывающей, включать в исполняемый файл отладочную конструкцию или нет.

## Листинг 17.3. Пример использования условной директивы компилятора

```
1: Sub Test ( )
2:   #Const DebuggingOn = True
3:   #If DebuggingOn Then
4:     Debug.Assert SomeBooleanValue
5:   iEnd If
6: End Sub
```

### Анализ

Листинг 17.3 содержит пример верного с формальной точки зрения использования директив компилятора. Поскольку значение константы `DebuggingOn` равно `True`, код строки 4 будет включен в откомпилированную версию программы. Если изменить значение константы `DebuggingOn` на `False`, при следующей компиляции строка 4 не будет включаться в исполняемый код. В любом случае (независимо от значения константы) строк 2, 3 и 5 в откомпилированном коде не будет.

Текст процедуры `Test` технически правильный, но он достаточно скромный в практическом отношении. Дело в следующем. Если вы будете действовать таким образом и далее, придется обрамлять конструкциями `#If ... #End If` буквально каждый вызов метода `Assert` и других отладочных функций. Поэтому более целесообразно заключить выражение верификатора в отдельную процедуру и обращаться к ней при необходимости. Листинг 17.4 показывает более удачный пример использования директив компилятора.

## Листинг 17.4. Пример процедуры верификации с условной директивой компилятора

```
1: #Const DebuggingOn = True 'или False
2: Sub Assert(ByVal Test As Boolean,
3:     ByVal FileName As String, ByVal AssertNumber As Long)
4:     ttf DebuggingOn Then
5:         Debug.Assert Test
6:     #End If
7: End Sub
```

### Анализ

Недостаток — Впрочем, единственный — этого метода заключается в том, что для определения места кода, вызвавшего приостановку программы верификатором Assert, придется анализировать стек вызовов процедур либо значения параметров FileName и AssertNumber. Но это все равно гораздо проще, нежели окружать каждое отладочное выражение парой условных директив.



Процедуры верификации хорошо развиты в языках С и С++, поскольку конструкция Assert реализована в них в виде макроса. Ее код автоматически включается компилятором в те места текста программы, в которых выполняется обращение к Assert. VBA такой возможностью не обладает. (В данном случае речь не идет об обычных макросах Access — говорится о макросах как о механизме развития директив компилятора.)

Чтобы увидеть содержимое стека вызовов (напомним, что программа в случае неудачной верификации приостанавливает свое выполнение в строке, содержащей Debug.Assert), обратитесь к команде меню View⇒Call Stack окна редактора VBA. Второй сверху элемент списка, отображаемого в диалоговом окне Call Stack, указывает на строку кода, вызвавшую приостановку в момент верификации. Рис. 17.1 соответствует ситуации, когда процедура Assert вызывается из некоторой процедуры SomeCodeToTest при условии, что в качестве аргумента Test передавалось значение False.

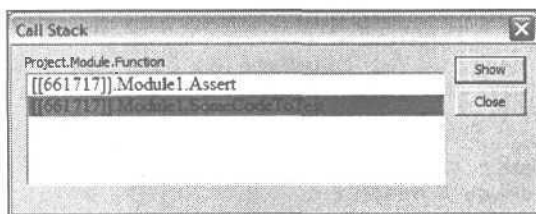


Рис. 17.1. Диалоговое окно Call Stack позволяет увидеть содержимое стека вызовов и быстро перейти к строке кода, содержащей обращение к процедуре или функции

Главная особенность структуры данных, называемой *стеком*, состоит в том, что данные поступают в нее в одном порядке, а извлекаются в обратном, т.е. последний "вошедший" элемент первым же и "выйдет". Информация о последовательности обращений к функциям и процедурам сохраняется в так называемом *стеке вызовов*. Всякий раз, когда одна функция или подпрограмма вызывает другую, стек пополняется, а при завершении последней вызванной функции (процедуры) элемент, распо-

женный на вершине стека, удаляется из него. Если щелкнуть на элементе списка, иллюстрирующего стек вызовов, а затем на кнопке Show диалогового окна Call Stack (см. рис. 17.1), текстовый курсор в окне редактора перейдет на строку, содержащую вызов. Так, например, щелчок на элементе `Module1.SomeCodeToTest` в представленном на рисунке примере приведет к переходу курсора на строку, в которой содержится обращение к процедуре `Assert`.

Теперь, умея обращаться с условными директивами компиляции, вы можете не беспокоиться о принудительном удалении отладочного кода. Если такой код написан, оставьте его в покое. Возложите обязанности по его сохранению или изъятию на компилятор. Для этого достаточно изменить значение единственной константы и перекompиллировать приложение.

Не нужно "обрамлять" условными директивами компиляции каждый вызов отладочной процедуры или функции — лучше разместить их внутри самой процедуры (функции). Другое немаловажное преимущество подобного подхода заключается в том, что отладочный код остается на месте — он всегда доступен для повторного использования и выполняет информативную, документирующую функцию. Впоследствии вы сами легко сможете определить, что именно тестировалось и каким образом. Листинг 17.5 представляет текст модуля `Debug.bas`, содержащего полный набор отладочных процедур, которые мы рассмотрели в ходе этого занятия.

#### Листинг 17.5. `Debug.bas` — набор отладочных процедур

```
1: Option Compare Database
2: Option Explicit
3: iConst DebuggingOn = True
4: Private Sub DoTrap(ByVal FileName As String, _
    ByVal TrapNumber As Long)
5:     Debug.Print "Ловушка: " & FileName & "(" & TrapNumber & ")"
6:     Stop
7: End Sub
8: Sub Trap(ByVal FileName As String, ByVal TrapNumber As Long)
9     #If DebuggingOn Then
10:         Call DoTrap( FileName, TrapNumber )
11:     #End If
12: End Sub
13: Private Sub DoTrace( ByVal FileName As String, _
14:     ByVal TraceNumber As Long, ByVal TraceMessage As Variant)
15:     Dim Output As String
16:     Output = "Трассировка: " & FileName & "(" & _
17:         TraceNumber & ") в " & Now & vbCrLf & _
18:         TraceMessage & vbCrLf
19:     Debug.Print Output
20: End Sub
21: Sub Trace( ByVal FileName As String,
22:     ByVal TraceMessage As String, ByVal TraceNumber As Long)
23:     #If DebuggingOn Then
24:         Call DoTrace(FileName, TraceMessage, TraceNumber)
25:     #End If
26: End Sub
27: Private Sub DoAssert(ByVal Test As Boolean, _
28:     ByVal FileName As String, ByVal AssertNumber As Long)
29:     Debug.Assert Test
30: End Sub
```

```

29:Sub Assert(ByVal Test As Boolean,
              ByVal FileName As String, ByVal AssertNumber As Long)
30:  If DebuggingOn Then
31:    Call DoAssert(Test, FileName, AssertNumber)
32:  #End If
33:End Sub

```



Как уже неоднократно отмечалось, предпочтительно писать короткие процедуры и функции. Просмотрев листинг 17.5, вы не увидите процедур длиннее трех-пяти строк. Функциональная часть (скажем, DoAssert) отделена от "условно-директивной" (Assert).

Встречаются программисты, которые буквально в штыки принимают подобный стиль работы. Я же мотивирую свои действия удобством и универсальностью подхода "разделяй-и-властвуй" • - мы уже говорили о нем: дели проблему на мелкие обозримые части, и ее решение упростится.

#### Анализ

В листинге 17.5 функциональная часть кода отладочных процедур отделена от условных конструкций компилятора. В этом случае можно воспользоваться процедурами тестирования даже тогда, когда отладочный режим отключен (т.е. константа DebuggingOn равна False). Эти процедуры, в названия которых введен префикс *Do*, объявлены посредством служебного слова-квалификатора Private, предотвращающего возможность их случайного использования за пределами "родного" модуля. (Подробнее о применении подобных средств см. главу "21-й час. Основы программирования классов".)

## Отладочный код — только для чтения

Отладочный код — это код только для чтения. Другими словами, наличие отладочных процедур в тексте программы не должно сказываться на результатах ее работы. Также недопустимо, чтобы Поведение программы менялось в зависимости от того, "срабатывают" условные директивы компилятора или нет. Программа должна работать одинаково в обоих случаях — с отладочным кодом и без такового.

## Различные типы кода обработки ошибок

Конструкции, обрабатывающие ошибки периода разработки программы, дают вам, автору, возможность выявить спорные ситуации на самой ранней стадии работы над проектом. Код обработки ошибок во время применения приложения пользователями должен гарантировать, что программа будет работать с различными наборами исходных данных и в любых ситуациях.

Чаше всего целесообразно наряду с отладочными выражениями использовать и конструкции обработки ошибок, служащие неотъемлемой частью кода приложения. Например, если файл должен существовать (это условие гарантирует возможность его дальнейшего открытия), можно применить верификатор, позволяющий протестировать код во время разработки, а также включить в текст постоянную конструкцию, проверяющую факт существования файла:

```

Call Assert( FileExists( FileName ), "ModuleName", 1 )
If (FileExists( FileName )Then
    ' Какой-то код

```

Else  
' Файла нет, но надо что-то сделать  
End If

Функция Assert поможет при тестировании программы на этапе ее разработки, а условная конструкция If ... End If позволит приложению сохранить работоспособность даже в том случае, если отладочный код во время компиляции был исключен.

## Резюме

Программирование — нелегкий хлеб. Даже если ваш опыт относительно невелик (а может быть, именно поэтому), вам наверняка приходилось тратить целые часы на решение каких-то, на первый взгляд, незначительных задач. Залог профессионального взросления — настойчивость в освоении передовых технологий, последовательность в их применении и практика.

На этом занятии вы изучили некоторые способы, которые помогут избавиться от ошибок в программах. Применяйте их правильно — и вы станете профессионалом. Правила таковы: старайтесь по возможности упрощать код, создавайте отладочный код одновременно с основным, не удаляйте отладочные конструкции из текста — пользуйтесь условными директивами компилятора для их включения или отключения.

Стратегии тестирования и отладки, рассмотренные нами, прошли проверку временем и доказали свою эффективность. Применяя их в сочетании с мощными средствами интегрированной среды программирования Visual Basic, вы добьетесь успеха, не прикладывая чрезмерных усилий. Вы сразу заметите собственный прогресс — ошибок времени компиляции станет меньше, задача тестирования облегчится, и в результате ваши пользователи получат более совершенное программное обеспечение.

Существует притча о том, как один вызывающе праздный турист, бродя по Нью-Йорку, обратился к замученному жизнью полисмену с вопросом: "Как добраться до Карнеги-холла?". Тот, почесав затылок, процедил сквозь зубы: "Тренируйся, тренируйся!". Практикуйтесь и вы в использовании рассмотренных приемов — и программирование станет не только работой, но и источником радостей. А теперь, чтобы все-таки дотащить до своего Карнеги-холла, ознакомьтесь с разделами "Вопросы и ответы" и "Задания".

## Вопросы и ответы

**Вопрос.** Имеются ли в составе библиотек классов VBA встроенные отладочные функции?

**Ответ.** Да. Класс Debug содержит два метода, предназначенных для подобных целей, — Print и Assert. Print оказывается полезным при реализации функции трассировки, а Assert предназначен для верификации условий. Я рекомендую использовать оба метода в виде отдельных процедур. В этом случае вы сможете легко настраивать интерфейс и при необходимости удалять отладочный код из исполняемой версии приложения с помощью условных директив компилятора.

**Вопрос.** Почему нельзя передать отладочной процедуре параметр, содержащий номер строки исходного текста программы?

**Ответ.** Да это было бы просто здорово! К сожалению, в VBA не поддерживается механизм внутреннего отслеживания номеров строк.

**Вопрос.** Можно ли построить дополнительные отладочные процедуры для облегчения тестирования?

**Ответ.** Конечно. Я, например, уже несколько лет пользуюсь двумя версиями функции Trace: одна выводит информацию в диалоговое окно, а другая — в тексто-

вый журнальный файл. Последняя особенно необходима, поскольку дает возможность позже внимательно проанализировать ход выполнения программы.

**Вопрос.** Что вы можете сказать по поводу сохранения отчета о результатах отладки в базе данных?

**Ответ.** Здравая мысль. Конечно, скорость работы программы при этом несколько снизится, но результат оправдывает подобные потери. Собственно говоря, аналогичный механизм реализован Microsoft в операционной системе Windows 2000. Сохранив отчет о результатах тестирования в базе данных, затем можно, например, легко преобразовать системные сообщения об ошибках в предложения общедоступного языка.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Как называется класс, содержащий основные программные средства отладки?
2. Какие методы, полезные во время тестирования, предлагаются классом, упомянутым в предыдущем вопросе?
3. Какие цели преследует трассировка?
4. Следует ли удалять отладочный код из окончательной версии приложения с помощью редактора или условных директив компилятора?

## Упражнения

1. Напишите функцию, возвращающую значение True или False в зависимости от того, существует ли файл с заданным именем.
2. Создайте тестовый код, использующий обращения к функции, построенной при выполнении п.1.
3. Разработайте версию процедуры Trace, предусматривающую вывод информации в текстовый файл.



# 18-й час

## Обработка ошибок во время выполнения программы



На прошлом занятии вы научились предотвращать, находить и устранять такие ошибки, которые принято называть термином *bug*, — в их появлении целиком повинен сам программист. Теперь мы рассмотрим приемы обработки ошибок другого рода. Важно понимать их существенные различия. Если в первом случае недостатки находятся в самом коде (из-за неверной реализации алгоритмов, неправильного использования объектов данных и т.д.), то во втором проблемы могут возникать, скажем, по причине ввода неправильных данных пользователем. (Да-да, пользователи — живые люди, и им свойственно совершать ошибки.)

В иных ситуациях пользователь попытается открыть ранее удаленный файл, на диске может не хватить свободного места либо соединение с Internet окажется "сброшенным". Подобные проблемы называют ошибками периода выполнения, и на этом занятии вы научитесь с ними справляться.

Если в области программирования эта книга — ваш первый опыт, текущая глава определено для вас. То же самое можно сказать и в случае, если программированием приходилось заниматься относительно давно — может быть, лет пять или шесть назад. С тех пор приемы борьбы с ошибками значительно изменились и получили существенное развитие. Вы узнаете, как писать устойчивый, "живучий" код, используя самые современные технологии обработки ошибок периода выполнения.

Основные темы занятия.

- Исключительные ситуации.
- Создание программных обработчиков ошибок.
- Создание ресурсозащитных блоков.
- Использование объектов класса `Err`.



# Сравнение технологий обработки ошибок

Когда я был студентом колледжа, нас учили, что обработка ошибок в программе сопряжена с построением наборов управляющих структур для проверки фактов выполнения тех или иных условий. Главная идея заключалась в том, что программист, приступая к решению конкретной задачи, должен был построить ряд условных выражений, чтобы убедиться в правильности исходных данных и надлежащем состоянии кода. Такой подход нельзя назвать абсолютно неправильным и непродуктивным — просто он соответствовал прежнему состоянию развития технологий и уровню понимания проблем программирования. К тому времени уже появилась и другая технология обработки ошибок, связанная с отслеживанием так называемых исключительных ситуаций, но она еще не поддерживалась большинством основных языков и систем программирования.

## Новый термин

*Исключительная ситуация*, или *исключение*, — это условие, приводящее к возникновению ошибки, а *обработчик исключений* — программный код, позволяющий обнаружить ошибку и устранить ее последствия.

С тех пор понятие надежности и устойчивости кода в сообществе программистов (и в индустрии программного обеспечения в целом) существенно изменилось. Теперь "живучим" называют код, в котором для борьбы с ошибками применяются обработчики исключений, а не условные конструкции. Это не значит, что код обработки исключений используется всеми и во всех возможных случаях, — просто данная технология более предпочтительна. Но прежде мы все-таки рассмотрим традиционные способы обработки ошибок, предусматривающие применение условных конструкций.

## Обработка ошибок с помощью условных выражений: старый подход

Этот подход подразумевает предварительное включение в код целого набора условных конструкций для тестирования ситуаций, потенциально грозящих возникновением ошибок. На этапе проектирования и кодирования сложно учесть абсолютно все возможные условия — одни из них могут выпасть из рассмотрения случайно, другие на момент написания программы зачастую просто неизвестны.

Еще один немаловажный фактор таков: программист, заранее предусматривающий различные нестандартные ситуации, склонен предполагать, что ошибки обязательно произойдут. Он насыщает код таким количеством проверок, что это не может не ухудшить производительности приложения — все дополнительные программные инструкции выполняются даже в том случае, когда в действительности все в порядке.

Если какая-либо из ситуаций, приводящих к возникновению ошибки, оказывается неучтенной, программа просто (говоря молодежным сленгом) "падает". Она либо прекращает реагировать на действия пользователя, либо приводит к зависанию или перезагрузке компьютера. Но шутки становятся неуместными, если авария программы сопровождается еще и потерей или повреждением данных.

Программист в таком случае обычно не может знать обо всех истинных причинах ошибки, поэтому лучший способ ее исправления — размышления и поиски.

## Обработка исключений: современная технология

Обработка исключений основывается на совершенно ином подходе. Идея, лежащая в его основе, заключается в написании кода, "отлавливающего" ошибки только в том случае, когда они действительно происходят. Обработчики исключительных ситуаций обеспечивают реакцию на все ошибки — без каких-либо "исключений".

Это означает, что не нужно тратить время на предварительный анализ потенциально опасных случаев и написание специализированных обработчиков — теперь ваш код способен справиться с любыми ошибками; достаточно сформулировать и зафиксировать те действия, которые программа должна выполнять в аварийных ситуациях по умолчанию. Обычное решение состоит в попытке вернуть программу в то состояние, в котором она пребывала непосредственно перед возникновением исключения. Хотя при этом не гарантируется устранение ошибки как таковой, подобная мера способна предотвратить "зависание" программы или сбой компьютера.

Помимо мер, предусматривающих поведение программы по умолчанию, обработчики исключений не могут не учитывать возможности реагирования на ошибки конкретного типа и применения специальных действий.

Если код не в состоянии полностью справиться с возникшей проблемой, вы как программист должны предложить пользователю сообщение с понятным описанием причин происшедшего и возможность корректного завершения работы, предотвращающую серьезные потери данных.



Один из видов деятельности, которой занимается фирма *Software Conceptions*, состоит в тестировании и оценке программного обеспечения по заказам других компаний. Если процедуры тестируемого приложения возвращают коды ошибок и предусматривают их обработку с помощью наборов условных выражений, приложение получает заведомо низкую оценку. Если же в программе используется технология обработки исключений, ей — при прочих равных условиях — дается более высокая оценка. Код, в котором применяются обработчики исключительных ситуаций, наверняка более "живучий".

Многие факторы, составляющие понятие хорошей программы (и об этом уже говорилось), в достаточной степени субъективны. Никто не запрещает вам применять — и с успехом — способы обработки ошибок с помощью условных конструкций, но почти невероятно, что программа в этом случае будет так же надежна, как и при использовании технологии обработки исключительных ситуаций. Любой, кто предпочтет громоздким наборам условий компактные и универсальные обработчики исключений, окажется в более выгодном положении.

Способность "отлавливать" все без исключения ошибки, возможность задания действий, предпринимаемых по умолчанию, средства устранения последствий ошибок либо корректного выхода из программы с сохранением данных, выполнение строк кода обработчика только в том случае, когда ошибка действительно произошла, — все эти факторы говорят о серьезном превосходстве методов обработки исключений над традиционными способами борьбы с ошибками на основе наборов условных конструкций. Компьютерное сообщество проголосовало, и новой технологии была присуждена победа (как в боксе) за явным преимуществом.

Впрочем, вынужден вас огорчить — VBA все еще не обладает возможностями полноценной поддержки механизмов обработки исключительных ситуаций. Но есть и хорошая новость: в языке программирования Visual Basic (и, как следствие, в VBA) изначально реализована весьма неплохая схема "отлова" ошибок, которая, в некотором весьма напоминает классические способы обработки исключений.



Visual Basic.NET поддерживает структурированную обработку исключений.

В следующей части этой главы речь пойдет о самых эффективных приемах борьбы с ошибками, поддерживаемых в VBA. Приведенный ниже материал поможет вам в создании образцов надежного, устойчивого и "живучего" кода.

# Как строить обработчики ошибок

Будем считать (в данном случае мы можем себе это позволить) понятия обработчика ошибок и обработчика исключений равнозначными. Microsoft в контексте темы программирования на VBA избегает употребления словосочетания *обработчик исключений*, поэтому так поступим и мы. (Подозреваю, что не только я, но и специалисты Microsoft понимают, что средства борьбы с ошибками, реализованные в VBA, нельзя назвать полноценным механизмом обработки исключительных ситуаций.) Сейчас в документации и литературе по VBA принят термин *обработчик ошибок*, так что мы вынуждены подчиниться и будем употреблять эту фразу для ссылок на образцы кода, которые изучаем. Но все равно ошибки, которые возникают в программе, — это, по существу, исключения.

Блок обработчика ошибок состоит из заголовка и тела. Строка заголовка должна располагаться непосредственно перед той областью текста функции или процедуры, в которой существует наибольшая опасность возникновения ошибок. Тело обработчика всегда располагается в конце функции или процедуры — в отличие от условных конструкций, где строки кода, имеющие отношение к обработке ошибки, ничем не отличаются от всех других и размещаются тут же, внутри выражения. Код обработчика ошибки выполняется (если не подразумевается иное) только в случае ее действительного возникновения.

## Конструкция заголовка обработчика ошибок



Заголовок обработчика ошибок задается посредством выражения `On Error GoTo`. Его синтаксис приведен ниже.

`On Error GoTo Метка`

В ходе дальнейшего повествования, употребляя слово "функция", имеем в виду как подлинные функции, так и процедуры (подпрограммы) или методы классов, при условии, если различия между ними в контексте обсуждения не существенны.

Чтобы задать заголовок обработчика ошибок, вам необходимо буквально воспроизвести словосочетание `On Error GoTo` и сопроводить его наименованием метки, указывающей на строку кода, с которой начинается тело обработчика. Тело всегда размещается в конце функции. Листинг 18.1 содержит пример со строками заголовка обработчика и его метки.

Листинг 18.1. Пример пустого обработчика ошибок

```
1: Sub Test( )
2:   On Error GoTo EXCEPT
3:   EXCEPT:
4: End Sub
```



Чаше всего я даю меткам имя `EXCEPT` (исключение) — это "исключает" тяжкие раздумья по поводу поиска названий, соответствующих конкретной ситуации. Имя метки должно быть уникальным в пределах функции. Если та же функция содержит и другие обработчики ошибок, можно предложить использовать названия типа `EXCEPT1`, `EXCEPT2` и т.д. Впрочем, если вы будете стараться писать короткие функции, проблема именования меток вообще исчезнет.

Строка 2 листинга 18.1 содержит заголовок обработчика ошибок, указывающий на метку EXCEPT. В строке 3 размещается собственно метка. Обратите внимание на использование в названии метки символа двоеточия (:); это обязательная часть конструкции метки, ее отличительный признак. (В заголовке обработчика двоеточие не употребляется!) После выполнения инструкции строки 2 любая ошибка, возникающая далее, заставит программу "перепрыгнуть" на строку тела обработчика, следующую за указанной меткой.

## Правила именования меток

Рекомендуется вводить названия меток символами верхнего регистра. В таком случае их легче распознавать в теле функций. Впрочем, это только совет. Вы вольны называть метки так, как хотите, но помните о необходимости быть последовательным в своих действиях — характерный (хотя, разумеется, не единственный и не самый главный) признак профессионализма. Следуйте принятым правилам неукоснительно — и ваша работа станет продвигаться вперед значительно быстрее.

## Не забывайте об инструкции Exit

Листинг 18.1 содержит один недостаток. Если после строки 2 вы вставите код, который планируете тестировать (вы обязаны это сделать — иначе процедура лишена смысла), то тело обработчика ошибки, как и остальные строки, будет выполняться всегда. Листинг 18.2 демонстрирует исправленную версию процедуры Test, снабженную дополнительной командой Exit Sub.

Листинг 18.2. Пример использования инструкции Exit Sub

```
1: Sub Test ( )
2:   On Error GoTo EXCEPT
3:   Exit Sub
4:   EXCEPT:
5: End Sub
```

Теперь все в порядке. При создании обработчика ошибок имеет смысл ввести в текст все эти строки (содержащие заголовок, команду Exit и метку) сразу, чтобы ничего не упустить. Если вы забудете вставить строку с Exit, программа, дойдя до метки, начнет выполнять инструкции тела обработчика и вместо устранения ошибок может вызвать их появление.

Конкретный вид команды Exit зависит от типа программного блока, в контексте которого она употребляется. Если блок представляет собой процедуру (подпрограмму), вы должны написать Exit Sub. Для функций используется конструкция Exit Function, а для методов Property — Exit Property. (Подробнее о методах Property см. главу "21-й час. Основы программирования классов".)

Чтобы процедура листинга 18.2 смогла выполнять что-то полезное, необходимо вставить между строками 2 и 3 функциональный код, а после строки 4 — код тела обработчика ошибок.

## Общие правила создания обработчиков ошибок

Обращайтесь к средствам построения обработчиков ошибок только в тех случаях, когда отчетливо представляете, какого рода ошибки могут возникнуть и как их устранить.

Действия, предусмотренные VBA по умолчанию, предполагают отображение окна сообщения с текстом, описывающим характер ошибки, и выдачу звукового сигнала. Если вы не можете придумать ничего лучшего по сравнению с операциями, выполняемыми по умолчанию, не создавайте обработчик ошибок. Исключением может служить необходимость документирования сообщений об ошибках в журнальном файле.

Иногда бывает полезным, наряду с собственными творческими достижениями, использовать и стандартные операции обработки ошибок. Но действия, предпринимаемые по умолчанию, выполняются только в случае отсутствия обработчика. Если обработчик создан, а вы хотели бы еще и имитировать стандартные операции, достаточно обратиться к объекту `Err`, который содержит информацию о возникшей ошибке (подробнее об этом см. раздел "Использование объектов класса `Err`" данной главы).

## Очистка состояния обработчика ошибок

Область действия обработчика ошибок ограничена пределами функции. Если вы построили обработчик, он будет "работать" только внутри текущей функции. При необходимости очистки состояния обработчика ошибок на уровне функции следует выполнить команду

```
On Error GoTo 0
```

Эта инструкция отменяет действие всех обработчиков, ранее установленных в пределах функции, и сбрасывает в исходное состояние атрибуты объекта `Err`. (Подробности, касающиеся объекта `Err` и способов его использования, приведены ниже, в одноименном разделе этой главы.)

## Пассивные обработчики ошибок



*Пассивным* называют такой обработчик, который позволяет программе просто перейти к строке кода, следующей за ошибочной, а не "прыгать" к метке, указывающей на специальный блок кода. Синтаксис задания пассивного обработчика приведен ниже:

```
On Error Resume Next
```

Приведенная команда трактуется исполняющей системой таким образом: в случае возникновения ошибки в любой последующей команде необходимо сразу перейти к очередной строке. Поскольку это все-таки обработчик **ошибок** — пусть и простой, — он отменяет все действия, предлагаемые по умолчанию.

В такой форме обработчик ошибок, использующий инструкцию `Resume`, применяется очень редко. Но если вас устраивает это простое решение, заключающееся в переходе к очередной строке кода (и условия задачи позволяют поступать именно так), пожалуйста, пользуйтесь на здоровье.

Например, `Resume Next` применяют при копировании набора данных, в случае, когда отдельные поля являются нулевыми. Вы можете игнорировать пустые поля посредством `Resume Next`.

## Использование команды `Resume`

Команда `Resume` весьма полезна и сама по себе, особенно в том случае, когда после выполнения кода обработчика необходимо вернуться к строке, вызвавшей ошибку, и попытаться выполнить ее заново. `Resume` просто запускает на повторное выполнение команду, которая явилась причиной последней ошибки, поэтому использовать `Resume` имеет смысл только после того, как предприняты какие-либо действия по устранению источников возникновения проблемы.

Листинг 18.3 демонстрирует пример использования команды Resume. Этот код достаточно прост.

### Листинг 18.3. Пример использования команды Resume

```
1: Sub Test ( )
2:   Call DeleteFile("SomeFile")
3: End Sub
4:
5: Sub RaiseError(ByRef Err As Object)
6:   Call Err.Raise(Err.Number, Err.Source, Err.Description, _
7:     Err.HelpFile, Err.HelpContext)
8: End Sub
9:
10: Function FileExists(Filename As String) As Boolean
11:   FileExists = Len(Dir(Filename)) > 0
12: End Function
13:
14: Sub DeleteFile(ByVal FileName As String)
15:   On Error GoTo EXCEPT
16:   Kill FileName
17:   Exit Sub
18: EXCEPT:
19:   Const FILE_NOT_FOUND = 53
20:   If (Err.Number <> FILE_NOT_FOUND) Then Call RaiseError(Err)
21:   If (MsgBox(Err.Description & "(" & FileName & ")" & _
22:     "Повторить?", vbRetryCancel) = vbRetry) Then
23:     FileName = InputBox("Введите имя файла:", "Имя файла", "")
24:     If (FileExists(FileName)) Then Resume
25:   End If
26: End Sub
```

#### Анализ

Строки 1–3 листинга 18.3 содержат текст вспомогательной подпрограммы, предназначенной для тестирования основной рассматриваемой в данный момент процедуры — DeleteFile. Процедура, расположенная в строках 5–8, в сущности, необязательна. Ее тело можно было расположить непосредственно в строке 20, но считаем, что процедура RaiseError вполне пригодна к повторному использованию, да и обработчик ошибок в процедуре DeleteFile стал заметно проще. Строки 10–12 содержат вариант функции проверки существования файла. Эта функция рассматривалась на прошлом занятии, а здесь нашла свое очередное применение — нам не понадобилось разрабатывать ее вновь. "Гвоздь программы" — процедура DeleteFile — занимает строки 14–26.

В строке 16 расположена основная команда процедуры, ради выполнения которой стоило осуществлять весь этот процесс, — директива Kill "убийства" файла с заданным именем. Конечно, можно было бы внести Kill FileName во все части кода, где это необходимо, но в таком случае пришлось бы загромождать текст строками обработки ошибок. Удобно и целесообразно выносить подобные действия в отдельные функции и строить в них обработчики ошибок — теперь обработчик не затеряется в массе других строк, не имеющих к нему отношения, и, в свою очередь, не будет привлекать к себе лишнего внимания в тех случаях, когда он не нужен. Всякий раз, когда необходимо удалить файл, я с легкой душой обращаюсь к процедуре DeleteFile, которая гарантирует полноценное решение задачи.

Строки 18-26 листинга 18.3 способны обработать любую возникшую ошибку. Если номер ошибки не 53 (*File Not Found* — файл не найден), предпринимается попытка сгенерировать сообщение об ошибке методом `Raise` объекта `Erg`. Если же ошибка связана именно с отсутствием файла, пользователю будет предложено ввести новое имя файла, приносимого в жертву. Теперь, если файл с введенным именем существует, команда `Resume`, расположенная в строке 24, передаст управление строке 16.

Разумеется, никто не запрещает вам написать нечто, похожее на строку, приведенную ниже:

```
If (FileExists(FileName)) Then Kill FileName
```

Нетрудно придумать множество других вариаций на заданную тему, и все они с формальной точки зрения окажутся верными. Вы могли бы, например, добавить предложение `Else`, позволяющее пользователю указать новое имя файла, проверить факт его существования, а затем попробовать удалить этот файл. Все это неплохо — именно неплохо, но и не очень хорошо. А что случится, если, скажем, и вновь заданный файл не существует? Как быть — создавать цикл, дающий возможность пользователю снова и снова вводить имя файла, а затем пытаться его удалить? Ну, а как справиться с ситуацией, когда файл существует, но он открыт другой программой? Или файл помечен атрибутом "только для чтения"?



Да, бывают случаи, когда обработчик основывается на наборе условных выражений — это как раз то, что нужно: например, требуется проверить, содержит ли выражение правильную дату. Если такое условие решит проблему — действуйте: достаточно написать `If (IsDate (Выражение)) Then` — и вы получите ответ на интересующий вас вопрос.

Примите к сведению такое общее правило: если необходимо решить некую конкретную проблему, используйте условные выражения; если же речь идет об обеспечении надежности кода в целом, обращайтесь к средствам построения обработчиков ошибок.

Существует масса причин, которые могут воспрепятствовать удалению файла. Вот почему обработчики ошибок, о которых мы говорим, так ценны. Обработчик "помалкивает", пока все в порядке, и вступает в действие, если возникает какая-либо ошибка. Если проблема устранена, команда `Resume` позволяет повторно выполнить соответствующую строку кода.

## Применение команды `Resume Next`

Команда `Resume Next` может использоваться как в строке заголовка обработчика ошибок в качестве директивы пассивных действий, так и внутри него. `Resume Next` при возникновении ошибки предписывает перейти к очередной строке кода.

Исправим рассмотренный ранее пример с учетом необходимости удаления файла до начала выполнения операций с ним (скажем, речь идет о файле, содержащем какой-либо ежедневно обновляемый отчет). В этом случае уместно написать следующее:

```
On Error Resume Next  
Kill FileName
```

'Создание нового экземпляра файла и обработка данных

Если файл существует, он удаляется (это как раз то, что нужно), а затем выполняются обычные действия. Команда `Resume Next` гарантирует, что попытка удаления старого файла не приведет к остановке программы, если этого файла случайно не окажется — ведь старый файл не является центральным звеном алгоритма.

# Использование объектов класса Err

Объект класса Err создается и поддерживается исполняющей системой автоматически. При возникновении ошибки свойствам объекта Err присваиваются соответствующие значения, позволяющие определить, что именно произошло. Объект Err содержит два метода, Clear и Raise, и шесть свойств — Description, HelpContext, HelpFile, Source, Number и LastDLLError. Методы дают возможность изменять состояние объекта, а свойства описывают характер ошибки.

## Свойства объекта Err

Свойство Description содержит текстовое описание ошибки. HelpFile и HelpContext — это, соответственно, имя файла оперативной справки и номер темы в нем. Такие данные могут быть заданы при вызове метода Raise для генерации сообщения об ошибке. В этом случае при нажатии клавиши <F1> пользователь получит возможность вызова приложения оперативной справочной системы с автоматическим открытием той страницы указанной вами темы, которая, по вашему мнению, должна содержать более подробную информацию об ошибке и способах дальнейших действий.

Свойство Source указывает на имя объекта или приложения, служащего источником ошибки. Number, номер ошибки, — это число, предопределенное в VBA (как, например, 53 — ошибка отсутствия файла, рассмотренная нами в листинге 18.3) либо заданное программистом.

Свойство LastDLLError содержит описание ошибки, если она возникла в динамической библиотеке, созданной средствами Visual Basic.

## Методы объекта Err

Метод Clear позволяет очистить содержимое объекта Err. Формат обращения к нему — Err.Clear. Clear вызывается автоматически при выполнении программой команд Resume, Exit или любой инструкции GoTo.

Формат вызова метода Raise описывается следующим синтаксическим выражением:

```
Call Err.Raise( Number [, Source [, Description [, HelpFile [, HelpContext]]]] )
```

(Наименования параметров метода Raise даны на английском языке, поскольку они совпадают по смыслу со свойствами объекта Err, рассмотренными выше. — Прим. перев.) При обращении к процедуре Raise должен быть указан хотя бы один параметр — Number, номер ошибки. Если вы введете, скажем, число 54, будет сгенерировано соответствующее сообщение об ошибке (номеру 54 отвечает ошибка задания неверного режима работы с файлом — *Bad File Mode*). Чтобы сгенерировать сообщение по номеру ошибки, определенному пользователем-программистом, необходимо применить конструкцию Call Err.Raise(vbObjectError + Number), где Number — это заданный пользовательский номер ошибки. Акцентируем ваше внимание: Number — обязательный аргумент, все остальные разрешается не указывать. Если задается параметр Source, он обычно обозначает имя объекта или модуля, в котором произошла ошибка. Аргумент Description применяется для определения строки текстового описания ошибки. Параметры HelpFile и HelpContext используются в тех случаях, если приложение содержит ссылки на файлы оперативной помощи, в которых находится подробная информация об ошибках и способах устранения их последствий. Процедура RaiseError, приведенная в тексте листинга 18.3, содержит пример обращения к методу Raise объекта Err.



Существует, по меньшей мере, две ситуации, оправдывающие практическое применение объекта `Egg`: в первом случае полезно исследовать свойства `Egg` после возникновения ошибки, а во втором с помощью метода `Raise` удобно сгенерировать сообщение об ошибке, чтобы дать пользователю информацию о характере проблемы. В листинге 18.3 мы обратились к `Raise`, чтобы предоставить пользователю данные об ошибке, для которой специальные средства обработки не предусмотрены (речь идет обо всех ошибках, кроме ошибки с номером 53).

## Применение обработчика ошибок для проверки данных, вводимых пользователем

Наиболее распространенный способ использования обработчиков связан с созданием блоков кода, оказывающих помощь в устранении последствий возникающих ошибок. В ходе этого занятия вы уже ознакомились с рядом примеров такого рода. Но есть и другие — обработчики иногда удобно использовать с целью проверки правильности данных, введенных пользователем. Вы можете применить соответствующую стандартную функцию преобразования типов, а наряду с ней — конструкцию обработчика, позволяющую избежать ошибок, если информация не поддается преобразованию. Просмотрите текст листинга 18.4.

**Листинг 18.4.** Пример использования обработчика в задаче определения признака високосного года

```
1: Sub Test ( )
2:     MsgBox IsLeapYear( 2001 )
3: End Sub
4:
5: Function IsLeapYear( ByVal Year As Integer ) As Boolean
6:
7:     On Error GoTo EXCEP
8:
9:     IsLeapYear = True
10:    Dim ADate As Date
11:    ADate = CDate( "2/29/" & Year )
12:    Exit Function
13:
14: EXCEPT:
15:    IsLeapYear = False
16:
17: End Function
```

### Анализ

Листинг 18.4 демонстрирует прием (правда, близкий к трюкачеству) определения того, является ли указанный год високосным. (Строки 1–3 содержат данные тестовой процедуры.) В строке 7 задается заголовок обработчика ошибок. Команда строки 8, свидетельствующая о неисчерпаемом оптимизме программиста, присваивает возвращаемой переменной значение `True`. В строке 11 объявляется вспомогательная переменная типа `Date`. Далее из литерала, задающего дату 29 февраля и значения тестируемого года, формируется строка, которая передается встроенной функции `CDate`. Эта функция предпринимает попытку преобразования значе-

ния типа `string` в `Date`. Если преобразование осуществляется безошибочно, выполнение функции завершается. В противном случае управление передается на метку `EXCEPT`, выполняется инструкция обработчика ошибки, и функция возвращает значение `False`.



В числе встроенных функций VBA имеется функция `IsDate` типа `Boolean`, выполняющая анализ корректности даты и возвращающая значение `True`, если дата верна, и `False` — в противном случае.

Конечно, помимо продолжительности года, следует проверять корректность значений месяца и числа. Существует и готовый алгоритм проверки того, является ли год високосным. Мне удалось ознакомиться с книгой Джека Пэдама (Jack Purdum) *C Programmer's Toolkit* 1989 года издания и преобразовать функцию с языка C на VBA. Листинг 18.5 содержит реализацию алгоритма определения признака високосного года на языке VBA.

#### Листинг 18.5. Функция проверки признака високосного года

```
1: Function LeapYear(ByVal Year As Integer) As Boolean
2:   LeapYear = (Year Mod 4 = 0) And ((Year Mod 100 <> 0) Or _
                                     (Year Mod 400 = 0))
3: End Function
```

Високосным считается год, значение которого нацело делится на четыре и четыреста, — за исключением тех, которые делятся на сто. Наверняка аналогичный код присутствует и в недрах функции `CDate`.

## Проблема экономии ресурсов компьютера

Применяются блоки обработчиков ошибок также с целью защиты ресурсов компьютера от истощения. Занимаясь программированием, необходимо заботиться об экономии любых ресурсов, объем которых заведомо ограничен. В вашем распоряжении имеется, например, только определенное количество свободных номеров файлов, некоторое конечное число ячеек памяти и ограниченный объем свободного дискового пространства. Если ваша программа не будет предпринимать **необходимых** мер по их экономии, то, не исключено, со временем ресурсы **окажутся** исчерпанными.

Чтобы решить задачу экономии ресурсов, можно воспользоваться механизмом создания обработчика ошибок — при условии, что его тело, которое должно выполняться *всегда*, будет содержать необходимые инструкции освобождения занятых ресурсов. VBA не предлагает специальных средств поддержки подобного подхода, но их легко построить. Хотя программа, написанная на VBA, способна истощить ресурсы памяти с меньшей долей вероятности, нежели аналогичное приложение, созданное, скажем, на языке C++, встречаются ситуации, когда необходимо проявлять особое внимание относительно корректного закрытия ранее открытых файлов или баз данных.

Блок строк кода, решающих задачу защиты ресурсов компьютера от истощения, создается как тело обычного обработчика ошибок. Но поскольку он должен выполняться всегда, инструкцию `Exit`, препятствующую переходу на метку обработчика, следует исключить. Листинг 18.6 демонстрирует пример обработчика, выполняющего закрытие файла в любом случае — независимо от того, произошла ошибка или нет.

```
1: Sub ProtectedResource( )
2:   On Error GoTo FINALLY
3:   Dim Handle As Double
4:   Handle = FreeFile
5:   Open "c:\autoexec.bat" For Input As #Handle
6:   Dim Line As String
7:   Do While (EOF(Handle) = False)
8:     Line Input #Handle, Line
9:     Debug.Print Line
10:  Loop
11: FINALLY:
12:   Close #Handle
13: End Sub
```



Ресурсосберегающие блоки не слишком часто применяются в программах на Visual Basic. Программисты, работающие с языками **структурированной** обработки ошибок, используют эту технику чаще. Компания Microsoft обеспечила структурированную обработку ошибок в Visual Basic.NET, некоторые элементы этой технологии реализованы и в VBA.

Тот факт, что данный простой метод требует поддерживать только один обработчик ошибок, является следствием разделения задач. Можно встретить и сложные процедуры, однако это не означает, что они хороши.

#### Анализ

Строка 2 задает заголовок обработчика ошибок. В этом примере речь идет об обеспечении гарантий закрытия файла. В строках 3-4 объявляется переменная для хранения номера открытого файла и в результате вызова функции FreeFile этот номер резервируется операционной системой. В строке 5 файл открывается. Строки 6-10 содержат ряд инструкций по обработке данных из файла (отдельные строки файла считываются и выводятся в окне Immediate). Команда строки 12, закрывающая файл, выполняется всегда (обратите внимание — инструкция Exit Sub отсутствует), поэтому выделенный файловый ресурс будет обязательно освобожден и возвращен операционной системе. В данном случае при именовании метки (FINALLY) мы придерживались иного соглашения — чтобы легко отличить обработчик ошибки, выполняющий дополнительную задачу обеспечения экономии ресурсов компьютера, от всех других.

## Применение объекта Debug

На страницах нашей книги мы уже неоднократно ссылались на объект Debug. Он относится к глобальным ресурсам программы и поэтому существует в единственном экземпляре. Объект Debug особенно полезен на этапе проектирования и отладки приложения. Он содержит два метода — Print и Assert; других атрибутов в его составе нет. Метод Print позволяет отобразить содержимое элемента данных в окне Immediate, а верификатор Assert дает возможность приостановить выполнение программы в том случае, если выражение типа Boolean, переданное ему в качестве параметра, дает в результате вычисления значение False.

Метод Print мы использовали уже много раз, и он хорошо вам знаком. Методу Assert был посвящен довольно подробный раздел предыдущей главы. Оба средства весьма полезны, и в процессе решения конкретных задач вы наверняка в этом убедитесь.

## Резюме

Существуют несложные правила и приемы, которые помогут вам в создании качественного программного кода. Прежде чем приступить к процессу непосредственного программирования, хорошо продумайте, что и как именно вы предполагаете делать. Старайтесь в максимальной степени упрощать код и при малейшей возможности повторно используйте ранее написанные фрагменты. Проверяйте код вновь и вновь и не отказывайтесь от помощи своих коллег. Активно внедряйте все приемы, о которых шла речь на этом занятии.

Если необходимо проверить некое специальное условие, используйте обычные условные конструкции. Если же речь идет о гарантиях надежности и устойчивости приложения, обращайтесь к средствам создания универсальных обработчиков, способных обеспечить защиту от любых возможных ошибок. Помните, что по умолчанию VBA отображает сообщение об ошибке с названием оператора, который вызвал ее появление. Если вы не в состоянии предложить ничего более совершенного, оставьте все как есть и не предпринимайте дополнительных действий. Используйте конструкции обработчиков ошибок для обеспечения функций закрытия файлов, наборов данных и освобождения ресурсов компьютера, ранее выделенных приложению.

Обращайтесь к атрибутам объекта Err, предоставляющим пользователю наиболее полную информацию о природе ошибки и ее причинах. Следуйте всем перечисленным выше советам аккуратно и последовательно, лишь тогда пользователи приложений сохранят собственные нервные клетки и не станут посягать на ваши. Не забудьте прочесть разделы "Вопросы и ответы" и "Задания", завершающие эту главу.

## Вопросы и ответы

**Вопрос.** Что представляет собой объект Err?

**Ответ.** Объект Err содержит информацию о последней ошибке, возникшей в текущем приложении или динамической библиотеке.

**Вопрос.** Где можно получить сведения о номерах ошибок?

**Ответ.** Найдите в оперативной справочной системе по Microsoft Visual Basic раздел *Trappable Errors*.

**Вопрос.** В чем отличие обычных обработчиков ошибок от тех, которые предполагают решение задачи защиты ресурсов компьютера от истощения?

**Ответ.** Отличие есть, но совершенно не существенное. Обычные обработчики предназначены для устранения ошибок и выполняются *только* в случае возникновения последних, а специальные, названные выше, "срабатывают" *всегда* и гарантируют возврат ресурсов, выделенных приложению. В обработчиках обоих видов заголовков одинаков — On Error GoTo.

**Вопрос.** Обязательно ли в каждую функцию включать код обработчика ошибок?

**Ответ.** Нет. Обработчики должны создаваться там, где есть риск возникновения ошибок, и предлагать действенные меры по решению проблем.

**Вопрос.** Как мне быть в случае, если я не могу предложить эффективных мер по обработке ошибок?

**Ответ.** Реализация предлагаемых по умолчанию функций обработки ошибок с каждой новой версией VBA и Windows становится все более совершенной. Во многих случаях стандартных средств такого рода может оказаться вполне достаточно. А со временем вы и сами научитесь писать надежный и "живучий" программный код.

## Задания

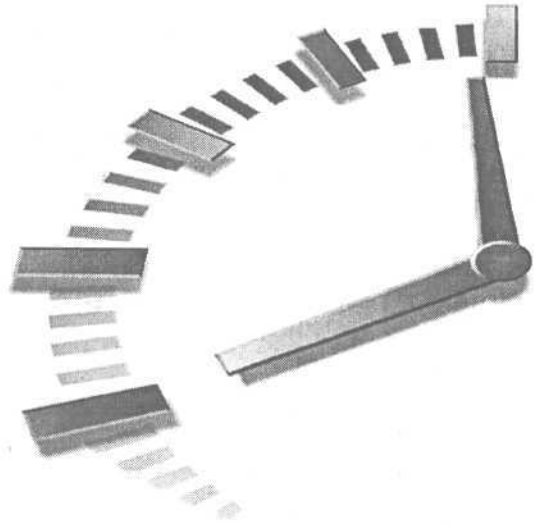
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Какие аргументы необходимо передать методу `Raise` объекта класса `Err`, чтобы связать сообщение об ошибке с файлами оперативной справки?
2. Как называются объект и его метод, позволяющие выводить информацию в окно `Immediate` во время отладки приложения?
3. Каково отличие обычного обработчика ошибок от того, который предназначен для решения задачи освобождения занятых ресурсов?
4. Можно ли сгенерировать стандартное сообщение об ошибке? Если да, то как и в каких случаях это целесообразно делать?

## Упражнения

1. Создайте обработчик, предусматривающий принудительное присваивание объекту класса `Recordset` значения `Nothing`.
2. Исправьте код листинга 18.3 с учетом того, что пользователь может удалять файл, помеченный атрибутом "только для чтения".
3. Напишите команду генерации сообщения об ошибке отсутствия файла, предусматривающую задание пользовательского номера ошибки.\



# Часть VII

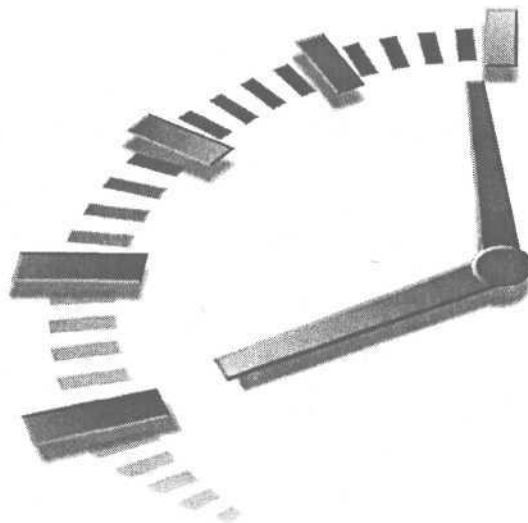
## Проектирование интерфейса пользователя

### Темы занятий

19-й час. Создание экранных форм

20-й час. Как связывать Web-страницы с базами данных





## 19-й час

# Создание экранных форм

Теперь, одолев 18 занятий, вы уже готовы к тому, чтобы, систематизировав полученные знания, реализовать их на практике в виде полноценных Windows-приложений, снабженных средствами пользовательских интерфейсов. Проектирование графических интерфейсов — это в определенном смысле один из наиболее захватывающих аспектов программирования, поскольку результаты вашей работы становятся видны всем и сразу.

Access обладает встроенной поддержкой графического пользовательского интерфейса. Этот факт имеет немаловажное значение, поскольку еще несколько лет назад создание графических интерфейсов было чрезвычайно трудной задачей и уделом исключительных личностей-“небожителей”, знавших толк в программировании на языке С и посвященных в глубокие таинства Windows API.

Однако праздник, наконец, пришел и на нашу улицу. Теперь вы можете взять на себя смелость заявить, что пользовательский интерфейс — нечто более легкое и второстепенное в сравнении с написанием кода, который решает задачу по существу. В среде Access 2002 большая часть работы по созданию интерфейса может быть выполнена с помощью мыши и средств мощных функций-мастеров, которые принимают на себя изрядную долю рутинных операций.

Подобный стиль программирования стал возможным и дееспособным благодаря появлению так называемых средств *быстрой разработки приложений* (Rapid Application Development — RAD). VBA — замечательный язык программирования, а Access — прекрасное средство RAD. В ходе этого занятия вы научитесь применять Access для построения пользовательских интерфейсов и создания готовых приложений, имеющих “товарный” вид.

Основные темы занятия.

- Использование мастеров создания форм.
- Настройка форм.



- Свойства и события — ключевые аспекты поведения программы.
- Как тестировать и выполнять приложение.

## Использование мастеров создания форм

Мощь и привлекательность средств быстрой разработки приложений обусловлены удобством выполнения общих задач создания интерфейсов и простотой построения приложений в целом. Если за привлекательной графической оболочкой вашей программы будут скрываться еще и возможности корректного (!) и эффективного решения конкретной задачи — вы достигнете просто-таки потрясающего результата.

Access 2002 предоставляет такие инструменты проектирования графических интерфейсов, которые предполагают написание минимальных фрагментов кода либо вовсе исключают необходимость специального программирования. Чтобы создать готовое приложение, вам останется только написать код, решающий задачу по существу. Если вы новичок, имеет смысл начать работу с обращения к средствам мастеров построения форм. Мастера предлагают большое количество готовых стилей форм, позволяющих придать приложению профессиональный унифицированный вид буквально за несколько минут.

### Новый термин

Термином *форма* мы называем экземпляр (объект) класса Form. Класс Form содержит в своем составе большое число свойств, методов и событий, определяющих внешний вид объекта формы и способы его поведения. В отличие от других классов, с которыми вы уже успели ознакомиться на страницах нашей книги, объект класса Form обладает визуальными характеристиками и поэтому содержит много "графической" информации, которая определяет особенности его отображения на экране компьютера. Операционная система Windows целиком построена на концепции окна, поэтому любые видимые объекты всегда располагаются в пределах определенной оконной формы.

В составе Access 2002 существует целый ряд мастеров создания форм, относящихся к самым различным стилям представления и поведения. Далее кратко рассмотрим особенности каждого из них, а затем перейдем к задаче проектирования пользовательского интерфейса.

## Обзор режимов проектирования форм

При создании новой формы вы можете обратиться к одному из семи возможных способов, перечисленных в списке диалогового окна Новая форма (New Form) (рис. 19.1): Конструктор (Design View), Мастер форм (Form Wizard), Автоформа: в столбец (Autoform: Columnar), Автоформа: ленточная (Autoform: Table), Автоформа: табличная (Autoform: Datasheet), Автоформа: сводная таблица (Autoform: PivotTable), Автоформа: сводная диаграмма (Autoform: PivotChart), Диаграмма (Chart Wizard) и Сводная таблица (Pivot Table Wizard). Чтобы открыть диалоговое окно Новая форма, выберите элемент Формы в списке Объекты окна База данных и щелкните на кнопке Создать панели инструментов.

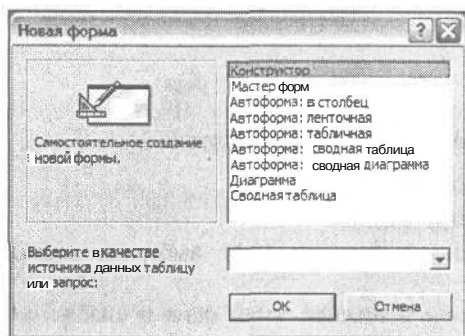


Рис. 19.1. Выберите в списке диалогового окна Новая форма требуемый режим создания формы — и все заботы Access примет на себя

## Режим Конструктор

В режиме Конструктор создается и открывается чистое окно формы. Форма — это окно в среде Windows. Каждый из графических элементов, которые вы видите на экране компьютера (работающего под управлением операционной системы Windows), принадлежит некоторой форме "территориально" либо связан с ней определенными неявными отношениями. Режим Конструктор позволяет работать в свободном стиле — выбрав его, вам придется многие функции выполнять самостоятельно.

## Режим Мастер форм

В режиме Мастер форм Access предложит пройти через серию диалоговых окон. При этом заполняются поля и выбираются значения параметров, которые, взятые в совокупности, помогают системе определить окончательный вид формы и особенно-сти ее поведения. Мастер автоматически помещает на форму все необходимые компоненты и присваивает ее свойствам требуемые значения, чтобы упростить дальнейшую работу.

Мы обратимся к услугам мастера совсем скоро. Режим Мастер форм — самый удобный старт в мир визуального программирования в среде Access.

## Режим Автоформа: в столбец

Режим Автоформа: в столбец позволяет создать форму простой структуры со строками, представляющими содержимое полей указанной таблицы.

Выберите в диалоговом окне Новая форма элемент Автоформа: в столбец, задайте в раскрывающемся списке наименование требуемой таблицы (см. рис. 19.1) и щелкните на кнопке ОК. Буквально в два приема вы получаете удобный графический инструмент просмотра содержимого таблицы. Соответствующий пример показан на рис. 19.2.

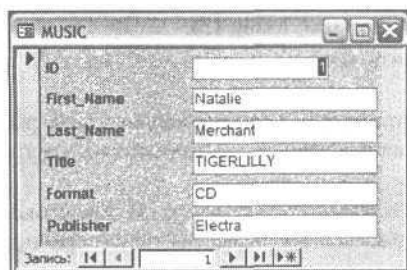


Рис. 19.2. Так выглядит форма, построенная в режиме Автоформа: в столбец

## Режим Автоформа: ленточная

Мастер Автоформа: ленточная действует подобно предыдущему. Чтобы построить форму в режиме Автоформа: ленточная, выполните следующие действия.

1. Выберите элемент **Формы** в списке **Объекты** окна **База данных**.
2. Щелкните на кнопке **Создать панели инструментов**.
3. В списке диалогового окна **Новая форма** (см. рис. 19.1) выберите элемент **Автоформа: ленточная**.
4. В раскрывающемся списке, расположенном в нижней части окна и снабженном пояснительной надписью **Выберите в качестве источника данных таблицу или запрос (Choose the table or query where the object's data comes from)**, укажите требуемый источник, информацию из которого вы хотели бы представить на форме.
5. Щелкните на кнопке **ОК**.

Это, собственно, все. Несколько первых пунктов инструкции являются общими для всех случаев, кроме режима **Конструктор**. Выполнив пп.1–5, вы получите готовую форму. Чтобы увидеть процесс в действии, обратитесь к любой из баз данных и таблиц, построенных нами на предыдущих занятиях. **Формы** всех рассматриваемых в книге типов заслуживают того, чтобы ознакомиться с ними поближе.

## Режим Автоформа: табличная

Мастер, поддерживающий режим **Автоформа: табличная**, создает форму, которая представляет информацию в том же виде, что и окна таблиц и запросов базы данных. Это самый простой стиль отображения данных. Чтобы создать форму в режиме **Автоформа: табличная**, выполните действия, описанные в предыдущей инструкции, выбрав соответствующий элемент списка диалогового окна **Новая форма**.

## Режим Автоформа: сводная таблица

Мастер, поддерживающий режим **Автоформа: сводная таблица**, позволяет динамически конфигурировать фильтр, строки и столбцы сводной таблицы и перестраивать организацию и презентацию данных. Сводные таблицы являются мощным средством представления данных, их можно добавлять в отчеты, формы и Web-страницы. (Подробнее см. главу "20-й час. Как связывать Web-страницы с базами данных".)

Предположим, необходимо создать настраиваемую сводную таблицу базы данных **Music Collection**. Можно создать сводную таблицу, представляющую все компании звукозаписи и организованную по списку исполнителей. Используя базу данных, построенную в главе "15-й час. **ADODB** — ваш верный помощник", создадим сводную таблицу.

1. Откройте или создайте базу данных **Music Collection** (см. главу 15).
2. В списке **Объекты** окна базы данных выберите **Формы** и щелкните на кнопке **Создать**.
3. В диалоговом окне **Новая форма** выберите **Автоформа: сводная таблица** и в раскрывающемся списке в нижней части окна выберите **MUSIC**. Щелкните на кнопке **ОК**.
4. На экране появится пустая сводная таблица в режиме конструктора. В диалоговом окне **Список полей сводной таблицы** будут содержаться все названия полей таблицы **MUSIC** (рис. 19.3).

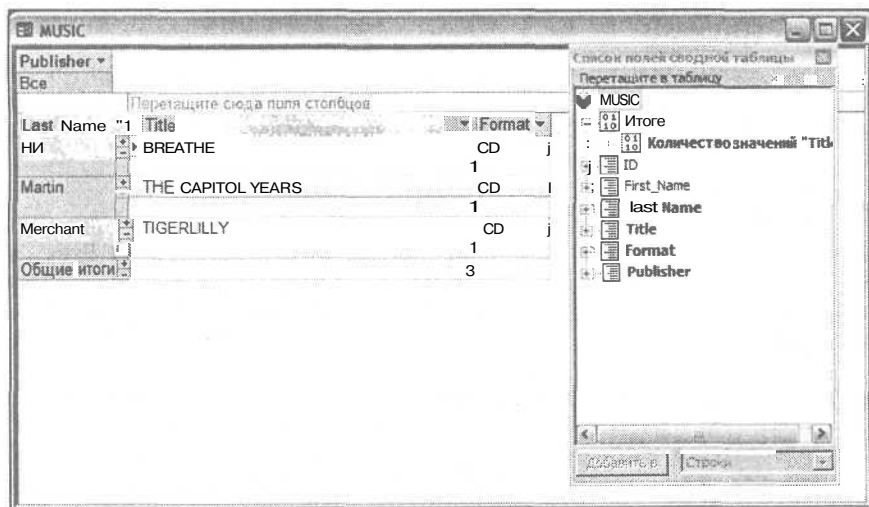


Рис. 19.3. Пример сводной таблицы, в котором вычисляется число записей каждого исполнителя

- Обратите внимание на раздел Перетащите сюда поля фильтра (Drop Filter Fields Here). Мы будем отсортировать поля по названию компании звукозаписи (PUBLISHER). Перетащите это поле из списка полей сводной таблицы в указанную область.
- Чтобы сгруппировать строки по фамилии исполнителя, перенесите LAST\_NAME из списка полей в область Перетащите сюда поля строк (Drop Row Fields Here).
- Чтобы просмотреть названия и формат записей, перенесите поля TITLE и FORMAT в область Перетащите сюда поля итогов или деталей (Drop Totals or Detail Fields Here). При этом сводная таблица примет вид, отображенный на рис. 19.3.
- Выберите одно из полей TITLE, щелкните на кнопке Автовычисления (Autocount), а затем на опции Число (Count). Программа определит число записей каждого из исполнителей.
- Закройте и сохраните сводную таблицу.

Обратите внимание, что в нижней части таблицы появилось общее число записей. Режим сводной таблицы позволяет реорганизовывать данные с целью выполнения анализа данных, например, определять промежуточные итоги. Можно также создавать сводные таблицы на основании более сложных наборов данных, например запросов.

## Режим Автоформа: сводная диаграмма

Мастер, поддерживающий режим Автоформа: сводная диаграмма, работает аналогично мастеру режима Автоформа: сводная таблица и позволяет представлять данные в графическом виде. Чтобы преобразовать пример предыдущего раздела в графический режим, выберите команду Вид⇒Сводная диаграмма (View⇒PivotChart View). Используется данный режим и тогда, когда настройка представления данных осуществляется в режиме сводной таблицы, а в режиме сводной диаграммы улучшается внешний вид формы (изменяются такие параметры, как цвет, стиль и формат вывода данных). Сводные таблицы и диаграммы можно добавлять в формы, Web-страницы и отчеты. Подробнее о сводных таблицах и диаграммах см. главу "20-й час. Как связывать Web-страницы с базами данных".

## Режим Диаграмма

Мастер построения диаграмм позволяет создать форму, представляющую данные в виде диаграммы. График — это объект. После его создания вы можете внести необходимые изменения, чтобы привести степень детализации и стиль отображения в соответствие с конкретными требованиями.

## Режим Сводная таблица

Мастер режима Сводная таблица строит форму, используя средства Excel, а затем сохраняет данные в объекте класса PivotTable. Сводная таблица допускает динамическое изменение параметров отображения и поддерживает функции автоматического пересчета данных.

## Создание формы в режиме Мастер форм

В ходе дальнейшего изложения мы будем ссылаться на базу данных с таблицей *MUSIC* (о ней шла речь в главе “15-й час. ADODB — ваш верный помощник”). Выбор не случаен, поскольку эту таблицу легко построить заново (если она не сохранилась). Если при построении будущих форм вы хотели бы обращаться к собственным таблицам и запросам — пожалуйста. Все, о чем будет рассказано далее, легко применимо к любым конкретным случаям.



Системы быстрой разработки приложений позволяют эффективно строить оконные графические интерфейсы пользователя. Достаточно нескольких щелчков мышью — и начало проекту положено. Но пусть вас не вводит в заблуждение внешняя простота — решать исходную задачу вам все равно придется. Прежде чем приступать к программированию, целесообразно потратить время на изучение проблемы и обдумывание деталей проекта.

Определение базы данных и создание форм — лишь малая часть решения. Необходим еще код, поддерживающий взаимодействие пользователя, формы и данных, но еще важнее, чтобы этот код решал поставленные перед разработчиком задачи.

Далее подробно описывается каждый шаг процедуры использования режима Мастер форм. Первое действие состоит в выборе базы данных и отображении диалогового окна Новая форма — с изучения его содержимого мы начали это занятие. Теперь, полагая, что окно уже открыто, продолжим работу.

1. Выберите в списке диалогового окна Новая форма элемент Мастер форм.
2. В раскрывающемся списке, расположенном в нижней части окна, укажите элемент (наименование таблицы) *MUSIC* (рис. 19.4).
3. Щелкните на кнопке ОК, чтобы перейти к первому из диалоговых окон мастера форм.
4. В диалоговом окне Создание форм укажите поля таблицы *MUSIC*, которые предполагается отображать на форме. (Напомню, что поле *MUSIC.ID* относится к типу *AutoNumber*, т.е. заполняется системой автоматически, поэтому пользователи не должны его видеть, а тем более — изменять.) С помощью кнопки с изображением стрелки вправо (рис. 19.5) перенесите из списка Доступные поля (*Available Fields*)

в список Выбранные поля (Selected Fields) все наименования полей таблицы MUSIC, за исключением ID.

5. Щелкните на кнопке Далее.
6. Во втором диалоговом окне мастера укажите способ **размещения** данных на форме. Мы выберем опцию В один столбец (Columnar), как показано на рис. 19.6.
7. Следующие окна мастера позволяют задать стиль представления данных с использованием различных визуальных эффектов и указать имя формы. После того как вся информация, необходимая для построения формы, задана, щелкните на кнопке Готово.

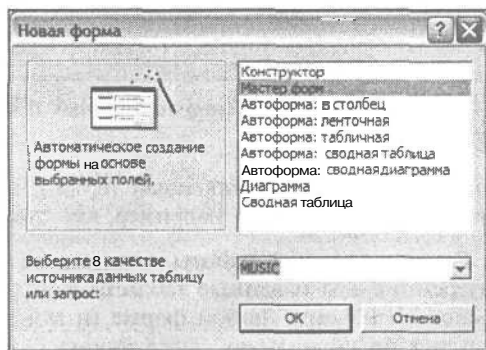


Рис. 19.4. Мастер поможет построить форму для отображения данных таблицы MUSIC

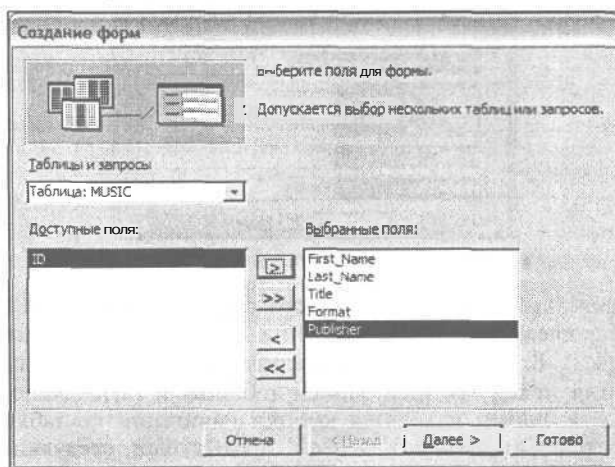


Рис. 19.5. Выберите те поля таблицы, которые хотите видеть на создаваемой форме

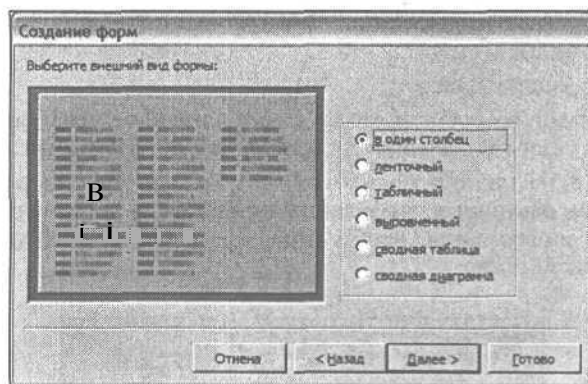


Рис. 19.6. Выбор опции размещения данных в один столбец

#### Новый термин

**Управляющий элемент** — это обобщенный термин, служащий для обозначения графических объектов, таких, например, как TextBox, Label или Form.

Мои поздравления! Результат нашей работы показан на рис. 19.7. Полученная форма, включающая внутренний код и данные таблицы MUSIC, — полноценное приложение, хотя и небольшое по объему. Любой форме (и MUSIC в том числе) можно присвоить признак открытия по умолчанию, тогда форма будет автоматически отображаться при каждом обращении к базе данных, в которой она содержится. Мастер создает форму с необходимыми управляющими элементами, позволяющими манипулировать данными соответствующей таблицы.

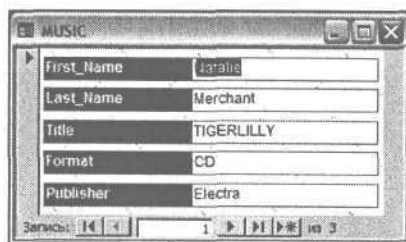


Рис. 19.7. Форма, построенная мастером

На рис. 19.8 представлены составные части формы. Управляющие элементы, размещенные в столбце слева, — это объекты типа Label, которые задают наименования полей таблицы MUSIC. В правом столбце располагаются объекты типа TextBox, отображающие значения полей текущей записи таблицы и позволяющие их редактировать. В нижней части формы находятся кнопки навигации по таблице. Они предоставляют возможность перемещения к первой, предыдущей, следующей, последней записям таблицы, а также позволяют добавлять новую запись.

Используя элементы меню Вид окна Access, вы можете переключаться в режимы конструктора, отображения формы как таковой и структуры таблицы. Режим конструктора (Вид⇒Конструктор) позволяет настроить параметры формы. Команда Вид⇒Режим формы приводит к открытию рабочего окна формы, пригодного для просмотра/редактирования данных, а результат выполнения команды Вид⇒Режим таблицы аналогичен открытию таблицы в обычном режиме — за исключением, что в данном случае будут видны только те столбцы, которые были выбраны на этапе проектирования формы.

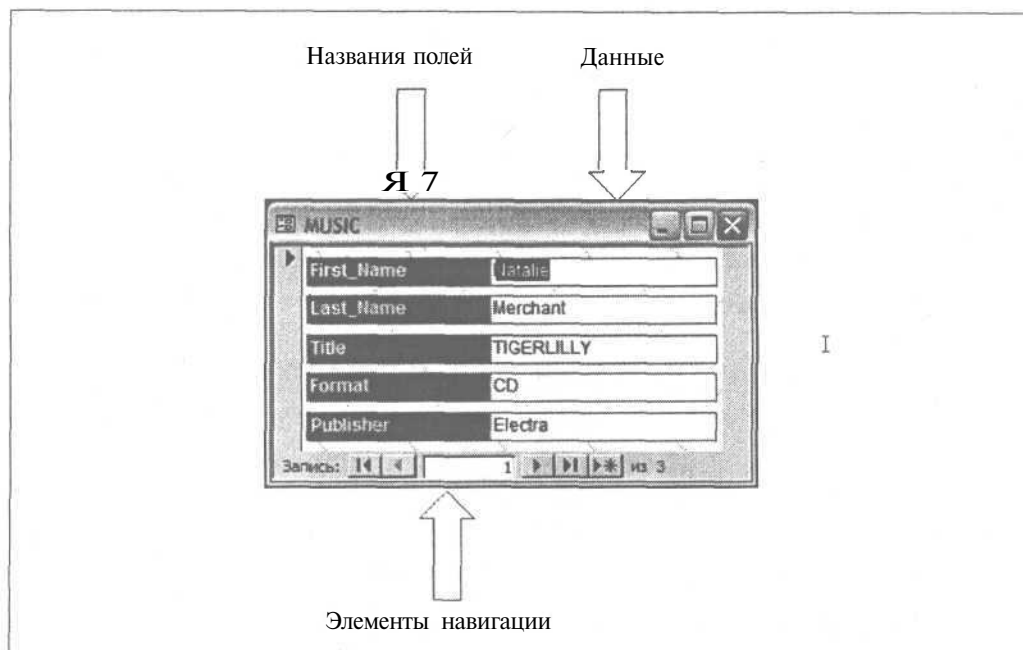


Рис. 19.8. Макет формы, созданной с помощью мастера

## Управление данными с помощью команд меню

В режиме отображения формы меню Access пополняется новыми элементами, которые помогают манипулировать данными. В меню Правка (Edit) появляются элементы Удалить запись, Выделить запись, Выделить все записи, Перейти; изменяется поведение команды Правка⇒Найти. Вставку новой записи можно выполнить как щелчком на соответствующей кнопке окна формы (см. рис. 19.8), так и посредством команды Вставка⇒Новая запись. Меню Записи (Records), доступное в режимах просмотра Режим формы и Режим таблицы, предоставляет возможности наложения и снятия фильтров, задания команд сортировки, сохранения внесенных изменений и обновления содержимого таблицы в окне формы.

Все названные элементы меню, кроме нескольких, ссылаются на команды прямого действия, т.е. приводят к непосредственному выполнению операций, без задания пользователем дополнительных данных.

## Фильтрация данных

Меню Записи⇒Фильтр (Records⇒Filter) предлагает четыре варианта фильтрации записей таблицы. (Напомним — наложение фильтра равносильно введению в команду SELECT предложения WHERE, которое уточняет критерии отбора данных.) В вашем распоряжении имеются такие инструменты: Изменить фильтр (Filter by Form), Фильтр по выделенному (Filter by selection), Исключить выделенное (Filter excluding selection) и Расширенный фильтр (Advanced Filter/Sort).



## Режим Изменить фильтр

При выборе команды Изменить фильтр открывается диалоговое окно, имеющее почти такой же вид, как и окно формы. Данные, которые вы введете в его поля, будут трактоваться системой как предикаты предложения WHERE запроса SELECT. Например, чтобы отфильтровать записи таблицы MUSIC по значению поля LAST\_NAME, выполните следующие действия.

1. Откройте окно формы MUSIC и выберите в меню Access команду Записи⇒Фильтр⇒Изменить фильтр. Откроется диалоговое окно Фильтр (рис. 19.9).

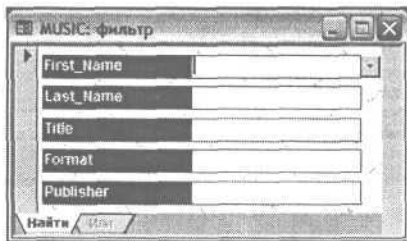


Рис. 19.9. Диалоговое окно Фильтр

2. Щелкните в поле ввода данных LAST\_NAME и введите выражение > "D". В данном случае мы предполагаем отобразить все записи таблицы MUSIC, кроме тех, в поле LAST\_NAME которых содержатся строки, начинающиеся с букв A, в или с.
3. Чтобы осуществить фильтрацию, выберите в строке меню команду Фильтр⇒Применить фильтр (Filter⇒Apply Filter).

Диалоговое окно Фильтр закроется, и система выполнит повторный запрос, уточняющий содержимое набора данных, которые отображаются формой. (В нашем примере в наборе останутся записи только о тех исполнителях, фамилии которых начинаются с букв D, E, F и т.д.) Чтобы снять фильтр и восстановить прежний состав набора данных, выберите в строке меню команду Записи⇒Удалить фильтр (Records⇒Remove Filter).

Наконец, пришло время обратить внимание на тот факт, что структура меню Access 2002 существенно зависит от текущего контекста, т.е. его состав и функции меняются в зависимости от осуществляемых пользователем операций. Одни элементы меню, имеющие отношение к конкретной ситуации, появляются, а другие исчезают. Например, в режиме просмотра формы в строке меню элемента Фильтр нет, а при открытии окна задания фильтра этот элемент уже присутствует.

## Режим Фильтр по выделенному

Команда Записи⇒Фильтр⇒Фильтр по выделенному доступна в режимах просмотра Режим формы и Режим таблицы. Отметьте содержимое любого из полей и выберите указанную выше команду меню. В результате будут отобраны только те записи, в одноименном поле которых содержатся данные, совпадающие с выделенными.

## Режим Исключить выделенное

Эта команда также доступна в режимах просмотра Режим формы и Режим таблицы, но ее результат противоположен предыдущему. Стоит выделить содержимое какого-либо поля и выбрать команду Записи⇒Фильтр⇒Исключить выделенное, и из набора данных будут изъяты те записи, в одноименном поле которых содержатся данные, совпадающие с выделенными.

## Режим Расширенный фильтр

Режим Расширенный фильтр позволяет определять более сложные выражения фильтрации, в которых значения полей могут сопоставляться с константами, содержанием других полей и результатами выполнения функций. Окно расширенного фильтра, показанное на рис. 19.10, во многом подобно (внешне и в отношении выполняемых функций) окну построения запросов.

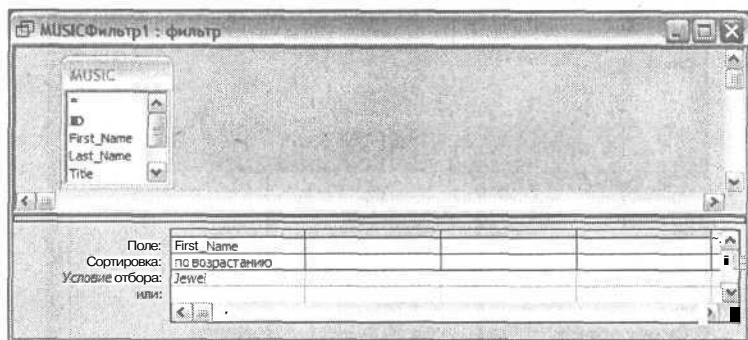


Рис. 19.10. Окно задания расширенного фильтра позволяет строить сложные выражения фильтрации, состоящие из нескольких предикатов

Чтобы созданный фильтр начал действовать, достаточно выбрать в строке меню команду **Фильтр⇒Применить фильтр**. Для построения выражений фильтрации, использующих подзапросы, обращения к другим таблицам и вызовы функций, применяются средства диалогового окна **Построитель выражений**. Чтобы воспользоваться ими, выполните следующие действия.

1. Откройте окно формы MUSIC и выберите в меню команду **Записи⇒Фильтр⇒Расширенный фильтр**.
2. Щелкните правой кнопкой мыши в пределах области определения критериев фильтрации, расположенной в нижней части окна задания расширенного фильтра.
3. Выберите в контекстном меню элемент **Построить** (рис. 19.11).

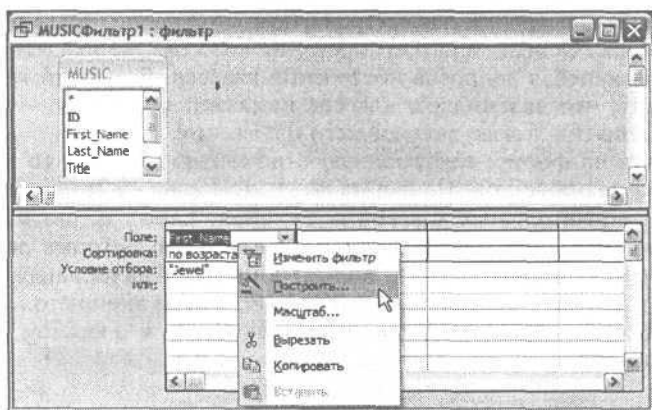


Рис. 19.11. Диалоговое окно **Построитель выражений** может быть легко вызвано из контекстного меню окна задания расширенного фильтра

Окно позволяет вводить условия отбора с клавиатуры либо использовать диалоговые средства — кнопки и списки. Возможные источники данных, на которые позволено ссылаться при построении условий фильтрации, таковы: Таблицы, Запросы, Формы, Отчеты, Функции, Константы, Операторы, Общие выражения — т.е. практически все объекты базы данных Access.

В некоторых случаях форму, построенную с помощью мастера, можно использовать сразу, не предпринимая дополнительных шагов. Но чаще требуется расширить ее функциональность и изменить некоторые свойства. В этом разделе рассказывается о возможных способах настройки параметров форм.

В главе "21-й час. Основы программирования классов" вы найдете много полезной информации, касающейся вопросов построения классов. В данный момент вам необходимо вспомнить, что экземпляры классов называют объектами — мы будем обращаться к этому понятию в ходе дальнейшего изложения.

Если взглянуть на форму, построенную с помощью мастера, то можно заметить, что она содержит ряд объектов. На форме MUSIC (см. рис. 19.7) присутствуют объекты трех наиболее употребительных классов — TextBox, Label и, разумеется, Form. Самое существенное различие между ними и объектами, о которых речь шла на прошлых занятиях (скажем, Recordset), состоит в наличии Визуальных свойств, т.е. дополнительных атрибутов, управляющих характеристиками внешнего вида.

## Часть VII. Проектирование интерфейса пользователя

# Знакомство с атрибутами визуальных объектов

*Класс* — это расширенный пользовательский тип данных (в роли пользователя в данном случае выступает программист). *Объектом* называется переменная, или экземпляр, класса. Например, выражение `Dim MyCollection As New Collection` определяет объект под названием `MyCollection`, служащий экземпляром класса `Collection`. Каждый объект некоторого класса хранит собственный набор атрибутов и способен к выполнению тех же функций, что и его родные "братья" — объекты того же класса.

Переменные, объявленные в составе класса, называются свойствами. Словом *свойство* в контексте лексики объектно-ориентированного программирования обозначаются *данные, служащие частью класса*. Речь уже шла о том, что объекты являются носителями "знаний" и "умений". "Знания" заключены во множестве свойств, а "умения" — в наборе реализованных в составе класса методов. В теории и практике объектно-ориентированного программирования *методами* называют процедуры и функции, принадлежащие классу.

Термины *свойство* и *метод* для вас не новы — мы неоднократно употребляли их на прошлых занятиях. К третьей группе атрибутов класса относятся *события*. Событие предполагает реакцию объекта на процессы, которые с ним происходят. Пользователь заполняет поле ввода текста или щелкает мышью на кнопке формы — это типичные примеры событий.

## Новый термин

*Сообщения* — это блоки данных, которыми операционная система Windows обменивается с каждой из прикладных программ, работающих в ее среде. Например, щелчок мышью в пределах формы приложения порождает сообщение, пересылаемое этому приложению.

## Новый термин

*Событие* — это термин, используемый для ссылки на факты приема и передачи приложением различных сообщений. Например, получение приложением сообщения о щелчке кнопкой мыши трактуется как событие.

## Новый термин

*Обработчик событий* — это процедура (функция), вызываемая в ответ на событие.

Когда вы нажимаете клавишу или щелкаете кнопкой мыши, первыми реагируют аппаратные устройства компьютера. Операционная система Windows преобразует физический сигнал в определенный блок информации, называемый сообщением. Сформировав сообщение, Windows определяет, какая из работающих в данный момент программ должна его получить, и отправляет сообщение по назначению. Блок кода приложения, принимающий сообщение, называют обработчиком события.

Напомним еще раз, что свойства — это данные, хранимые в составе объектов, методы — это функции и *процедуры*, объявленные внутри класса, а обработчики событий — это процедуры, взаимодействующие с Windows в ответ на получаемые приложением сообщения. Последний атрибут несколько более тонок. С событиями связаны собственные порции данных и кода. Процедура вообще и обработчик события в частности с точки зрения компьютера — это некоторый адрес, т.е. те же данные. Поэтому событие в составе объекта можно считать свойством, содержащим адрес процедуры обработчика.

Если не все понятно, не беспокойтесь. Визуальные объекты нам с вами использовать намного проще, чем их создавать. Надеемся, примеры, предлагаемые далее, помогут прояснить ситуацию.

## Свойства объектов

Access 2002 — это инструментальная среда визуального программирования. Иначе говоря, значительная часть работы по заданию характеристик поведения форм выполняется системой автоматически.

Каждый визуальный объект обладает определенными свойствами. Со свойствами формы можно ознакомиться, обратившись к средствам диалогового окна Форма (Form) (рис. 19.13), которое открывается командой Вид⇒Свойства (View⇒Properties).

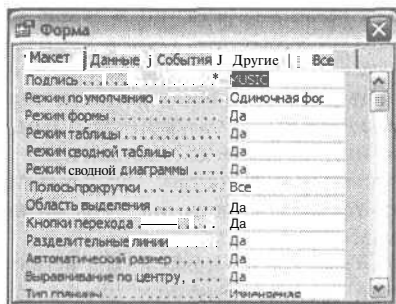


Рис. /Р. 13. Диалоговое окно свойств позволяет управлять свойствами формы

Свойства, перечисленные на вкладках диалогового окна Форма, — это данные. В ходе проектирования формы вы можете легко изменять содержимое тех или иных свойств, находя наименование свойства в левой половине списка и редактируя его значение в правой. Чтобы, например, изменить заголовок формы с *MUSIC* на *Music*, выполните следующие действия.

1. Откройте форму *MUSIC*.
2. Выберите команду Вид⇒Свойства.
3. Перейдите на вкладку Макет (Format) диалогового окна Форма.
4. Найдите свойство Подпись (Caption) (оно располагается в самом начале списка) и измените его значение с *MUSIC* на *Music*.

Немного попрактиковавшись, вы поймете, что все это довольно просто. Визуальная среда предоставляет возможности интерактивной работы — не нужно заниматься кодированием в привычном понимании этого слова. (При необходимости свойства объекта могут изменяться и с помощью строк кода — динамически, в ходе выполнения программы.)

Свойства, перечисленные в окне Форма, относятся к тому объекту, который находится в фокусе. Не закрывая окна Форма, щелкните, скажем, на поле ввода текста (фактически, на объекте типа TextBox) *FIRST\_NAME* формы *MUSIC* — и окно свойств Форма изменит свое название (теперь оно будет именоваться Поле: *FIRST\_NAME*), а также содержимое.

## Методы

Методы объектов вызываются с указанием наименования объекта, оператора точки (.) и названия метода. В главе “13-й час. Коллекции данных” речь шла об использовании объектов и их методов. Вспомните, скажем, что метод *Add* класса *Collection* вызывается посредством конструкции *ИмяОбъекта.Add* с передачей требуемых аргументов. (Строка 4 листинга 13.2, например, демонстрирует пример добавления в коллекцию элементов типа *String*.)

Аналогичным образом используются и другие методы классов — указывается имя объекта, сопровождаемое оператором точки и названием метода. Методы применяются точно так же, как и обычные процедуры. То же справедливо и в отношении методов визуальных объектов. Каждый объект класса Form, например, имеет в своем составе метод Repaint, позволяющий обновить изображение окна формы со всем его содержимым. Имя объекта формы MUSIC — Form\_MUSIC (оно хранится в свойстве Name). Для вызова метода Repaint вы могли бы написать:

```
Form_MUSIC.Repaint
```

Метод Repaint не требует задания аргументов. Во многих классах наименования и назначение методов одинаковы. Чтобы прояснить для себя детали использования конкретного метода, обращайтесь к материалам оперативной справочной системы.

## Программирование обработчиков событий

Процедуры обработчиков событий сохраняются в модулях, наряду со всем остальным кодом приложения. Прежде чем приступить к созданию обработчиков событий для формы, следует установить признак того, что форма может иметь код. (Ранее отмечалось, что форма способна выполнять некоторые базовые операции даже при отсутствии собственного кода.)



Чтобы получить информацию об определенном свойстве объекта, откройте диалоговое окно свойств, укажите требуемое свойство и нажмите клавишу <F1>. Отобразится окно контекстной оперативной справки, содержащее исчерпывающие данные о свойстве, ссылки на родственные темы и примеры кода.

Чтобы установить для формы признак наличия кода, выполните следующие действия.

1. Откройте окно формы, выберите в строке меню команду Вид⇒Конструктор (View⇒Design View), а затем щелкните на темном квадрате в левом верхнем углу окна формы, чтобы передать фокус объекту формы (рис. 19.14).

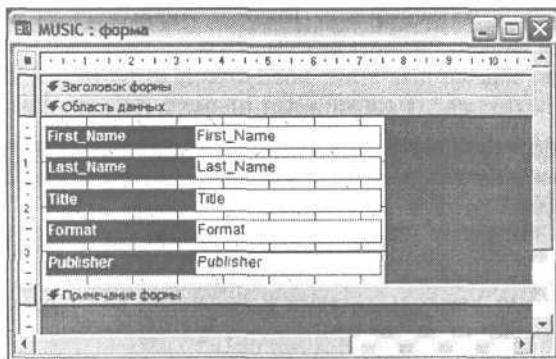


Рис. 19.14. Выберите объект формы в целом, щелкнув на темном квадрате в левом верхнем углу окна формы

2. Выберите команду меню Вид⇒Свойства, чтобы открыть диалоговое окно свойств Форма.
3. Перейдите на вкладку Другие (Other).

4. Найдите в нижней части списка свойств элемент Наличие модуля (Has Module) и установите для него значение Да (Yes), воспользовавшись раскрывающимся списком (рис. 19.15).

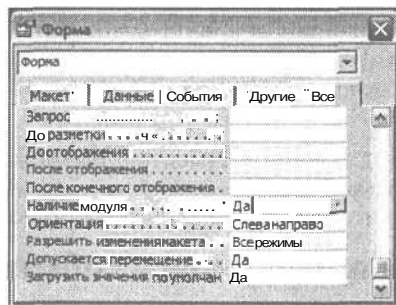


Рис. 19.15. Прежде чем снабдить форму собственным кодом, установите для свойства *Наличие модуля* значение *Да*

Теперь вы сможете создавать обработчики событий и "привязывать" их к форме.

В свойствах событий формы находят отражение все процедуры обработки, которые вы создали. Чтобы написать обработчик конкретного события, отыщите название события на вкладке События диалогового окна свойств, щелкните на соответствующем элементе списка, а затем — на кнопке справа от него, помеченной пиктограммой многоточия. Далее приводится пример динамического изменения заголовка формы. Выполните следующие действия.

1. Щелкните на форме, чтобы передать ей фокус ввода.
2. Выберите в строке меню команду Вид⇒Свойства с целью открыть диалоговое окно свойствФорма.
3. Переидите на вкладку События (Event).
4. Щелкните на элементе Загрузка (On Load) списка событий.
5. Щелкните на кнопке Построитель (Build) (с изображением многоточия). Откроется окно редактора модуля текущей формы.
6. Добавьте в тело обработчика `Form_Load` строку `Form_MUSIC.Caption = "Моя музыкальная коллекция"`. Полный текст примера показан в листинге 19.1.

Листинг 19.1.Пример обработчика, динамически изменяющего заголовок формы

```
1: Private Sub Form_Load( )
2:   Form_MUSIC.Caption = "Моя музыкальная коллекция"
3: End Sub
```

Совершенно неважно, насколько прост или сложен код, который вы напишете, -- процесс создания обработчиков единый.

## Размещение управляющих элементов на форме

Чтобы разместить на форме новые управляющие элементы, обратитесь к средствам панели инструментов Панель элементов, которая может быть открыта командой меню Вид⇒Панель элементов (рис. 19.16).



*Рис. 19.16. Панель элементов позволяет быстро пополнить форму необходимыми стандартными интерфейсными элементами — объектами Поле, Надпись и т.п.*

Размещение интерфейсного элемента в форме называют *рисованием*. Чтобы нарисовать элемент, щелкните на соответствующей ему кнопке Панели элементов, а затем в нужном месте формы. Положение, форму и размеры элемента в форме легко изменить с помощью мыши. Диалоговое окно свойств предоставляет доступ ко всем атрибутам визуального объекта. Чтобы, например, разместить на форме MUSIC объект, содержащий растровое графическое изображение, выполните следующие действия.

1. Откройте форму MUSIC в режиме конструктора.
2. С помощью команды Вид⇒Панель элементов откройте одноименную панель инструментов.
3. Щелкните на кнопке Рисунок (Image) (все кнопки панели снабжены всплывающими подсказками, облегчающими поиск нужного элемента).
4. Переместите указатель мыши в нужное место формы и щелкните левой кнопкой мыши.
5. Откроется диалоговое окно Выбор рисунка (Insert Picture), позволяющее указать требуемый графический файл.
6. Выберите файл, содержимое которого должно отображаться в границах вновь построенного объекта, и щелкните на кнопке ОК.

Приведенная инструкция — с незначительными поправками — пригодна для описания действий по размещению на форме объектов всех других видов. Программисту, использующему VBA, ныне доступны тысячи разнообразных компонентов, хотя Панель элементов представляет всего 18 "самых" стандартных. Каждый день в мире создаются новые полезные визуальные компоненты.

## Выбор дополнительных элементов управления

Панель элементов позволяет обратиться к дополнительным визуальным компонентам, зарегистрированным в вашей копии операционной системы Windows. Для этого щелкните на кнопке Другие элементы (More Controls), помеченной пиктограммой в виде пересекающихся молотка и гаечного ключа.

Существует настолько широкое множество дополнительных компонентов, что для исчерпывающего описания их свойств, методов и событий требуются десятки тысяч страниц технической документации. Обращаясь к "нестандартным" средствам, полагайтесь на фирменные руководства и системы оперативной справки, содержащие всю необходимую информацию.

## Использование невизуальных компонентов

В составе Панели элементов имеются ссылки только на визуальные интерфейсные компоненты. Впрочем, в своем приложении вы можете использовать и невизуальные элементы. Например, элементы библиотеки Microsoft ActiveX Data Objects 2.5 Library не представлены на панели элементов в виде пиктограмм.

Чтобы снабдить приложение дополнительными возможностями, выберите в окне редактора Visual Basic команду Tools⇒References и в одноименном диалоговом окне установите флажки тех интерфейсных библиотек, средствами которых хотите воспользоваться. Нетрудно заметить, например, что библиотеки Visual Basic for Applications,



Microsoft Access 10.0 Object Library, OLE Automation, Microsoft ActiveX Data Objects 2.5 Library выбраны по умолчанию.

В диалоговом окне References перечислены названия файлов динамических библиотек, компонентов ActiveX и серверов OLE Automation, зафиксированных в базе данных реестра Windows. Эти библиотеки располагаются на жестком диске компьютера в виде файлов с расширениями *.DLL*, *.OCX* или *.EXE*. Каждая из библиотек содержит классы, характеризующиеся наборами свойств и методов, а некоторые обеспечивают и реакцию на определенные события. Все классы, как правило, располагают описаниями в виде файлов оперативной справки — ищите и изучайте.

## Создание программного кода формы

После присваивания свойству Наличие модуля формы значения Да Access создает модуль, связанный с формой. Подобные модули не отображаются в списке Модули окна базы данных. Чтобы получить доступ к модулю формы, откройте форму, а затем выберите в строке меню команду Вид⇒Программа (View⇒Code).

Окно редактора модуля формы используется для написания кода, расширяющего возможности формы. Разновидностями кода, который может присутствовать в модуле формы, являются обработчики событий и обычные процедуры. Код формы создается точно так же, как и любой другой. Следует особо отметить единственное обстоятельство — код модуля формы имеет отношение только к тем объектам, которые принадлежат форме.

Из тела обработчика можно вызвать любые процедуры. Рекомендуем последовательно придерживаться такого правила: код обработчика события целесообразно размещать в отдельной процедуре или функции, а не в самом обработчике. Например, листинг 19.1 соответствует ситуации, когда заголовок формы изменяется в обработчике события Load. Это не очень удачный вариант действий, поэтому следует внести небольшие изменения, которые помогут сделать код более наглядным и дадут возможность его повторного использования. Листинг 19.2 содержит исправленную версию примера, представленного в листинге 19.1.

### Листинг 19.2. Исправленный вариант примера листинга 19.1

```
1: Private Sub SetCaption( )
2:   Caption = "Моя музыкальная коллекция"
3: End Sub
4: Private Sub Form_Load( )
5:   SetCaption
6: End Sub
```

#### Анализ

Внешне код листинга 19.2 выглядит более сложным, нежели в предыдущем примере. Однако новая процедура SetCaption, которая должна изменить заголовок формы, предельно проста и (что немаловажно) удачно названа. При необходимости она впоследствии может быть вызвана и из других частей программы. Следует отметить, что код вовсе не нуждается в дополнительных комментариях.

Конечно, листинг 19.2 решает довольно упрощенную задачу. На практике приходится иметь дело с более сложными проблемами, и именно в таких случаях рассмотренный подход проявляет всю свою плодотворность и эффективность. Код, который не имеет прямого отношения к управлению объектами формы, лучше сохранять в отдельном модуле. Избегайте построения чрезмерно обширных обработчиков событий и не "привязывайте" к форме слишком большие фрагменты кода. Размышляя над тем, где целесо-

образно разместить тот или иной блок кода — в модуле формы или в модуле общего назначения, — продумайте, понадобится этот блок только в контексте текущей формы или есть надежда на его более широкое применение. Впрочем, впоследствии вы в любом случае сможете переместить код из одного модуля в другой, и сделать это будет тем легче, чем более краткими и простыми окажутся ваши процедуры и функции.

## Расширенные возможности редактирования

Access 2002 имеет дополнительные комбинации клавиш, которые помогут при редактировании форм, а именно:

- клавиши <F7> и <Shift+F7> позволяют переключаться между режимами объекта и кода активной формы соответственно;
- клавиша <F4> вызывает окно свойств;
- клавиши <Ctrl+<> и <Ctrl+>> выполняют циклическое переключение между доступными для активного объекта режимами. Например, при работе с формой посредством этих клавиш легко переключиться из режима конструктора в режим формы. Что еще удобнее, данные комбинации клавиш позволяют переключаться между всеми режимами выбранного объекта. Например, при работе с таблицами они производят переключение между режимами Конструктор (Design View), Режим таблицы (Table View), Сводная таблица (PivotTable) и Сводная диаграмма (PivotChart).

Немного практики — и вы поймете, насколько удобно пользоваться этими горячими клавишами.

## Расширенные возможности форм и отчетов

В Access 2002 формы и отчеты имеют новые свойства, методы и события. По сравнению с предыдущей версией программы, новшества небольшие, но весьма полезные.

### Новые свойства форм и отчетов

В Access 2002 меню Сервис⇒Параметры запуска (Tools⇒Startup) имеет опцию Значок приложения (Application Icon). Вы можете использовать для форм и отчетов собственные пиктограммы.

Новые свойства появились и у объекта Report. Отчеты могут отображаться в модальных окнах. (*Модальным* называется окно, находящееся в фокусе до тех пор, пока его не закроют. Прежде чем выполнять иные действия, пользователь должен закрыть окно с отчетом.) Кроме того, можно изменять свойства Тип границы (BorderStyle), Автоматический размер (AutoResize), Выравнивание по центру (AutoCenter), Кнопки размеров окна (MinMaxButtons), Кнопка закрытия (CloseButton) и Кнопка оконного меню (ControlBox) объекта Report.

Чтобы изменить свойства объекта Report, выберите его из списка объектов. Откройте отчет, который вы хотите изменить в режиме конструктора, и нажмите клавишу <F4> с целью вызвать окно свойств. Большинство указанных выше свойств имеют значения Да/Нет (Yes/No). Чтобы кнопка оконного меню (рис. 19.17) не работала, установите значение соответствующего свойства равным Нет.

О создании модальных окон см. раздел "Построение отчета" далее в этой главе.

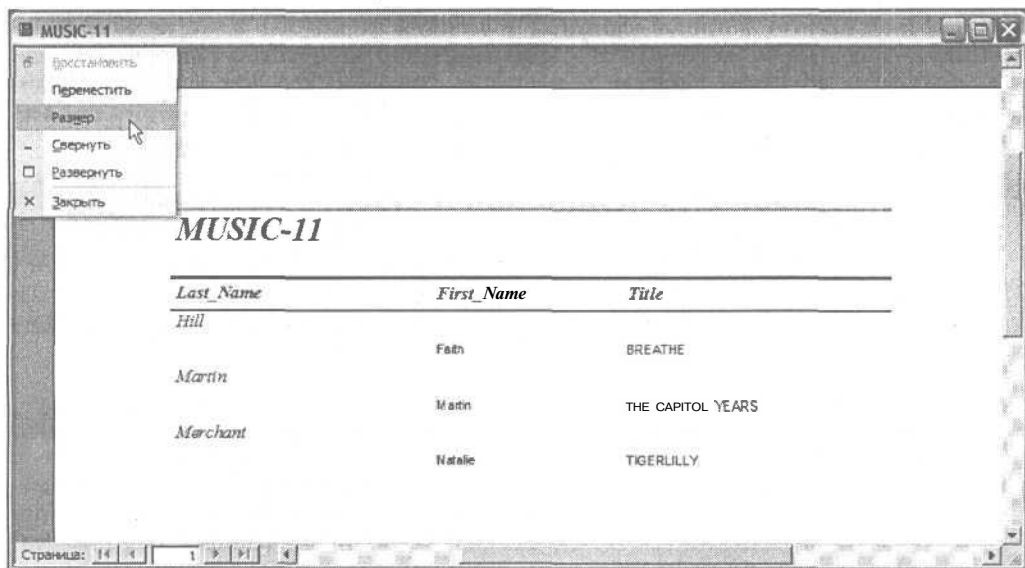


Рис. 19.17. Выбрано меню Размер

## Новые события форм

В Access 2002 имеются новые события OnUndo и OnRecordExit на уровне формы, а также OnDirty и OnUndo на уровне элемента управления. Событие OnUndo формы вызывается, когда пользователь в Access нажимает клавишу <Esc> или выбирает команду меню Правка⇒Отменить (Edit⇒Undo). Событие OnUndo на уровне элемента управления происходит при нажатии клавиши <Esc> во время редактирования имеющихся в нем значений. Событие OnDirty происходит, когда изменяется содержимое формы или поля со списком, а пользователь переходит к другому элементу управления посредством нажатия клавиши <Tab>. Событие OnRecordExit происходит при перемещении к другой записи.

Каждое из этих событий имеет один целочисленный аргумент Cancel. Если Cancel равен True, любое событие — OnUndo, OnDirty, OnRecordExit — не происходит. Например, если обработчик события Form\_OnDirty устанавливает Cancel = True, вы не сможете изменить данные в форме.

## Новые методы форм и отчетов

Объекты Form и Record имеют теперь новый метод Move, принимающий аргументы Left, Top, Width и Height, которые определяют положение и размер объектов. Вызывается метод следующим образом:

```
Call Me.Move (0, 0, 2000, 1500)
```

Новые значения параметров Left и Top равны 0, ширина (width) — 2000, а высота (Height) — 1500 твипсов (1 твипс равен 1/1440 дюйма).

## Форма подотчета

Теперь при добавлении подформы или подотчета их можно увидеть в отдельном окне режима конструктора. Находясь в этом режиме, выберите подотчет или подформу, щелкните правой кнопкой мыши и в контекстном меню выберите команду Вид⇒В отдельном окне (View⇒Subform in own window).

# Тестирование формы

Если форме соответствует программный код, то он должен быть протестирован. Получив из "рук" мастера готовую форму, вы можете, используя режим просмотра формы (команда Вид⇒Режим формы), быстро оценить, что еще необходимо сделать — добавить или удалить какие-либо объекты, отредактировать их свойства либо, скажем, наложить условия фильтрации набора данных. Если вы решили добавить в модуль формы обработчик событий или другие процедуры, их необходимо протестировать с использованием методов и приемов, о которых было рассказано в главах "17-й час. Отладка кода" и "18-й час. Обработка ошибок во время выполнения программы". В противном случае вы не будете иметь никаких гарантий правильности и надежности разработанных программных решений.

## Построение отчета

Отчеты, как и формы, можно построить с помощью мастеров. Помните, что существующий отчет — это объект, со своими собственными свойствами, методами и событиями. В Access всеми свойствами объекта можно управлять во время разработки из окна свойств, а также во время исполнения с помощью кода. Чтобы использовать объект, например Report, ознакомьтесь с его свойствами, методами и событиями и напишите код для управления отчетом так, чтобы добиться ожидаемого результата.

Откройте Access и из списка Объекты выберите Отчеты (Reports), дважды щелкните на ссылке Создание отчета с помощью мастера (Create report by using wizard). Появится мастер отчетов, который работает аналогично мастеру форм. В базе данных MUSIC выберите одноименную таблицу. Задайте все нужные поля и щелкните на кнопке Готово (Finish). Вы получите образец отчета, представленный на рис. 19.18 в режиме предварительного просмотра (Print Preview). Чтобы изменить отчет, перейдите в режим конструктора, выбрав команду Вид⇒Конструктор. Отчет в режиме конструктора похож на форму. Настройте отчет, изменив его свойства и создав код, влияющий на его поведение.

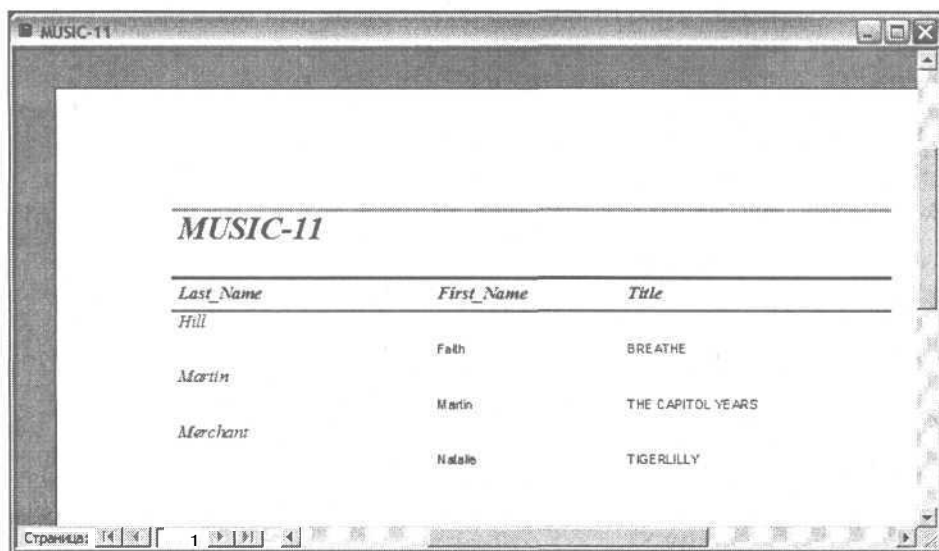


Рис. 19.18. Макет отчета, созданный с помощью мастера

# Использование объекта Printer

Объект Printer был добавлен для облегчения управления свойствами печати. Он позволяет настраивать порт принтера, поля страницы, число копий, ориентацию и пр. Чтобы напечатать отчет, созданный в предыдущем разделе на базе таблицы MUSIC, используйте код листинга 19.3.

## Листинг 19.3. Открытие и печать отчета в альбомной ориентации с использованием объектов Printer и DoCmd

```
1: Sub PrintReport ()
2:   Dim Report As New Report_All_MUSIC
3:   Printer.Orientation = acPRORLandscape
4:   Printer.Copies = 1
5:   Call DoCmd.OpenReport("All_MUSIC", acViewPreview)
6:   DoCmd.PrintOut
7: End Sub
```

Добавьте этот код в модуль базы данных MUSIC и протестируйте пошагово, нажимая клавишу <F8>. Листинг 19.3 демонстрирует пример кода, необходимого для программного открытия отчета и его печати. Новый объект Printer используется для управления свойствами принтера, а объект DoCmd мы использовали многократно в течение нескольких последних занятий. Чтобы добавить возможности печати отчета в форму, поместите в нее кнопку, например с названием Печать, и используйте код листинга 19.3 для печати по щелчку пользователя на кнопке.

## Добавление элементов меню в Access для печати отчета

Access позволяет добавлять пользовательские меню в базы данных. Чтобы добавить элемент в меню Сервис, создайте макрос, печатающий отчет, и поместите его в меню. Чтобы добавить элемент меню Печать отчета, отображающий отчет MUSIC, выполните следующие действия.

1. Преобразуйте процедуру листинга 19.3 в функцию, возвращающую значение типа Variant. Остальной код не изменяйте.
2. В списке Объекты выделите элемент Макросы (Macros) и щелкните на кнопке Создать (New) для создания нового макроса.
3. Выберите действие Запуск кода (Run Code). В области аргумента Имя функции (Function Name) щелкните на символе многоточия, чтобы вызвать Построитель выражений (Expression Builder).
4. В диалоговом окне Построитель выражений (рис. 19.9) выберите функцию PrintReport. Сохраните макрос под именем PrintReport.
5. В окне программы Access выделите команду Сервис⇒Настройка (Tools⇒Customize), затем перейдите на вкладку Команды (Commands).
6. В списке Категории (Categories) выберите Все макросы (All Macros) — в списке Команды (Commands) появится макрос PrintReport. Перетащите его поверх меню Сервис, при этом указанное меню откроется.
7. Не отпуская кнопку мыши, перенесите макрос PrintReport в конец списка команд меню Сервис (рис. 19.20).

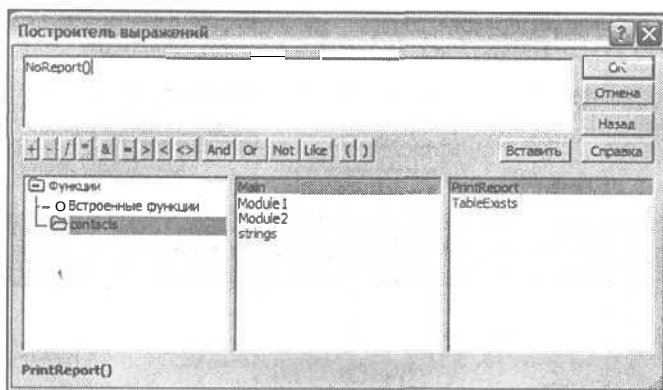


Рис. 19.19. 5 диалоговом окне Построитель выражений выберите нужную функцию

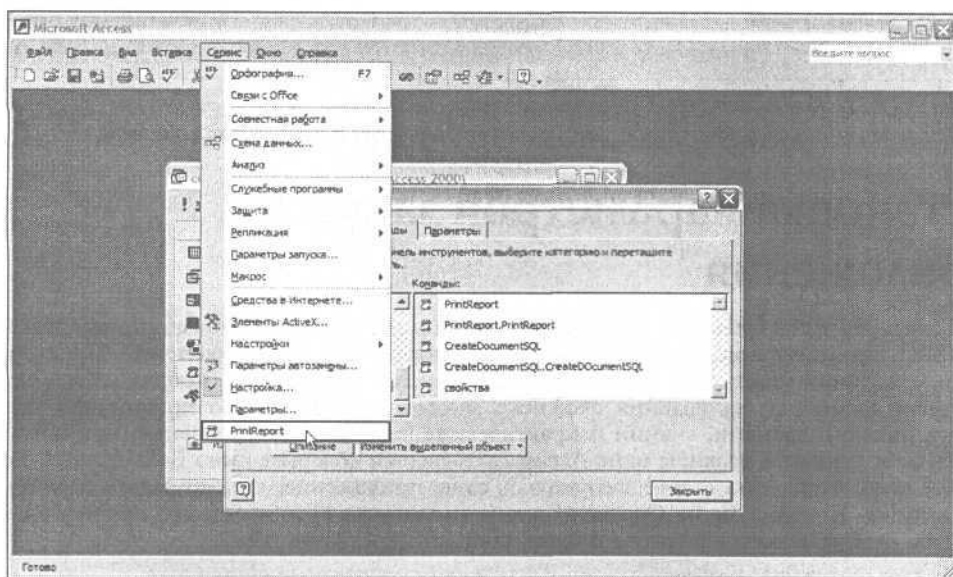


Рис. 19.20. Добавление пользовательской команды в меню Сервис методом перетаскивания

8. Измените название элемента меню на Печать отчета, для этого введите данное название в поле Имя (Name) или измените свойство Caption в диалоговом окне Параметры элемента управления (Tools Control Properties).

По завершении этой процедуры отчет будет полностью интегрирован в базу данных MUSIC, поэтому его легко использовать как любую другую функцию Access. Для добавления, удаления и изменения пользовательских элементов управления воспользуйтесь диалоговым окном Настройка. На рис. 19.21 показано, как изменять текст на кнопке пиктограммы, добавлять в меню гиперссылки, пользуясь раскрывающимися меню. Диалоговое окно Настройка позволяет добавлять новые меню и панели инструментов, а также изменять уже существующие.

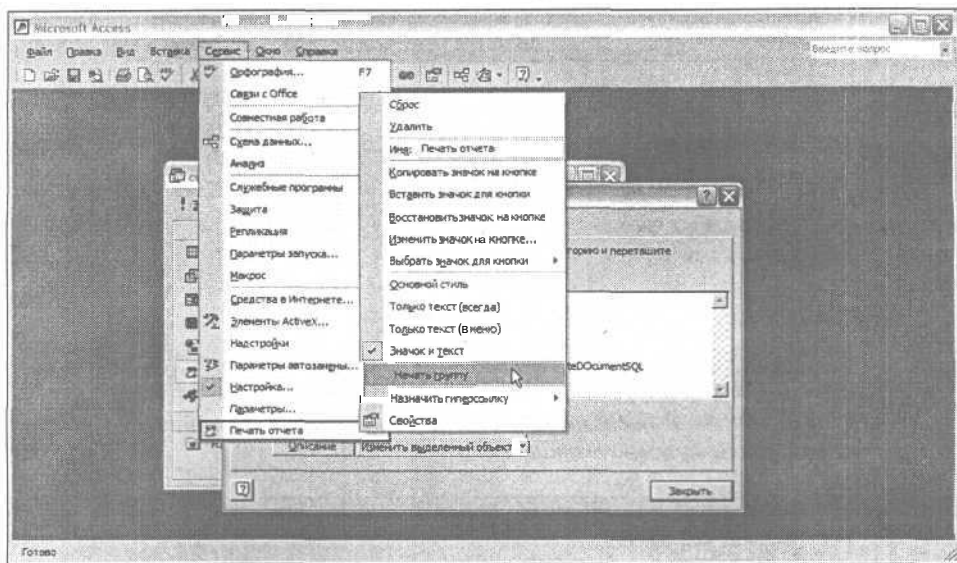


Рис. 19.21. Когда открыто диалоговое окно *Настройка*, можно изменять элементы меню и панелей инструментов, пользуясь опциями раскрывающихся меню

## Установка параметров запуска приложения

Теперь вы ознакомлены с основными аспектами проектирования программ для Access 2002, предполагающих интерактивное взаимодействие с пользователем. Принципиальных различий между процессами создания простой прогаммы, отображающей единственную форму, и составления сложного многофункционального приложения нет — дело в наличии времени, знаний и практических навыков. По умолчанию при обращении к базе данных в главном окне Access открывается дочернее окно *База данных*. Если необходимо, чтобы при старте загружалось ваше приложение, следует задать параметры его запуска. Команда меню **Сервис**⇒**Параметры запуска** (**Tools**⇒**Startup**) приводит к открытию одноименного диалогового окна, показанного на рис. 19.22.

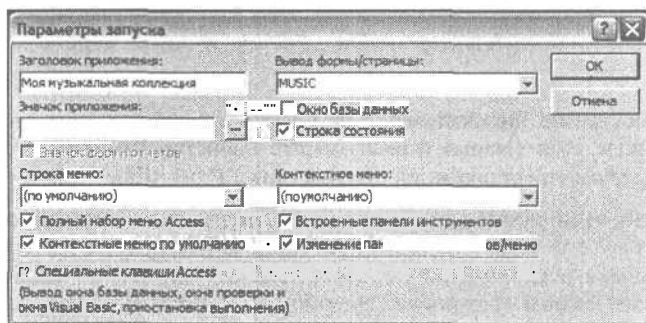


Рис. 19.22. Диалоговое окно *Параметры запуска* позволяет указать форму или блок кода, которые будут отображаться и выполняться при открытии текущей базы данных

Чтобы при открытии базы данных отображалось окно базы данных Моя музыкальная коллекция и выполнялся ее код, необходимо осуществить настройку с помощью средств диалогового окна Параметры запуска.

1. Введите в поле Заголовок приложения (Application Title) строку Моя музыкальная коллекция.
2. В раскрывающемся списке Вывод формы/страницы (Display Form/Page) выберите ссылку на форму MUSIC.
3. Снимите флажок Окно базы данных (Display Database window). Остальные флажки в диалоговом окне должны быть установлены.
4. Щелкните на кнопке ОК.

При очередном обращении к базе данных будет автоматически открыта форма с заголовком Моя музыкальная коллекция и выполнены соответствующие фрагменты кода модуля формы, если таковые созданы.

Если вы приобрели дополнительный пакет разработчика Developers Tools for Office XP, то сможете распространять приложения для Access вместе с соответствующими системными динамическими библиотеками, исключаящими необходимость установки на компьютерах пользователей полнофункциональных версий Access 2002.

## Резюме

Большинство приложений для Windows снабжено графическими пользовательскими интерфейсами. Access 2002 предоставляет средства создания баз данных, блоков кода и визуальных форм в единой среде. Одним словом, Access содержит все необходимое для построения полноценных Windows-приложений, предусматривающих возможности обработки информации из баз данных.

Формы и размещаемые на них управляющие элементы — это визуальные компоненты, допускающие настройку свойств как на этапе проектирования, так и во время выполнения программы. Вы ознакомились с тремя типами атрибутов, присущих визуальным компонентам. С помощью свойств описываются внешние признаки объектов, методы дают возможность управления их поведением, а обработчики событий выполняют прием сообщений, посланных приложению операционной системой Windows. Windows — это система, управляемая событиями. Автор приложения предусматривает реакцию на определенные события с помощью специальных процедур-обработчиков.

Панель инструментов Панель элементов содержит ссылки на ряд общеупотребительных стандартных управляющих элементов. С ее помощью вы можете построить форму с нуля либо подправить результат, полученный при использовании мастера создания форм. Чтобы разместить на форме любой из объектов, нужно щелкнуть на соответствующей кнопке Панели элементов, а затем указать положение объекта в пределах формы. После этого вам, возможно, придется настроить некоторые свойства объекта или создать обработчики соответствующих событий (только тех, на которые, по вашему мнению, следует реагировать). Все эти функции выполняются с помощью диалогового окна свойств.

Задача проектирования пользовательского интерфейса сводится, по существу, к "рисованию" управляющих элементов на форме и обеспечению реакции на события с помощью обработчиков. Построением интерфейса, однако, проблема далеко не исчерпывается — вы должны предложить качественное и надежное решение самой задачи. Прочтите приведенные ниже разделы "Вопросы и ответы" и "Задания", чтобы еще раз повторить и закрепить в памяти пройденный материал.



# Вопросы и ответы

**Вопрос.** Каким образом можно "привязать" к форме тот или иной источник данных?

**Ответ.** Откройте диалоговое окно свойств Форма, перейдите на вкладку Данные (Data), найдите свойство Источник записей (Record Source) и выберите в качестве его значения ссылку на любой допустимый источник данных — таблицу, запрос и т.п.

**Вопрос.** Можно ли задать определенный источник данных для некоторого объекта, принадлежащего форме?

**Ответ.** Многие объекты, подобные компонентам TextBox, обладают свойством Данные, которое ссылается на имя поля в "главном" источнике данных формы (см. предыдущий вопрос). Объект "связывается" с этим полем и получает возможность считывать информацию из поля и сохранять ее в базе данных.

**Вопрос.** Можно ли ссылаться из формы на несколько таблиц базы данных?

**Ответ.** Да. Эта возможность реализуется при выполнении п.1 процедуры создания формы с помощью мастера. Чтобы предоставить мастеру информацию о том, как следует отображать данные, вы должны описать взаимосвязи таблиц с помощью средств окна Схема данных (Relationships), вызываемого командой Сервис⇒Схемаданных.

**Вопрос.** Допускается ли возможность распространения приложений для Access без использования самой среды Access 2002?

**Ответ.** Да. Дополнительный пакет разработчика Developers Tools for Office XP, поставляемый отдельно, позволяет включать в состав приложения системные динамические библиотеки, обеспечивающие возможность автономного использования приложения вне среды Access 2002.

## Задания

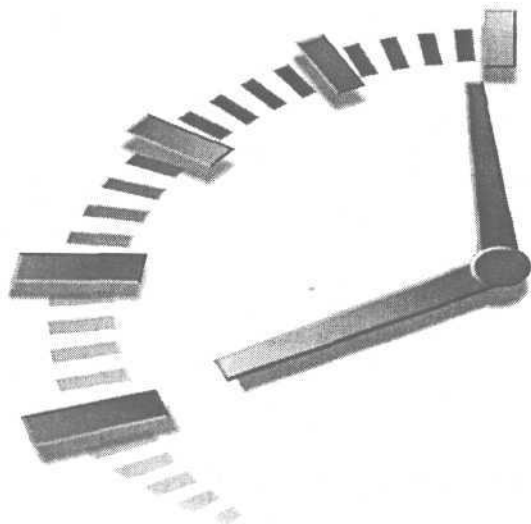
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Каким общим термином обозначаются данные, принадлежащие объекту класса?
2. При обращении к методу класса следует указывать имя объекта. Верно ли это?
3. Что такое событие в контексте лексики объектно-ориентированного программирования?
4. Что представляет собой обработчик события?
5. Каким образом можно разместить в форме новый управляющий элемент?

## Упражнения

1. Опишите процедуру установления взаимосвязей между таблицами MUSIC и TRACKS, созданными на 15-м занятии.
2. Используя схему данных, построенную при выполнении предыдущего упражнения, создайте с помощью мастера новую форму, представляющую информацию из двух таблиц.
3. Сохраните созданную форму (см. предшествующее упражнение) и обеспечьте ее автоматическое открытие при последующих обращениях к текущей базе данных.



## 20-й час

### Как связывать Web-страницы с базами данных

Инструментальные средства программирования развиваются настолько же быстро, как и индустрия программирования в целом. Сегодня в центре внимания находятся проблемы разработки и применения инструментов, обеспечивающих возможности реализации программных решений в среде World Wide Web. Если вы следите за рынком ценных бумаг, то наверняка обратили внимание, какими невероятными темпами растут котировки акций компаний, работающих в сфере электронной коммерции (e-commerce). Интерес к этой области заключается как в получении прямых прибылей от торговли акциями, так и в поиске сугубо профессиональной выгоды от знакомства с самыми передовыми Web-решениями. Программирование для среды Web — сегодня самая горячая тема в индустрии информационных технологий, нескончаемый сериал, труднопредсказуемый сюжет которого закручивается день ото дня.

До середины 90-х в центре всеобщего внимания были вопросы создания Windows-приложений, обладающих графическим интерфейсом пользователя. Но с недавних пор интересы и потенциал информационной индустрии сместились в сторону Web-программирования. Технологии программирования для Web предполагают создание распределенных приложений "тонких" клиентов, интерфейс которых, построенный, скажем, на языке HTML, отображается в окне броузера. Windows-программа представляет собой набор откомпилированных или интерпретируемых графических оконных интерфейсов, а Web-приложение содержит гиперссылки на Web-страницы. Web-страница состоит преимущественно из текста на языке HTML. На Web-сервере, вероятнее всего, выполняется еще и программный код других видов. Web-приложения носят название "тонких" клиентов потому, что в окне броузера отображаются только элементы внешнего интерфейса Web-страницы, а ее внутренний код хранится и исполняется большей частью на сервере. Мощь и привлекательность приложений Web обусловлены несколькими причинами. Одна из самых очевидных заключается в том, что разработчикам достаточно исправить или дополнить программный код на сервере, и внесенные изменения сразу становятся доступны всем пользователям Web-сайта. Стив Балмер (Steve Balmer), президент компании Microsoft, предрекает, что в самом

ближайшем будущем большинство интерактивных компьютерных приложений обретет форму "тонких" Web-клиентов.

Microsoft Access 2002 поддерживает средства разработки "тонких" клиентских приложений, основанных на стандарте *страницы доступа к данным* (Data Access Page). Страница доступа к данным представляет собой Web-страницу, которую пользователь может просматривать, пополнять и исправлять в среде Internet (intranet) или пересылать по электронной почте. Страницы доступа к данным разрабатываются примерно так же, как и формы Access, поэтому приобретенные ранее знания вам обязательно пригодятся. Внутренний HTML-код страницы генерируется с помощью мастеров и может редактироваться в среде Access 2002.

Основные темы занятия.

- Internet и intranet.
- Создание страниц доступа к данным с помощью мастеров.
- Индивидуальная настройка страниц в среде Access 2002.

## Internet и intranet

*Internet* — это глобальная компьютерная сеть. Компьютеры подключаются к Internet с помощью программно-технических средств, предоставляемых провайдерами услуг Internet (Internet Service Provider — ISP), при наличии так называемого *адреса IP* (Internet Protocol), который состоит из четырех групп цифр, разделенных символом точки. Например, моему компьютеру присвоен адрес IP, равный 198.109.162.177.

*Intranet* — это усеченный, замкнутый вариант Internet, "Internet в миниатюре". Существенно то, что intranet представляет собой IP-сеть, к компьютерам которой имеет доступ ограниченная группа пользователей.

Новый термин

*Гипертекст* — это обычный текст, который дополнен встроенным кодом, регламентирующим правила восприятия и отображения данных программными системами-броузерами, например Microsoft Internet Explorer или Netscape Navigator.



Web-броузеры (Internet Explorer) отображают текст в формате HTML. Другие устройства, обеспечивающие доступ к Internet, например сотовые телефоны, используют формат WML (сокр. от Wireless Markup Language — язык разметки для беспроводных систем). WML — это разновидность XML (Extensible Markup Language, расширяемая спецификация языка, предназначенного для создания Web-страниц). Текст в любом из этих форматов является гипертекстом.

Главный инструмент пользователей Internet и intranet — Web-броузер. Один из основных языков, применяемых для управления данными в среде Web, носит название *Hypertext Markup Language (HTML)*. HTML дополняет собственно текст специальными инструкциями по его обработке программами-интерпретаторами, например Microsoft Internet Explorer и Netscape Navigator. Приложения Microsoft Word и Microsoft Outlook также поддерживают возможности воспроизведения гипертекста.

Гипертекст намного более полезен и гибок в сравнении с обычным текстом, поскольку встроенные в него команды дают возможность включить в состав документа графические объекты, аудио- и видеоданные, информацию из баз данных и т.д.

# Установка и использование приложения Internet Information Services

Самый простой способ создания intranet предлагает установку и применение приложения Microsoft Personal Web Server (PWS) или Internet Information Services (IIS) для Windows 2000, которые позволяют использовать Web-сервер в пределах локальной сети. Хотя программа PWS не предоставляет всех возможностей, доступных, скажем, при использовании приложения Internet Information Server, она позволяет поддерживать в рабочем состоянии сайты Internet и intranet, к тому же удобна в качестве среды разработки и отладки. Если есть возможность, рекомендуем отдать предпочтение программе Internet Information Services.



В Software Conceptions приложение PWS было впервые применено для создания Web-сайта компании еще в 1995 году, а совсем недавно я обратился к нему вновь, решая задачу построения сети intranet для одной из страховых компаний штата Мичиган. Имея собственный сайт intranet, удобно поддерживать данные о текущих проектах и предоставлять информацию пользователям программных продуктов.

Чтобы установить Microsoft Internet Information Services на компьютере с операционной системой Windows 2000 Professional, выполните следующие действия.

1. Щелкните на кнопке Пуск⇒Настройка⇒Панель управления (Start⇒Settings⇒Control Panel).
2. В окне Панель управления щелкните на апплете Установка и удаление программ (Add/Remove Programs).
3. В окне Установка и удаление программ щелкните на кнопке Установка компонентов Windows (Add/Remove Windows Components).
4. Появится окно Мастер компонентов Windows (Windows Components Wizard). Установите флажок Internet Information Services (IIS) и щелкните на кнопке Далее (Next).
5. Мастер установит IIS и сконфигурирует его.

После завершения инсталляции Internet Information Services можно управлять конфигурацией, используя апплет Администрирование (Administrative Tools) Панели управления (рис. 20.1).



Если на первый взгляд Internet Information Services имеет верную конфигурацию, но вам все же не удастся просматривать содержимое компьютера, проверьте, работает ли *проxy-сервер* — приложение, которое обеспечивает компьютерам сети совместный доступ в Internet.

Если проxy-сервер установлен, необходимо сообщить Internet Explorer о том, что его не нужно использовать для локальных адресов. Откройте Internet Explorer, выберите команду Сервис⇒Свойства обозревателя⇒Подключения (Tools⇒Options⇒Connections). Щелкните на кнопке Настройка LAN (LAN Settings). Если в группе Прокси-сервер (Proxy Server) установлен флажок Использовать прокси-сервер (Use a proxy server), установите также флажок Не использовать прокси-сервер для локальных адресов (Bypass proxy server for local addresses).

Если вы не знаете, как обеспечивается доступ в Internet с вашего компьютера, а один термин проxy-сервер приводит вас в замешательство, обратитесь к администратору сети.

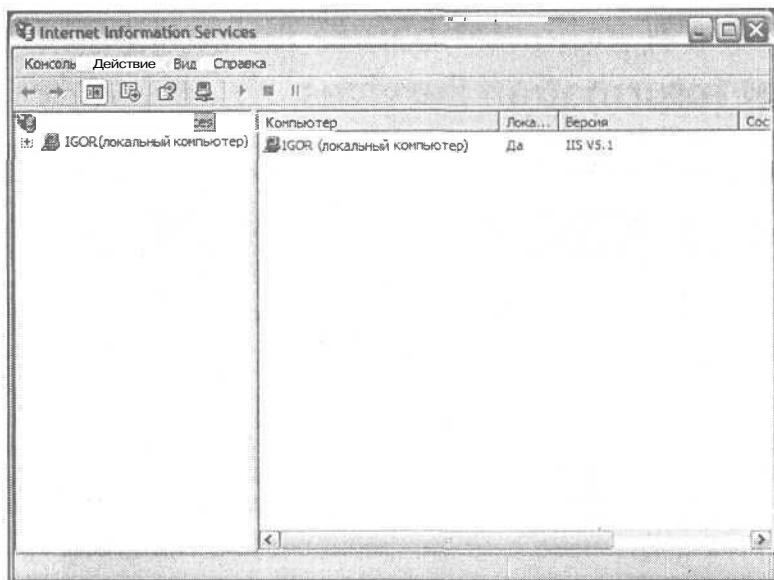


Рис. 20.1. Internet Services Manager

Добавлять пользовательские страницы в intranet очень просто: добавьте HTML-страницу в папку `c:\inetpub\wwwroot` — локальную папку, заданную по умолчанию для Web-страниц в IIS.

Сохранив страницу доступа к данным в корневом каталоге Web-узла рабочей станции, вы сможете ее протестировать и сделать доступной для пользователей вашего узла. Как уже отмечалось ранее, PWS и IIS представляют собой идеальную среду для тестирования. Такие средства, как FrontPage 2002 и Dreamweaver могут значительно упростить процесс создания Web-страниц.

## Создание Web-сайта

Personal Web Server — хороший выбор для поддержки небольших групп пользователей в среде intranet и прекрасная платформа для разработки и тестирования Web-решений. Чтобы построить Web-сайт, необходимо спроектировать начальную страницу. На ней должны находиться гиперссылка на другие страницы (в том числе и страницы доступа к данным), содержащие информацию, которую вы планируете предоставить в распоряжение пользователей.



Если ваши профессиональные намерения серьезны (например, вас беспокоят проблемы обеспечения безопасного доступа, участия в проектах, связанных с электронной коммерцией, и т.п.), выберите более мощную платформу — скажем, Windows NT или Windows 2000 в сочетании с Internet Information Services.

## Знакомство с Web-страницами

Web-сайт состоит, как минимум, из одной страницы. Большинство Web-сайтов содержит несколько страниц, и их структура все более усложняется. Творчество Web-мастера ограничено только его воображением. Web в целом — это заданный посредст-

вом гиперссылок набор логических связей между страницами. Гиперссылка представляет собой специальный фрагмент текста HTML, содержащий адрес указываемой Web-страницы — так называемый *Uniform Resource Locator* (URL).

Удобно создавать Web-страницы в программе Microsoft FrontPage 2002. Листинг 20.1 демонстрирует пример использования основных конструкций кода HTML, которые должны присутствовать в любом HTML-файле.

#### Листинг 20.1. "Шаблон" файла HTML

```
1: <html>
2: <head>
3: </head>
4: <body>
5: </body>
6: </html>
```

#### Анализ

Строки 1–6 представляют содержимое пустой Web-страницы. Если наполнить ее "телом", вставив соответствующий HTML-код и данные между метками `<body>` и `</body>`, мы получим страницу, которую можно будет просмотреть с помощью Web-браузера. Специальные инструментальные средства, такие как FrontPage, значительно упрощают процесс создания Web-страниц. Web-сайт содержит одну или несколько страниц, взаимосвязанных гиперссылками. (Слово Web в переводе с английского означает *паутина*, и отдельные "паутинки" — это гиперссылки.) В своей основной форме гиперссылка описывается следующим синтаксическим выражением:

```
<a href=http://WebАдрес/ИмяФайлаСтраницы.htm>ИмяСтраницы</a>
```

Метка `a href` указывает на начало строки гиперссылки. Вместо словосочетания *WebАдрес* вводится реальный адрес (или URL) в формате доменного имени узла Web, если речь идет о ссылке Internet, либо имени ресурса локальной сети intranet. Например, чтобы создать гиперссылку на Web-сайт компании Software Conceptions, достаточно в качестве параметра *WebАдрес* указать строку `www.softconcepts.com`. Другой пример — для ссылки на сайт компании Microsoft следует использовать адрес `www.microsoft.com`. Параметр *ИмяФайлаСтраницы* задает полный путь к нужной странице Web-сайта. Вот реальный пример обращения к конкретной странице сайта:

```
<a href=http://www.softconcepts.com/index.htm>SoftwareConceptions
Homepage</a>
```

Параметр *ИмяСтраницы* содержит текст, который должны видеть пользователи на Web-странице, собираясь проследовать по указанной ссылке.

## Что представляет собой страница доступа к данным

*Страница доступа к данным* — термин, применяемый для обозначения Web-страниц, отображающих информацию из баз данных. Код HTML, который содержит инструкции для создания графических управляющих элементов и привязки к ним объектов базы данных, особенно сложен. Хотя теоретически вы могли бы написать его самостоятельно, с нуля, подобная процедура требует довольно серьезных усилий, поэтому вы значительно ускорите работу, если обратитесь к "услугам" мастеров Access 2002.



Погрузиться в таинства программирования на HTML вы сможете, ознакомившись со специальной литературой и изучив код, написанный вашими коллегами либо сгенерированный при помощи мастеров создания страниц доступа к данным.

## Расширенные возможности проектирования Web-страниц

Ранее уже говорилось о том, что индустрия программирования для Web сегодня развивается самыми быстрыми темпами. Сайты, созданные с применением новейших технологических достижений, помимо HTML, в большом количестве содержат также блоки программного кода других видов. Сайты, решающие задачи электронной коммерции, позволяют совершать покупки, осуществлять поисковые операции, отправлять и принимать электронную почту. Для реализации подобных функций требуется применение специальных средств.

Microsoft разработала стандарт под названием Active Server Pages (ASP). Web-страницы, созданные на его основе, могут содержать код, написанный на языках VBScript и JavaScript. Последний находит применение и при создании обычных страниц HTML. Сайты, обладающие расширенными возможностями, предполагают выполнение на Web-сервере дополнительных программ — речь идет о таких серверах, как, скажем, ISAPI, NSAPI и CGI.

Если вас привлекают задачи электронной коммерции, обратитесь к услугам профессионального разработчика или как минимум приобретите несколько специализированных изданий, посвященных технологиям построения и администрирования Web-серверов, а также освещающих проблемы обеспечения безопасного доступа к ресурсам Internet и программирования в стандартах HTML и ASP. Подобные вопросы выходят за рамки предмета нашего обсуждения.

## Создание демонстрационной базы данных

Задача создания корпоративной базы данных возникнет перед вами даже в том случае, если речь идет о компании скромных размеров. Если вам приходится пользоваться внутренним телефонным справочником фирмы, который хранится на традиционных бумажных носителях, то на поддержание его в активном состоянии придется тратить немалые усилия и средства. Если же разместить справочник в виде таблицы данных, доступной на узле intranet, многие проблемы просто исчезнут, и задача администрирования сведется к выполнению ряда простых операций.



С любыми информационными потоками, действующими в среде небольшой компании, справиться гораздо легче при наличии сайтов intranet. Здесь могут публиковаться штатные расписания, внутрикорпоративные стандарты и требования, технические бюллетени и другие самые разнообразные материалы, которые представляют интерес для широкого круга сотрудников.

Поскольку правильность сказанного не поддается сомнению, в ходе дальнейшего изложения будут приведены примеры использования простой корпоративной базы данных, содержащей сведения о структуре компании.

**Схема** — обобщенный термин, применяемый для обозначения различных совокупностей объектов, из которых состоит база данных — таблиц, полей, индексов, запросов, хранимых процедур и т.п.

Представим себе вымышленную компанию, именуемую **eSoft**, которая занимается разработкой специализированного программного обеспечения. База данных, содержащая сведения о структуре и персонале eSoft, состоит из трех таблиц — DEPARTMENT, EMPLOYEE и ROLES. Схема базы данных компании приведена в табл. 20.1, 20.2 и 20.3.

**Таблица 20.1. Структура таблицы DEPARTMENT, содержащей данные о подразделениях компании**

Название поля	Тип	Размер	Индекс
DEPARTMENT_ID	AutoNumber		Первичный ключ
NAME	Text	50	

**Таблица 20.2. Структура таблицы ROLES, в которой находится информация о номенклатуре должностей компании**

Название поля	Тип	Размер	Индекс
ROLE_ID	AutoNumber		Первичный ключ
NAME	Text	20	
DESCRIPTION	Text	50	

**Таблица 20.3. Структура таблицы EMPLOYEE, содержащей данные о служащих компании**

Название поля	Тип	Размер	Индекс
EMPLOYEE_ID	AutoNumber		Первичный ключ
FIRST_NAME	Text	20	
LAST_NAME	Text	20	
SSN	Text	11	
DEPARTMENT_ID	Number		Внешний ключ
PHONE_NUMBER	Text	14	
EXTENSION	Text	6	
ROLE_ID	Number		Внешний ключ

Основную роль в схеме играет таблица EMPLOYEE. Между таблицами EMPLOYEE и DEPARTMENT установлена связь типа *многие-к-одному*, т.е. для каждой записи DEPARTMENT (подразделения компании) может существовать несколько записей EMPLOYEE (служащих). Аналогичная связь (рис. 20.2.) задана и между таблицами EMPLOYEE и ROLES: каждый служащий занимает определенную должность, причем (естественно) допускается ситуация, когда у нескольких служащих должности совпадают.



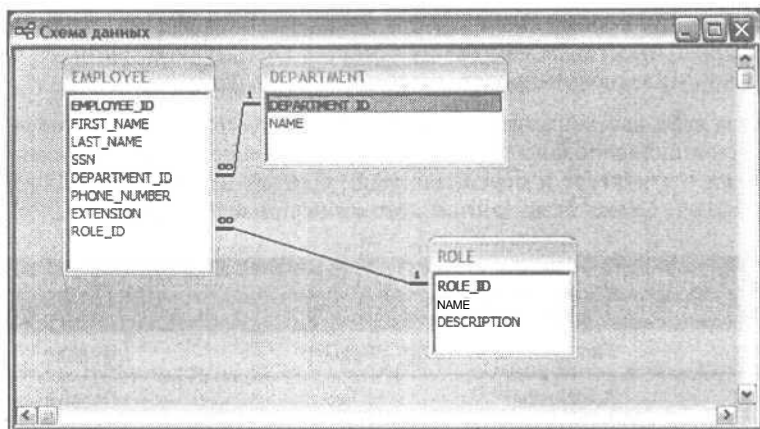


Рис. 20.2. Связи между таблицами базы данных eSoft

Чтобы наш последующий разговор приобрел более предметные очертания, вам необходимо создать базу данных Access (далее мы будем называть ее eSoft) и построить в ней таблицы в соответствии со схемой, приведенной выше. Затем вы узнаете, как с помощью мастера создать страницу доступа к данным.

## Создание Web-страниц с помощью мастеров

Решение проблем, легко поддающихся формализации, может быть автоматизировано — этот тезис сомнений не вызывает. Если в такую категорию попадают задачи построения форм, почему такой вывод нельзя сделать о Web-страницах простой структуры? По мере развития инструментальных средств меняются и наши взгляды на то, решается ли определенная задача, и если да, то каким образом.

Как и во многих других случаях, при построении страниц доступа к данным на помощь приходят вседесущие мастера Access 2002. Чтобы ознакомиться с их перечнем, откройте недавно построенную базу данных eSoft, выберите в списке Объекты (Objects) элемент Страницы (Pages) и щелкните на кнопке Создать (New) панели инструментов. Откроется диалоговое окно Новая страница доступа к данным (New Data Access Page), в котором представлены следующие режимы работы: Конструктор (Design View), Существующая Web-страница (Existing Web page), Мастер страниц (Page Wizard) и Автостраница: в столбец (Autopage: Columnar) (рис. 20.3).

В режиме Конструктор открывается чистая страница. Теперь всю работу придется выполнить самостоятельно — это гораздо сложнее, нежели отредактировать страницу, сформированную мастером. При выборе опции Существующая Web-страница открывается диалоговое окно Поиск Web-страницы (Locate Web Page), позволяющее найти и открыть одну из ранее созданных страниц. Имейте в виду, что Web-страницы сохраняются вне базы данных — Access запоминает лишь ссылки на их физические адреса. Режим Автостраница: в столбец требует от вас минимальных усилий, но платой за это будут ограниченные возможности дальнейшей настройки параметров полученной страницы. Для его применения достаточно выбрать источник данных (таблицу или запрос) и щелкнуть на кнопке ОК. Режим Мастер страниц предлагает компромиссный вариант действий, обеспечивающий простоту использования и разнообразие возможностей (в дальнейшем внимание будет сконцентрировано именно на нем).

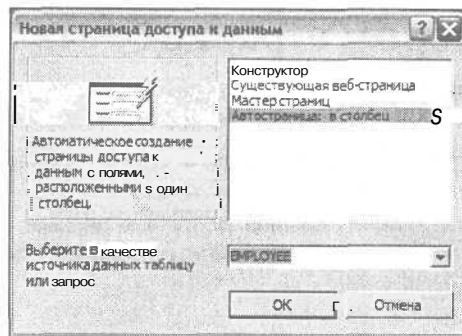


Рис. 20.3. Диалоговое окно Новая страница доступа к данным предлагает на выбор несколько инструментальных средств

Чтобы продемонстрировать способы практического применения мастера страниц, мы построим запрос, объединяющий информацию из всех таблиц базы данных eSoft — EMPLOYEE, DEPARTMENT и ROLES.

## Выбор данных из нескольких таблиц

Запрос, который мы создадим, должен связать все три таблицы — EMPLOYEE, DEPARTMENT и ROLES, чтобы вместо скупых идентификаторов (содержимого полей DEPARTMENT\_ID и ROLE\_ID) пользователи видели наименования должности каждого сотрудника и подразделения компании, в котором тот служит.



Ключевые поля, DEPARTMENT\_ID, ROLE\_ID и EMPLOYEE\_ID, которые обеспечивают уникальность записей и используются для логического объединения таблиц, обычно не следует отображать на формах и Web-страницах, кроме тех случаев, когда информация предназначена для администраторов баз данных.

Создайте запрос, текст которого приведен ниже, в листинге 20.2, и сохраните его под именем QUERY\_EMPLOYEE\_PHONELIST\_BY\_DEPARTMENT.

### Листинг 20.2. Пример запроса, объединяющего данные из нескольких таблиц

```
1: SELECT DEPARTMENT.DEPARTMENT_ID, DEPARTMENT.NAME,
2: EMPLOYEE.FIRST_NAME, EMPLOYEE.LAST_NAME,
3: EMPLOYEE.PHONE_NUMBER, EMPLOYEE.EXTENSION, ROLE.NAME
4: FROM ROLE RIGHT JOIN (DEPARTMENT RIGHT JOIN EMPLOYEE ON
5: DEPARTMENT.DEPARTMENT_ID = EMPLOYEE.DEPARTMENT_ID) ON
6: ROLE.ROLE_ID = EMPLOYEE.ROLE_ID);
```

#### Анализ

Запрос возвратит данные следующих полей таблиц: DEPARTMENT.DEPARTMENT\_ID, DEPARTMENT.NAME, EMPLOYEE.FIRST\_NAME, EMPLOYEE.LAST\_NAME, EMPLOYEE.PHONE\_NUMBER, EMPLOYEE.EXTENSION и ROLE.NAME. Поскольку в команде SELECT используются конструкции объединения таблиц (операторы JOIN), запрос допускает только чтение данных — это, собственно, то, что нам нужно.

## Работа с мастером страниц

Чтобы построить страницу доступа к данным с помощью мастера на основе созданного ранее запроса, выполните следующие действия.

1. Открыв базу данных eSoft, выберите в списке Объекты элемент Страницы и щелкните на кнопке Создать панели инструментов.
2. В диалоговом окне Новая страница доступа к данным выберите опцию Мастер страниц и в качестве источника данных укажите запрос `QUERY_EMPLOYEE_PHONELIST_BY_DEPARTMENT`.
3. Щелкните на кнопке ОК.
4. В первом диалоговом окне Мастер страниц перенесите из списка Доступные поля (Available Fields) в список Выбранные поля (Selected Fields) все поля запроса, за исключением `DEPARTMENT_ID`.
5. Щелкните на кнопке Далее (Next).
6. Второе диалоговое окно мастера дает возможность добавления уровней группировки. Выполните все необходимое и щелкните на кнопке **Далее**.
7. Третье диалоговое окно предлагает указать критерии сортировки набора данных. Выберите наименование поля `LAST_NAME`. Щелкните на кнопке Далее.
8. В четвертом диалоговом окне мастера можно указать таблицы, которые позволено обновлять пользователю. Поскольку страница основана на запросе, допускающем только чтение данных, страница доступа к базе данных должна оставаться тоже в режиме "только для чтения". Щелкните на кнопке Далее.
9. В последнем диалоговом окне мастера измените название создаваемой страницы **Сотрудники компании**. Щелкните на кнопке Готово (Finish).

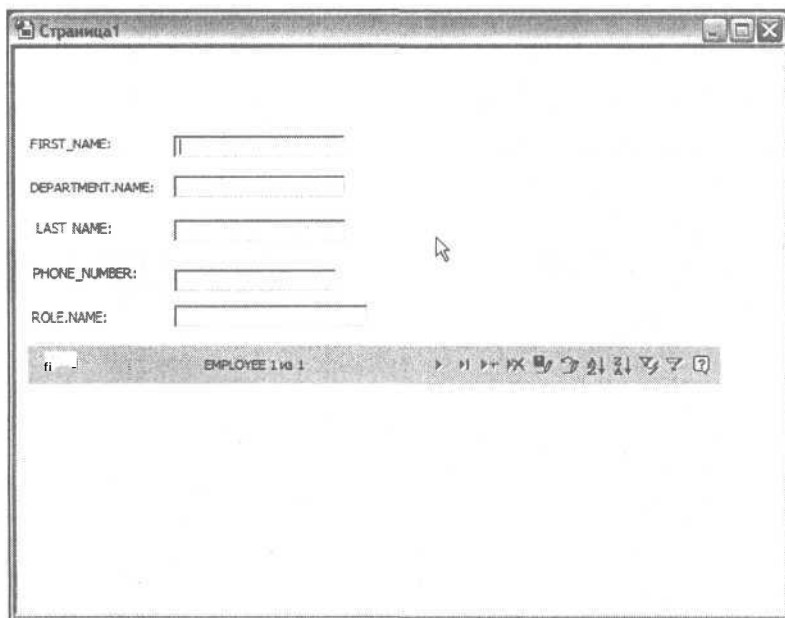


Рис. 20.4. Так выглядит страница доступа к данным, созданная с помощью мастера



Поскольку запрос связывает несколько таблиц, он допускает только чтение результатов. Ввод данных в таблицы следует выполнить предварительно.

Готовая Web-страница показана на рис. 20.4. Чтобы увидеть, как будет выглядеть страница в окне браузера, выберите в строке меню команду Вид⇒Просмотр страницы (View⇒Page View). Управляющие элементы навигации, расположенные в нижней части страницы, позволяют перемещаться по записям набора данных.

## Инструменты Web-дизайна в среде Access

Access 2002 предоставляет разнообразные возможности настройки параметров созданной Web-страницы. Впрочем, страницу можно использовать и без дополнительных преобразований.

## Использование конструктора страниц доступа к данным

Режим конструктора страниц доступа к данным в Access 2002 претерпел изменения. Новые возможности данного режима описаны в следующих разделах.

### Режим структуры данных и серверные фильтры

Режим структуры данных, показанный на рис. 20.5, позволяет задать свойства полей страницы, изменить связи между группами и задать серверные фильтры.

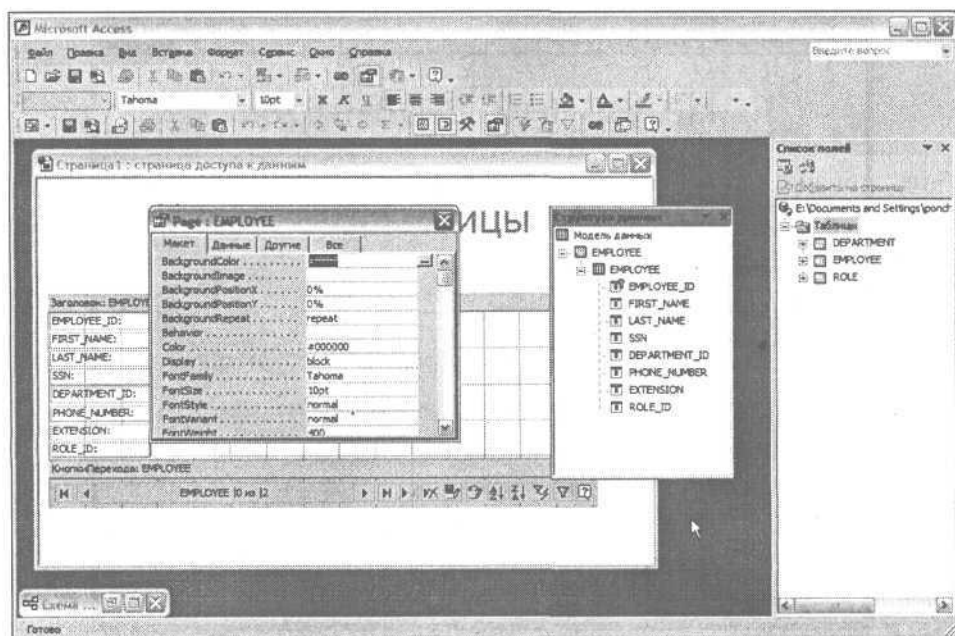


Рис. 20.5. Режим структуры данных в Access 2002

Поведение серверных фильтров во многом схоже с запросами SQL WHERE, относящимися к наборам данных. Чтобы задать серверный фильтр на наборе записей, определенном в модели данных, щелкните на наборе записей правой кнопкой мыши (в нашем примере это QUERY PHONELIST BY DEPARTMENT), в контекстном меню выберите команду Свойства (Properties) и добавьте серверный фильтр (рис. 20.6). Он имеет ВИД ИмяПоля оператор Значение. Например LAST\_NAME > "Kimmel".

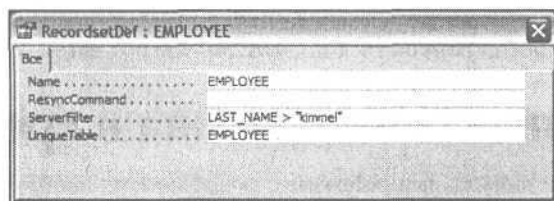


Рис. 20.6. Во время разработки страницы добавьте серверный фильтр, используя диалоговое окно свойств

## Визуальное управление размерами

При изменении размеров элементов управления в Access 2002 происходит их выравнивание и привязывание по линиям сетки.

## Раскрывающиеся элементы

Во время разработки страниц доступа к данным удобно пользоваться раскрывающимся меню. На рис. 20.7. показан пример такого меню в области заголовка. Данное меню упрощает разработку отдельных деталей страницы.

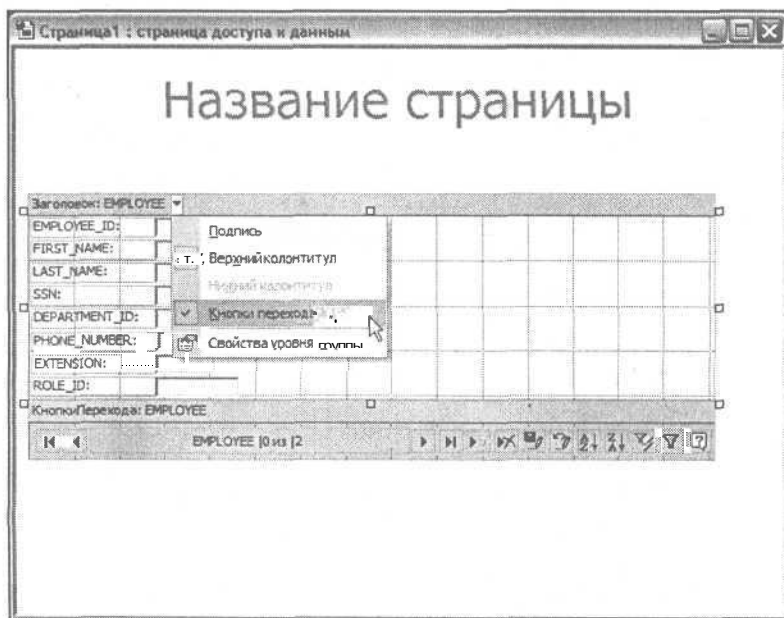


Рис. 20.7. Раскрывающееся меню области заголовка

## Автосуммирование и фильтрация

Access 2002 упрощает процесс добавления на страницы итоговых полей. Щелкните на элементе управления поля, для которого необходимо создать итог, а затем — на кнопке Автосумма (Autosum) на панели инструментов. В нижнем колонтитуле появится надпись, а на страницу будет добавлено текстовое поле, содержащее итог.

По умолчанию выполняется суммирование, но, щелкнув на стрелке справа от кнопки Автосумма, можно выбрать нужное действие — Сумма (Sum), Среднее (Average), Минимум (Min), Максимум (Max), Количество значений (Count), СТАНДОТКЛОН (Standard Deviation), Любой (Any).

## Наследование расширенных возможностей баз данных

Конструктор страниц доступа к данным наследует расширенные возможности баз данных Jet и SQL Server 2000. Это означает, что при составлении поля подстановки во время создания страницы доступа к данным будет введен элемент управления подстановки.

Вернемся к примеру базы данных eSoft. Поля DEPARTMENT\_ID и ROLE\_ID являются ключевыми. Неразумно и жестоко заставлять пользователя помнить значение поля-счетчика, когда можно вывести на экран соответствующий текст. Если изменить таблицу так, чтобы DEPARTMENT\_ID и ROLE\_ID стали полями подстановки, существующими в связанной таблице (DEPARTMENT и ROLE соответственно), то пользователь сможет выбрать текстовое значение из поля со списком. При сохранении такой таблицы в виде страницы доступа к данным, на Web-странице указанная возможность сохранится.

Чтобы задать поле подстановки для EMPLOYEE.DEPARTMENT\_ID, выполните следующие действия.

1. В списке Объекты (Objects) окна базы данных выберите таблицу EMPLOYEE и щелкните на кнопке Конструктор (Design View).
2. В столбце Имя поля (Field Name) выберите поле DEPARTMENT\_ID.
3. В нижней части окна таблицы перейдите на вкладку Подстановка (Lookup).
4. На вкладке Подстановка задайте значение Тип элемента управления (Display control) равным Поле со списком (Combo Box). Заполните остальные поля следующим образом: Тип источника строк = Таблица или запрос (Row source type = Table/Query), Источник строк (Row Source) = DEPARTMENT, Присоединенный столбец (Bound Column) = 1, Число столбцов (Column Count) = 2, Заглавия столбцов = Да (Column Heads = Yes), Ограничиться списком = Да (Limit to list = Yes). Значения остальных свойств, заданные по умолчанию, не изменяйте.
5. Сохраните таблицу. Дважды щелкните на таблице EMPLOYEE и убедитесь: теперь поле EMPLOYEE.DEPARTMENT\_ID позволяет выбирать значения из раскрывающегося списка.
6. В списке Объекты окна базы данных выберите Страницы (Pages). Щелкните на кнопке Создать (New).
7. Выберите Автостраница: в столбец (Autopage: Columnar) и задайте таблицу EMPLOYEE.
8. Щелкните на кнопке ОК.
9. Удалите со страницы надпись и текстовое поле DEPARTMENT\_ID.
10. Выберите команду Вид⇌Список полей (View⇌Field List). Перетащите DEPARTMENT\_ID из списка полей на прежнее место одноименного поля. Появится поле со списком и надпись.
11. Выберите поле со списком DEPARTMENT\_ID и откройте диалоговое окно свойств командой Вид⇌Свойства. На вкладке Другие (Other) измените свойство TabIndex так, чтобы он следовал за предыдущим элементом управления (в нашем случае он был равен 1 и следовал за полем SSN).

Теперь страница доступа к данным содержит редактируемую версию таблицы EMPLOYEE и наследует свойство подстановки исходной таблицы. Дополнительные возможности настройки страниц доступа к данным описаны в следующих разделах.

## Меню и панели инструментов для настройки страницы доступа к данным

Меню Access 2002 зависят от контекста. Другими словами, состав меню и набор их функций динамически меняются в зависимости от рода выполняемых пользователем операций. В этом разделе рассмотрено назначение всех меню, панелей инструментов и диалоговых окон, способных оказать помощь в настройке свойств страницы доступа к данным.

Табл. 20.4 содержит описания элементов меню, доступных в том случае, когда построенная в Access 2002 Web-страница открыта в режиме Конструктор.

**Таблица 20.4. Команды меню, предназначенные для настройки параметров страниц доступа к данным**

Меню	Элемент/команда	Описание
Файл (File)	Предварительный просмотр Web-страницы (Web Page Preview)	Открывает текущую страницу в окне Web-браузера, заданного в Windows по умолчанию
Вид (View)	Конструктор (Design View)	Режим настройки параметров страницы
Вид	Просмотр страницы (Page View)	Позволяет просмотреть и отредактировать данные
Вид	Свойства(Properties)	Открывает диалоговое окно свойств того объекта страницы, который находится в фокусе (рис. 20.5)
Вид	Список полей (Field List)	Открывает диалоговое окно Список полей, позволяющее выбрать поля любых таблиц и границы запросов текущей базы данных для привязки к компонентам, размещенным на странице (рис. 20.6)
Вид	Источник HTML (HTML Source)	Отображает окно редактора исходного кода страницы на языке HTML
Вид		Открывает одноименное диалоговое окно, позволяющее задать одноименные критерии обработки представляемых на странице данных
Вид	Панель элементов (Toolbox)	Открывает панель элементов, которая дает возможность размещать на странице визуальные компоненты (рис. 20.7)
Вид	Структура данных (Data Outline)	Отображает на странице иерархическую структуру записей и полей
Вставка (Insert)	Фильм из файла (Movie from File)	Добавляет на страницу данных файл формата .mov, .avi, .mpeg или .asf
Вставка	Рисунок (Picture)	Добавляет на страницу образ графического файла в формате .gif, .jpg, .jpeg, .bmp, .xmp или .png

Меню	Элемент/команда	Описание
Вставка	Диаграмма Office (Office Chart)	Открывает диалоговое окно Мастер диаграмм Microsoft Office, позволяющее вставить в страницу графическую диаграмму и привязать ее к данным выбранной таблицы или запроса
Вставка	Сводная таблица Office (Office Pivot Table)	Вставляет в страницу компонент сводной таблицы, допускающей динамическое изменение параметров отображения данных
Вставка	Электронная таблица Office (Office Spreadsheet)	Вставляет в страницу компонент электронной таблицы, которая способна к динамическому обновлению в соответствии с текущим содержанием базы данных
Вставка	Гиперссылка (Hyperlink)	Добавляет гиперссылку на файл или Web-страницу
Вставка	Элемент ActiveX (ActiveX Control)	Позволяет выбрать для размещения на странице внешний объект ActiveX, например Microsoft NetShow Player, позволяющий проигрывать файлы аудио- и видеоданных
Вставка	Несвязанный раздел (Unbound Section)	Добавляет часть страницы доступа к данным, например заголовок, колонтитул или отдельный раздел
Формат (Format)	Тема (Theme)	Открывает диалоговое окно Тема, позволяющее задать predetermined набор признаков оформления страницы
Формат	Фон⇒Цвет (Background⇒Color)	Предлагает выбор цвета фона страницы
Формат	Фон⇒Рисунок (Background⇒Picture)	Позволяет выбрать графический файл для отображения в качестве фона страницы
Формат	Фон⇒Звук (Background⇒Sound)	Дает возможность выбрать звуковой файл для проигрывания во время демонстрации страницы

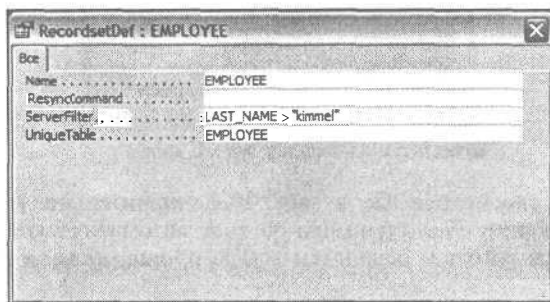


Рис. 20.8. Это диалоговое окно предоставляет возможность выбрать поля любой таблицы или запроса базы данных для привязки к визуальным компонентам страницы



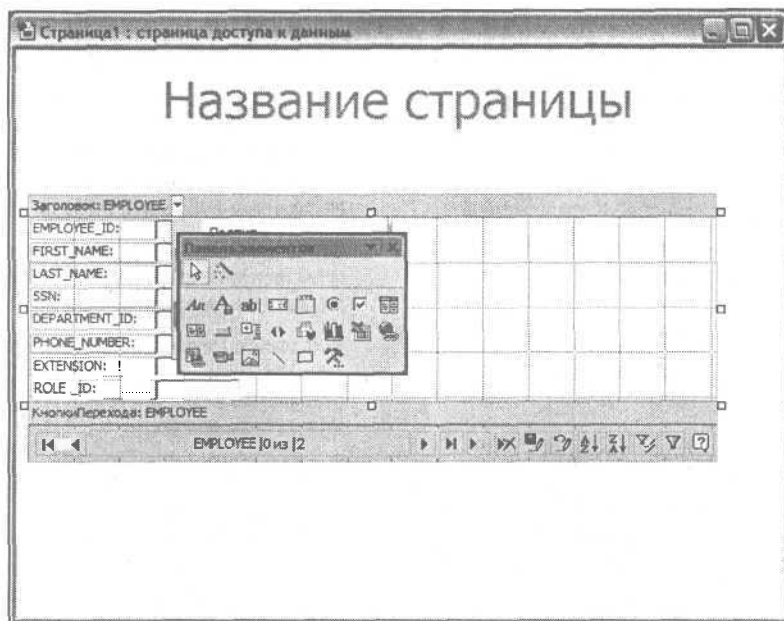


Рис. 20.9. Панель элементов содержит кнопки, соответствующие стандартным визуальным компонентам

В Access 2000 была доступна команда меню Вид⇌Сортировка и группировка. Функции сортировки и группировки перенесены в раскрывающееся меню области заголовка. Щелкните на кнопке со стрелкой "вниз" справа от имени записи и выберите команду Свойства уровня группы (Group Level Properties) (рис. 20.8).

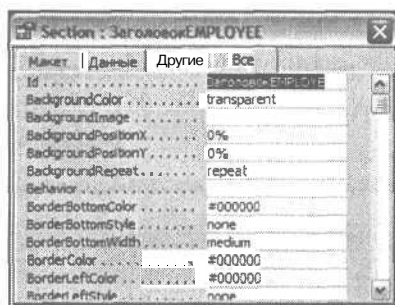


Рис. 20.10. Свойства можно задать, используя элементы этого меню

Команды меню, рассмотренные в табл. 20.4, значительно упрощают задачу построения и модификации Web-страницы за счет автоматической генерации HTML-кода. Воспользоваться готовым решением всегда проще, нежели программировать самостоятельно.

## Размещение дополнительных компонентов на странице

Характеристики внешнего вида и функциональные возможности Web-страницы определяются составом используемых при ее построении визуальных компонентов. Чтобы разместить на странице новый компонент, выполните следующие действия.

1. Выберите в строке меню команду Вид⇒Панель элементов, чтобы отобразить одноименную панель инструментов.
2. Щелкните на нужной кнопке панели (все кнопки снабжены всплывающими подсказками).
3. Переместите курсор мыши к требуемой части страницы и щелкните левой кнопкой мыши, чтобы вставить выбранный компонент интерфейса.

Щелкните правой кнопкой мыши и отметьте в контекстном меню элемент Свойства (Properties) (либо воспользуйтесь командой Вид⇒Свойства главного меню) — откроется диалоговое окно свойств, с помощью которого вы сможете внести все необходимые изменения, касающиеся характеристик нового компонента (подробнее см. следующий раздел).

## Диалоговое окно свойств

Диалоговое окно свойств позволяет установить все необходимые характеристики выбранного компонента Web-страницы заранее, на этапе проектирования (впрочем, то же можно сделать и с помощью соответствующего кода, написанного вручную).

Среда быстрой разработки приложений — в отличие от традиционных инструментов программирования — предполагает возможность создания визуального интерфейса непосредственно на этапе проектирования. Чтобы отредактировать свойства определенного компонента Web-страницы, откройте ее в режиме Конструктор, щелчком выберите требуемый элемент интерфейса и выполните следующие действия.

1. Укажите в строке меню на команду Вид⇒Свойства.
2. В списке диалогового окна свойств найдите требуемое свойство.
3. Введите для выбранного свойства новое значение.

Результат очевиден. Попробуйте для примера (мы собираемся изменить содержимое текстовых меток, представляющих наименования полей) выполнить следующее.

1. Откройте построенную ранее страницу доступа к данным базы eSoft.
2. Щелкните на объекте `LAST_NAME`.
3. Выберите в строке меню команду Вид⇒Свойства.
4. Перейдите на вкладку Другие (Other) диалогового окна свойств.
5. Найдите элемент списка `InnerText` (наименования свойств перечислены в левой колонке списка, а их значения — в правой).
6. Измените значение свойства с `LAST_NAME` на Фамилия.

В качестве упражнения аналогичным образом измените содержимое всех меток полей запроса, введя более понятные названия.

# Привязка полей данных к компонентам интерфейса

Говоря о привязке поля данных к определенному компоненту интерфейса страницы, мы имеем в виду, что в диалоговом окне свойств этого компонента параметру ControlSource (вкладка Данные (Data)) соответствует имя поля таблицы, значения которого будут считываться, отображаться посредством компонента и, возможно, сохраняться в базе данных (если страница допускает выполнение операций записи). Кроме того, свойство RecordSource страницы хранит ссылку на общий источник данных, который был выбран в ходе работы с мастером создания страницы.

Проиллюстрируем технику размещения на странице нового компонента и привязки его к полю данных. Откройте Панель элементов, щелкните на кнопке Поле, а затем — в требуемом месте свободной области страницы. Чтобы привязать поле (элемент интерфейса) к полю (объекту базы данных), выполните следующие действия.

1. Щелкните в пределах вновь созданного объекта интерфейса поля.
2. Выберите команду Вид⇌Свойства.
3. Перейдите на вкладку Данные диалогового окна свойств.
4. Щелкните на элементе ControlSource списка свойств, а затем откройте список, расположенный справа.
5. Выберите в списке элемент DEPARTMENT\_ID.
6. Щелкните на тексте надписи и измените ее на Department ID.

Процесс привязки компонентов к полям данных выполняется мастером создания страниц автоматически, но вы должны знать, как при необходимости осуществить подобные операции самостоятельно.

Простейший способ связать элемент страницы доступа к данным с полем данных — просто перетащить название поля из списка полей в нужное место страницы. Чтобы добавить на страницу элемент DEPARTMENT\_ID, выберите команду Вид⇌Список полей. Откроется одноименное диалоговое окно. В режиме конструктора щелкните на поле EMPLOYEE DEPARTMENT\_ID и перетащите его на страницу — простой и наглядный метод связывания данных.

## Сортировка и группировка данных

Диалоговое окно GroupLevel, показанное на рис. 20.11, предлагает возможности изменения структуры представления данных на странице. Доступ к нему обеспечивается из раскрывающегося меню области заголовка (см. предыдущий раздел). Свойства сортировки и группировки описаны в табл. 20.5.

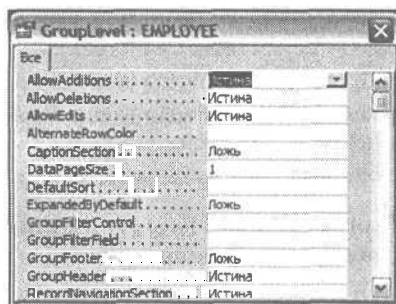


Рис. 20.11. Свойства группировки и сортировки страницы доступа к данным

**Таблица 20.5. Свойства сортировки и группировки данных на странице**

Наименование свойства	Описание
AllowAddition	Булево значение, определяющее, может ли пользователь добавлять записи. По умолчанию равно истина
AllowDeletion	Булево значение, определяющее, может ли пользователь удалять записи. По умолчанию равно Истина
AllowEdits	Булево значение, определяющее, может ли пользователь изменять записи. По умолчанию равно Истина
AlternateRowColor	Позволяет выбрать цвет строки данных
CaptionSection	Булево значение, определяющее, является ли группа разделом заголовка. По умолчанию равно Ложь
DefaultPageSize	Задаёт число записей на странице. По умолчанию равно 1. Чтобы отобразить все записи, измените это значение на All
DefaultSort	Определяет имя поля, по которому выполняется сортировка по умолчанию
ExpandedByDefault	Булево значение, определяющее, должны ли раскрываться подгруппы при просмотре страницы. По умолчанию равно Ложь
GroupFilterControl	Значение, которое используется для указания записей, возвращаемых в наборе записей
GroupFilterField	Имя поля, по которому производится фильтрация записей
GroupFooter	Булево значение, определяющее, отображается ли на странице нижний колонтитул. По умолчанию равно ложь
GroupHeader	Булево значение, определяющее, отображается ли на странице заголовок. По умолчанию равно истина
Record NavigationSection	Булево значение, определяющее, отображаются ли на элементы навигации. По умолчанию равно истина
RecordSelector	Булево значение, определяющее, отображается ли индикатор выбора записи. По умолчанию равно ложь

С помощью указанных выше свойств можно управлять всеми аспектами поведения страниц доступа к данным. Например, можно добавить раскрывающийся список для осуществления фильтрации в папке EMPLOYEE. Выполните следующие действия (здесь предполагается, что вы задали подстановку поля EMPLOYEE.DEPARTMENT\_ID, как это было описано в разделе "Наследование расширенных возможностей баз данных").

1. Создайте страницу доступа к данным на основе таблицы EMPLOYEE.
2. Перетащите поле DEPARTMENT\_ID из списка полей таблицы EMPLOYEE на созданную страницу.
3. Если была создана подстановка для поля, на странице появится поле со списком, связанное с DEPARTMENT\_ID. По умолчанию поле со списком будет называться DEPARTMENT\_ID1. Посмотрите, какое значение имеет свойство ID поля со списком.
4. Откройте окно свойств GroupLevel. Задайте свойство GroupFilterControl равным значению свойства ID поля со списком.

5. В окне свойств GroupLevel задайте свойство GroupFilterField равным DEPARTMENT\_ID.
6. Чтобы увидеть образец страницы, воспользуйтесь командой Вид⇒Просмотр страницы (View⇒Page View).

Если вы выберете элемент поля со списком, он будет работать как фильтр по записям (аналог HAVING в операторах SQL).

## Добавление сводных таблиц на Web-страницы

Сводная таблица — это компонент. Поэтому поведение сводной таблицы одинаково в любом контексте, где бы она ни использовалась. Таким образом, ее поведение и атрибуты, представленные в виде свойств и методов, доступны нам для того, чтобы определить вид компонента и его взаимодействие с пользователем.

Добавление сводной таблицы в страницу доступа к данным аналогично ее добавлению в форму. Чтобы использовать сводную таблицу на странице доступа к данным, откройте страницу в режиме конструктора и выберите команду Вставка⇒Сводная таблица Office (Insert⇒Office PivotTable). Все остальные приемы работы со сводными таблицами в точности повторяют те, которые вы изучали на прошлом занятии, "19-й час. Создание экранных форм". Для примера создайте сводную таблицу на странице доступа к данным, пользуясь следующей схемой.

1. Откройте базу данных eSoft, созданную ранее в этой главе.
2. Щелкните на кнопке Страницы в списке Объекты.
3. Выберите Создание страницы доступа к данным в режиме конструктора и щелкните на кнопке Создать.
4. В новой странице доступа к данным перейдите в режим конструктора.
5. Выберите команду Вставка⇒Сводная таблица Office.
6. Если список полей не находится на экране, выберите команду Вид⇒Список полей.
7. Перенесите поля FIRST\_NAME, LAST\_NAME, PHONE\_NUMBER и EXTENSION таблицы EMPLOYEE в раздел Перетащите сюда поля итогов или деталей.
8. Переместите поле EMPLOYEE.DEPARTMENT\_ID в раздел Перетащите сюда поля фильтра.
9. Выберите команду Файл⇒Предварительный просмотр веб-страницы (File⇒Web Page Preview), чтобы просмотреть созданную страницу доступа к данным в окне Web-браузера.

При создании настоящей Web-страницы вам потребуется, во-первых, дополнить страницу HTML кодом (чтобы привести ее в соответствие с дизайном других страниц узла); и во-вторых, разместить страницу доступа к данным на сервере.

Опубликовать страницу доступа к данным можно на своем ПК или разместить в intranet, скопировав ее на Web-сайт. По умолчанию папка Web-сайта — c:\inetpub\wwwroot\. При распространении базы данных Access и страниц доступа к данным на другие компьютеры не забудьте обновить информацию о подключении к источнику данных с помощью файла Office Data Connection (ODC) или с помощью адреса Universal Naming Convention (UNC) (подробнее см. справочную систему Access).

# Universal Naming Convention

Определяя информацию о соединении, указывайте физический адрес вида \\компьютер\папка\базаданных.mdb.

При размещении страниц доступа к данным обновите информацию о соединении и используйте адрес UNC, а не буквы диска или подключенного сетевого диска в проводнике Microsoft Windows. Предположим, сервер имеет имя SERVER, а папка вашего Web-узла — MYWEB. Чтобы изменить соединение страницы, используя UNC, укажите \\SERVER\MYWEB\eSoft.mdb.

Для изменения свойства соединения страницы откройте ее в режиме конструктора и выберите команду Вид⇒Список полей. Над списком расположена пиктограмма Свойства подключения для страницы (Data Link Properties). Щелкните на ней, чтобы отобразить диалоговое окно Свойства связи с данными (Data Link Properties), и перейдите на вкладку Подключение (Connection) (рис. 20.12). Щелкните на кнопке с многоточием справа от поля Выберите или введите имя базы данных (Select or enter database name). После этого найдите в сети файл eSoft.mdb и щелкните на кнопке Открыть (Open). Если база данных расположена на другом компьютере сети, в указанном поле появится ее UNC-адрес.

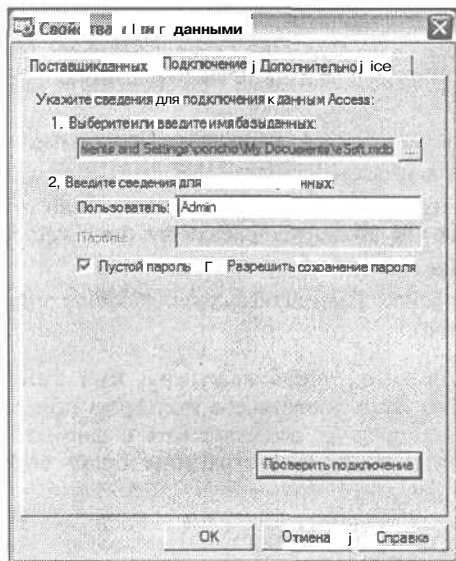


Рис. 20.12. Это диалоговое окно позволяет применить UNC-адрес для подключения базы данных, что гарантирует использование адреса, не доступного для других пользователей

Прежде чем закрыть диалоговое окно Свойства связи с данными, щелкните на кнопке Проверить подключение (Test Connection). Получив сообщение о корректности подключения, щелкните на кнопке ОК. Теперь, работая со страницей доступа к данным, можете быть уверены, что пользуетесь информацией, хранящейся в нужной базе данных.

# Office Data Connection

Файлы подключения в формате Office Data Connection (ODC) служат для привязки одной или нескольких страниц к источнику данных. Помните, что страницы доступа к данным хранятся вне базы данных, а информация о соединении находится в виде сценария в HTML-файле.

Чтобы создать файл подключения в формате ODC, выберите нужную страницу и перейдите в режим конструктора. Щелкните правой кнопкой мыши на странице и выберите в контекстном меню команду Свойства страницы (Page Properties). В диалоговом окне свойств перейдите на вкладку Данные (Data). Щелкните на свойстве ConnectionFile, а затем — на кнопке с многоточием справа от него. Откроется диалоговое окно Выбор источника данных (Data Source Explorer), после чего выполните следующие действия.

1. В диалоговом окне выберите +Подключение к новому источнику данных (+Connect to New data Source.odc).
2. Щелкните на кнопке ОК. Появится окно мастера подключения данных (Data Connection Wizard).
3. Выберите тип источника данных Дополнительно (Other/Advanced). Щелкните на кнопке Далее (Next).
4. Появится диалоговое окно Свойства связи с данными (см. рис. 20.12). На вкладке Поставщик данных (Provider) выберите Microsoft Jet 4.0 OLE DB. Щелкните на кнопке Далее.
5. Откроется то же диалоговое окно на вкладке Подключение.
6. Введите путь к базе данных в формате UNC.
7. Щелкните на кнопке Проверить подключение. Если тест завершится успешно, щелкните на кнопке ОК. Появится диалоговое окно мастера.
8. Щелкните на кнопке Далее.
9. Введите имя ODC-файла. Если необходимо, добавьте описание. Затем щелкните на кнопке Готово (Finish).

В свойстве ConnectionFile теперь находится имя файла подключения. Создав файл подключения, можно использовать его повторно при создании других страниц доступа к данным. Применяв файл подключения в формате ODC, вы можете быть уверены, что все созданные вами Web-страницы будут содержать информацию из нужной базы данных.

## Роль XML в Access

XML — это язык разметки, подобный HTML. Extended Markup Language (XML) используется для обмена данными между Access 2002 и страницами доступа к данным, а также для импорта и экспорта данных.

XML — промышленный стандарт, а не стандарт Microsoft, поэтому с ним могут работать многие другие приложения и протоколы. Использование промышленного стандарта для передачи данных упрощает распространение информации между существующими инфраструктурами, в особенности в Web.

XML — это просто текст, но специальным образом отформатированный, следовательно, его можно передавать и получать с помощью HTTP-соединений так же, как Web-страницы. при этом не требуются специальные программы для его конфигурации и Поддержки.

XML используется в трех важных областях. Во-первых, как средство передачи данных из баз данных на Web-страницы и обратно. Во-вторых, XML, будучи стандартом, применяется для импорта и экспорта данных. Многие производители приложения знают, как работать со стандартом XML, поэтому существенно возрастает вероятность возможной интеграции приложений других производителей и приложений Microsoft. И наконец, XML используют в качестве посредника при работе со страницами доступа к данным в автономном режиме.

Очень важно, что XML работает как будто за сценой. При создании страниц доступа к данным Access передает данные в формате XML, не требуя дополнительных трудозатрат от своего пользователя.

## Резюме

Страницы доступа к данным можно воспринимать как формы. Разместив на них текст, графику и различные управляющие элементы, вы значительно расширите и усилите возможности интерфейса своих Web-приложений. Страницы доступа к данным отличаются от форм тем, что их легко просмотреть в окне Web-браузера или переслать по электронной почте.

На этом занятии вы научились создавать простые страницы с помощью мастера Access и выполнять некоторые операции по их настройке. Проектирование формы обычно сопровождается написанием фрагментов VBA-кода. При создании страницы доступа к данным большая часть работы выполняется мастером, который автоматически генерирует внутренний код страницы на языке HTML. При необходимости вы можете исправить текст HTML в окне редактора. Помимо HTML, для построения Web-страниц применяются и другие средства программирования — например, JavaScript и VBScript. Созданный с их помощью текст может быть успешно интерпретирован некоторыми развитыми Web-браузерами. Это самостоятельные и серьезные языки программирования, требующие специального изучения. VBScript, впрочем, весьма напоминает VBA. К сожалению, код VBScript не поддерживается браузером Netscape Navigator — для его интерпретации необходим Microsoft Internet Explorer.

Если вы решили поучаствовать в построении корпоративной сети intranet или проектировании Web-страниц для опубликования в Internet, инструменты создания страниц доступа к данным помогут достичь вполне приемлемых результатов быстро и эффективно. Напоследок обратитесь к приведенным ниже разделам "Вопросы и ответы" и "Задания", чтобы еще раз освежить в памяти пройденный материал.

## Вопросы и ответы

**Вопрос.** Как определить, нужен ли мне более мощный программный Web-сервер?

**Ответ.** Вы всегда имеете основания использовать самые совершенные программные продукты, и это особенно справедливо в том случае, если ваши намерения связаны с предоставлением услуг электронной коммерции, созданием общедоступных Web-сайтов в Internet и т.п. В подобных ситуациях просто необходимо обращаться к более мощным инструментам, таким, например, как Internet Information Services.

**Вопрос.** Могу ли я создать Web-сайт в Internet на основе обычного персонального компьютера?

**Ответ.** Да. Приложение Personal Web Server появилось еще в составе Windows 95. Все, что вам нужно, — это Personal Web Server, IP-адрес, подключение к провайдеру услуг Internet и зарегистрированный URL. За дополнительной информацией обращайтесь к провайдеру.



**Вопрос.** Обязательно ли для создания intranet использовать выделенный сервер Windows 2000?

**Ответ.** Нет. Персональные компьютеры, оснащенные сетевыми картами и подключенные к кабельной системе 5-й категории, вполне способны работать в одноранговой сети.

**Вопрос.** Можно ли пользоваться страницами доступа к данным в приложениях, ориентированных на выполнение в среде Access?

**Ответ.** Безусловно. Хотя формы обеспечивают более высокую производительность и расширенные возможности программного управления, вы можете применять страницы доступа к данным в сочетании с формами или даже вместо них.

## Задания

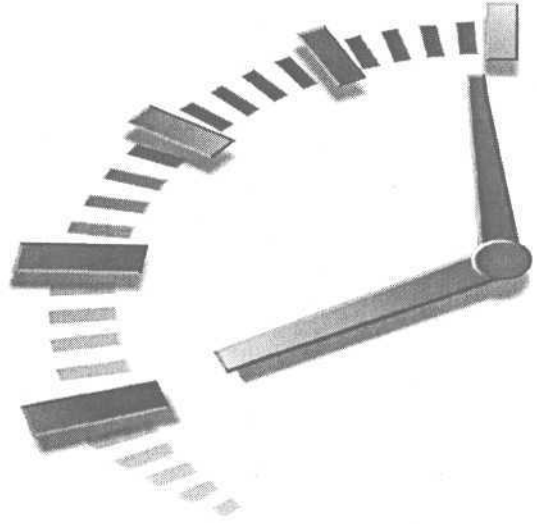
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Для каких целей применяется язык HTML?
2. Назовите другие языки программирования, используемые в Web-дизайне.
3. Для чего необходим URL?
4. Если существует необходимость в создании Web-сайта с внутрикорпоративной информацией, какого рода сайт вы построите — intranet или Internet?
5. При создании страниц доступа к данным можно ссылаться на те же таблицы и запросы, что и при проектировании форм. Верно?
6. Что означает фраза *привязка компонента к полю данных*?

## Упражнения

1. Установите для построенной страницы доступа к данным одну из тем — предопределенных наборов параметров внешнего оформления.
2. Добавьте в построенную страницу новый элемент управления — Номер подразделения.
3. Измените цвет фона текстового поля Наименование подразделения, чтобы визуально подчеркнуть, что оно допускает только операцию чтения.
4. Измените параметр TabIndex текстового поля Наименование подразделения таким образом, чтобы оно оказалось в начале списка компонентов, управляющего порядком перемещения фокуса при нажатии пользователем клавиши <Tab>.



# Часть VIII

## Объектно-ориентированное программирование в Access

### Темы занятий

21-й час. Основы программирования классов

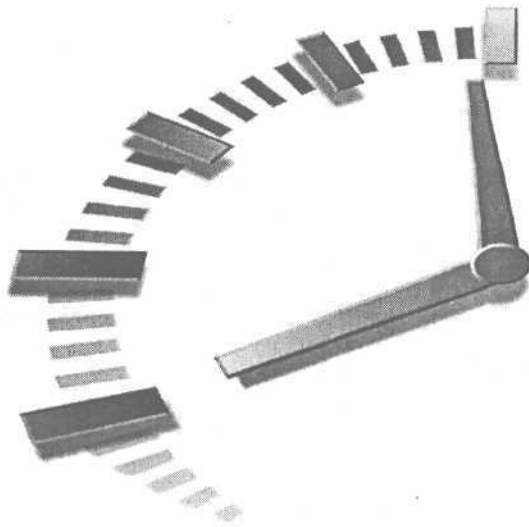
22-й час. Совершенствование типов данных

23-й час. Надстройки Access

24-й час. Управление информацией о контактах Outlook



# 21-й час



## Основы программирования классов

Вам наверняка уже доводилось видеть и слышать много разной информации об объектно-ориентированном программировании (нет, не только на страницах нашей книги!). До сих пор мы пользовались классами, которые были созданы другими людьми. Класс — это расширенный пользовательский тип данных. Экземпляр класса называют объектом. Класс — это базовое, краеугольное понятие парадигмы объектно-ориентированного программирования; не будь классов, термин *объектно-ориентированный* лишился бы своей главной части — *объектно-*. Впрочем, несмотря на зловещее название, которое не обещает новичку спокойной жизни, за этой терминологией не скрывается ничего чрезмерно сложного и недоступного для понимания. Очень скоро (думаем, к концу этого занятия) вы определенно научитесь создавать собственные классы — более того, вы, по существу, узнаете, как писать объектно-ориентированные программы.

Понятие объекта основывается на реалиях окружающей жизни. В самом деле, объекты настолько *реальны*, что создатели большинства языков программирования уже обеспечили их средствами поддержки объектно-ориентированной парадигмы либо настойчиво к этому стремятся. Даже дедушка COBOL — и тот, прихрамывая на обе ноги, желает поспеть за своими молодыми и более резвыми конкурентами. Объектно-ориентированный стиль дает в руки программистов мощные орудия, позволяющие справляться с задачами, сложность которых — в угодку требованиям эпохи — неуклонно возрастает.

Объектно-ориентированное программирование далеко не так страшно, как кажется. Просто оно требует некоторой подготовки — частично по той причине, что человеку приходится отвыкать от стереотипов. Если же вы ранее вообще не занимались программированием, считайте, что вам крупно повезло, поскольку переучиваться и ломать привычный ход мыслей попросту незачем.

Основные темы занятия.

- Почему объектный подход так важен и ценен.
- Еще раз о терминах объектно-ориентированного программирования.
- Создание новых классов и расширение возможностей существующих.
- Отладка объектного кода в среде Access 2002.

# В чем состоит важность объектного подхода

Прежде чем привести аргументы в пользу важности объектно-ориентированной парадигмы, начнем с того, как развивалась область программирования.

До возникновения "объектных" идей в индустрии программирования господствовал так называемый *структурный* подход, который был обязан своим появлением (так же, как и его будущий преемник) все возрастающей сложности прикладных задач, какую он и призван был Преодолеть. В центре внимания "структурных" программистов находился *процесс*. Решая задачу, программист определял множество данных и процессов их обработки.

Процедуры и данные никак не соотносились друг с другом — кроме, пожалуй, тех ситуаций, когда процедура и порция данных в какой-то момент времени выполнения программы "соприкасались" для осуществления конкретной операции. После определения множества процессов и круга необходимых данных создавались процедуры, поддерживавшие выполнение отдельных процессов. Затем строилась "главная" процедура, из которой вызывались вспомогательные, решавшие частные задачи в рамках общей проблемы. Все это именовалось *потокм вычислений*.

Структурное программирование поддерживалось средствами структурного проектирования. На этапе структурного проектирования предполагалось построение *блоков*, в которых с помощью специальных символов отображался каждый конкретный процесс с учетом его места в общем потоке вычислений.

Сторонники структурного подхода столкнулись с трудностями, порожденными чрезвычайной сложностью задач, которые предстояло решать. Крупное приложение могло содержать в своем составе тысячи процедур и функций, а языки структурного программирования, такие как C и Basic (предшественники Visual Basic), не поддерживали развитых средств агрегации кода. (Помните, мы как-то говорили о любви? Нет, конечно, о любви всегда приятно поразмышлять, но тогда речь шла, простите за прозу, об агрегации понятий.)

Впрочем, еще до появления объектно-ориентированных систем программисты осознали необходимость агрегации данных в более крупные структуры — типы, определяемые пользователем. Агрегатный тип позволяет связать в единую целостную сущность несколько разнородных порций данных. Программисты, кроме того, уяснили (что в настоящее время для нас совершенно естественно) и полезность агрегации строк кода в именованные блоки-процедуры, позволяющие упростить структуру программы и уменьшить ее объем. Агрегатные типы данных и процедуры (функции) составили основу структурного подхода к программированию, но некоторые принципиальные моменты все равно остались за пределами изучения.

Программирование все более тяготело к решению задач, настолько же сложных, как и мир, который нас окружает. Физический мир состоит из "вещей". Каждая "вещь" характеризуется определенными атрибутами, описывающими ее структуру и особенности поведения. Человек — это, в известном смысле, одна из *сущностей* физического мира. Человеку свойственны определенные способности — скажем, речь и движение. Каждый человек отличается внешними характеристиками — цветом волос, ростом, весом и т.п. Это справедливо и в отношении других "вещей" физического мира: все они обладают некоторыми функциональными возможностями и ощутимыми, поддающимися оценке свойствами.

Однако отдельно взятые процедуры и данные не в состоянии обеспечить полноценных средств описания вещей. Ученые-естествоиспытатели доказали, что физический мир (вероятно, самая сложная из всех сущностей) не может быть описан набором процессов — мир состоит из взаимодействующих между собой вещей. Это обстоятельство заставило задуматься и математиков, которые вполне обоснованно реши-

ли, что языки программирования должны поддерживать средства создания абстракций, описываемых множествами свойств и способных к взаимодействиям. Подобные системы программирования получили название *объектно-ориентированных*.

Развитие компьютерной науки и индустрии доказало правоту кибернетиков-первопроходцев. Сложная программа становится более управляемой, если написана в объектно-ориентированном стиле. Впрочем, плохие образцы объектного кода — как и любая другая плохо выполненная работа — не станут лучше от того, что мы назовем их модным словечком.

Не все программисты работают сейчас в объектном стиле, но подавляющее большинство новых проектов создается на языках, в которых поддерживаются средства объектно-ориентированного программирования. Пока еще отдельные компании (по крайней мере, так было в 2001 году) еще не могут оценить всех достоинств новых технологий. Что ж, это их воля. Объектно-ориентированный подход к программированию остается в силе, он может развиваться, изменяться, но уже не будет отброшен, и компьютерная индустрия никогда не повернет вспять, к прошлому.

Уделите особое внимание этой главе — пользу гарантируем. Прежде чем продолжить, несколько советов. Избегайте построения объектов просто так, без особой на то нужды. Сосредоточьте внимание на сущности проблемы, а не на количестве классов, которые вы построили или, наоборот, не построили. Примите к сведению, что хороший объектно-ориентированный "блин" редко удастся с первого раза уберечь от превращения в "ком". Качественные результаты достигаются только в процессе практики — непрерывной и, может быть, даже утомительной.

## В первый раз — первый класс

Работа с модулем класса ничем не отличается от привычных приемов использования обычных модулей. Единственная особенность состоит в том, что законченные фрагменты кода обычных модулей независимы, а содержимое модуля класса трактуется Access как единое целое. Чтобы использовать класс в программе, необходимо построить экземпляр (объект) класса.

Вы уже осведомлены о том, как создаются объекты классов. Еще раз воспроизведем пример из главы "13-й час. Коллекции данных":

### Dim Strings As New Collection

В приведенной строке кода объявляется и создается экземпляр класса Collection под названием strings. Это и есть объект класса. Вводя код в модуле класса, вы определяете структуру класса. Чтобы построить простой класс, выполните следующие действия.

1. Создайте новую или откройте существующую базу данных.
2. Выберите элемент Модули (Modules) в списке Объекты (Objects) окна База данных (Database) и щелкните на кнопке Создать (New) панели инструментов.
3. Введите текст листинга 21.1 в окне нового модуля.
4. Сохраните модуль под именем Test.
5. Не закрывая окна редактора, выберите в строке меню команду Insert⇒Class Module, чтобы создать новый модуль класса и добавить его в базу данных.
6. Введите текст листинга 21.2 в окне нового модуля класса.
7. Сохраните модуль под именем FirstClass.
8. Выберите в строке меню окна редактора команду Window⇒Test, чтобы перейти к окну модуля Test.
9. Выполните процедуру TestMyFirstClass — откроется окно, содержащее сообщение В первый раз — первый класс!.

### Листинг 21.1. Процедура для тестирования класса, описанного в листинге 21.2

```
1: Option Compare Database
2: Sub TestMyFirstClass
3:   Dim MyFirstClass As New FirstClass
4:   MyFirstClass.Greetings
5:   Set MyFirstClass = Nothing
6: End Sub
```

#### Анализ

Листинг 21.1 содержит текст простой процедуры, в строке 3 которой создается объект MyFirstClass класса FirstClass, а далее, в строке 4, вызывается метод Greetings этого класса. Директива строки 5 освобождает фрагмент памяти, выделенный для хранения объекта. Обратите внимание, что имя объекта (MyFirstClass) отделяется от названия его метода (Greetings) оператором точки (.).

Листинг 21.2 демонстрирует содержимое модуля класса.

### Листинг 21.2. Содержимое модуля класса

```
1: Option Compare Database
2: Sub Greetings( )
3:   MsgBox "В первый раз — первый класс!"
4: End Sub
```



В VBA все члены класса содержатся в одном модуле. Модуль класса создается при выборе в строке меню окна редактора команды Insert⇒Class Module.

#### Анализ

Листинг 21.2 представляет код класса FirstClass. Любой код, содержащийся в модуле класса, становится неотъемлемой частью класса. Построенный нами класс FirstClass весьма прост: он содержит единственный метод — процедуру Greetings.

Несмотря на то, что первый класс, FirstClass, нельзя назвать особенно полезным с практической точки зрения, формально он верен и может использоваться в качестве руководства по построению других, более реальных и сложных классов. Чтобы приступить к созданию нового класса, достаточно обратиться к окну редактора Microsoft Visual Basic и выбрать в строке меню команду Insert⇒Class Module, которая приводит к открытию окна нового модуля. Теперь, работая в окне модуля класса, вы можете действовать в той же манере, как и при написании обычного — не объектно-ориентированного — кода. Наиболее существенное различие подходов состоит в том, что, намереваясь включить определенный код в состав класса, вы должны поместить его в соответствующий модуль — и код станет естественной частью интерфейса класса.

Прежде чем перейти к вопросам создания более сложных классов, необходимо кратко осветить проблему обеспечения доступа к членам классов.

## Соккрытие информации

Слыхали ли вы когда-нибудь фразу: *Слишком много информации?* Да, временами бывает, что излишек информации определенно вреден. Вообразите на мгновение, что для обеспечения жизнедеятельности своего организма вы должны будете отдавать мысленные приказы типа "Желудок, дружище, займись-ка перевариванием передан-

ного тебе бутерброда" или "Легкие, ну-ка, вдохните поглубже, та-а-а-к, а теперь выдохните". Просто замечательно, что не нужно заботиться о подобных вещах! А как насчет управления всеми деталями механизма автомобиля в момент экстренного торможения? Если бы такая задача была целиком возложена на человека, сидящего за рулем, ни о какой экстренности не могло бы быть и речи.

Соккрытие информации (или соккрытие сложности) — безусловно, полезно и здорово. Жизнь стала бы просто невыносимой или даже невозможной, если бы нам с вами приходилось ежесекундно думать обо всех аспектах окружающей действительности.

Системы объектно-ориентированного программирования предоставляют в наше распоряжение средства сокращения сложных подробностей структуры объектов. Создавая класс, вы как программист обязаны написать весь его код, т.е. предусмотреть определение всех необходимых свойств и методов, которые позволят будущим объектам правильно и эффективно решать возложенные на них задачи. А теперь о том, чего следует избегать, — вы не должны требовать от потребителей интеллектуальной продукции глубокого знания всех деталей внутренней организации класса. Некоторые детали могут (и, вероятно, должны) быть скрыты от посторонних глаз.

Например, на прошлых занятиях мы неоднократно обращались к объектам класса `ADODB` и нам вовсе не мешало незнание каких-то особенностей его реализации. Собственно говоря, а зачем знать все тонкости? Нам как пользователям класса необходимо и достаточно изучить доступные свойства и методы. Скажем, все, что следует сделать для открытия набора данных `Recordset`, — это вызвать метод `Open`.

Приступая к созданию класса, имейте в виду следующее: вам, вероятно, придется написать целые горы кода, но вашим будущим счастливым пользователям, заранее обреченным на успех (остается только им *позавидовать!*), вовсе нет нужды доподлинно знать или даже понимать, какие внутренние силы поддерживают на плаву все это великолепие. Соккрытие некоторых деталей, несомненно, пойдет во благо, поскольку облегчит задачу использования вашего класса — точно так же, как и любого другого, в том числе стандартного. Ведь не задумываются же люди, изучающие программирование для `Access`, о том, как именно работает система?

*Класс* — это молекула объектно-ориентированного мира. В `VBA` класс представляет собой модуль, подобный другим модулям. Впрочем, если вы захотите экспортировать содержимое обычного модуля во внешний файл, он получит расширение `.BAS`, в то время как файлы модулей классов обозначаются расширением `.CLS`.

В классе `VBA` допускаются две степени доступности его атрибутов. Первая обозначается служебным словом `Public`, а вторая — `Private`. Перед объявлением свойства или метода класса **квалификатором** `Private` вы указываете, что это внутренний атрибут класса и пользователям не стоит о нем беспокоиться. Если объявления данных или методов снабжены служебным словом `Public`, они входят в состав внешнего интерфейса класса и предназначены для общего использования. **Квалификаторы** `Public` и `Private` в конструкциях объявлений членов класса указываются первыми.

#### Новый термин

Любой из атрибутов (свойство или метод) класса называется *членом*.

Данные и методы, определенные в составе класса, часто называют *интерфейсом класса*. Если в начале конструкции объявления члена класса используется служебное слово `Public`, говорят, что такой член служит частью *внешнего интерфейса*. Если же член класса объявлен с применением **квалификатора** `Private`, он становится частью *внутреннего интерфейса* класса.

Итак, словом `Public` вы указываете на объекты, предназначенные для всеобщего обозрения, а атрибуты (свойства или методы), которые обозначены квалификатором `Private`, составляют скрытую, внутреннюю, приватную часть класса. Создав объект класса, даже вы, автор и полноправный владелец этого класса, не сможете получить доступ к его внутреннему интерфейсу. Если вы все-таки попытаетесь обратиться к приват-



ному члену класса за пределами модуля класса, то получите сообщение об ошибке: `Method or data member not found` (метод или член-элемент данных не найден).

Если член объявлен с использованием признака `Private`, его следует трактовать как деталь внутренней реализации класса. Другими словами, с подобными атрибутами класса может работать только его автор. Всем другим программистам даже необязательно знать об их существовании.

До появления и широкого распространения объектно-ориентированного подхода создавались программы, весь код которых был открыт и доступен, и мы как участники проекта должны были бы полностью его изучать. Теперь, взяв на вооружение объектную модель и приступая к проектированию приложения, мы обязаны обращать внимание только на внешний интерфейс тех классов, объекты которых собираемся создавать. Понятно, что в этом случае задача значительно упрощается.

При создании класса вам надлежит заботиться обо всех самых мелких частностях, и это, конечно, правильно. Важно, чтобы построенный класс был самодостаточным и независимым блоком кода; в нем должны найти отражение все стороны моделируемой сущности — в вашей воле сделать их описания открытыми или приватными. В ходе работы над классом вы **концентрируете** внимание на коде класса в целом, но при его использовании достаточно обращаться только к атрибутам внешнего интерфейса.

В последующих разделах мы более конкретно поговорим о способах и особенностях применения служебных слов `Private` и `Public`. Приведем еще несколько советов, которые могут оказаться полезными. Вы вправе пользоваться **квалификатором** `Public` до тех пор, пока не почувствуете себя уверенно при обращении с членами классов, объявленными посредством `Private`. Впрочем, при необходимости всегда можно изменить статус доступности того или иного члена класса.

## Определение методов класса

Методы класса — это процедуры и функции. Они объявляются и строятся в пределах модуля класса. Синтаксические правила обычны и ничем не отличаются от тех, которыми мы руководствовались ранее. Изучите листинг 21.3.

### Листинг 21.3. Расширенный вариант класса `FirstClass`

```
1 Option Compare Database
2 Option Base 0
3 Option Explicit
4 Enum LanguageType
5     ltTechnoid
6     ltEspanol
7     ltDeutsch
8 End Enum
9 Public Sub GreetingsAufDeutsch( )
10: MsgBox GetGreetingsText( ltDeutsch )
11:End Sub
12:Public Sub GreetingsEspanol( )
13: MsgBox GetGreetingsText(ltEspanol)
14:End Sub
15:Public Sub GreetingsTechnoid( )
16: MsgBox GetGreetingsText(ltTechnoid)
17:End Sub
18:Private Function GetGreetingsText(ByVal Language As _
    LanguageType) As String
19: Select Case Language
```

```

20:      Case ltTechnoid
21:          GetGreetingsText = "Hello!"
22:      Case ltEspanol
23:          GetGreetingsText = "Buenos diaz!"
24:      Case ltDeutsch
25:          GetGreetingsText = "Guten tag!"
26:  End Select
27:End Function

```



В пределах модуля открытые и приватные члены класса группируйте отдельно, а внутри каждой из групп располагайте их в алфавитном порядке следования названий. Тогда и вам, и вашим коллегам не придется долго искать все то, что нужно в данный момент.

#### Анализ

Тестовую процедуру листинга 21.1 нетрудно приспособить и для проверки расширенной версии класса `FirstClass`, приведенной в листинге 21.3. В классе теперь содержится перечислимый тип `LanguageType`. В состав класса входят три общих метода: `GreetingsEnglish`, `GreetingsEspanol`, `GreetingsDeutsch`, а в его внутренний интерфейс включена функция `GetGreetingsText`, компактно хранящая все литеральные строки в одном месте и выбирающая нужную.



Пользователь объекта класса `Firstciass` не сможет вызвать метод `GetGreetingsText`. Эта функция использует оператор `Select Case` для выбора строки текста.

Поскольку внутренние члены класса не используются за пределами его модуля, вы как автор имеете возможность смело изменять их названия и способы реализации, не рискуя нарушить внешний код.

Например, можно изменить код трех методов внешнего интерфейса, вызвав новый `Private`-метод, `GetFastGreetingsText`, и удалив метод `GetGreetingsText`. Эффект воздействия такого рода изменений ограничен рамками модуля класса. Члены внутреннего интерфейса класса недоступны извне, поэтому их модификация или даже удаление (если эти операции корректны сами по себе, в контексте модуля класса) никак не скажутся на работоспособности других модулей.

Листинг 21.4 представляет более эффективный вариант реализации класса `Firstciass`.

#### Листинг 21.4. Класс `Firstciass` - а может быть, вот так?

```

1: Option Compare Database
2: Option Base 0
3: Option Explicit
4: Enum LanguageType
5:     ItTechnoid
6:     ItEspanol
7:     ItDeutsch
8: End Enum
9: Public Sub GreetingsAufDeutsch( )
10:    MsgBox GetFastGreetingsText( ItDeutsch )
11:End Sub
12:Public Sub GreetingsEspanol( )

```

```

13: MsgBox GetFastGreetingsText( ltEspanol )
14:End Sub
15:Public Sub GreetingsTechnoid( )
16: MsgBox GetFastGreetingsText( ltTechnoid )
17:End Sub
18:Private Function GetFastGreetingsText(ByVal Language As _
        LanguageType) As String
19:   GetFastGreetingsText = Array( "Hello!", "Buenos días!", _
20:       "Guten tag!") (Language)
21:End Function

```

### Анализ

Листинг 21.4 почти идентичен предыдущему, за исключением изменений в внутренней функции класса — теперь она получила название `GetFastGreetingsText`. Параметр функции (значение `Language` перечислимого типа `LanguageType`) выступает в роли индекса массива, позволяющего заменить громоздкую управляющую структуру `Select Case` ... `End Case` простым и изящным выражением присваивания.

Итак, изменения, вносимые в скрытый интерфейс класса, не оказывают влияния на внешний код.

Если же исправлениям подвергаются внешние методы класса, потребуется пересмотреть и протестировать код других модулей, в которых содержатся обращения к объектам рассматриваемого класса.



Когда создается код класса, вы являетесь производителем, а когда используется класс — пользователем. Помните об этой особенности при создании классов. Даже если вы единственный потенциальный пользователь класса, сделайте его более простым в применении посредством скрытого интерфейса.

## Определение свойств класса

Данные — это то, с чем работает программа. Если информация содержит ошибки, результат окажется неверным. Издавна программисты придерживаются мнения, что данные должны проверяться. Например, произвольная комбинация цифр не обязательно дает в результате правильную дату, -10 не может быть значением возраста, а 1000 — числом одновременно продаваемых акций компании Microsoft. Эти примеры элементов данных верны далеко не во всех случаях; поэтому информация подлежит проверке.

На первых порах для проверки данных использовались функции. К несчастью, такой подход порождал существенные проблемы. Что произойдет, если функции, которая должна проверять корректность информации, в программе просто нет? Либо она не используется? Ведь так велик соблазн избежать лишней работы и пренебречь необходимостью проверки данных, положившись на "авось".

Решение пришло — в виде концепции *свойств* и *методов* *Property*. Свойства используются для хранения данных в составе класса; им легко по мере необходимости поставить в соответствие специальные методы того же класса, обозначаемые служебным словом `Property` и предназначенные, в частности, для выполнения автоматической проверки корректности данных.

Методы `Property` обладают собственным уникальным синтаксисом. Один из двух методов используется для чтения значений соответствующего свойства, а другой — для их записи. Листинг 21.5 представляет пример нового класса, в котором определены два свойства, `Name` и `EmailAddress`, и соответствующие им пары методов `Property`.

## Листинг 21.5. Пример класса Contact, содержащего определения методов Property

```
1: Option Compare Database
2: Option Explicit
3: Private FName As String
4: Private FEmailAddress As String
5: Property Get Name( ) As String
6:     Name = FName
7: End Property
8: Property Let Name( ByVal Value As String )
9:     FName = Value
10:End Property
11:Property Get EmailAddress( ) As String
12:    EmailAddress = FEmailAddress
13:End Property
14:Private Sub RaiseError( ByVal MESSAGE As String )
15:    Call Err.Raise(vbObjectError, "Contact", Message)
16:End Sub
17:Private Sub ValidateEmailAddress(ByVal EmailAddress As String)
18:    Const Message = "Неверный почтовый адрес"
19:    If (InStr( 1, EmailAddress, "@" ) = 0) Then _
        RaiseError(Message & EmailAddress)
20:End Sub
21:Property Let EmailAddress( ByVal Value As String )
22:    Call ValidateEmailAddress( Value )
23:    FEmailAddress = Value
24:End Property
```

### Анализ

Напомним: чтобы создать класс Contact, код которого приведен в листинге 21.5, следует открыть окно редактора Microsoft Visual Basic и выбрать в меню команду **Insert⇒Class Module**. Строки 3 и 4 содержат объявления свойств класса — переменных FName и FEmailAddress типа String. Они снабжены квалификатором Private, чтобы исключить возможность непосредственного доступа извне. Мы предлагаем пользователям класса альтернативный вариант — методы Property.

В строках 5–7 размещен текст Property-метода Get для свойства FName. Предположим, что создан объект MyContact класса Contact. Тогда к свойству можно обратиться опосредованно, с помощью конструкции MyContact.Name. Когда она употребляется в качестве правостороннего операнда, вызывается Property-метод Get Name и срабатывает строка кода 6. Строки 8–10 определяют код метода записи, Let. Метод Let вызывается в том случае, если свойство используется в качестве левостороннего операнда выражения. Следующие строки кода демонстрируют обращение к свойству из левой и правой частей выражения:

```
MyContact.Name = "Noah Kimmel"
MsgBox MyContact.Name
```

Приведенные строки выглядят как обычные обращения к переменной, предполагающие присваивание и использование ее текущего значения (это, собственно, нам и нужно), но на самом деле при их выполнении программа вызывает соответствующие Property-методы, Let и Get. Все, как и ранее, удобно, к тому же, может быть проведена автоматическая проверка данных.

В строках 11–13 и 21–24 содержатся методы Get и Let, обеспечивающие доступ к свойству FEmailAddress. (Обратите внимание, что синтаксис аналогичен рассмотренному выше. Property-метод Get оформляется в виде функции, возвращающей значе-

ние требуемого типа, а **Property-метод** `Let` имеет структуру процедуры, которой передаются аргументы.) Строка 22 демонстрирует, как данные, которые предполагается присвоить переменной `FEmailAddress`, подвергаются незаметной для пользователя проверке. Следующая строка иллюстрирует возможную операцию присваивания:

```
MyContact.EmailAddress = "pkimmel@hotmail.com"
```

В этот момент выполняется код строки 22, вызывающей процедуру `ValidateEmailAddress`, которой передается значение параметра `"pkimmel@hotmail.com"`. (Обратите внимание, что процедура `ValidateEmailAddress` объявлена выше, в строках 17–20, как скрытый (`Private`) член класса.) Если результат проверки успешен, в строке 23 значение присваивается внутренней переменной `FEmailAddress`.

Методы `Property`, будучи простыми в использовании, предоставляют широкие возможности управления данными. Вы можете построить только те конструкции проверки, которые отвечают природе конкретного элемента данных, и сосредоточить их компактно, внутри соответствующего метода `Property`.

Еще одно преимущество механизма обращения к свойствам класса посредством методов `Property` состоит в том, что они позволяют ограничить круг возможных операций, выполняемых со свойством. Например, чтобы обеспечить доступ к переменной только по чтению, достаточно просто "забыть" об определении соответствующего метода `Let`. Если же необходимо, чтобы пользователь, наоборот, мог только присваивать значения, но не считывать их, следует отказаться от построения метода `Get`. Создавая класс, объявляйте все свойства с помощью квалификатора `Private`, а для доступа к ним предлагайте пары соответствующих методов `Get` и `Let`. Концепция методов `Property` весьма эффективна, и именно благодаря ей получили развитие средства визуального программирования и быстрой разработки приложений.

## Использование свойств объектов

Среди свойств классов могут быть как переменные простых типов, так и объекты других классов. Конструкция объявления метода `Property`, используемого для чтения содержимого переменной простого типа или объекта, снабжается служебным словом `Get`. При создании `Property-методов`, предназначенных для присваивания значений свойствам простых типов, используется служебное слово `Let`. Но его нельзя употреблять, если речь идет о свойстве, представляющем собой объект класса. В этом случае применяется служебное слово `Set`.

Таким образом, `Set` заменяет `Let` в тех случаях, когда в качестве свойства используется объект класса. Например, чтобы построить несколько `Property-методов` для доступа к свойству, представляющему собой объект класса `Collection`, следует употребить служебные слова `Get` и `Set`.

## Статические свойства

Создавая класс, вы определяете пользовательский тип. Построив экземпляр класса, вы получаете объект этого типа. Каждый объект обладает собственным набором всех элементов данных (свойств) и методов класса. Поэтому для двух объектов класса `Contact` — скажем, `MyContact` и `YourContact` — выражения `MyContact.Name` и `YourContact.Name` будут ссылаться, как правило, на два совершенно разных элемента данных.



Разница между классами и объектами примерно такая же, как между и клонами. ДНК — это набор генов, а каждый отдельный набор генов — это клон. Класс — это ДНК, а объект — клон.

Статические члены класса отличаются от остальных и периодом "жизни". Обычные свойства и методы доступны только после создания объектов класса и исключительно в контексте последних. Но к статическим членам класса разрешается обращаться и в отсутствие объектов. Листинг 21.6 демонстрирует пример использования статического члена класса.

#### Листинг 21.6. Пример объявления и использования статического члена класса

```
1: Static Property Get UniqueID( ) As Integer
2:   Dim ID As Integer
3:   ID = ID + 1
4:   UniqueId = ID
5: End Property
```

#### Анализ

При каждом выполнении программы, содержащей данный код, значение переменной ID увеличивается на 1. Это значение хранится в период между обращениями к UniqueID.

Статические атрибуты класса допускают самое разнообразное применение и оказываются довольно необходимыми, например, для хранения уникальных идентификаторов таблиц базы данных или экземпляров класса.

Используйте квалификатор Static на уровне свойства, когда требуется, чтобы все переменные свойства были статическими.

## Обработчики событий создания и удаления объекта

В каждом классе содержатся две предопределенные процедуры, которые можно использовать для задания последовательности действий, выполняемых в ответ на события инициализации и удаления объекта. Процедура инициализации обрабатывает событие создания объекта, а код обработчика события удаления объекта вызывается, когда переменной-объекту присваивается значение Nothing. Эти процедуры срабатывают только в том случае, если в них содержится какой-либо код. Явно вызывать их не следует.

Подпрограммы обработки событий создания и удаления объекта носят стандартные названия Class\_Initialize и Class\_Terminate. Чтобы получить возможность создания кода любой из них, выполните следующие действия.

1. Откройте окно редактора Microsoft Visual Basic.
2. Выберите существующий или создайте новый модуль класса.
3. Откройте список Object, расположенный в левой верхней части окна (в нем изначально отображается слово (General)), и выберите элемент Class (рис. 21.1).
4. Откройте список Procedure, находящийся в правой верхней части окна (см. рис. 21.1), и выберите один из элементов — Initialize или Terminate — в зависимости от того, обработкой какого события, создания объекта либо его удаления вы намерены заниматься.

В результате выполнения указанной инструкции вы получите пустой шаблон обработчика. Остается самая малость — наполнить его. Процедура Class\_Initialize — как раз то место, где удобно расположить фрагмент кода, который должен выполняться в момент создания объекта, срабатывать раньше других блоков кода и/или действовать только один раз. Код процедуры Class\_Terminate выполняется в последнюю очередь, непосредственно перед удалением объекта из памяти. Естественно, он также срабатывает один раз.

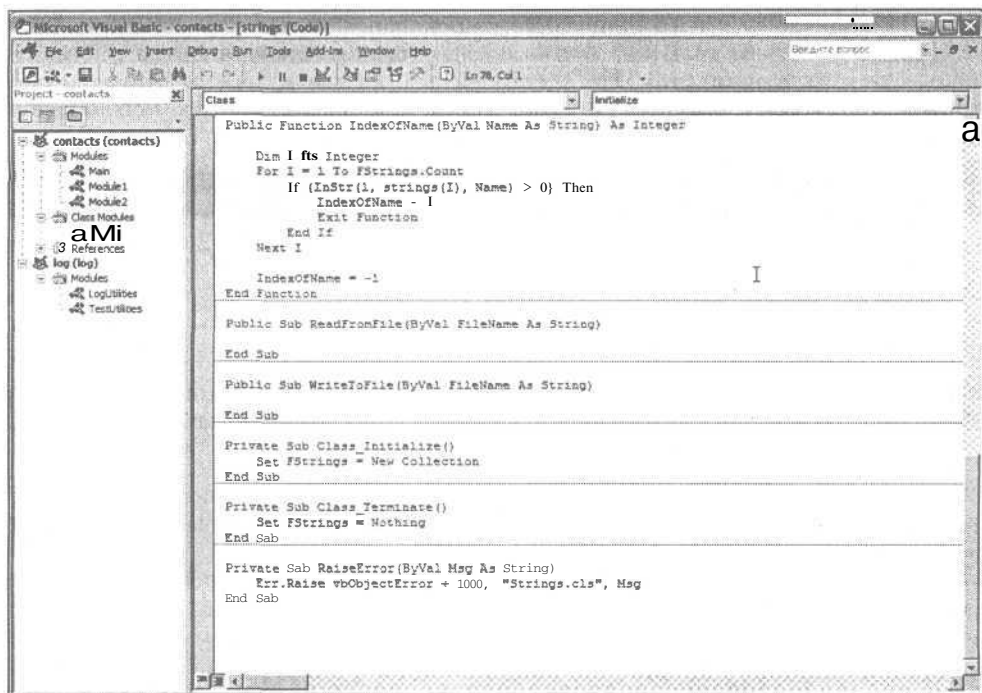


Рис. 21.1. Окно редактора Visual Basic позволяет быстро составить шаблоны обработчиков системных событий создания и удаления объектов класса

Названия рассмотренных именованных блоков кода достаточно прозрачны — процедура `Class_Initialize` удобна для выполнения некоторых операций инициализации, например открытие баз данных или иных файлов; а обработчик `class_Terminate` — как раз то место, где имеет смысл располагать директивы закрытия баз данных, файлов и т.п. Пример использования процедуры `Class_Initialize` содержится в листинге 21.7, приведенном в следующем разделе.

## Создание нового класса

Настоящий раздел посвящен рассмотрению примера построения реального и — с моей точки зрения — полезного. Класс, которому дано название `FileStream`, реализует механизм файлового потокового ввода-вывода. Методы класса обеспечивают интерфейс, удобный для выполнения операций открытия и закрытия файлов, чтения и записи текстовых данных, перемещения к заданной позиции внутри файла. Листинг 21.7 содержит полное определение класса `FileStream`.

### Листинг 21.7. Класс, реализующий операции файлового потокового ввода-вывода

```
1: Option Compare Database
2: ' FileStream.Bas -- Defines a file-management streaming class
3: ' Copyright © 1999. All Rights Reserved.
4: ' By Software Conceptions, Inc.
5: ' Written by Paul Kimmel, Okemos, MI USA 800-471-5890
6: Option Explicit
```

```

7: Private FFileName As String
8: Private FHandle As Double
9: Private Type Buffer
10:   S As String
11:End Type
12:Public Property Get Count( ) As Long
13:   Count = FileLen( FFileName )
14:End Property
15:Public Property Get FFileName( ) As String
16:   FFileName = FFileName
17:End Property
18:Public Property Let FFileName( ByVal Value As String )
19:   FFileName = Value
20:End Property
21:Public Property Get FHandle( ) As Double
22:   FHandle = FHandle
23:End Property
24:Public Property Get Position( ) As Long
25:   Position = Seek( FHandle )
26:End Property
27:Public Property Let Position( ByVal Value As Long )
28:   Call SeekStream( Value )
29:End Property
30:Private Sub CheckFile( ByVal FFileName As String )
31:   If (FileExists( FFileName ) = False) Then
32:       Err.Raise 1, Me, "Invalid filename " & FFileName
33:   End If
34:End Sub
35:Private Sub Class_Initialize( )
36:   FHandle = 0
37:End Sub
38:Public Sub CloseStream( )
39:   Close #FHandle
40:   FHandle = 0
41:End Sub
42:Private Function FileExists(ByVal FFileName As String) As Boolean
43:   FileExists = Len( Dir( FFileName ) ) > 0
44:End Function
45:Public Sub OpenStream( ByVal FFileName As String )
46:   FFileName = FFileName
47:   FHandle = FreeFile
48:   Open FFileName For Random As #FHandle Len = 1
49:End Sub
50:Public Sub ReadStream(ByVal Str As String, ByVal Length As Long)
51:   Dim I As Integer
52:   Dim Buf As Buffer
53:   For I = 1 To Length
54:       Get #FHandle, , Buf
55:       Str = Str + Buf.S
56:   Next I
57:End Sub
58:Public Sub SeekStream( ByVal Length As Long )
59:   On Error GoTo EXCEPT
60:   Seek #FHandle, Length
61:   Exit Sub
62: EXCEPT:
63:   If (Length = 0) Then Resume Next

```



```

64: Err.Raise Err.Number, Err.Source, Err.Description,
    Err.HelpFile, Err.HelpContext
65:End Sub
66:Public Sub WriteStream( ByVal Str As String )
67:    Dim I As Long
68:    Dim Buf As Buffer
69:    For I = 1 To Len( Str )
70:        Buf.S = Mid$( Str, I, 1 )
71:        Put #FHandle, , Buf
72:    Next I
73:End Sub

```

#### Анализ

Класс `FileStream` содержит определения всех методов, необходимых для выполнения операций чтения и записи файловых данных. Класс содержит четыре свойства: `Count`, `FileName`, `Handle` и `Position`. `Handle` и `Count` — это переменные, которые позволяют только чтение, а также предназначены для хранения номера открытого файла и счетчика символов соответственно. `FileName` — имя файла, а `Position` — текущее значение позиции внутри файла, относительно которой будет выполняться очередная операция чтения или записи данных. `Property`-метод `Position` позволяет свободно перемещаться внутри файла при считывании/записи информации.

В составе класса определено восемь методов: `CheckFile`, `Class_Initialize`, `CloseStream`, `FileExists`, `OpenStream`, `ReadStream`, `SeekStream` и `WriteStream`. `CheckFile` — это внутренняя процедура, в отсутствие файла открывающая окно сообщения об ошибке. Обработчик `Class_Initialize` выполняет инициализацию переменной `FHandle`, предназначенной для хранения номера открытого файла. Метод `CloseStream` закрывает файл, заданный номером `FHandle`. `FileExists` обеспечивает непосредственное выполнение проверки существования файла. Процедура `OpenStream`, получающая имя файла, сохраняет его во внутренней переменной и открывает файл в режиме `Random`. Метод `ReadStream` используется для чтения указанного числа символов, начиная с текущей позиции в файле. Процедура `SeekStream` дает возможность переместить указатель позиции в файле на заданное число символов относительно текущего положения. Метод `WriteStream` осуществляет вывод данных в файловый поток.

Применяемые в совокупности, перечисленные выше методы позволяют комплексно решать задачи управления файловым вводом-выводом. Листинг 21.8 демонстрирует пример практического использования класса `FileStream`.

#### Листинг 21.8. Пример использования класса `FileStream`

```

1: Sub TestFileStream( )
2:    Dim Stream As New FileStream
3:    Call Stream.OpenStream( "Test.txt" )
4:    Call Stream.WriteStream( "Это тестовый пример" )
5:    Call Stream.CloseStream
6:    Set Stream = Nothing
7: End Sub

```

#### Анализ

Строка 2 листинга 21.8 содержит конструкцию объявления и создания объекта `Stream` класса `FileStream`; при этом срабатывает процедура `Class_Initialize`. При выполнении строки 3 открывается файл с именем `Test.txt`. Команда строки 4 осуществляет запись данных в файл, а в строке 5 файл закрывается. Строка 6 содержит директиву удаления объекта

Stream и освобождения **отведенного** для него фрагмента памяти — если бы в составе класса была реализована процедура `Class_Terminate`, то в этот момент она была бы автоматически вызвана.



Примите к сведению: существует объект ActiveX под названием `Scripting.FileSystemObject`. Чтобы воспользоваться им в прикладной программе, необходимо объявить соответствующую переменную и вызвать функцию `CreateObject`. Другой вариант — следует добавить ссылку на библиотеку `Microsoft Scripting Runtime Library` и создать экземпляр объекта `FileSystemObject` с помощью ключевого слова `New`. Примеры использования объекта `Scripting.FileSystemObject` вы найдете в оперативной справочной системе `Microsoft Visual Basic`, задав в качестве ключевого слова поиска строку `FileSystemObject`.

Класс `FileStream`, рассмотренный в этом разделе, достаточно полезен и эффективен. Но существует также и более развитый и гибкий стандартный класс — `FileSystemObject`. Это весьма удачная альтернатива, если в недалеком будущем ваши потребности превзойдут уровень возможностей, предоставляемых классом `FileStream`.

Мощь объектно-ориентированного подхода к программированию заключается, в частности, в том, что он позволяет сосредоточить внимание на целостной проблеме. Код класса обладает высокой степенью переносимости (поскольку размещается в отдельном модуле и решает задачу комплексно) и управляемости — вы можете, например, легко исправлять внутренний интерфейс класса, не повредив внешний код.

## Тестирование класса

Создаваемые классы следует обязательно тестировать. В этой ситуации вполне применимы все методы и приемы, рассмотренные ранее в главах "17-й час. Отладка кода" и "18-й час. Обработка ошибок во время выполнения программы". По завершении проектирования класса постройте тестовую процедуру, которая позволит "прощупать" все его методы и свойства — если класс протестирован, смело создавайте его объекты, не боясь ошибок.

## Расширение возможностей существующих классов

Один из важных аспектов объектно-ориентированного программирования связан с возможностью агрегации классов (т.е. создания новых классов, расширяющих потенциал существующих). Под *агрегацией* в данном случае понимается включение класса в состав атрибутов нового класса.

Глава "22-й час. Совершенствование типов данных" содержит примеры расширения набора функциональных средств класса за счет агрегации кода.

## Создание экземпляра класса

Приемы построения экземпляров стандартных и пользовательских классов абсолютно одинаковы. Следующая строка кода демонстрирует команду создания объекта Stream класса `FileStream`:

```
Dim Stream As New FileStream
```



Процесс создания объекта путем добавления ссылки с применением ключевого слова *New* называется *ранним связыванием* (на этапе компиляции). Вызов *CreateObject* с присвоением значения переменной типа *Variant* называется *динамическим связыванием*.

При использовании раннего связывания редактор Visual Basic предлагает список свойств и методов объекта, поскольку тип объекта известен уже на этапе разработки. Для динамически связанных объектов такого рода помощью редактор не располагает.

Объекты ActiveX требуют применения функции *CreateObject* с передачей ей строки наименования класса. Листинг 21.9 содержит пример создания ActiveX-объекта Excel. (Предварительно включите библиотеку Microsoft Excel 10.0 Object Library в состав проекта — для этого выберите в строке меню окна редактора VBA команду *Tools⇒References* и в одноименном диалоговом окне установите соответствующий флажок.)

Листинг 21.9. Пример создания ActiveX-объекта Excel

```
1: Sub CreateExcelInstance( )
2:   Dim Excel As Object
3:   Set Excel = CreateObject( "excel.application" )
4:   Excel.Visible = True
5:   Excel.Quit
6:   Set Excel = Nothing
7: End Sub
```

#### Анализ

Строка 2 листинга 21.9 содержит конструкцию объявления объекта с именем *Excel*. (Думается, это название достаточно подходящее.) В строке 3 вызывается функция *CreateObject*, которой в качестве аргумента передается строка с наименованием класса — *"excel.application"*. С помощью выражения строки 4, содержащего обращение к Property-методу *Visible*, объект отображается на экране, а затем сеанс работы Excel завершается (строка 5) и объект удаляется из памяти (строка 6). Аналогичным образом можно создать в среде прикладной программы экземпляры Word, Outlook и даже Access (!), поскольку функция *CreateObject* позволяет строить объекты любых серверов OLE Automation.

OLE Automation — одно из достижений протокола Component Object Model (COM). Сервер OLE Automation представляет собой программу (построенную в соответствии с требованиями стандарта COM), которая может предоставить в общее пользование часть своего интерфейса. Серверы OLE Automation обеспечивают сервис для других программ или используются в качестве самостоятельных приложений. Access 2002 — это полноценный сервер OLE Automation.

## Резюме

Простота и сложность объектно-ориентированного подхода к программированию заключается в огромных возможностях создания новых полезных типов данных. В отсутствие классов не было бы *ADODB* и компонентов, задача создания графических интерфейсов оставалась бы чрезвычайно сложной и — куда уж страшнее! — не появился бы язык программирования VBA. VBA — это дитя, плоть от плоти, объектной модели.

На прошлых занятиях рассказывалось о том, как пользоваться готовыми классами. Но теперь вы изучили способы создания собственных. Класс — это тип данных, определяемый пользователем-программистом. Код класса сохраняется в отдельном модуле. Классы состоят из свойств и методов, в совокупности называемых атрибутами, или членами. Свойства — это элементы данных, а методы — правила их обработки.

Уровень доступности атрибутов класса задается с помощью служебных слов-квалификаторов `Public` и `Private`. Члены класса, помеченные словом `Private`, доступны исключительно для методов этого же класса, и забота о них целиком возложена на плечи автора-программиста. Атрибуты `Public` служат частью внешнего, общего интерфейса класса.

Объектно-ориентированный подход предоставляет большое количество преимуществ и целым коллективам программистов, работающих над серьезными проектами, и даже любителям-одиночкам, позволяя концентрировать внимание на самых значительных частях общей задачи. Классы VBA предоставляют вам все необходимое.

Следующая глава, "22-й час. Совершенствование типов данных", посвящена рассмотрению ряда усовершенствованных возможностей объектно-ориентированного подхода, причем в ходе ее изложения мы будем опираться на материал текущего занятия. Не упустите случая закрепить свои "объектно-ориентированные" навыки и ознакомьтесь с разделами "Вопросы и ответы" и "Задания", приведенными ниже.

## Вопросы и ответы

**Вопрос.** Как только процедуры и функции становятся частью класса, их перестают называть процедурами и функциями. Почему?

**Ответ.** Это не просто игра слов. Новые термины подразумевают расширенное толкование. Методы — да, это процедуры и функции, но за ними стоит кое-что еще. `Property`-методы можно трактовать и как данные. А свойства — это больше чем обычные элементы информации. Новая терминология требует и нового осмысления. Без нее мы не сможем понимать друг друга.

**Вопрос.** Так ли принципиален вопрос освоения способов объектно-ориентированного программирования в контексте использования Access?

**Ответ.** И да, и нет. Практически все, с чем вы имеете дело, работая в Access, связано с теми или иными аспектами объектно-ориентированного программирования. Разумеется, можно заниматься программированием для Access, ограничившись применением существующих классов, — для решения многих задач этого достаточно. Но наиболее эффективные результаты достигаются только тогда, когда вы способны использовать для решения конкретной проблемы все имеющиеся возможности.

**Вопрос.** Должен ли я досконально овладеть всеми идиомами объектно-ориентированного подхода, чтобы суметь построить собственный класс?

**Ответ.** Нет. На этом занятии были рассмотрены базовые понятия и приемы, и теперь вы вполне в состоянии создавать собственные классы, хотя в VBA существуют и более сложные процессы. Важно понимать, что различные объектно-ориентированные языки программирования далеко не равноценны. В `Object Pascal` или `C++`, например, поддерживается множество других идиом, и поэтому каждый из них гораздо более труден для изучения — разумеется, они более мощные. Access VBA предлагает разумный компромисс между гибкостью и эффективностью, с одной стороны, и простотой использования — с другой.

**Вопрос.** Как узнать, когда именно следует создавать класс?

**Ответ.** Хороший вопрос. Далеко не обязательно, чтобы весь код программы принадлежал классам. Если вы сталкиваетесь с повторяющейся задачей, охватывающей

определенный круг данных и способов их обработки, соответствующая группа строк кода становится подходящей "кандидатурой" для оформления в виде класса. Процесс перехода структурного кода в объектно-ориентированный может происходить постепенно. Наглядный пример — класс `FileStream`. Работа с файлами предполагает наличие средств их открытия, чтения, записи и т.п. И со временем вполне естественно возникает мысль об объединении всех требуемых элементов данных и процедур в единое целое, класс.

## Задания

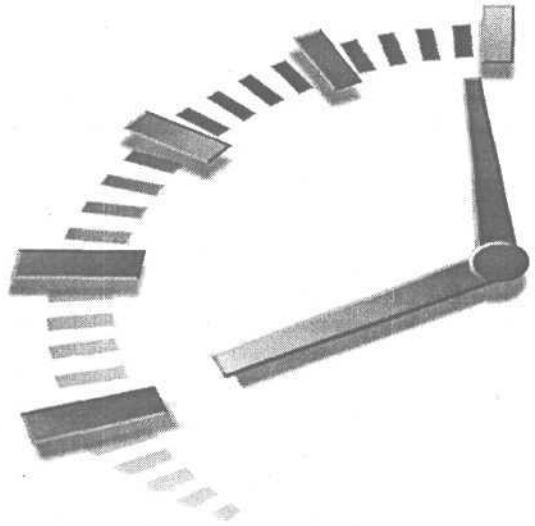
Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Тип и переменная находятся в такой же взаимосвязи, как класс и ... (вставьте пропущенный термин).
2. Класс может содержать свойства, для которых не определены методы `Property`. Верно ли это?
3. Каким обобщенным термином обозначаются процедуры и функции-члены класса?
4. Перечислите действия, необходимые для создания нового модуля класса.
5. С помощью каких служебных слов обозначаются `Property`-методы, предназначенные для чтения переменных и присвоения им значений?

## Упражнения

1. Определите класс, содержащий переменные-свойства для хранения данных о телефонных номерах и именах их владельцев.
2. Наберите код класса `FileStream` в окне редактора. Постройте тестовую процедуру, которая в целях проверки корректности кода класса должна записывать в файл строку текста, возвращать указатель в начало файла и считывать ту же строку.
3. Примените объект `Scripting.FileSystemObject` для чтения файла с данными, сохраненными при выполнении предыдущего упражнения.



## 22-й час

# Совершенствование типов данных

Анализ в программировании — это процесс постижения проблемы. Синтез — ее реализация. Примеры формального анализа возникают там, где аналитик, общаясь с заказчиком или пользователем будущей программы, выясняет природу проблемы. Неформальный подход к анализу проявляется в тех случаях, когда вы сами, совершенно самостоятельно решаете, что для достижения результата должна быть написана определенная программа. Подобные примеры характерны для ситуаций, когда количество участников рабочей группы не превышает двух-трех человек. В крупномасштабных проектах анализ может выполняться одними людьми, а практическая реализация принятых решений — другими. В настоящий момент вы будете выступать и в роли аналитика, и в роли программиста.

Существует опасность бессистемного создания классов, а также бессмысленного их наполнения. На этом занятии речь идет о том, как создать удачный класс и его объекты.

Основные темы занятия.

- Критерии "классового" подхода.
- Принципы объектно-ориентированного проектирования.
- Расширение возможностей существующих классов посредством агрегации.
- Использование новых классов.

## Определение цели

На этом занятии основной причиной, побуждающей строить новый класс, будем считать потребность в практической реализации ассоциативного массива. *Ассоциативным* называется массив, в котором каждому значению соответствует именованный ключ. Об ассоциативном массиве можно говорить как о разновидности базы данных. Окружающая действительность предлагает немало образцов задач, нуждающихся в решении с помощью ассоциативных массивов. Знакомые всем файлы INI, предназна-

ченные для хранения информации о параметрах настройки программ, — это примеры ассоциативных массивов. Такой же структурой обладают и базы данных реестров операционных систем Windows 95, 98, NT и 2000. Реестр содержит ключи и соответствующие им значения. Если вы никогда не видели, как выглядит реестр Windows, щелкните на кнопке Пуск (Start) панели задач, выберите в меню элемент Выполнить (Run) и посредством команды RegEdit загрузите программу Редактор реестра. Не вносите в реестр никаких изменений, поскольку это может привести к разного рода нарушениям работоспособности операционной системы и прикладных программ.

Класс Collection также можно считать разновидностью ассоциативных массивов, поскольку каждому элементу коллекции вы ставите в соответствие определенный ключ. В данном случае, правда, существует одно ограничение — у программиста нет доступа к ключам как к значениям. Вы должны заранее знать значения ключей, чтобы суметь ими воспользоваться. Далее вы узнаете, как применять Collection в новом классе для расширения его возможностей. Чтобы облегчить задачу сохранения ассоциативного массива в файле и чтения его из файла, добавим средства класса FileStream, который мы построили в главе "21-й час. Основы программирования классов".

## Когда именно следует создавать классы

Никто не запрещает создавать классы при каждом удобном случае. Если говорить о формальной стороне вопроса, вы можете построить класс, содержащий единственный элемент данных, хотя, разумеется, это нельзя назвать удачным решением. В качестве общего правила можно указать следующее: целесообразно рассматривать возможность построения класса в тех случаях, когда вы способны определить три или более порций данных и предложить, по меньшей мере, столько же методов их обработки, полезных для решения определенной частной задачи.



Помните о различии классов и объектов. *Класс* определяет вид и поведение отдельного элемента. Термин *объект* часто используют там, где с технической точки зрения, правильнее было бы применить термин *класс*. Как уже указывалось в начале этой главы, совершенно не обязательно определять классы во всех приложениях в Access.

Впрочем, если вы заглянете в окно оперативной справочной системы и отыщете тему, посвященную классу Collection, то обнаружите, что этот класс содержит всего одно свойство и три метода. Тем не менее, класс Collection находит самое широкое применение. С другой стороны, неплохо, если класс содержит не более десятка общих методов, поскольку в противном случае он становится слишком сложным для восприятия и использования. Число скрытых членов класса особого значения не имеет, поскольку пользователи могут даже не знать об их существовании.

Если в программе повторяется один и тот же фрагмент кода, эта программа *дивергентная*. Код должен быть сходящимся, т.е. каждый отдельный блок кода должен появляться в программе только один раз. Если набор из нескольких строк встречается в коде неоднократно, попробуйте организовать этот фрагмент в виде отдельного класса. Такой подход не только уменьшает объем кода, но и облегчает его сопровождение.

Конечно, это самые общие правила, поэтому еще раз сформулируем основные критерии пригодности задачи к тому, чтобы быть оформленной в виде класса, и общие количественные требования к структуре класса.

- Существует три или более элементов данных, совместно описывающих частную задачу.
- Количество членов внешнего интерфейса класса не превышает десятка.
- Удастся обосновать необходимость нарушения двух предыдущих правил.

Если вы станете создавать классы только в тех случаях, которые укладываются в предложенную схему, вы достигнете вполне приемлемых результатов.

Теперь вернемся к поставленной цели — нам необходим класс ассоциативной коллекции, содержащей пары ключевых имен и значений и допускающей возможность файлового ввода-вывода данных.

## Принципы объектно-ориентированного проектирования

Один из основных принципов *объектно-ориентированного проектирования* состоит в необходимости построения классов, которые способны предоставить решение задач, относящихся к определенной *области*. Мы, например, собираемся решить задачу хранения пар данных, состоящих из имени и значения, обеспечив возможность доступа к значению по имени или по индексу и, наоборот, к имени по значению.



С объектно-ориентированным программированием неразрывно связаны дисциплины объектно-ориентированного *проектирования* и *анализа*, обладающие собственными языками, методами и инструментальными средствами. Одна из самых популярных систем такого рода — Rational Rose.

Проектирование во многом сводится к процессу моделирования. Стандартом языков моделирования служит Unified Modeling Language (UML). UML использует собственные средства описания классов и взаимоотношений между ними. За более подробной информацией о языке UML и способах его применения обращайтесь к специальной литературе (например, к книге *Teach Yourself UML in 24 Hours*, изданной Sams).



Объектно-ориентированное проектирование — это самостоятельная область знаний, отличная от собственно программирования. Конечно, если проект прост и реализуется силами небольшого коллектива, стадии проектирования и программирования часто неразрывно связаны, чего не скажешь о крупномасштабных и сложных проектах, когда этапу программирования предшествуют серьезные и длительные процессы анализа, моделирования и проектирования.

Основной принцип объектно-ориентированного проектирования заключается в определении набора свойств и методов, необходимого и достаточного для решения определенной задачи с помощью будущего класса. (На этом этапе программисту помогут лист бумаги и карандаш.) Ниже перечислены все требования к классу ассоциативных массивов, который будет проектироваться.

## Проектирование класса ассоциативных массивов

Класс ассоциативных массивов, который необходимо построить, предназначен для хранения пар символьных значений. Поэтому уместно назвать его Strings. Класс обязан обеспечить реализацию возможностей, перечисленных ниже.

- Хранение строк.
- Доступ к значениям по ключевым именам.



- Доступ к именам по значениям.
- Чтение строк из файла.
- Запись строк в файл.
- Редактирование значений.

Теперь, когда круг задач определен, имеет смысл придумать названия для соответствующих членов класса и решить, *что* именно должен представлять собой каждый из них — свойство или метод.

Нам необходима возможность доступа к строкам по индексу, поэтому символьное значение должно быть свойством, допускающим индексацию. Кроме того, придется обращаться к строкам по ключевому имени, которое также следует представить в виде свойства. Указанное замечание справедливо и в отношении ключей — их надлежит оформить как свойства. Теперь подытожим: нам необходимы отдельные свойства `Names` и `Values`, а также символьная переменная `Strings`, предназначенная для одновременного хранения ключевого имени и значения. В наименовании свойства `Strings` форма множественного числа существительного выбрана по той причине, что это индексированное значение — в объекте класса их может быть несколько.

Класс должен поддерживать возможность выполнения операций чтения и записи файловых данных. Поэтому потребуются два соответствующих метода — `ReadFromFile` и `WriteToFile`. Табл. 22.1 представляет описание элементов внешнего интерфейса класса `Strings`.

Таблица 22.1. Описание внешнего интерфейса класса `Strings`

Имя	Свойство или метод	Описание
<code>Names</code>	Свойство	Ключевое имя
<code>Values</code>	Свойство	Значение
<code>Strings</code>	Свойство	Общая строка данных
<code>ReadFromFile</code>	Метод	Процедура чтения данных из файла
<code>WriteToFile</code>	Метод	Процедура записи данных в файл

Табл. 22.1 в общих чертах представлена структура класса `strings`. Перечисленные здесь свойства и методы образуют внешний интерфейс класса. Какой окажется практическая реализация — зависит от возможностей и желаний программиста (в данном случае — моих).

## Несколько советов по реализации класса

Приступая к проектированию класса, придерживайтесь такого правила: внешний интерфейс должен быть настолько простым, насколько это возможно. Другая составляющая успеха — максимальное использование существующего кода. Такой код заведомо привлекателен по той причине, что большая часть работы уже проделана — код прошел тестирование и проверку временем. В результате новый класс окажется проще в использовании и дешевле в сопровождении.

Теперь, используя данные, приведенные в табл. 22.1, можно составить "скелет" искомого класса — просмотрите листинг 22.1.

Листинг 22.1. Определение членов внешнего интерфейса класса `Strings`

```
1: Option Compare Database
2: Option Explicit
3: Public Property Get Names(ByVal Index As Integer) As String
4:
```

```

5: End Property
6:
7: Public Property Let Names(ByVal Index As Integer, ByVal Name As_
   String)
8:
9: End Property
10:
11: Public Property Get Strings(ByVal Index As Integer) As String
12:
13: End Property
14:
15: Public Property Let Strings(ByVal Index As Integer, ByVal Str As_
   String)
16:
17: End Property
18:
19: Public Property Get Values(ByVal Name As String) As String
20:
21: End Property
22:
23: Public Property Let Values(ByVal Name As String, ByVal Value As_
   String)
24:
25: End Property
26:
27: Public Sub ReadFromFile(ByVal FileName As String)
28:
29: End Sub
30:
31: Public Sub WriteToFile(ByVal FileName As String)
32:
33: End Sub

```

#### Анализ

В строках 1–33 листинга 22.1 определяется *внешний* интерфейс класса `strings`, поскольку все методы снабжены квалификатором `Public`. Вы, разумеется, уже обратили внимание, что сами методы еще не реализованы и класс не содержит приватных членов. Вся оставшаяся часть работы будет выполнена далее.

## Расширение возможностей существующих классов

Удачный способ построения нового класса заключается в расширении функциональных возможностей существующего. Если говорить о классе `strings`, отметим два аспекта проблемы, требующих решения. Во-первых, нам необходима структура данных для хранения значений и, во-вторых, нужны способы файлового ввода-вывода этих данных. Если ничего не помешает, решения непременно будут найдены.

## Использование коллекции для хранения данных ассоциативного массива

Класс `Collection` уже создан, и вполне пригоден для хранения символьной информации. Поэтому его можно рассматривать в качестве подходящей кандидатуры на роль "хранилища" данных (строк, содержащих ключевое имя и значение) в классе

**Strings.** Решив использовать класс **Collection**, следует предложить способы "добычи" данных из него — в листинге 22.1 уже "заготовлены" определения соответствующих **Property**-методов. (Конечно, вместо коллекций можно было бы обратиться к средствам построения массивов, но в последнем случае мы создали бы себе лишние хлопоты — зачем?)

Включение объекта существующего класса в состав нового называют *агрегацией*. Чтобы воспользоваться возможностями коллекций в контексте проектируемого класса, необходимо объявить переменную типа **Collection** и предусмотреть ее инициализацию, т.е. выполнение операции создания объекта класса **Collection**. В данном случае переменная типа **Collection** будет объявлена как приватная. Для ее инициализации и очистки удобно воспользоваться средствами стандартных обработчиков событий **Class\_Initialize** и **Class\_Terminate**.

С учетом сказанного, код класса, знакомый по тексту листинга 22.1, нетрудно наполнить новым содержанием. Результат представлен в листинге 22.2.

#### Листинг 22.2. Расширенный вариант кода класса **Strings**

```
1: Option Compare Database
2: Option Explicit
3: Option Base 1
4:
5: Private FStrings As Collection
6:
7: Public Property Get Names(ByVal Index As Integer) As String
8:     On Error GoTo EXCEPT
9:     Names = Mid$(Strings(Index), 1, _
        InStr(1, Strings(Index), "=") - 1)
10: Exit Property
11: EXCEPT:
12: Call RaiseError(Err.Description)
13: End Property
14:
15: Public Property Let Names(ByVal Index As Integer, _
        ByVal Name As String)
16: On Error GoTo EXCEPT
17: Strings(Index) = Name & "=" & Values(Names(Index))
18: Exit Property
19: EXCEPT:
20: Call RaiseError(Err.Description)
21: End Property
22:
23: Public Property Get Strings( ) As Collection
24: Strings = FStrings
25: End Property
26:
27: Public Property Get Values(ByVal Name As String) As String
28: Dim Index As Integer
29: Index = IndexOfName(Name)
30: If (Index > -1) Then
31:     Values = Strings(Index)
32:     Values = Right$(Values, _
        Length(Values) - InStr(1, Values, "="))
33: Else
34:     Call RaiseError(Name & " не найдено" )
35: End If
36: End Property
```

```

37:
38:Public Property Let Values(ByVal Name As String,
                               ByVal Value As String)
39:  On Error GoTo EXCEPT
40:  FStrings(IndexOfName(Name)) = Name & "=" & Value
41:  Exit Property
42:EXCEPT:
43:  If (Err.Number = 5) Then
44:    FStrings.Add(Name & "=" Value)
45:  Else
46:    Call RaiseError(Err.Description)
47:  End If
48:End Property
49:
50:Public Function IndexOfName(ByVal Name As String) As Integer
51:
52:  Dim I As Integer
53:  For I = 1 To FStrings.Count
54:    If (InStr(1, Strings( I ), Name) > 0) Then
55:      IndexOfName = I
56:      Exit Function
57:    End If
58:  Next I
59:
60:  IndexOfName = -1
61:End Function
62:
63:Public Sub ReadFromFile(ByVal FileName As String)
64:
65:End Sub
66:
67:Public Sub WriteToFile(ByVal FileName As String)
68:
69:End Sub
70:
71:Private Sub Class_Initialize( )
72:  Set FStrings = New Collection
73:End Sub
74:
75:Private Sub Class_Terminate( )
76:  Set FStrings = Nothing
77:End Sub
78:
79:Private Sub RaiseError(ByVal Msg As String)
80:  Err.Raise vbObjectError + 1000, "Strings.cls", Msg
81:End Sub

```

#### Анализ

Как вы заметили, объем кода существенно вырос. Выбрав класс коллекций, мы значительно облегчили задачу программирования. Строка 5 содержит объявление переменной FStrings класса Collection. Объект коллекции инициализируется в момент выполнения директивы New (срабатывает процедура Class\_Initialize, расположенная в строках 71–73), уничтожается он при выполнении команды присваивания значения Nothing (автоматически выполняется процедура Class\_Terminate, строки 75–77).

Если свойство `Names` выступает в роли правостороннего операнда выражения, вызывается соответствующий `Property`-метод `Get` (строка 9), который возвращает часть строки указанного индексом элемента коллекции от начала до символа равенства (=). Если `Names` употребляется в левой части выражения присваивания, срабатывает `Property`-метод `Let` (строки 15-21), перезаписывающий содержимое заданного элемента коллекции символьной строкой, которая состоит из ключевого имени и значения, разделенных символом =.

Реализация свойства `Strings` претерпела изменения. Табл. 22.1 описывает `Strings` как общую строку данных. В начале процесса проектирования класса у меня еще не было предложений по поводу добавления строк в будущее "хранилище" и удаления их. Поскольку в классе `Collection` подобные методы уже реализованы, достаточно было оставить `Property`-метод `Get Strings` (метод `Let Strings` я исключил во избежание непреднамеренного присвоения объекту коллекции нового значения типа `Collection`) и возложить обязанности по изменению элементов коллекции на `Property`-методы `Let` для свойств `Names` и `Values`. Может показаться, что указанные выходы могут способствовать тому, чтобы "задним числом" исправить листинг 22.1.

Но зачем? Он может послужить хорошей иллюстрацией методов работы, поддерживаемых парадигмой объектно-ориентированного программирования. На самом деле, вполне приемлема и даже желательна практика непрерывного – "диалектического" •- улучшения структуры и способов реализации класса по мере постижения проблемы и осознания ее особенностей. В рамках объектно-ориентированного подхода затраченные усилия, направленные на развитие и улучшение программного кода, всецело "окупятся".



Рассматривая возможность использования объекта `Collection` в качестве свойства `Strings` в настоящем приложении, я бы сначала изменил дизайн с целью реализовать методы `Add` и `Remove` как методы класса `Strings`.

Использование в явном виде объекта `Collection` исключает применение интерфейса класса `Strings`.

Еще несколько замечаний. Достаточно полезна специальная функция `IndexOfName` (определенная в строках 50–61). Она вызывается из строк 29 и 40 и возвращает индекс элемента коллекции по значению ключевого имени.

Заслуживает внимания реализация обработчика ошибок в строках 43–47. Если `Property`-метод `Let Value` предпринимает попытку изменения такого элемента коллекции, которого просто нет, выполняется условие обработчика ошибок в строке 43 и вызывается стандартный метод `Add` добавления в коллекцию нового элемента. Все другие обработчики просто констатируют ошибку и обращаются к процедуре `RaiseError`, определенной в строках 79–81.

## Реализация функций файлового ввода-вывода

Для практического воплощения средств файлового ввода-вывода, предусмотренных в интерфейсе класса `Strings`, целесообразно обратиться к готовому решению — классу `FileStream` (мы его строили на прошлом занятии). Листинг 22.3 демонстрирует ту часть кода класса `Strings`, которая содержит тексты процедур потокового вывода данных в файл и чтения их из файла. Если объединить тексты листингов 22.2 и 22.3 воедино, мы получим цельный интерфейс класса, предназначенного для реализации ассоциативного массива с поддержкой файловых операций.

**Листинг 22.3. Методы класса Strings, обеспечивающие выполнение файловых операций**

```
1: Private Sub ParseStrings(ByVal Text As String)
2:   Dim P As Integer
3:   Do While (Len(Text) > 0)
4:     P = InStr(1, Text, vbNewLine)
5:     If (P = 0) Then
6:       Call FStrings.Add(Text)
7:       Exit Sub
8:     Else
9:       Call FStrings.Add(Left$(Text, P - 1))
10:      Text = Mid$(Text, P + Len(vbNewLine))
11:    End If
12:  Loop
13::End Sub
14:
15::Public Sub ReadFromFile(ByVal FileName As String)
16:; Clear
17:; Dim F As New FileStream
18:; Call F.OpenStream(FileName)
19:; Dim Text As String
20:; Call F.ReadStream(Text, F.Count)
21:; Call ParseStrings(Text)
22:; Call F.CloseStream
23:; Set F = Nothing
24::End Sub
25:
26::Public Sub WriteToFile(ByVal FileName As String)
27:; Dim I As Integer
28:; Dim F As New FileStream
29:; Call F.OpenStream( FileName )
30:; For I = 1 To FStrings.Count
31:;   Call F.WriteStream(Strings( I ) & vbNewLine)
32:; Next I
33:; Call F.CloseStream
34:; Set F = Nothing
35:; End Sub
36:
37::Public Sub Clear()
38:; Do While(FStrings.Count >0)
39:;   Call FStrings.Remove(FStrings.Count)
40:; Loop
41::End Sub
```

**Анализ**

Листинг 22.3 демонстрирует пример реализации методов `ReadFromFile` и `WriteToFile`. Добавьте этот текст в состав модуля класса `Strings` (листинг 22.2) — и получите полнофункциональный вариант организации ассоциативного массива, снабженного средствами файлового ввода-вывода. Листинг 22.3 содержит дополнительную процедуру `ParseString`, осуществляющую считывание отдельных строк данных из общего текстового буфера и сохранение их в элементах коллекции. Разделителем строк в файле служит предопределенная константа `vbNewLine`, состоящая из пары символов возврата каретки и перевода строки. В процедурах `ReadFromFile` и `WriteToFile` создаются и применяются объекты класса `FileStream`, построенного на прошлом занятии. Еще одна процедура, `Clear`, используется для проверки того факта, что в коллекции `strings` находится только содержимое считанного файла.

Чтобы получить возможность использовать ранее созданный класс, хранящийся в другой базе данных, выполните следующие действия.

1. Откройте необходимый модуль (в нашем случае — `FileStream`) в окне редактора Microsoft Visual Basic.
2. Выберите файл для экспорта (в нашем случае — `FileStream.cls`).
3. Выберите команду `File⇒Export File`.
4. В диалоговом окне `Export File` (рис. 22.1) введите имя файла и щелкните на кнопке Сохранить (Save).

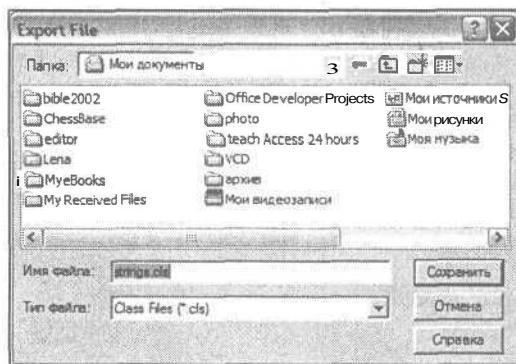


Рис. 22.1. Диалоговое окно `Export File` позволяет сохранить код класса во внешнем файле

Благодаря применению готовых классов `Collection` и `FileStream` задача реализации класса `Strings` оказалась относительно простой. Агрегация позволяет использовать средства существующих классов при создании новых. Возможности кода вырастают весьма ощутимо, но это слабо сказывается на его сложности. Причина в том, что новый профаммный текст писать почти не приходится — вы пользуетесь готовыми и протестированными решениями, прошедшими проверку временем и гарантирующими более высокую степень надежности.

## Тестирование нового класса

Подобно любому новому профаммному коду, класс `strings` должен быть протестирован. Однако класс `Collection` тестировать не нужно (впрочем, исходного кода у вас все равно нет), да и наш собственный класс `FileStream` вряд ли нуждается в проверке — это уже сделано на прошлом занятии. При повторном использовании существующего кода достаточно проверить точки его взаимодействия со строками нового.

Конечный итог повторного применения профаммного кода выражается в увеличении производительности труда профаммиста, уменьшении затрат на тестирование и повышении степени надежности и функциональной полноты.

Листинг 22.4 содержит фрагмент кода, который можно применить для тестирования класса `Strings`.

### Листинг 22.4. Процедура для тестирования класса `Strings`

```
1: Sub TestStrings( )
2:   Dim S As New Strings
3:   S.Values( "Ключ1" ) = "Значение1"
4:   S.Values( "Ключ2" ) = "Значение2"
```

```
5: Call S.WriteToFile( "Test.txt" )
6: Call S.ReadFromFile( "Test.txt" )
7: MsgBox S.Strings(1)
8: End Sub
```

#### Анализ

Строка 2 листинга 22.4 содержит конструкцию объявления и создания объекта `S` класса `strings`. Строки 3 и 4 тестируют правильность выполнения функции `IndexOfName`, `Property`-метода `Let Values` и обработчика ошибок в нем, поскольку выражения содержат обращения к элементам коллекции, которых еще нет (в этом случае элементы будут созданы). Строка 5 проверяет средства записи данных в файл, а строка 6 — их чтения. Строка 7 позволяет увидеть содержимое считанного из файла элемента коллекции с номером 1 в окне сообщения. Тестовый код способен выявить немало ситуаций, о которых вы даже не подозревали.

## Применение класса Strings

Класс `Strings` может оказаться особо нужным в тех случаях, когда необходимы средства организации доступа к значениям по ключевым именам. В Windows, например, все еще находят широкое применение файлы `INI`, содержащие информацию о параметрах настройки приложений. Ассоциативный массив в виде объекта коллекции — удобный способ хранения данных, позволяющий обращаться к элементам информации как по индексам, так и по ключевым именам.



В главе "8-й час. Декомпозиция задач" вы изучили процедуры `Get Settings` и `SaveSettings`, позволяющие выполнять чтение реестра и запись в реестр. Эти методы работы с реестром более предпочтительны для сохранения информации о приложении, чем использование `INI`-файлов.

Класс `Strings` полезен для хранения ассоциативных данных, которые должно отслеживать приложение. Однако не рекомендуется использовать класс `Strings` для работы с реестром вместо специально созданных для этой цели методов.

Для чтения и записи `INI`-файлов существуют API-процедуры — `GetPrivateProfileString` и `WritePrivateProfileString`. Как уже отмечалось, предпочтительнее работать с реестром, в этом случае указанные API-процедуры не понадобятся.

## Резюме

На этом занятии на примере класса `strings` вы изучили способы построения коллекций, обеспечивающих возможности обращения к элементам данных по индексным значениям и ключевым именам. Что еще более важно — вы усвоили основные аспекты объектно-ориентированного проектирования и методы повторного применения существующего кода при построении новых классов. Достаточно приложить небольшие усилия — и за счет добавления простых проверенных решений класс приобретает новые полезные качества.

Вы убедились, как включение существующего кода в состав нового класса помогает добиться приемлемых результатов с минимальными затратами: так, при построении класса `Strings` мы воспользовались ранее созданным классом `FileStream`, обеспечивающим средства выполнения файловых операций.



Приложения, которые имеют дело с парами данных, состоящими из ключа и значения, широко используются в среде Windows. Реестр Windows — это файл данных, содержащий информацию об установленных в системе приложениях и состоящий из пар ключевых имен и соответствующих им значений различных типов. Win.INI и другие конфигурационные INI-файлы имеют аналогичную структуру. Класс strings наверняка найдет применение и в ваших собственных разработках.

При создании новых классов стремитесь к максимальной простоте. Если, имея готовый класс, вы хотели бы расширить его функциональные возможности, постройте новый класс, включив в состав интерфейса атрибуты существующего. Не пытайтесь просто исправить ранее созданный и протестированный класс. Теперь в вашем распоряжении по-прежнему останется старое, проверенное временем решение, и вы сможете на его основе выстроить новое.

А теперь рекомендуем прочесть материал приведенных ниже разделов “Вопросы и ответы” и “Задания”.

## Вопросы и ответы

**Вопрос.** Так ли важно уметь создавать классы?

**Ответ.** В принципе, нет, но это в ваших собственных интересах. Создавая классы, вы получите такие же преимущества, как и при использовании стандартных классов Access, — удобство, надежность, компактность и возможность повторного использования кода.

**Вопрос.** Можно ли переносить модули вообще и классы в частности из одной базы данных в другую?

**Ответ.** Да. В меню File редактора Microsoft Visual Basic есть элементы Import File и Export File. Соответствующие им команды предоставляют возможность переноса кода VBA между различными базами данных, в том числе и созданными другими разработчиками.

**Вопрос.** Что лучше — создавать код класса с нуля или всегда пытаться использовать существующие решения?

**Ответ.** Ответ очевиден — если в вашем распоряжении имеются готовые и протестированные фрагменты кода, решающего хотя бы часть общей задачи, используйте их. Вы получите ощутимые преимущества даже в том случае, если существующий код слишком избыточен, чтобы точно соответствовать условиям конкретной задачи. Готовый код уже написан и проверен, и вам не придется тратить время на изобретение “велосипеда” и его “обкатку” — этих аргументов вполне достаточно.

**Вопрос.** Как узнать, когда именно следует создавать класс?

**Ответ.** Если вам удастся определить круг взаимосвязанных элементов данных и способов их обработки, доступно и последовательно очерчивающих проблему, — постройте соответствующий класс. Пусть вашим лучшим советчиком будет собственный опыт.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. В чем заключаются преимущества агрегации кода?
2. Сколько атрибутов внешнего интерфейса целесообразно определить при создании нового класса?

3. Как отличить атрибуты внешнего интерфейса класса от остальных?
4. Назовите стандартные процедуры, которые вызываются при создании объекта класса и его уничтожении.

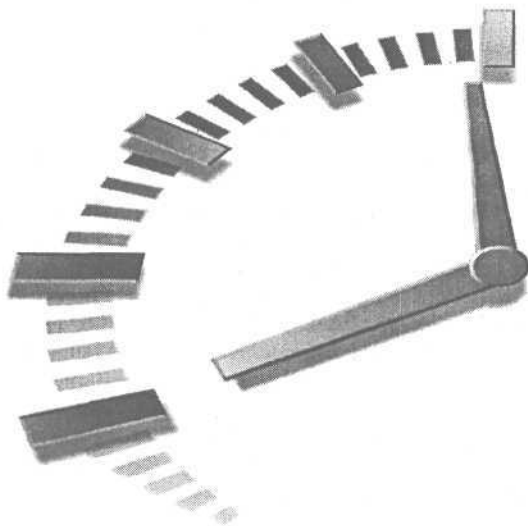
## Упражнения

1. Добавьте в состав класса `Strings` атрибут `Text`, позволяющий вернуть содержимое всех элементов коллекции в виде единой строки, в которой частные значения разделяются константой `vbNewLine`.
2. Добавьте в интерфейс класса `Strings` Property-метод `Let Text`, который дает возможность расчленить общую строку `Text` на отдельные сегменты и сохранить их в элементах коллекции.
3. Дополните текст методов `ReadFromFile` и `WriteToFile` класса `Strings` таким образом, чтобы обеспечить поддержку свойства `Text` (см. предыдущие упражнения).



## 23-й час

# Надстройки Access



Свидетельством качества инструментального средства программирования служит его способность к самопополнению и быстрому расширению собственных возможностей. Access 2002 подобным критериям, безусловно, отвечает — система допускает надстройку с помощью дополнительных баз данных и кода на языке VBA. Главная цель таких операций — облегчение труда программиста и пользователя системы. Круг подобных вопросов и составит предмет последующего обсуждения.

### Новый термин

*Надстройка* (add-in) — это обобщенный термин, используемый для обозначения кода, расширяющего и пополняющего возможности программной системы. В Access этот термин подразумевает ссылку на объект COM или базу данных формата Access, содержащую код, который создан разработчиками системы. В ходе этого занятия вы узнаете о том, как пользоваться готовыми надстройками Access и создавать собственные.

Обычно надстройки включают в себя модули, классы и фрагменты вспомогательного программного кода, хранящиеся в базе данных. Мы рассмотрим процесс создания надстройки, реализующей механизм автоматического ведения протокола ошибок, которая окажется полезной при разработке и отладке приложений. Завершив изучение материала этой главы, вы получите средство, которое существенно облегчит функции тестирования прикладных программ в среде Access и позволит переключить внимание на более важные процессы — скажем, на техническую поддержку собственных приложений.

Основные темы занятия.

- Индивидуальная настройка Access с помощью средств add-in.
- Создание надстройки для ведения журнала ошибок.
- Тестирование кода надстройки.
- Установка и удаление надстроек.

## Знакомство с надстройками Access

Мы уже говорили о том, что надстройка Access представляет собой базу данных формата Access или объект COM. Далее будет рассмотрена процедура создания надстройки в виде базы данных — вопросы построения объектов COM в форме исполняемых модулей

(EXE или DLL) ActiveX выходят за рамки нашего обсуждения, поскольку предполагают использование, как минимум, языка программирования Visual Basic.

Мы будем говорить о надстройках в формате базы данных Access, обозначаемой расширением имени *.MDA* (вместо традиционного *.MDB*). Кроме обозначения, никаких других отличий не существует — в надстройках, как и в обычных базах данных, могут содержаться модули с процедурами и классами, макросы, таблицы, запросы, страницы доступа к данным, отчеты, формы и т.д.

Самое значительное преимущество, обеспечиваемое надстройками, состоит в том, что они — об этом говорит само их название — добавляются в среду Access, становятся ее частью и существенно расширяют и углубляют ее возможности. Другими словами, Access наделена способностями роста — для инструментальных систем программирования это большое преимущество. Прежде чем приступить к созданию надстройки, вы должны определить круг задач, которые ей предстоит решать; создать базу данных и включить в нее программный код, соответствующие формы и другие необходимые объекты; протестировать базу данных; и, наконец, уведомить систему Access о том, что вы намереваетесь ее "расширить". Все названные операции в той или иной степени вам уже знакомы, кроме, пожалуй, последней — непосредственного включения надстройки в состав Access.

Ниже кратко перечислены основные действия, которые предстоит осуществить в процессе создания надстройки, обеспечивающей выполнение функций ведения журнала ошибок.

- Динамическое создание журнальной таблицы.
- Реализация функции автоматического добавления в журнал записи о возникшей ошибке.
- Тестирование кода SQL и схемы базы данных.
- Создание интерфейсных средств, позволяющих динамически включать и отключать в Access механизм ведения протокола ошибок.
- Оформление результатов работы в виде надстройки.
- Установка и тестирование надстройки в среде Access.

В конечном итоге вы получите полезный инструмент отладки, способный оказать существенную помощь в работе над будущими проектами, и пополните свой профессиональный арсенал отнюдь не лишними знаниями.

## Создание базы данных для ведения протокола ошибок

База данных, построением которой мы собираемся заняться, должна содержать таблицу, способную оказать помощь в выяснении причин ошибок, возникших в прикладной программе. Исходя из практических соображений, предполагается создать средства ведения протокола таких ошибок, информацию о которых способен предоставить программный код на языке VBA. Подобные сведения можно "добыть", если обратиться к интерфейсу класса `Err`.

Прежде чем обратиться к примерам, которые приведены далее, вам необходимо создать новую базу данных (назовем ее *Log.mda*). С этой целью выполните следующие действия.

1. Загрузите приложение Access 2002.
2. Выберите в строке меню команду **Файл⇒Создать (File⇒New)**.

3. В диалоговом окне Файл новой базы данных (File New Database) в раскрывающемся списке Тип файла (Save As Type) выберите элемент Все файлы (\*.\*). Если оставить тип Базы данных Microsoft Access (Microsoft Access Database) (\*.mdb), при сохранении файла базы данных с расширением .mda, Access автоматически добавит еще и .mdb. Таким образом, вы получите Имя\_файла.mda.mdb.
4. В поле Имя файла (File Name) введите строку Log.mda и щелкните на кнопке Создать (Create).

Воспринимайте созданную базу данных, Log.mda, как обычную базу данных Access.

Из материала предыдущих глав вы уже знаете, что объект класса Егг содержит переменные для хранения номера ошибки, названия источника ее происхождения, описания, имени соответствующего файла оперативной справки и идентификатора темы в файле справки. Вся эта информация должна найти отражение в таблице, которую нам предстоит построить. Помимо указанных данных, таблица будет содержать сведения о пользователе приложения и дате/времени возникновения ошибки. Обладая таким набором сведений, вы сможете без особого труда восстановить ход всех событий, которые привели к ошибке, даже в том случае, если программа потерпела полный крах. Предвидя дальнейший ход событий, отметим необходимость в SQL-процедуре динамического создания журнальной таблицы — в этом случае вам будет легко построить ее в любой базе данных, приложения которой нуждаются в отладке с поддержкой протокола ошибок. Текст процедуры создания журнальной таблицы приведен в листинге 23.1.

#### Листинг 23.1. Процедура создания журнальной таблицы

```
1: CREATE TABLE LOG
2: (ID AUTOINCREMENT PRIMARY KEY,
3:  ERROR_ID NUMBER,
4:  SOURCE TEXT(50),
5:  DESCRIPTION TEXT(255),
6:  HELPPFILE TEXT(255),
7:  HELPCONTEXT NUMBER,
8:  WHEN DATETIME,
9:  USER TEXT(25)
10: );
```

#### Анализ

Листинг 23.1 содержит текст команды CREATE TABLE на языке SQL, которая позволяет создать таблицу с именем LOG в произвольной базе данных. Сохраните его в виде запроса под именем CREATE\_LOG\_TABLE. Теперь вы всегда сможете, обратившись к запросу, быстро создать в текущей базе таблицы LOG, исключив необходимость выполнения "ручных" операций. -

## Динамическое создание журнальной таблицы

В каждой базе данных, которая будет взаимодействовать с создаваемой надстройкой, должна содержаться копия таблицы LOG. Чтобы избежать возникновения ошибки при попытке повторного создания таблицы, следует написать код, проверяющий факт наличия таблицы в базе данных. Если таблицы нет, ее необходимо построить. Листинг 23.2 демонстрирует текст VBA-процедуры динамического создания таблицы LOG.

### Листинг 23.2. Процедура динамического создания таблицы LOG

```
1: Option Compare Database
2: Option Explicit
3:
4: Private Function CreateLogQuery () As String
5:     CreateLogQuery = _
6:         "CREATE TABLE LOG " & _
7:         "(ID AUTOINCREMENT PRIMARY KEY, ERROR_ID NUMBER, " & _
8:         " SOURCE TEXT(50), DESCRIPTION TEXT(255), " & _
9:         "HELPFILE TEXT(255), HELPCONTEXT NUMBER, " & _
10:        "WHEN DATETIME, USER TEXT(25));"
11:End Function
12:Private Sub CreateTable(ByVal QueryText As String)
13:    Call DoCmd.RunSQL(QueryText)
14:End Sub
```

#### Анализ

Листинг 23.2 содержит только часть будущего решения, поэтому мы создадим в базе данных Log.mda новый модуль и сохраним его под именем LogUtilities. Код SQL, представленный ранее в листинге 23.1, теперь перенесен в строки 4–10 листинга 23.2. Процедура CreateTable, текст которой размещен в строках 12–14, предназначена для создания таблицы LOG в текущей базе данных и использует с этой целью метод RunSQL класса DoCmd из состава библиотеки DAO.

При реализации общего решения (в этом вы убедитесь позднее) мы исходим из того, что таблица LOG в базе данных существует. Если ее нет, генерируется исключительная ситуация. Процедура обработки ошибки создает таблицу и возвращает управление строке, вызвавшей исключение, для повторного выполнения. В следующем разделе приведен фрагмент кода, в котором обеспечивается вызов процедуры CreateTable и реализуется механизм добавления в таблицу LOG записи, содержащей информацию об ошибке.

## Ведение протокола ошибок

Мы будем полагать, что таблица LOG в базе данных уже создана. Код, приведенный ниже, выполняет операцию добавления в журнальную таблицу записи с данными об ошибке (листинг 23.3). Мы определили три процедуры: две из них снабжены квалификаторами Public (WriteErrorEntry и WriteEntry), а третья - Private (DoWriteEntry).

### Листинг 23.3. Процедуры добавления записей в таблицу LOG

```
1: Public Sub WriteErrorEntry( ByVal Error As ErrObject, _
2:     Optional UserName As String = "Admin" )
3:
4:     With Err
5:         Call WriteEntry(.Number, .Source, .Description, _
6:             .HelpFile, .HelpContext, UserName)
7:     End With
8: End Sub
9:
10:Private Function TableExists(TableName As _
11:    String) As Boolean
```

```

11:
12: Dim Table As Variant
13:
14: For Each Table In CurrentData.AllTables
15:     TableExists = Table.Name = TableName
16:     If (TableExists) Then Exit For
17: Next Table
18:
19:End Function
20:
21:Public Sub WriteEntry(ByVal Number As Long, _
                        ByVal Source As String, _
22:    ByVal Description As String, ByVal HelpFile As String, _
23:    ByVal HelpContext As Long, Optional UserName As String )
24:
25: On Error GoTo EXCEPT
26: Call DoWriteEntry(Number, Source, Description, _
27:    HelpFile, HelpContext, UserName)
28:
29: Exit Sub
30:EXCEPT:
31: If (Not TableExists("Log")) Then
32:     Call CreateTable(CreateLogQuery)
33:     Resume
34: Else
35:     Call Err.Raise(Err.Number, _
36:        "LogUtilities.WriteEntry", Err.Description)
37: End If
38:End Sub
39:
40:Private Sub DoWriteEntry (ByVal Number As Long, _
41:    ByVal Source As String, ByVal Description As String, _
42:    ByVal HelpFile As String, ByVal HelpContext As Long, _
43:    Optional UserName As String)
44:
45: Dim SQL As String
46: SQL = "SELECT * FROM LOG"
47: Dim Recordset As New ADODB.Recordset
48:Call Recordset.Open(SQL, CurrentProject.Connection, _
49:    adOpenDynamic, adLockPessimistic)
50: Recordset.AddNew
51: Recordset( "ERROR_ID" ) = Number
52: Recordset( "SOURCE" ) = Source
53: Recordset( "DESCRIPTION" ) = Description
54: Recordset( "HELPPFILE" ) = HelpFile
55: Recordset( "HELPCONTEXT" ) = HelpContext
56: Recordset( "WHEN" ) = Now
57: Recordset( "USER" ) = UserName
58: Recordset.Update
59: Recordset.Close
60: Set Recordset = Nothing
61: End Sub

```

#### Анализ

I Код листинга 23.3, рассматриваемый совместно с текстом листинга 23.2, I составляет содержимое модуля LogUtilities. Процедуре WriteError-Entry, расположенной в строках 1–8, в качестве параметра передается



объект Err, а она, в свою очередь, обращается к процедуре WriteEntry. Операция добавления записи выполняется приватной процедурой DoWriteEntry, вызываемой в строке 14. Оставшаяся часть кода процедуры WriteEntry связана с обработкой возможных ошибок. Если ошибка произошла по причине отсутствия в базе данных таблицы LOG, вызывает-ся процедура CreateTable, рассмотренная в листинге 23.2, и управление вновь передается строке, породившей ошибку (строки 32-33). Если при-чина ошибки заключается в другом, вызывается стандартный метод Raise объекта Err. В строках 10–19 выполняется поиск таблицы LOG в коллекции AllTables.

Самая важная часть работы возложена на процедуру DoWriteEntry — об этом го-ворит префикс Do ее названия. Строки обработки ошибок отделены от кода, выпол-няющего собственно операцию пополнения таблицы, с целью упростить обе процеду-ры. Код подпрограммы DoWriteEntry достаточно прозрачен и не нуждается в под-робных комментариях — создается объект класса DAO.Recordset, с его помощью в таблицу добавляется новая запись, ее полям присваиваются значения, переданные процедуре в виде аргументов, а затем внесенные изменения сохраняются в базе дан-ных командой Update.

Чтобы протестировать рассмотренный выше код, создайте новый модуль и помес-тите в него процедуру, приведенную в листинге 23.4, или ей подобную. (Я обычно стараюсь сохранять тестовый код в отдельных модулях, не смешивая его с кодом, не-посредственно решающим задачу, — то же рекомендую делать и вам.)

#### Листинг 23.4. Тестовая процедура для проверки кода модуля LogUtilities

```
1: Sub Test ( )
2:   On Error GoTo Except
3:   Call Err.Raise(1, "LogUtilities", "Test", "No Help", 0)
4:   Exit Sub
5:   Except:
6:     Call WriteErrorEntry(Err)
7: End Sub
```

#### Анализ

Тестовый код действует следующим образом: в строке 3 с помощью при-нудительного вызова Err.Raise имитируется возникновение ошибки и провоцируется выполнение кода обработчика исключений, который, об-ращаясь к процедуре WriteErrorEntry, сохраняет информацию об ошибке в таблице LOG базы данных. Теперь, снабдив любую из реальных программ конструкциями обработки ошибок, содержащими обращения к процедурам WriteErrorEntry или WriteEntry, вы получите надежный код и удобное средство анализа ошибок. В случае возникновения любого сбоя вы без труда восстановите картину происшедшего, просмотрев со-держимое соответствующей записи таблицы LOG. В следующем разделе мы обсудим способы оперативного включения/отключения функций ве-дения протокола ошибок и просмотра содержимого журнальной таблицы.

## Интерфейс просмотра ошибок

Диспетчер настроек, вызываемый из меню Сервис⇒Надстройки (Tools⇒Add-Ins) ис-пользуют для добавления внешних ссылок на базы данных, которые необходимо инкор-порировать в текущий проект. Добавив с помощью диспетчера надстроек (вы ознакоми-

тесь с ним чуть позже) код `Log.mda` в базу данных, вы получите доступ к средствам, которые предлагает эта надстройка. Включение механизма ведения протокола обеспечивается вызовом любой из процедур, `WriteErrorEntry` либо `WriteEntry`. Ниже приведен код, дающий возможность просмотра содержимого журнала ошибок.

После установки надстройки в среде Access в меню **Сервис**⇒**Надстройки** добавляется новый элемент. Щелчок на нем приводит к срабатыванию определенной процедуры или функции в модуле надстройки — в данном случае в этой роли выступает функция, открывающая таблицу LOG для просмотра. (Вы можете развить мою мысль и придумать что-то еще.) Код функции просмотра содержимого журнальной таблицы приведен в листинге 23.5.

Листинг 23.5. Функция просмотра содержимого таблицы LOG, выполняемая при выборе соответствующего элемента меню **Сервис**⇒**Надстройки**

```
1: Function EntryPoint ( )
2:   Call DoCmd.OpenTable("LOG", acViewNormal, acReadOnly)
3: End Function
```

#### Анализ

Как видите, код листинга 23.5 нельзя назвать особенно сложным. В строке 2 используется обращение к методу `DoCmd.OpenTable` для открытия указанной таблицы в режиме просмотра. Если ошибок в тестируемой программе не было, значит, таблица LOG в базе данных еще не создана и при выполнении строки 2 откроется окно с сообщением об “интерфейсной” ошибке — это вполне нормально. Функцию `EntryPoint` также следует включить в состав модуля `LogUtilities`.

## Тестирование кода надстройки

Подобно любому программному коду, процедуры и функции из состава надстройки должны быть подвергнуты всестороннему тестированию. Поскольку мы имеем в виду надстройку в виде базы данных Access, никаких принципиальных инноваций вы не встретите. Как и ранее, откройте модуль в окне редактора VBA, поместите текстовый курсор в пределах процедуры, которую хотите выполнить, и нажмите клавишу <F5> или <F8>. Гораздо легче протестировать процедуры надстройки заблаговременно, перед инсталляцией ее в среде Access.

Еще один способ тестирования внешнего кода связан с построением ссылки из одной, рабочей, базы данных на другую. Чтобы добавить ссылку на базу данных `Log.mda` (или произвольную базу), выполните следующие действия.

1. Откройте новую или существующую базу данных, в которой необходимо создать ссылку.
2. Выберите в строке меню окна Access команду **Сервис**⇒**Макрос**⇒**Редактор Visual Basic** (**Tools**⇒**Macros**⇒**Visual Basic Editor**).
3. В строке меню окна редактора Microsoft Visual Basic выберите команду **Tools**⇒**References**.
4. В диалоговом окне **References** щелкните на кнопке **Browse**.
5. Откроется диалоговое окно **Add Reference**. В раскрывающемся списке **Тип файлов** (**Files of Type**) выберите опцию **Надстройки (\*.mda) (Add-Ins)**.
6. С помощью средств навигации окна отыщите файл `Log.mda` и щелкните на кнопке **Открыть** (**Open**), а затем на кнопке **ОК** в диалоговом окне **References**.

7. Изменения в структуре проекта вы сможете увидеть, если откроете дочернее окно Project Explorer среды Microsoft Visual Basic (рис. 23.1).

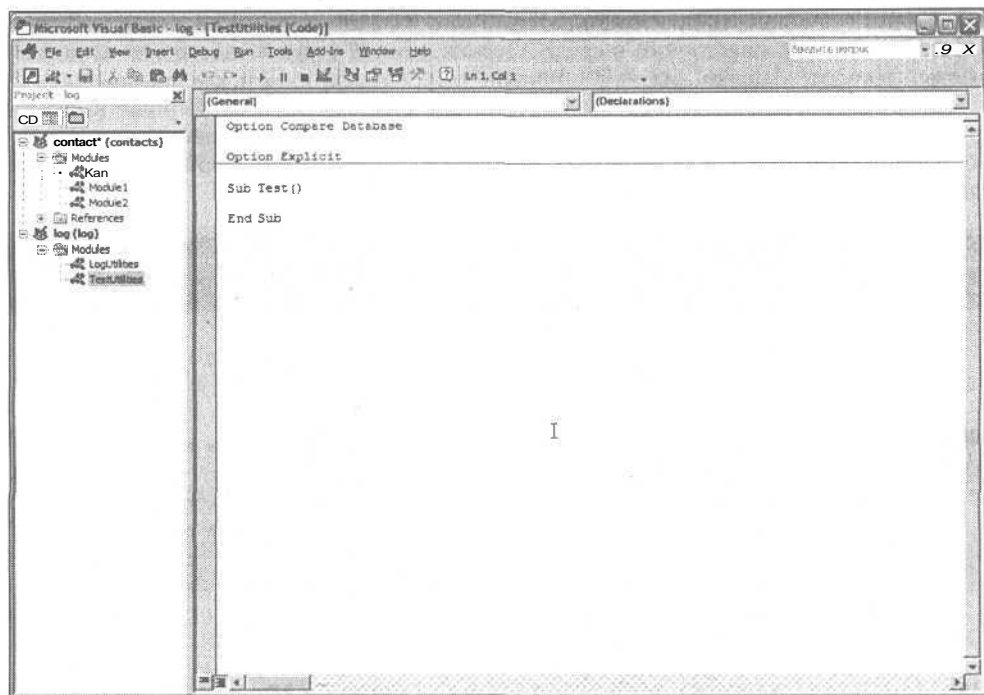


Рис. 23.1. Создав ссылку, вы сможете использовать в новом проекте ранее составленный код

После построения ссылки код адресуемой базы данных становится доступным для решения задач, связанных с текущей базой. Это удобный способ повторного использования кода (в том числе, кода надстроек), исключающий необходимость выполнения операций экспорта и импорта модулей.

## Установка и отключение надстроек

После создания и тестирования кода надстройки вы можете установить его в среде Access. Откройте базу данных Log.mda и откомпилируйте код модуля LogUtilities — его полный текст приведен в листинге 23.6.

Листинг 23.6. Полный текст модуля LogUtilities

```
1: Option Compare Database
2: Option Explicit
3:
4: Function EntryPoint()
5:   Call DoCmd.OpenTable("LOG", acViewNormal, acReadOnly)
6:   End Function
7:
8:
```

```

9: Property Get UserName( ) As String
10:   UserName = Workspaces( 0 ).UserName
11:End Property
12:
13:Private Function CreateLogQuery ( ) As String
14:   CreateLogQuery = _
15:     "CREATE TABLE LOG " & _
16:     "(ID AUTOINCREMENT PRIMARY KEY, ERROR_ID NUMBER, " & _
17:     "SOURCE TEXT(50), DESCRIPTION TEXT(255), HELPPFILE TEXT(255),
18:     " S = "HELPCONTEXT NUMBER, WHEN DATETIME, USER TEXT(25));"
19:End Function
20:
21:Private Sub CreateTable()
22:   Call DoCmd.RunSQL(CreateLogQuery)
23:End Sub
24:
25:Public Sub WriteErrorEntry(ByVal Error As ErrObject, _
26:   Optional UserName As String = "Admin")
27:
28:   With Err
29:     Call WriteEntry(.Number, .Source, .Description, _
30:       .HelpFile, .HelpContext, UserName)
31:   End With
32: End Sub
33:
34:Private Function TableExists(ByVal TableName As String)_
   As Boolean
35:
36:   Dim Table As Variant
37:
38:   For Each Table In CurrentData.AllTables
39:     TableExists = Table.Name = TableName
40:     If (TableExists) Then Exit For
41:   Next Table
42:
43:End Function
44:
45:Public Sub WriteEntry(ByVal Number As Long, _
   ByVal Source As String, _
46:   ByVal Description As String, ByVal HelpFile As String, _
47:   ByVal HelpContext As Long, Optional UserName As String )
48:
49:   On Error GoTo EXCEPT
50:   Call DoWriteEntry(Number, Source, Description, _
51:     HelpFile, HelpContext, UserName)
52:
53:   Exit Sub
54:EXCEPT:
55:   If (Not TableExists("LOG")) Then
56:     Call CreateTable
57:     Resume
58:   Else
59:     Call Err.Raise(Err.Number, "LogUtilities.WriteEntry", _
60:       Err.Description)
61:   End If
62:End Sub
63:

```

```

64: Private Sub DoWriteEntry ( ByVal Number As Long, _
65:   ByVal Source As String, ByVal Description As String, _
66:   ByVal HelpFile As String, ByVal HelpContext As Long, _
67:   Optional UserName As String)
68:
69:   Dim SQL As String
70:   SQL = "SELECT * FROM LOG"
71:   Dim Recordset As New ADODB.Recordset
72:   Call Recordset.Open(SQL, CurrentProject.Connection
       adOpenDynamic, adLockPessimistic)
73:   Recordset.AddNew
74:   Recordset( "ERROR_ID" ) = Number
75:   Recordset( "SOURCE" ) = Source
76:   Recordset( "DESCRIPTION" ) = Description
77:   Recordset( "HELPPFILE" ) = HelpFile
78:   Recordset( "HELPCONTEXT" ) = HelpContext
79:   Recordset( "WHEN" ) = Now
80:   Recordset( "USER" ) = UserName
81:   Recordset.Update
82:   Recordset.Close
83:   Set Recordset = Nothing
84:
85: End Sub

```

#### Анализ

Практически весь этот код вам уже знаком, за исключением свойства `UserName`, в котором вызывается глобальный метод `Application.Rentser`. Набирая текст в окне модуля, сопоставляйте его с содержимым листинга 23.6.

Перед компиляцией кода проверьте, чтобы база данных `Log.mda` была открыта. В окне редактора Microsoft Visual Basic выберите команду **Debug⇒Compile**. Если система выдаст сообщения об ошибках компиляции, у вас появится шанс их исправить. Прежде чем установить созданную надстройку в среде Access, необходимо выполнить еще несколько операций, о чем речь пойдет в следующем разделе.

## Создание таблицы с данными о регистрации

Каждая надстройка Access требует наличия таблицы `USysRegInfo`, которая содержит сведения, необходимые системе для создания определенных ключей реестра Windows. Таблица `USysRegInfo` может быть построена в базе данных `Log.mda` с помощью процедуры SQL. Ее текст приведен в листинге 23.7.

**Листинг 23.7.** Процедура создания таблицы `USysRegInfo`, которая необходима для регистрации надстройки, устанавливаемой в Access

```

1: CREATE TABLE USysRegInfo
2:   (SubKey Text(255),
3:    Type Number,
4:    ValName Text(50),
5:    Value Text(50)
6:   );

```

#### Анализ

Текст листинга 23.7 не нуждается в подробных разъяснениях, за исключением разве что квадратных скобок. Поскольку существует ключевое слово `Value`, чтобы сообщить Access о необходимости создания поля с таким именем, его название необходимо заключить в квадратные скобки.

После того, как таблица построена, вы должны добавить в нее три записи, содержащие данные о так называемой точке входа надстройки и местоположении ее файла на диске. Содержимое каждого из полей всех записей таблицы USysRegInfo представлено в табл. 23.1.



Если вы не видите таблицу USysRegInfo, выберите команду Сервис⇒Параметры (Tools⇒Options), перейдите на вкладку Вид (View) и установите флажок Системные объекты (System Objects).

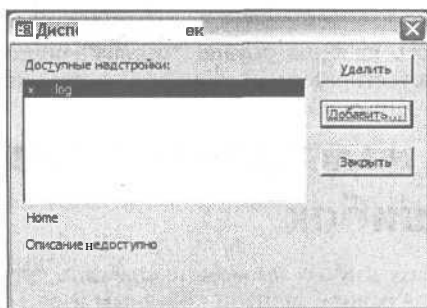
**Таблица 23.1. Содержимое полей таблицы USysRegInfo**

SubKey	Type	ValName	Value
HKEY_CURRENT_ACCESS_PROFILE\Menu Add-Ins\Log	0		
HKEY_CURRENT_ACCESS_PROFILE\Menu Add-Ins\Log	1	Expression	=EntryPoint( )
HKEY_CURRENT_ACCESS_PROFILE\Menu Add-Ins\Log	1	Library	ACCDIR\Log.mda

Первая запись таблицы USysRegInfo (описанная первой же строкой табл. 23.1) используется для создания ключа регистрации надстройки Log. Вторая запись определяет функцию из состава надстройки, вызываемую при щелчке на том элементе меню Сервис⇒Надстройки, который появляется после установки надстройки в Access. Третья запись указывает место расположения файла надстройки на диске компьютера. В поле Value третьей записи следует сохранить значение |ACCDIR\Log.mda. Конструкция |ACCDIR служит обозначением стандартной папки для размещения надстроек, предусмотренной в Access. Обычно это f:\MCK\Windows\Application Data\Microsoft\AddIns, где Диск — буква дисковод, на котором располагаются файлы операционной системы Windows. Важно помнить, что профамма диспетчера при установке надстроек в Access создает копии файлов в папке AddIns.

## Установка надстройки

Инсталляция надстройки производится с помощью средств диспетчера надстроек Access, диалоговое окно которого показано на рис. 23.2. Чтобы установить надстройку в Access, выполните следующие действия.



*Рис. 23.2. Диалоговое окно диспетчера надстроек Access*

1. Загрузите Access.
2. Выберите в строке меню команду Сервис⇒Надстройки⇒Диспетчер надстроек (Tools⇒Add-Ins⇒Add-In Manager).

3. Щелкните на кнопке Добавить (Add New) диалогового окна Диспетчер надстроек, с помощью средств навигации диалогового окна Открытие надстройки (Open) проследуйте к файлу Log.mda, выберите его и щелкните на кнопке Открыть (Open).
4. Убедитесь, что в список Доступные надстройки (Available Add-Ins) окна Диспетчер надстроек добавлена запись с наименованием надстройки и предшествующим символом x. Кроме того, кнопка Установить (Install) должна изменить свое название на Удалить (Uninstall). Щелкните на кнопке Закреть.



Если после установки надстройки Log.mda возникли проблемы, необходимо ее деинсталлировать, закрыть Access и удалить копию базы данных Log.mda из папки надстроек. По умолчанию это папка C:\Documents and Settings\имя\_пользователя\Application Data\Microsoft\Add-Ins\Log.mda, где имя\_пользователя необходимо заменить действительным именем пользователя.

Внесите изменения в исходную базу данных, вновь установите надстройку и проверьте правильность ее работы.

Чтобы проверить корректность установки надстройки, протестируйте ее. Откройте меню Сервис⇒Надстройки — в нем должен появиться новый элемент (в нашем случае — Log), который следует выбрать. Если надстройка Log установлена правильно, вы получите сообщение об ошибке подобного содержания: Приложению 'Microsoft Access' не удастся найти объект 'LOG'. Не расстраивайтесь, это нормально — надстройка работает и пытается открыть для просмотра таблицу LOG, но, поскольку какое приложение еще не тестировалось (и не провоцировало ошибок), таблица LOG в базе данных еще не создана.

## Модификация и удаление надстройки

Удалить надстройку из среды Access (как и выполнить любое другое действие, направленное на разрушение, а не созидание) очень просто. Достаточно с помощью команды Сервис⇒Надстройки⇒Диспетчер надстроек открыть диалоговое окно Диспетчер надстроек, выбрать нужную (или, точнее, уже ненужную) надстройку и щелкнуть на кнопке Удалить. Чтобы внести в настройку какие-либо изменения, первым делом следует выгрузить все экземпляры Access и физически удалить соответствующий файл из папки AddIns. Завершив модификацию надстройки, повторите все описанные выше действия по ее установке.

Теперь, когда у вас есть удобное орудие "отлова" ошибок, пора отправиться на охоту, т.е. попробовать протестировать инструмент на практике.

## Применение надстройки для ведения протокола ошибок

Каждую собственную базу данных вы можете снабдить средством ведения протокола ошибок, которое окажется в равной степени полезным и во время разработки приложений, и в период их эксплуатации. Часто происходит так, что пользователи, мельком взглянув на окно сообщения об ошибке и не утруждая себя излишними размышлениями, щелкают на кнопке ОК и пытаются продолжить работу. В результате спохватываются только тогда, когда программы терпят полное фиаско. Обладая инструментом ведения журнала ошибок, вы всегда, даже спустя некоторое время, сможете восстановить картину происшедшего, выявить причины сбоев и принять соответствующие меры.

После установки надстройки желательно протестировать ее в реальных условиях. Обращайтесь к процедурам модуля Logutilities во всех случаях, когда посчитаете необходимым. Подобные вызовы естественно размещать внутри блоков обработчиков ошибок — здесь практически всегда в вашем распоряжении имеется объект Err, обладающий исчерпывающей информацией о последней ошибке. Достаточно вставить строку кода Call WriteErrorEntry(Err) сразу после строк, содержащих обычные инструкции обработки ошибок. Пример использования кода приведен в листинге 23.4. Чтобы просмотреть таблицу LOG, выберите соответствующий элемент меню Сервис⇒Надстройки.

Еще одно дополнительное замечание. Средства ведения протокола ошибок легко включать и отключать с помощью условных директив компилятора, добавив модуль с двумя процедурами, аналогичными WriteEntry и WriteErrorEntry. Подробнее о директивах компилятора см. главу "18-й час. Обработка ошибок во время выполнения программы".

## Подведем итоги

Вам абсолютно не помешает еще раз повторить все действия, которые необходимо выполнить в процессе создания надстройки Access. Сказанное далее, справедливо в отношении любой надстройки, оформленной в виде базы данных. Не забывайте о двух моментах: чтобы воспользоваться возможностями повторного применения кода, вовсе не обязательно создавать надстройки; для построения надстроек в формате объектов COM следует прибегнуть к более мощным средствам программирования — например, Visual Basic. (Visual Basic — это самостоятельный язык, обладающий собственной интегрированной средой; он распространяется в составе Microsoft Visual Studio. Завершив изучение нашей книги, вы наверняка сможете изучить и Visual Basic.)

Итак...

1. Создайте базу данных с расширением имени файла .MDA.
2. Постройте в базе данных надстройки все необходимые модули, классы, формы и другие объекты.
3. Создайте таблицу USysRegInfo и не забудьте указать в ней имя процедуры или функции, служащей точкой входа надстройки. (Для построения таблицы вы можете регулярно пользоваться командой листинга 23.7 и содержимым табл. 23.1. Вероятно, имеет смысл всегда называть процедуру, используемую в качестве точки входа надстройки, одним и тем же именем.)
4. Перед установкой надстройки тщательно ее протестируйте.
5. Инсталлируйте надстройку средствами диалогового окна Диспетчер надстроек.

Перечисленные выше моменты на самом деле более полезны, чем может показаться на первый взгляд. Надстройки — одно из самых мощных средств автоматизации выполнения рутинных операций. Именно таким образом созданы хваленые мастера Microsoft. И наконец, последнее — если позволяет бюджет, покупайте готовые надстройки Access, чтобы избежать необходимости самостоятельного программирования.

## Резюме

Настройки Access предлагают в ваше распоряжение мощные средства управления всеми компонентами интегрированной среды системы. Применив навыки программирования, вы сможете настроить (или *настроить*?) Access самым непредвиденным образом — разумеется, если в этом есть серьезный практический смысл.

В ходе занятия вы ознакомились со способами создания, тестирования и установки надстроек в среде Access. Надстройки — полноценный товар и удобное средство обмена готовыми решениями.

Не забудьте прочесть разделы "Вопросы и ответы" и "Задания", завершающие эту главу.



# Вопросы и ответы

**Вопрос.** Возможно ли одновременное использование средств DAO и ADO в пределах одного приложения?

**Ответ.** Да. Воспринимая DAO и ADO как обычные классы (именно так и следует к ним относиться), вы, разумеется, можете создавать и их объекты. Единственная потенциальная опасность заключается в том, что названия некоторых атрибутов этих классов совпадают.

**Вопрос.** Можно ли воспользоваться классом из состава надстройки?

**Ответ.** Да, но не напрямую. Потребуется создать в модуле надстройки дополнительную функцию, возвращающую объект класса. Определение переменной такого класса в модуле, не принадлежащем надстройке, не допускается.

**Вопрос.** Удаляется ли исходный файл .MDA после установки надстройки в папку AddIns?

**Ответ.** Нет. У вас остается две копии базы данных — не забывайте о необходимости их синхронизации.

**Вопрос.** Как поступить, если надстройка содержит процедуру или функцию с таким же именем, какое используется в рабочем модуле приложения?

**Ответ.** Перед именем процедуры из состава надстройки введите префикс в виде названия надстройки и символа точки, например Log.FileExists.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. С какой целью файлы баз данных надстроек Access обозначаются расширением имени .MDA?
2. Как называется таблица, которую необходимо создать в базе данных надстройки для обеспечения возможности ее регистрации?
3. Методы Property могут быть созданы в обычном, не "классовом", модуле. Верно ли это?
4. Использование квалификаторов Public и Private допустимо в пределах обычного модуля. Верно ли это?
5. Что означает константа IACCDIR?

## Упражнения

1. Создайте условную конструкцию, которая проверяет факт существования таблицы LOG и в случае необходимости вызывает процедуру CreateTable.
2. Приведите выражение для тестирования процедуры WriteEntry.
3. Создайте пустую базу данных, определите ссылку на базу Log.mda и вызовите процедуру WriteEntry. Будет ли в результате этого обращения построена таблица LOG?



## 24-й час

# Управление информацией о контактах Outlook

Outlook, приложение из состава Microsoft Office XP, с точки зрения особенностей построения относится к числу *серверов OLE Automation* (или просто *серверов автоматизации*). Это означает, что программа Outlook способна выполняться самостоятельно и вызываться из среды других приложений. Outlook служит для управления самыми разнообразными данными — сообщениями электронной почты, сведениями о контактах, расписаниями, задачами и т.п. На этом занятии будут рассмотрены способы создания кода на языке **VBA** в среде Access для использования Outlook в качестве программного агента, управляющего информацией о контактах в приложениях баз данных.

Код VBA, который мы построим в ходе занятия, предназначен для управления информацией о контактах с помощью средств Outlook. Это наглядный пример операций, для выполнения которых, по большому счету, и предназначен пакет приложений Office XP — мощный комплекс гибко взаимодействующих программ, позволяющих автоматизировать решение множества офисных задач.

Навыки, полученные вами на прошлых занятиях, непременно пригодятся. К настоящему моменту вы уже достаточно хорошо осведомлены, например, о способах построения и использования классов и их объектов. Создавая экземпляр Outlook в среде приложения Access, мы будем "общаться" с объектами самым "тесным образом".

Основные темы занятия.

- Объектная модель Outlook.
- Применение папок Outlook.
- Просмотр, изменение и поиск данных в папках Outlook.
- Использование почтовых средств Outlook в прикладных программах.

# Знакомство с Outlook 2002

Microsoft Outlook 2002 — это стандартное Windows-приложение, которое поддерживает функции сервера OLE Automation. Программа Outlook была создана, чтобы помочь пользователю справиться с лавиной информации — почтовыми сообщениями, сведениями о контактах, расписаниями и прочими данными, а в конечном итоге — сберечь драгоценное время.

Приложение Outlook просто незаменимо в управлении телефонными номерами и адресами электронной почты. Обращайтесь к помощи Outlook, если необходимо составить расписание заданий, которые предстоит упорядочить во времени и завершить в срок. Если вы хотели бы обмениваться почтовыми сообщениями с коллегами по работе или внешними абонентами, Outlook и тут придет вам на помощь. Функции ведения календаря "поучаствуют" в планировании встреч, а в разделе заметок вы сможете быстро завязать "узелки" на память.

Новый термин

*Контроллер автоматизации* — это приложение, которое использует или управляет серверами автоматизации, например Outlook.

Нет ничего удивительного в том, что многим приложениям требуются одни и те же функции. Но вовсе не обязательно всякий раз создавать их с нуля — вероятно, просто глупо пытаться заново воспроизвести все возможности, предлагаемые тем же Outlook. Если в программе для Access, над которой вы трудитесь, понадобятся те или иные (а может быть, даже все) функции, выполняемые Outlook, просто воспользуйтесь копией Outlook. Воспринимайте Outlook как еще один объект, код которого доступен для повторного использования. Ниже рассказывается, как этого добиться.

## Объектная модель Outlook

Пытаясь представить Outlook в виде объекта, средства которого можно применить в собственном приложении, помните, что это не совсем обычный объект, как, скажем, переменная Collection. Outlook — весьма громоздкое сооружение, состоящее из множества других объектов, каждый из которых, в свою очередь, содержит свойства, методы... и вложенные объекты. Впрочем, все это отнюдь не означает, что "класс" Outlook принципиально неуправляем или непостижим. Немного терпения, чуть-чуть времени на знакомство с документацией — и работа закипит.

Итак, рассмотрим объектную модель, лежащую в основе Outlook. (Напомним, что объектная модель охватывает аспекты внутренней организации приложения, а не его внешнего представления, доступного пользователям.)

1. Запустите на выполнение Microsoft Outlook 2002.
2. Выберите в строке меню команду **Сервис⇒Макрос⇒Редактор Visual Basic (Tools⇒Macros⇒Visual Basic Editor)**. (Да-да, здесь доступен тот же самый, хорошо знакомый редактор Visual Basic. Если вы все еще не удосужились его устанавливать — мы допускаем и такую, почти невероятную, мысль, — сделайте это хотя бы сейчас. Программа инсталляции Office XP по умолчанию устанавливает далеко не все компоненты пакета.)
3. Перейдя в окно приложения Microsoft Visual Basic, нажмите клавишу <F1>.
4. В окне запроса Помощника введите ключевую фразу поиска *Microsoft Outlook Objects* и щелкните на кнопке Найти (Search).



Если Помощник Office (Скрепыш) находится на экране, его можно скрыть с помощью команды контекстного меню. Щелкните на Помощнике правой кнопкой мыши и в контекстном меню выберите команду Параметры (Options). На вкладке Параметры снимите флажок Использовать помощника (Use the Office Assistant). Помощник будет скрыт, но вы по-прежнему можете обращаться к мастеру вопросов. (Более подробную информацию о Помощнике можно получить по адресу [www.officeclippy.com](http://www.officeclippy.com).)

Кроме того, примите во внимание, что ссылка на объектную модель может не работать корректно. Если в строку поиска справочной системы Visual Basic ввести Microsoft Outlook Objects, раздел справки найден не будет. И только заменив Outlook на Outloo, вы получите нужный результат.

В окне приложения оперативной справочной системы Microsoft Visual Basic откроется страница указанной темы, отображающая графическую диаграмму иерархии классов Outlook. Щелчок на любом из элементов диаграммы вызывает переход к соответствующей странице справки. Легко заметить (труднее понять!), что "класс" Outlook состоит из множества различных объектов. За каждым из них скрывается собственный, зачастую достаточно сложный, интерфейс.

Хотя на первый взгляд ситуация кажется довольно мрачной и обескураживающей, мы все же выйдем на верную дорогу, и вскоре, вы наверняка сможете воспользоваться инструментами Outlook для успешного решения собственных прикладных задач.

## Создание экземпляра Outlook

В основе иерархии объектов Outlook лежит объект класса Application. Объект с тем же именем образует верхний уровень интерфейса всех других компонентов Microsoft Office XP, поддерживающих архитектуру серверов OLE Automation. Экземпляр Outlook строится совершенно так же, как и объекты всех других классов. Следуя приведенной ниже инструкции и используя код листинга 24.1, вы сможете создать объект Outlook в среде прикладной программы Access и загрузить приложение Помощник Microsoft Office.

1. Загрузите Access и создайте базу данных.
2. В окне База данных (Database) выберите элемент Модули (Modules) списка Объекты (Objects).
3. Щелкните на кнопке Создать (New) панели инструментов.
4. Введите код, приведенный в листинге 24.1, и выполните его.

Листинг 24.1. Демонстрационная процедура для создания объекта Outlook и запуска приложения Помощник Microsoft Office

```
1: Sub ShowOutlookAssistant( )
2:   Const ClippitFile = "rocky.acs"
3:   Dim Outlook As Object
4:   Set Outlook = CreateObject( "Outlook.Application" )
5:   Outlook.Application.Assistant.FileName = ClippitFile
6:   Outlook.Assistant.Visible = True
7:   Outlook.Assistant.Animation = 22 'msoAnimationGreeting
8: End Sub
```



Необходимо установить файл анимации `rocky.acs`, который не устанавливается по умолчанию вместе с Access.

#### Анализ

Строка 2 листинга 24.1 содержит объявление константы с наименованием файла Помощника Office, в строке 3 определяется переменная Outlook типа Object, которой далее присваивается экземпляр Outlook. Подобный стиль создания объектов носит название *динамического связывания* — объект ассоциируется с переменной в процессе выполнения программы. Альтернативный способ — *статическое связывание* — предполагает предварительное объявление переменной заведомо известного типа. Чтобы получить возможность статического связывания объектов Outlook, в диалоговом окне References необходимо установить флажок Microsoft Outlook 10.0 Object Library. В случае статического связывания строки 3–4 листинга 24.1 должны быть заменены следующим выражением:

```
Dim Outlook As New Outlook.Application
```

Способ динамического связывания более гибок — он позволяет присваивать переменной типа Object объект любого класса. При статическом связывании методы и свойства объектов класса доступны уже на этапе проектирования программы. Поскольку в этом случае тип заранее известен, редактор Visual Basic по мере набора текста открывает контекстные меню, в которых находятся, в частности, и атрибуты интересующего вас класса.

Строка 5 содержит инструкцию задания имени файла Помощника, далее фигурка последнего изображается на экране, а команда строки 7 заставляет Помощника выйти "на бис".

## Использование объекта NameSpace

Сейчас вы уже знаете, как создать экземпляр Outlook. (Доступ к ранее созданному объекту можно получить, воспользовавшись функцией `GetObject`.) Далее необходимо освоить способы построения объектов, представляющих сообщения электронной почты, задачи, сведения о контактах и данные календаря. К сожалению, в объектной модели подобные возможности явно не предусмотрены.

Все объекты Outlook, располагающие информацией о размещении данных, подчинены абстрактному корневому объекту NameSpace. Для доступа ко всем разновидностям данных Outlook необходимо использовать вызов `GetNameSpace( "MAPI" )`, возвращающий объект класса NameSpace. Обладая объектом NameSpace, вы сможете добраться до источников данных (скажем, информации о контактах) с помощью коллекции Folders. Листинг 24.2 демонстрирует пример построения объекта NameSpace.

#### Листинг 24.2. Пример построения объекта NameSpace

```
1: Sub CreateNameSpace( )
2:   Dim N As NameSpace
3:   Set N = GetNameSpace( "MAPI" )
4: End Sub
```

#### Анализ

В строках 1–4 листинга 24.2 переменная класса NameSpace объявляется, а затем инициализируется соответствующим объектом MAPI (*Mail API*). Допускается существование только одного объекта NameSpace MAPI в пределах текущего сеанса Access. Наименование функции и факт использования в ней текстового аргумента дают основание предположить, что

в будущем Microsoft, вероятно, предоставит средства создания объектов `NameSpace` и других разновидностей. Планируя обратиться к данным Outlook, используйте конструкцию вызова, приведенную выше.

Построив объект `NameSpace`, вы получаете возможность обращения к данным Outlook с помощью коллекции `Folders`. Об этом — далее.

## Использование коллекции `Folders`

В составе объекта `NameSpace` `MAPI` содержится коллекция `Folders` объектов класса `MAPIFolder`. Каждый элемент списка папок, отображаемого в окне приложения Microsoft Outlook (рис. 24.1), представляется отдельным объектом `MAPIFolder`. Чтобы получить доступ к содержимому любой из папок, необходимо обратиться к соответствующему объекту коллекции. Табл. 24.1 приводит список предопределенных констант, обозначающих различные объекты `MAPIFolder`.

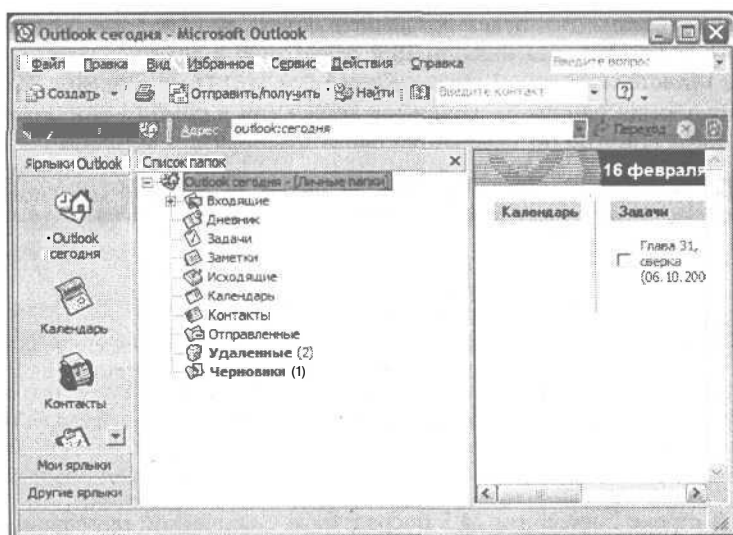


Рис. 24.1. Прикладная программа способна получить доступ к содержимому стандартных папок Outlook

Таблица 24.1. Предопределенные константы для обозначения объектов `MAPIFolder`

Константа	Название лапки и описание объекта
<code>olFolderCalendar</code>	Календарь; коллекция <code>items</code> содержит объекты <code>AppointmentItem</code>
<code>olFolderContacts</code>	Контакты; коллекция <code>items</code> содержит объекты <code>ContactItem</code>
<code>olFolderDeletedItems</code>	Удаленные; коллекция <code>items</code> содержит объекты <code>MailItem</code>
<code>olFolderDrafts</code>	Черновики; коллекция <code>items</code> содержит объекты <code>MailItem</code>
<code>olFolderInbox</code>	Входящие; коллекция <code>Items</code> содержит объекты <code>MailItem</code>
<code>olFolderJournal</code>	Дневник; коллекция <code>items</code> содержит объекты <code>JournalItem</code>

Константа	Название папки и описание объекта
<code>olFolderNotes</code>	Заметки; коллекция <code>Items</code> содержит объекты <code>NoteItem</code>
<code>olFolderOutbox</code>	Исходящие; коллекция <code>items</code> содержит объекты <code>MailItem</code>
<code>olFolderSentMail</code>	Отправленные; коллекция <code>items</code> содержит объекты <code>Mailitem</code>
<code>olFolderTasks</code>	Задачи; коллекция <code>items</code> содержит объекты <code>TaskItem</code>

Подробности, касающиеся объектов `MAPIFolder` каждого из указанных в таблице видов, вы сможете найти в оперативной справочной системе Outlook.

Объект `MAPIFolder` содержит коллекцию `Items`. Вы уже знаете, что такое коллекции и как с ними работать. Тип элементов коллекции (см. табл. 24.1) зависит от разновидности объекта `MAPIFolder`. Маленький пример: в результате вызова `GetDefaultFolder(olFolderContacts)` будет возвращен объект `MAPIFolder`, который содержит информацию о контактах. Эта информация оформлена в виде коллекции, каждый элемент которой представляет собой объект класса `ContactItem`. Листинг 24.3 демонстрирует способ получения одного из элементов коллекции в объекте `MAPIFolder`, который ссылается на предлагаемую по умолчанию папку с данными о контактах.

#### Листинг 24.3. Пример получения информации о контактах

```

1: Sub PrintOneContact( )
2:   Dim Outlook As New Outlook.Application
3:   Dim NameSpace As NameSpace
4:   Set NameSpace = Outlook.GetNameSpace( "MAPI" )
5:   Dim Folder As MAPIFolder
6:   Set Folder = NameSpace.GetDefaultFolder(olFolderContacts)
7:   Dim Contact As ContactItem
8:   Set Contact = Folder.Items("Lori Kimmel")
9:   Debug.Print Contact.FullName
10:End Sub

```

#### Анализ

В строке 2 листинга 24.3 посредством статической привязки создается экземпляр Outlook. Строка 3 содержит объявление переменной типа `NameSpace`, которая инициализируется в строке 4. В строке 5 объявляется переменная `Folder` типа `MAPIFolder`; в строке 6 ей присваивается значение объекта `MAPIFolder`, который ссылается на папку с информацией о контактах. Строка 7 содержит выражение объявления переменной `Contact` типа `ContactItem`. В строке 8 выполняется поиск элемента коллекции, в свойстве `FullName` которого хранится значение "Lori Kimmel", и присвоение этого элемента переменной `Contact`. В качестве индекса вместо строки допускается задавать целочисленное выражение, значение которого не превышает величину, хранящуюся в `Items.Count`. Команда строки 9 отображает значение свойства `FullName` выбранного объекта-элемента коллекции в окне Immediate. Более лаконичный вариант той же процедуры приведен в листинге 24.4.

#### Листинг 24.4. Сокращенный вариант процедуры листинга 24.3

```

1: Sub BriefPrintOneContact( )
2:   Dim Outlook As New Outlook.Application

```

```

3:   Debug.Print _
      Outlook.GetNameSpace("MAPI").GetDefaultFolder(
          olFolderContacts ).Items("Lori Kimmel")
4: End Sub

```

#### Анализ

В строке 2 объявляется и создается экземпляр Outlook, а строка 3 содержит выражение, иллюстрирующее иерархию членства объектов, относящихся к классу NameSpace.

## Просмотр информации о контактах Outlook в Access

В настоящем и последующих разделах главы содержится ряд примеров практического использования информации Outlook во внешнем приложении. Листинг 24.5 демонстрирует способы доступа к данным о контактах посредством манипуляций с соответствующим объектом MAPIFolder.

**Листинг 24.5. Пример процедуры для просмотра информации о контактах Outlook**

```

1: Sub OpenContactsFolder( )
2:   Dim Outlook As New Outlook.Application
3:   Dim NameSpace As NameSpace
4:   Set NameSpace = Outlook.GetNameSpace( "MAPI" )
5:   Dim ContactFolder As MAPIFolder
6:   Set ContactFolder =
       NameSpace.GetDefaultFolder( olFolderContacts )
7:   ContactFolder.Display
8: End Sub

```

#### Анализ

В строках 2–6 листинга 24.5 выполняются операции создания экземпляра Outlook и объекта NameSpace, а также построения ссылки на папку, содержащую информацию о контактах. Метод Display класса MAPIFolder, который вызывается в строке 7, открывает окно экземпляра Outlook, созданного ранее, и передает фокус элементу Контакты (Contacts) дерева, отображаемого в дочернем окне Список папок. Листинг 24.5 демонстрирует один из способов управления пользовательским интерфейсом Outlook из внешней прикладной программы.

Приведенный пример, текст которого отображен в листинге 24.6, иллюстрирует возможности доступа к каждому из элементов коллекции контактов. Обладая ссылкой на элемент данных, вы можете решать какие угодно задачи.

**Листинг 24.6. Пример процедуры циклической обработки элементов данных о контактах**

```

1: Sub IterateContacts( )
2:   Dim Outlook As New Outlook.Application
3:   Dim NameSpace As NameSpace
4:   Set NameSpace = Outlook.GetNameSpace( "MAPI" )
5:   Dim ContactFolder As MAPIFolder
6:   Set ContactFolder = NameSpace.GetDefaultFolder(olFolderContacts)
7:   Dim ContactItem As Variant
8:   For Each ContactItem In ContactFolder.Items

```



```

9:      Debug.Print ContactItem
10:   Next ContactItem
11:End Sub

```

#### Анализ

Строки 1–6 вы видели уже несколько раз. Строка 6 содержит инструкцию получения ссылки на объект `MAPIFolder`, имеющий отношение к информации о контактах. Атрибут `Items` класса `MAPIFolder` — это обычная коллекция, и к ней применимы все известные вам операции. Например, задача прохождения по элементам коллекции легко решается с помощью цикла `For Each ... Next`, индекс которого задается переменной типа `Variant` (строка 7). В этой ситуации, кроме того, применима и циклическая конструкция `For ... Next`, на каждом шаге которой очередной элемент коллекции должен быть присвоен переменной типа `Variant`.

Процедура листинга 24.6 осуществляет вывод содержимого элемента коллекции контактов в окно `Immediate`; впрочем, внутри цикла вы можете выполнять любые другие операции — скажем, редактирование элемента данных, его форматирование или печать, отображение в графических визуальных компонентах, занесение в файл отчета или пересылку по электронной почте. В следующем разделе предлагаются примеры, демонстрирующие способы изменения данных Outlook.

## Обновление информации Outlook из внешних приложений

Изменяется наша жизнь — изменяется и информация. Телефонные компании, озабоченные и одновременно озабоченные ростом количества абонентов, вынуждены добавлять к телефонным номерам новые префиксы. Фирмами и физическими лицами ежедневно открываются сотни Web-сайтов. Адресаты переходят на новую работу, компании переезжают в новые офисы — как уследить за непрерывно обновляющейся информацией?

Производительность будет минимальной, если вы будете перебирать вручную десятки, а то и сотни адресов или телефонных номеров. Напишите несколько процедур и возложите все эти неприятные обязанности на плечи Access. Листинг 24.7 демонстрирует простой пример, связанный с модификацией телефонных номеров. Немного фантазии — и вы сможете расширить эту процедуру, снабдив ее средствами поддержки графического интерфейса, ведения протокола изменений и т.п.

Листинг 24.7. Пример процедуры обновления данных о контактах

```

1: Sub UpdateAreaCodes ( )
2:   Dim Outlook As New Outlook.Application
3:
4:   Dim NameSpace As NameSpace
5:   Set NameSpace = Outlook.GetNameSpace( "MAPI" )
6:
7:   Dim ContactFolder As MAPIFolder
8:   Set ContactFolder = NameSpace.GetDefaultFolder(olFolderContacts)
9:
10:  Dim I As Integer, Offset As Integer
11:
12:  For I = 1 To ContactFolder.Items.Count

```

```

13:      With ContactFolder.Items( I )
14:          If (InStr(1, .BusinessTelephoneNumber, "(313)" > 0)
            Then
15:              .BusinessTelephoneNumber =
16:                  Replace(.BusinessTelephoneNumber,
                        "(313)", "(810)")
17:              .Save
18:          End If
19:      End With
20:  Next I
21:End Sub

```

#### Анализ

Строки 1–8 получают объект `MAPIFolder` для контактов. Строка 10 содержит объявление целочисленной переменной, которая будет использоваться в цикле. Заголовок цикла размещен в строке 12. Следующая за ним конструкция `with` призвана исключить необходимость повторения постоянного префикса `ContactFolder.Items (I)` при наборе кода.

Принадлежность переменной очередному элементу коллекции, `Items(I)`, обусловливается символом точки (`.`), предшествующим имени, и самим фактом упоминания переменной внутри блока `With ... End With`. (Если мне не изменяет память, конструкция `With ... End with` на страницах нашей книги еще не появлялась. Более подробные сведения о ней вы найдете далее, рядом с пиктограммой примечания.) Остальная часть текста процедуры листинга 24.7 должна быть вам понятна.



Конструкция на основе служебного слова `with` облегчает задачу многократного обращения к атрибутам одного и того же глубоко вложенного объекта. Например, выражение `With ContactFolder.Items (I)` позволяет избежать необходимости повторения постоянного префикса, ссылающегося на элемент коллекции `Items` из состава `ContactFolder`. Чтобы указать на принадлежность атрибута этому объекту, достаточно ввести оператор точки (`.`) — скажем, `.BusinessTelephoneNumber`.

## Поиск заданной строки в сообщениях электронной почты

Объекты класса `MailItem` используются в нескольких "почтовых" коллекциях данных (или, с точки зрения пользователя, папках) **Outlook** — это (см. рис. 24.1 и табл. 24.1) Входящие (Inbox), Исходящие (Outbox), Отправленные (Sent Items), Черновики (Drafts) и Удаленные (Deleted Mail). Объект `MailItem` сохраняет информацию об отправителе и получателях сообщения, а также о его содержимом. В этом разделе будет рассмотрен пример процедуры, решающей задачу поиска литеральной символьной строки в сообщениях электронной почты.

Вы наверняка мечтали о волшебном решетке, которое смогло бы помочь быстро просеивать сообщения электронной почты. Теперь вы получаете реальную возможность самостоятельного решения подобной задачи. Все, что необходимо — это небольшой блок кода на языке VBA и Automation-сервер Outlook. Внимательно изучите листинг 24.8.



Свойство Body объекта MailItem содержит текст сообщения. Если текст написан на языке HTML, при поиске или просмотре данных возможны нежелательные эффекты и искажения.

#### Листинг 24.8. Пример процедуры поиска символьной строки в сообщениях электронной почты

```
1: Sub Test ( )
2:   Dim Outlook As New Outlook.Application
3:   Dim Namespace As NameSpace
4:   Set Namespace = Outlook.GetNameSpace( "MAPI" )
5:   Dim DeletedItems As MAPIFolder
6:   Set DeletedItems =
       Namespace.GetDefaultFolder( olFolderDeletedItems )
7:   Dim MailItem As MailItem
8:
9:   Dim I As Integer
10:
11:   Dim RefCount As Long
12:   Dim As Long
13:   Dim Count As Long
14:
15:   RefCount = 0
16:   ItemRefCount = 0
17:
18:   For I = 1 To DeletedItems.Items.Count
19:
20:     If TypeOf DeletedItems.Items(I) Is MailItem Then
21:       Set MailItem = DeletedItems.Item(I)
22:       Count = CountWord( "Microsoft", MailItem)
23:       If (Count > 0) Then
24:         ItemRefCount = ItemRefCount + 1
25:         RefCount = RefCount + Count
26:       End If
27:     End If
28:
29:   Next I
30:
31:   Debug.Print "Всего удаленных: " & _
       DeletedItems.Items.Count
32:   Debug.Print "Всего ссылок на Microsoft: " & RefCount
33:   Debug.Print "Удаленных, содержащих ссылку " & _
34:       Round((ItemRefCount / DeletedItems.Items.Count) _
       * 100, 2) & "%"
35:
36: End Sub
37:
38:
39: Function CountWord(ByVal Word As String, _
40:   ByVal MailItem As MailItem ) As Long
41:
42:   Dim P As Long
43:   P = 1
44:   Do While (True)
45:     P = InStr( P + 1, MailItem.Body, Word )
46:     If (P > 0) Then
```

```

47:         CountWord = CountWord + 1
48:     Else
49:         Exit Do
50:     End If
51: Loop
52: End Function

```

#### Анализ

Процедура Test, представленная в листинге 24.8, создает объект Outlook, получает ссылку на папку Удаленные и вызывает функцию CountWord, передавая в качестве параметров искомую литеральную строку и адрес одного из объектов коллекции удаленных сообщений. В окне Immediate выводится общее число удаленных сообщений, число удаленных сообщений, содержащих ссылку на Microsoft и их процентное соотношение в общем количестве удаленных сообщений.

Ранее вы не встречались с оператором TypeOf, используемым в строке 22. Выражение TypeOf Объект Is Класс используют для того, чтобы динамически определять, является ли указанный объект элементом заданного класса. Вспомните: коллекции могут хранить данные любого типа. В листинге 24.8 заложена информация о том, что коллекция DeletedItems.Item содержит объекты папок Outlook — нам нужно лишь найти объект Mailltem. В ходе работы программы проверяются только удаленные объекты Mailltem, а, например, удаленные заметки (объекты Noteltem) коллекции DeletedItems.Item не проверяются.

Функции CountWord, расположенной в строках 39–52, передаются в качестве аргументов Word и Mailltem. Здесь, используя простой цикл вида Do While ... Loop, стандартная функция InStr отыскивает в теле сообщения экземпляры строки аргумента и подсчитывает их количество. Если результат поиска неуспешен, цикл завершается и функция возвращает значение 0.

## Восстановление удаленных сообщений

Объекты MAPI Folder разрешается перемещать из одной коллекции (или папки) в другую. Например, вам может понадобиться сохранить архив сообщений в новой папке MAPI Folder. Другой пример связан с необходимостью автоматического восстановления ранее удаленных сообщений на основе определенных критериев поиска. Листинг 24.9 содержит текст процедуры, предназначенной для восстановления (посредством команды Move) сообщений с определенным именем отправителя.

Листинг 24.9. Пример процедуры восстановления удаленных сообщений, отвечающих заданному критерию поиска

```

1: Sub TestUndelete( )
2:     Call UndeleteMailFrom( "Иван Иванов" )
3: End Sub
4:
5: Sub UndeleteMailFrom( ByVal SenderName As String )
6:     Dim Outlook As New Outlook.Application
7:     Dim Folder As MAPIFolder
8:     Set Folder =
9:         Outlook.GetNamespace( "MAPI" ) _
            .GetDefaultFolder( olFolderDeletedItems )
10:    Dim DeletedItem As Mailltem
11:
12:    For Each DeletedItem In Folder.Items
13:        If (TypeOf DeletedItem Is Mailltem) Then

```

```

14:             If (DeletedItem.SenderName = SenderName) Then
15:                 Call DeletedItem.Move(GetNameSpace("MAPI") _
16:                     .GetDefaultFolder(olFolderInbox) )
17:             End If
18:         End If
19:     Next
20:End Sub

```



Не удивляйтесь, если Outlook отобразит сообщение с требованием подтвердить, что вы позволяете другой программе, в данном случае Access, отправлять сообщения от вашего имени. Это часть новой системы безопасности Outlook. Щелкните на кнопке Да (Yes) и позвольте VBA-коду немного поработать со своими сообщениями.

При создании контроллеров автоматизации для Outlook посоветуйтесь с администратором сети по поводу возможности отключить средства безопасности для почтовых сообщений.

#### Анализ

В строках 1–3 листинга 24.9 размещена подпрограмма, предназначенная для тестирования основной процедуры (UndeleteMailFrom), приведенной ниже. В строках 5–8 создаются экземпляр Outlook и объект MAPIFolder, ссылающийся на папку DeletedItems. Конструкция цикла For Each была выбрана по той причине, что при удалении элемента коллекции изменяется значение атрибута Count. Применение же в подобном случае цикла вида For ... Next требует наличия дополнительных проверок, позволяющих избежать возникновения ошибки выхода индекса за допустимые границы. Условное выражение в строке 13 проверяет, является ли удаленный элемент почтовым сообщением, а в строке 14 осуществляется сопоставление имени отправителя сообщения с содержимым аргумента процедуры. В случае успеха сообщение перемещается из папки Удаленные в папку Входящие.



При удалении элементов коллекции можно использовать цикл For Next. Просто выполняйте итерации от элемента с индексом Count до первого элемента с шагом, равным -1, т.е. от последнего элемента к первому. При удалении элемента размер коллекции уменьшится, но наименьший индекс (равный 1) останется неизменным.

В листинге 24.9 цикл For Each использован для ясности, здесь элемент коллекции присваивается временной переменной. Подробнее о циклах и условных операторах см. главу "5-й час. Программирование управляющих структур".

Еще один вариант выполнения операции сравнения связан с использованием функции strCmp, позволяющей управлять признаком чувствительности к регистру символов.

## Автоматическая рассылка почтовых сообщений

Задача автоматической рассылки почты возникает в самых разных ситуациях. Изобретать что-то новое вовсе не обязательно, да и просто нецелесообразно — воспользуйтесь готовыми средствами Outlook. Подобный подход особенно эффективен для небольших компаний, не имеющих возможность финансировать крупные программные проекты. Листинг 24.10 показывает, насколько простым может быть решение такой задачи.



Я крайне отрицательно отношусь к использованию высокотехнологичных программных средств в сомнительных целях — в данном случае имеется в виду автоматическая рассылка так называемых нежелательных почтовых сообщений (spam). Некоторые личности, пытающиеся заявить миру о своем существовании, засоряют каналы Internet и почтовые ящики добропорядочных граждан лавиной электронного мусора, причиняя тем самым немалый материальный и моральный ущерб.

Новая система безопасности Outlook 2002 препятствует доступу VBA-кода к контактам. Это связано с распространением в последние годы вирусов, которые, являясь VBA-приложениями, считывают ваши контакты и рассылают по ним сообщения, выполняя подобную операцию на компьютерах всех получателей таких писем.

#### Листинг 24.10. Пример процедуры автоматической рассылки почтовых сообщений

```
1: Sub MailEveryone( )
2:   Dim Outlook As New Outlook.Application
3:   Dim Folder As MAPIFolder
4:   Set Folder = Outlook.GetNameSpace( "MAPI" ) _
5:     .GetDefaultFolder( olFolderContacts )
6:   Dim Contact As Variant
7:   Dim MailItem As MailItem
8:   For Each Contact In Folder.Items
9:     Set MailItem = Outlook.CreateItem( olMailItem )
10:    MailItem.To = Contact.FullName
11:    MailItem.Subject = "Тест массовой рассылки"
12:    MailItem.Body = "Тестовое сообщение"
13:    MailItem.Send
14:  Next
15: End Sub
```

#### Анализ

Код листинга 24.10 предполагает просмотр адресной книги и автоматическую отправку каждому зарегистрированному в ней абоненту тестового сообщения. Минимальный набор действий таков: следует создать объект сообщения (строка 9) и указать в нем адрес получателя (строка 10). Если этим ограничиться, адресаты получат пустые сообщения. Команды строк 11 и 12 позволяют указать тему сообщения и наполнить его реальным содержанием. Метод Send, вызываемый в строке 13, выполняет отправку сообщения по назначению.

## Резюме

Неужели? Да-да, это правда. Вы одержали победу — примите поздравления! Приобретенный опыт дался вам, смеем надеяться, относительно легко — пустившись в свободное плавание на свой страх и риск, вы затратили бы гораздо больше времени и усилий.

Главная цель этого занятия состояла в том, чтобы продемонстрировать вам всю мощь Access, VBA и объектно-ориентированного подхода в целом. Для вас наверняка не был новым тот факт, что приложения Microsoft Office XP могут осуществлять взаимный обмен информацией — теперь же вы узнали о способах коллективного доступа к объектам на программном уровне. Outlook — мощное самостоятельное приложение, и, разумеется, совершенно невозможно втиснуть исчерпывающее описание лежащей

в его основе объектной модели в тесные рамки одной-единственной главы. Поэтому автор уделит внимание особенно важным вопросам.

Столкнувшись с задачей управления данными, имеющими отношение к электронной почте, смело обращайтесь к функциям Outlook — вряд ли имеет смысл изобретать что-то новое.

Теперь, по завершении последнего занятия, вы можете создавать и использовать объекты — и это самое главное. В данном случае неважно, идет речь об объектах Access, Outlook или каких-либо других. Каждый объект содержит методы и свойства — достаточно научиться правильно их применять.

Благодарю вас за приятное общество. Позвольте напоследок еще раз воспользоваться правом автора и попросить вас все-таки дочитать главу до конца.

## Вопросы и ответы

**Вопрос.** Можно ли программировать на языке VBA непосредственно в среде Outlook?

**Ответ.** Да, конечно. Хотя наша книга не об этом, тот же Visual Basic for Application, уже знакомый вам по Access, поддерживается в Outlook, Word, Excel, PowerPoint и FrontPage. Чтобы приобщиться к таинствам программирования в среде любого из этих приложений, вам потребуется приобрести и изучить соответствующую литературу, поскольку имеют место различия как в структурах моделей объектов, так и в способах их использования.

**Вопрос.** Можно ли в среде Outlook создавать формы?

**Ответ.** Да. Инструментальная среда Outlook достаточно богата — для подробного освещения всех ее возможностей одной главы, подобной этой, явно недостаточно.

## Задания

Ниже приведены тестовые задания и упражнения, которые помогут вам самостоятельно проверить уровень освоения материала, изложенного в этой главе. Ответы на вопросы вы найдете в приложении.

## Тесты

1. Как называется объект верхнего уровня, предоставляющий средства доступа к папкам Outlook?
2. Какая из разновидностей (пока единственная) объектов `NameSpace` существует в настоящее время?
3. Назовите тип универсального объекта для хранения коллекций сообщений всех видов.
4. Как называется метод, позволяющий перемещать документы Outlook из одной папки (коллекции) в другую?
5. Каково наименование объекта для хранения отдельных почтовых сообщений?

## Упражнения

1. Напишите фрагмент кода, позволяющий создать новый объект типа `JournalItem`.
2. Сохраните созданный объект `JournalItem` в папке Outlook.
3. Укажите выражение, позволяющее передать фокус объекту Дневник (`Journal`) в списке папок окна Microsoft Outlook.



# Приложение

## Ответы

Ниже приведены ответы на тестовые вопросы и упражнения, которыми завершается каждая глава книги.

### 1 -й час. Новинки Access 2002

#### Тесты

1. Для чего предназначен элемент меню Развернуть?  
Развернуть — это элемент меню, позволяющий отобразить те пункты меню, которые в данный момент скрыты. Access обычно отображает только ограниченное множество элементов меню.
2. Если элементу меню или кнопке панели инструментов назначить гиперссылку, то "поведение" этого интерфейсного объекта изменится. Верно ли это?  
Верно. Функция объекта, предусмотренная по умолчанию, изменится. Вероятно, более целесообразно пользоваться для подобных целей вновь созданными кнопками панелей инструментов или элементами меню.
3. Функция проверки правописания подвергает анализу выражения SQL и такие простые образцы текстовых данных, как сокращения и произвольные последовательности символов. Верно ли это?  
Неверно. Функция проверки правописания воздействует только на поля данных и игнорирует ту информацию, которую не в состоянии интерпретировать как набор слов.
4. Каково наименование библиотеки, содержащей объекты Procedure, предназначенные для создания хранимых процедур?  
ActiveX Data Objects Extensions 2.7 for DDL and Security (ADOX).
5. Access 2002 имеет в своем составе функции-мастера, позволяющие создавать Web-страницы, связанные с объектами баз данных. Верно ли это?  
Верно. Мастера Data Pages Wizards помогут в создании Web-страниц, представляющих в окне браузера как простые, так и сложные объекты данных.



## Упражнения

1. Откройте панель инструментов Собрание по сети (Online Meeting) и закрепите ее у верхней границы окна Access.

Выберите в строке меню команду Сервис⇒Настройка (Tools⇒Customize).

Перейдите на вкладку Панели инструментов (Toolbars) диалогового окна Настройка (Customize), пролистайте одноименный список и установите флажок для панели Собрание по сети.

Щелкните на кнопке Закрыть (Close), а затем перетащите выбранную панель к верхней границе окна Access.

2. Добавьте новое меню в конец строки меню.

Выберите в строке меню команду Сервис⇒Настройка.

Перейдите на вкладку Команды (Commands) диалогового окна Настройка, пролистайте список Категории (Categories) вниз до конца, щелкните на элементе Новое меню (New Menu), а затем перетащите мышью одноименный элемент из списка Команды в конец строки меню.

3. Дайте созданному меню новое имя — Контакты.

Не закрывая диалогового окна Настройка, щелкните правой кнопкой мыши в пределах нового меню, перейдите к элементу Имя (Name) контекстного меню, введите в текстовом поле новое название, Контакты, и нажмите клавишу <Enter>.

4. Назначьте интерфейсному объекту гиперссылку, указывающую на Web-сайт издательства, которое выпустило оригинальную версию нашей книги (<http://www.samspublishing.com>).

Откройте диалоговое окно Настройка и перейдите на вкладку Команды.

Подберите категорию команд, подходящую для решения поставленной задачи (открытия страницы Web-сайта). Рекомендуем обратиться к категории Web.

Найдите в списке справа подходящую команду выбранной категории (например, Открыть), перетащите мышью соответствующий элемент списка и опустите его в пределах ранее созданного меню Контакты.

С помощью контекстного меню, вызываемого щелчком правой кнопки мыши, измените имя нового элемента меню на &Открыть Web-сайт Sams и назначьте для него гиперссылку <http://www.samspublishing.com>, введя команду Назначить гиперссылку⇒Открыть (Assign Hyperlink⇒Open) и заполнив поле одноименного диалогового окна.

5. Установите посредством NetMeeting соединение с компьютером пользователя в Internet либо в пределах корпоративной сети.

Загрузите приложение Access 2002 и выберите в строке меню команду Сервис⇒Совместная работа⇒Начать собрание (Tools⇒Online Collaboration⇒Meet Now).

Следуя рекомендациям программы-мастера, настройте параметры NetMeeting.

Установив соединение, выберите собеседника с помощью средств диалогового окна Вызов (Find Someone).

Если у вас нет микрофона и громкоговорителя, обратитесь к строке меню окна приложения Microsoft NetMeeting и введите команду Сервис⇒Разговор (Tools⇒Chat), чтобы инициировать общение в режиме *chat*.

## 2-й час. Познакомимся с VBA

### Тесты

1. Каков наиболее предпочтительный тип данных для хранения чисел с плавающей запятой?  
Double.
2. К какому типу должна относиться переменная, предназначенная для присвоения результата вещественного деления?  
Double.
3. В чем состоит отличие операторов (+ и &) сложения строк?  
Оператор сложения строк, обозначаемый символом **амперсанда (&)**, дополнительно осуществляет неявное преобразование типов данных.
4. Чем отличается объявление переменной от ее инициализации?  
Объявление переменной имеет **целью** указать ее имя и тип. В процессе инициализации переменной присваивается некоторое значение.
5. Какое служебное слово применяется для объявления и инициализации постоянных значений (констант)?  
Const.

### Упражнения

1. Напишите выражения объявления переменной для хранения даты и инициализации этой переменной значением текущей даты.  

```
Dim ADate As Date  
ADate = Date
```
2. Составьте выражение для вычисления суммы числа и того же числа, помноженного на дробь.  

```
ADouble = 10 + (10 * .06)
```
3. Напишите выражение для сложения строк, выполняющее неявное преобразование типов данных.  

```
AString = "Сегодня не " & 4 & " июля!"
```

## 3-й час. Принципы работы программы с данными

### Тесты

1. Какое отличие существует между явным и неявным определением переменной?  
Перед первым использованием явного определения переменной применяются выражения, содержащие служебные слова Dim, ReDim, Const и Global. Если переменная просто вводится в текст (без дополнительных разъяснений относительно ее типа), это воспринимается компилятором как неявное определение.
2. С помощью каких средств языка можно исключить потенциально опасные возможности неявного определения переменных?

Если в верхнюю часть текста модуля вводится выражение `Option Explicit`, то все последующие определения переменных должны быть явными, иначе программа выдаст сообщение об ошибке.

3. Для каких целей предназначен формат объявления переменных, использующий служебное слово `ReDim`?

Формат объявления с помощью служебного слова `ReDim` применяется при определении динамических массивов.

4. В каком месте кода следует объявлять глобальные переменные?

Переменные `Global` объявляются в верхней части текста модуля.

5. Данные каких типов могут быть присвоены переменным типа `Variant`? Следует ли широко использовать объекты типа `Variant`?

Переменным типа `Variant` могут быть присвоены значения любых типов. Применение объектов типа `Variant`, однако, сопряжено с потерей "прозрачности" кода, накладными расходами, связанными с необходимостью дополнительного анализа типов переменных во время выполнения программы, а также потенциальной опасностью непредвиденного искажения значений.

## Упражнения

1. Напишите выражение объявления константы `PI` за пределами подпрограммы `SquareOfCircleCalc`, приведенной в листинге 3.3.

```
Const PI = 3.14159
Sub CalculateCircumference( )
    Dim Circumference As Double, Radius As Double
    ' далее по тексту листинга 3.3
```

2. Исправьте код листинга 3.3 таким образом, чтобы в выражениях использовалась новая глобальная константа `PI`.

```
Const PI = 3.14159
Sub CalculateCircumference ( )
    Dim Circumference As Double, Radius As Double
    Radius = 10
    Circumference = PI * Radius ^ 2
End Sub
```

3. Постройте выражение объявления динамического массива для хранения десяти чисел двойной точности.

```
ReDim MyDoubles(9) As Double
```

По умолчанию первым элементом массива будет `MyDoubles(0)`.

## 4-й час. Последовательность действий и выполнение вычислений

### Тесты

1. Для каких целей применяется оператор `+`?  
Для сложения чисел и строк.
2. Назовите четыре основные группы операторов.  
Арифметические, логические, операторы сравнения и конкатенации строк.

3. Каков приоритет выполнения групп операторов?  
В порядке убывания — арифметические, операторы сравнения, логические. Оператор конкатенации строк (&) здесь условно отнесен к группе арифметических операторов.
4. Как изменить естественный порядок выполнения операций?  
Заклучите отдельные высказывания всего выражения в круглые скобки.
5. Операторы какой группы обладают наивысшим приоритетом выполнения?  
Первыми выполняются арифметические операторы.

## Упражнения

1. Напишите текст подпрограммы, демонстрирующей все возможные сочетания значений аргументов оператора And.  

```
Sub TruthTable( )
    Dim Test As Boolean
    Test = False And False
    Test = False And True
    Test = True And False
    Test = True And True
End Sub
```
2. Используя окно непосредственного исполнения кода (Immediate) интегрированной среды программирования Microsoft Visual Basic, вычислите значения следующих выражений на основе побитового оператора Or: 2 Or 3; 4 Or 5; 6 Or 7.  

```
2 Or 3 = 2
4 Or 5 = 4
6 Or 7 = 6
```
3. С помощью средств окна Immediate вычислите остаток от деления пар чисел, приведенных в предыдущем упражнении.  

```
2 Mod 3 = 0
4 Mod 5 = 0
6 Mod 7 = 0
```

## 5-й час. Программирование управляющих структур

### Тесты

1. Как определить цикл для обработки элементов массива из 10 целочисленных значений?  

```
Dim I As Integer
For I = 1 To 10
    ' тело цикла
Next I
```
2. Если требуется создать такой цикл, код которого должен выполняться по меньшей мере один раз, то какую конструкцию вы выберете?  
Следует применить одну из конструкций — Do ... Loop While либо Do ... Loop Until — поместив условное выражение в предложение, содержащее Loop.
3. Как убедиться, что код выражения If работает верно?  
Необходимо подобрать соответствующие значения данных и выполнить код в режиме отладки.

4. Для каких целей предназначена инструкция Exit For?  
Она используется для принудительного прерывания цикла вида For . . . Next.
5. Что представляют собой средства прерывания циклов? При каких обстоятельствах их следует использовать?  
Средства прерывания циклов — это предопределенные инструкции, позволяющие покинуть цикл до его естественного завершения. Они используются в таких случаях, когда, например, по условию задачи невозможно построить тестовое выражение, позволяющее корректно завершить выполнение цикла, либо в процессе вычислений возникают особые обстоятельства, диктующие необходимость выполнения подобных действий.

## Упражнения

1. Применив средства оперативной справки Microsoft Visual Basic, отыщите информацию о функции Switch. Напишите код на основе Switch, позволяющий найти имя человека по заданной фамилии среди нескольких пар строк вида *имя-фамилия*.

```
Switch( [FirstName] = "Paul", "Kimmel", _
      [FirstName] = "Robert", "Dearman", _
      [FirstName] = "Robert", "Golieb", _
      [FirstName] = "Greg", "Smith")
```

2. Обратившись к системе оперативной справки, найдите сведения о функции Iff. Приведите конкретный пример ее использования.

```
Iff(Income > 50000, "Высокий доход", "Низкий доход")
```

3. Объясните объект коллекции и примените конструкцию For Each для отображения содержимого каждого элемента коллекции с помощью процедуры MsgBox.

```
Sub For Each Demo()
Dim C As New Collection
Call C.Add ("Январь")
Call C.Add ("Февраль")
Dim Elem As Variant
For Each Elem In C
MsgBox Elem
Next Elem
```

## 6-й час. Управление базами данных

### Тесты

1. Какие объекты данных, помимо таблиц, могут адресоваться средствами RecordSet? Запросы и курсоры.
2. Как в среде Access создать новый программный модуль?  
Откройте окно базы данных. В списке Объекты (Objects) выберите элемент Модули (Modules). Щелкните на кнопке Создать (New) панели инструментов.
3. Каким образом можно протестировать код модуля?  
В окне редактора Microsoft Visual Basic переместите текстовый курсор к нужной строке процедуры или функции, а затем используйте клавишу <F8> для прохождения по строкам кода в режиме отладки.
4. Для каких целей применяется объект класса Catalog?  
Объект класса Catalog применяется для ссылки на таблицы, курсоры, учетные записи пользователей и групп пользователей.

5. Каким образом установить указатель записей объекта класса `Recordset` на первую запись?  
Если создан объект `MyRecordSet` класса `Recordset`, следует использовать вызов `MyRecordSet.MoveFirst`.
6. Какую разновидность циклов предпочтительно использовать для обработки заранее известного количества объектов данных?  
Если известны нижняя и верхняя границы изменения переменной цикла, наиболее удобно использовать циклическую конструкцию вида `For ... Next`.

## Упражнения

1. Исправьте код процедуры `CreateTable`, приведенной в листинге 6.2, таким образом, чтобы добавить в определение таблицы `CONTACTS` столбцы для хранения адреса (`ADDRESS`), названия города (`CITY`), области (`REGION`) и почтового индекса (`ZIP`).

Необходимо добавить в текст листинга 6.2 после строки 19 следующие строки:

```
Table.Column.Append "ADDRESS", adVarChar, 30
Table.Column.Append "CITY", adVarChar, 20
Table.Column.Append "REGION", adVarChar, 20
Table.Column.Append "ZIP", adVarChar, 5
```

2. Определите SQL-запрос, отображающий данные из всех столбцов таблицы `CONTACTS`, упорядоченные по возрастанию почтового индекса (`ZIP`).

```
SELECT * FROM CONTACTS ORDER BY ZIP;
```

3. Применив в качестве примера код листинга 6.5 и воспользовавшись выражением SQL, построенным при выполнении предыдущего упражнения, напишите процедуру поиска записи по заданному почтовому индексу (`ZIP`).

В строку 12 листинга 6.5 вставьте выражение

```
Const SQL = "SELECT * FROM CONTACTS ORDER BY ZIP"
```

В строке 13 замените литерал `"CONTACTS"` идентификатором константы SQL:

```
RecordSet.Open SQL, Catalog.ActiveConnection, _
adOpenDynamic, adLockOptimistic
```

Измените содержимое строк 21–35 следующим образом:

```
Temp = InputBox( "Введите почтовый индекс (Q=Выход):", _
    "Поиск по почтовому индексу")
Do While (RecordSet.EOF = False And Temp <> "Q")
    If (Temp = RecordSet( "ZIP").Value) Then
        MsgBox "Найдена: " & RecordSet( "ZIP").Value & _
            " в " & RecordSet( "ID" ).Value
        Exit Do
    End If
    RecordSet.MoveNext
Loop
```

## 7-й час. Расширенные типы данных

### Тесты

1. На какое количество записей способен одновременно ссылаться объект типа `RecordSet`?  
Только на одну.
2. Какому классу принадлежит коллекция `Tables` — `Catalog` или `Connection`?  
`Catalog`.

3. С какими объектами данных позволяет работать объект Catalog?  
С таблицами, курсорами, хранимыми процедурами, учетными записями пользователей и групп пользователей.
4. Назовите два общих вида услуг (функций), предоставляемых объектами класса Catalog.  
Определение данных и ограничение доступа к ним.
5. Какие объекты и методы применяются для циклического прохождения по набору записей таблицы?  
**Объект** ADODB.Recordset **и его методы** MoveFirst, MoveLast, MovePrevious, MoveNext, Move, BOF и EOF.

## Упражнения

1. Создайте запрос для рабочей базы данных и сохраните его. Откройте запрос с помощью фрагмента кода, использующего объект типа Recordset.  

```
Dim Rs As New ADODB.Recordset
Rs.Open "LIBRARY_Query", CurrentProject.Connection
```
2. Напишите процедуру, предусматривающую возможность циклического прохождения по всем ключам, хранимым в объекте Catalog.  
Исправьте строки 5-7 листинга 7.5 следующим образом:  

```
5: For I = 0 To Catalog.Keys.Count - 1
6:     Debug.Print Catalog.Keys( I ).Name
7:     Next I
```
3. Создайте подпрограмму для прохождения по набору записей в обратном порядке, начиная с последней.  

```
Sub QueryAll( )
    Dim Rs As New ADODB.Recordset
    Rs.Open "LIBRARY_Query", _
        CurrentProject.Connection, adOpenKeyset
    Rs.MoveLast
    Do While (Rs.BOF = False)
        Rs.MovePrevious
    Loop
    Rs.Close
End Sub
```

## 8-й час. Декомпозиция задач

### Тесты

1. Назовите приемлемое среднее число строк кода функции или процедуры.  
В пределах 5–10, и чем меньше — тем лучше. Назначение процедуры или функции должно быть отражено в ее названии.
2. Предположим, что вы хотели бы снабдить некоторый аргумент функции (процедуры) квалификатором Optional. В каком месте списка параметров может находиться такой аргумент?  
Аргументы, помеченные квалификатором Optional, должны располагаться в конце списка параметров.

3. Придумайте название, уместное для функции, которая предусматривает операции чтения почтовых адресов из базы данных.  
Довольно удачным может оказаться такое: `ReadEmailAddress`.
4. Каково имя объекта, указывающего на текущую открытую базу данных?  
`CurrentProject` в `ADODB`.
5. Как называется подкласс в составе класса `Application`, содержащий множество полезных свойств и методов?  
`DoCmd`.

## Упражнения

1. В начале раздела "Определение типов аргументов" был приведен пример процедуры, выполняющей расчет **полной** стоимости изделия. Реализуйте его в виде функции.  

```
Function CalculateTotalSales (By Val SaleAmount As Double, _
    By Val SalesTax As Double ) As Double
    CalculateTotalSales = SaleAmount * (1 + SalesTax)
End Function
```
2. В разделе "Создание таблицы" мы рассматривали процедуру динамического построения таблицы. Напишите подпрограмму удаления таблицы из базы данных.  

```
Sub DropTable( ByVal TableName As String )
    Call DoCmd.DeleteObject (actable, TableName)
End Sub
```
3. В разделе "Импорт текста из файла" мы говорили о процедуре импорта в таблицу Access текста с разделителями. Используя материалы оперативной справочной системы, перепишите процедуру, имея в виду возможности импорта данных из таблицы Excel.

Следует воспользоваться методом класса `DoCmd`, описанным ниже:

```
DoCmd.TransferSpreadsheet ( [ТипПреобразования] _
    [, ТипЭлТаблицы], ИмяТаблицы, ИмяФайла _
    [, ФлагЗаголовков] [, Диапазон] )
```

## 9-й час. Работа с макросами

### Тесты

1. Какую команду следует применять для импорта данных из таблицы Excel?  
`DoCmd.TransferSpreadsheet`.
2. Как создается спецификация импорта данных?  
С помощью средств мастера Импорт текста и диалогового окна Спецификация импорта.
3. Какая команда использует спецификацию импорта данных?  
`TransferText`.
4. Как называется объект, применяемый для программирования макрокоманд в коде?  
`DoCmd`.
5. Могут ли объекты, подобные `DoCmd`, применяться в программах, ориентированных на использование с другими приложениями Office?  
Да. Объекты, подобные `DoCmd`, доступны в любом приложении, поддерживающем VBA или возможности обращения к объектам OLE Automation (скажем, в Visual Basic или Delphi).



## Упражнения

1. Создайте макрос для резервного копирования всех таблиц базы данных.

```
Sub Backup (ByVal Table As TableDef)
    On Error Resume Next
    DoCmd.DeleteObject acTable, Table.Name & "_Backup"
    DoCmd.CopyObject , Table.Name & "Backup", _
        acTable, Table.Name
End Sub
Sub BackupAllTables( )
    Dim I As Integer
    Dim Table As TableDef
    For Each Table In CurrentDb.TableDefs
        Call Backup( Table )
    Next
End Sub
```

2. Напишите фрагмент кода, позволяющего пользователю ввести наименование файла, данные из которого предполагается импортировать.

```
FileName = InputBox( "Введите имя файла для импорта:", "Имя файла", "Default.Txt" )
```

3. Напишите код, аналогичный описанному выше макросу, для создания резервной копии.

См. процедуру Backup из упражнения 1.

## 10-й час. Как использовать готовые решения

### Тесты

1. Как называется функция, предназначенная для поиска подстрок?  
InStr.
2. В чем состоит принципиальное отличие модальных окон от остальных?  
Открытое модальное окно препятствует использованию остальных окон приложения. Чтобы получить возможность продолжения работы, модальное окно необходимо **заккрыть**.
3. Как должна выглядеть команда открытия текстового файла в режиме чтения?  
Open Имя файла For Input As #НомерФайла
4. Какую функцию можно использовать для интерактивного ввода данных?  
InputBox.

## Упражнения

1. Напишите выражение для форматирования 9-значного почтового индекса.  
Format( PostalCodeVar, "#####-####" )
2. Постройте процедуру, добавляющую указанную строку в заданный файл.

```
Sub WriteToFile (ByVal FileName As String,
    ByVal Text As String)
    Dim Handle As Double
    Handle = FreeFile
```

```

    Open FileName For Append As ttHandle
    Print #Handle, Text
    Close #Handle
End Sub

```

3. Создайте подпрограмму поиска заданного имени в двоичном файле, предполагая, что при сохранении данных использовался тип `Email` и файл содержит более одной записи.

```

Sub FindByName (ByVal Name As String)
    Dim Handle As Double
    Handle = FreeFile
    On Error GoTo FINALLY
    Open "A:\10\test.bin" For Binary Access Read As #Handle
    Dim Mail As Email
    Do While Not EOF( Handle )
        Get #Handle, , Mail
        If (Mail.Name = Name) Then
            MsgBox "Найдено: " & Mail.Name & ", " & _
                Mail.Email
        Exit Do
    End If
Loop
FINALLY:
    Close #Handle
End Sub

```

## 11-й час. От сложного к простому: создание собственных типов данных

### Тесты

1. Может ли пользовательский тип данных содержать переменную перечислимого типа?  
Да. Типы данных, объявленные с помощью служебного слова `Type`, могут содержать экземпляры других пользовательских и перечислимых типов.
2. Для каких целей применяются пользовательские и перечислимые типы и в чем их различие?  
Типы позволяют собрать в единое целое ряд различных элементов данных, а перечисления используются для удобства представления ограниченных наборов целочисленных значений.
3. Могут ли в объявлениях типов содержаться функции или процедуры?  
Нет, но это возможно в конструкциях классов.
4. Допускаются ли в определениях типов члены-константы?  
Нет. Постоянные значения не могут быть членами типа, определяемого пользователем.
5. Позволяется ли включать в объявление типа ссылочную переменную?  
Формально это не **запрещено**, хотя и не является общеупотребительной практикой.

### Упражнения

1. Создайте объявление типа для хранения данных об имени человека, его адресе и номере телефона.

```

Type Contact
  Name As String
  Address1 As String
  Address2 As String
  City As String
  State As String
  PostalCode As String
End Type

```

2. Определите тип, одним из членов которого будет экземпляр типа, созданного в предыдущем упражнении. Что может послужить причиной для осуществления подобных действий? Почему бы просто не исправить объявление исходного типа?

```

Type InternetContact
  Who As Contact
  Email As String
End Type

```

Здесь мы создали новый тип, одним из членов которого является экземпляр ранее построенного типа. При таком подходе нам не придется исправлять существующий код, чтобы приспособить его к появлению в составе типа новой порции данных, описывающей адрес электронной почты.

3. Объявите перечислимый тип, представляющий все сорта мороженого.

```

Enum IceCreamFlavors
  Vanilla
  Strawberry
  Chocolate
  ButterPecan
  Boisenberry
End Enum

```

## 12-й час. Управление данными переменного объема

### Тесты

1. Как называется функция, позволяющая построить и вернуть массив?  
Array.
2. Какая функция используется для инициализации элементов массива значением, равноценным null?  
Erase.
3. Действует ли функция Erase одинаково в отношении массивов данных различных типов?  
Нет. "Поведение" функции Erase зависит от типа массива. Так, например, элементы массива объектов получают значение Nothing, а массиву типа Variant целиком присваивается значение Empty.
4. Каким образом можно динамически, т.е. в ходе выполнения программы, изменить размер массива?  
Следует использовать команду повторного объявления массива с помощью служебного слова ReDim.
5. Какой из алгоритмов сортировки более эффективен — метод "пузырька", выбора или "быстрой сортировки"?

Наиболее быстродействующим считается алгоритм "быстрой сортировки", правда, при условии, если данные не упорядочены предварительно. В последнем случае более высокую производительность может продемонстрировать алгоритм выбора.

## Упражнения

1. Напишите выражение сравнения строк для использования в процедуре сортировки.  
If (StrComp( Data(I), Data(J)) > 0) Then
2. Исправьте текст процедуры Dump таким образом, чтобы вывод данных осуществлялся в файл. Почему лучше выносить подобный код в отдельный именованный блок (функцию или процедуру) а не вносить его непосредственно в то место программы, где он необходим?

```
Sub Dump( ByVal Data( ) As Long)
    Dim Handle As Double
    Handle = FreeFile
    Open "dump.txt" For Output As #Handle
    Dim Elem As Variant
    For Each Elem In Data
        Print #Handle, Elem
    Next Elem
    Close #Handle
End Sub
```

3. Внесите изменения в текст процедуры BubbleSort, чтобы осуществить сортировку данных в порядке их убывания. (Имейте в виду: в листинге 12.9 предполагается вариант сортировки по возрастанию.)

```
Sub BubbleSort( ByRef Data( ) As Long )
    Dim I As Integer, J As Integer
    For I = Lbound( Data ) To Ubound( Data ) - 1
        For J = I + 1 To Ubound( Data )
            If (Data(J) > Data(I)) Then
                Call Swap( Data, I, J )
            End If
        Next J
    Next I
End Sub
```

## 13-й час. Коллекции данных

### Тесты

1. Чем отличается метод от обычной функции (процедуры)?  
Метод — обобщенное название функций и процедур, служащих членами класса.
2. Как называется метод, позволяющий добавлять в коллекцию новые элементы?  
Эту функцию выполняет базовый метод класса коллекций, который носит название Add.
3. Каково назначение конструкции присвоения переменной-объекту значения Nothing?  
Операция присвоения переменной-объекту класса предопределенного значения Nothing позволяет вернуть фрагмент памяти, отведенный объекту, в общий пул динамически распределяемой памяти.

4. Обязательно ли перед операцией очистки памяти, отведенной объекту коллекции, удалять все его элементы?  
 Нет. Собственно, и саму операцию очистки выполнять необязательно, поскольку компилятор VBA снабжен специальными средствами *сборки мусора*.
5. Назовите способы циклического прохождения по элементам коллекции.  
 С этой целью могут применяться циклы For Each ... Next и For ... Next, последний из которых учитывает значение, хранящееся в Count.

## Упражнения

1. Создайте коллекцию и добавьте в нее 10 целочисленных значений.

```
Sub AddTenInegers ( )
    Dim Integers As New Collection
    Dim I As Integer
    For I = 1 To 10
        Integers.Add( Int( Rnd * 10 ) )
    Next I
    Set Integers = Nothing
End Sub
```

2. Напишите процедуру, позволяющую отобразить содержимое коллекции, построенной при выполнении предыдущего упражнения, в окне Immediate.

Добавьте в код приведенной выше процедуры AddTenIntegers (непосредственно перед строкой Set Integers = Nothing) следующий фрагмент:

```
For I = 1 To Integers.Count
    Debug.Print Integers.Item(I)
Next I
```

3. Исправьте текст рассмотренной в главе 12 процедуры сортировки по методу "пузырька" с учетом возможности обработки элементов коллекции.

```
Sub Swap( ByRef ACollection As Collection,
    ByVal I As Integer, ByVal J As Integer )
    Dim Temp As Integer
    Temp = ACollection(J)
    Call ACollection.Add( ACollection(I) , , J )
    Call ACollection.Remove( J + 1 )
    Call ACollection.Add( Temp, , I )
    Call ACollection.Remove( I + 1 )
End Sub

Sub BubbleSortCollection(ByRef Integers As Collection)
    Dim I As Integer, J As Integer
    For I = 1 To Integers.Count - 1
        For J = I + 1 To Integers.Count
            If (Integers(I) > Integers(J)) Then
                Call Swap( Integers, I, J )
            End If
        Next J
    Next I
End Sub
```

Процедура сортировки практически не изменилась. Реализация же процедуры Swap в значительной степени зависит от типов данных – в нашем случае использованы стандартные методы Add и Remove класса Collection.

## 14-й час. Стил ь программирования:

### "Что такое хорошо, и что такое плохо"

#### Тесты

1. Назовите удобный способ именования процедур и функций.  
Следует использовать словосочетания, состоящие из глагола, который описывает операцию, и имени существительного, указывающего на объект воздействия.
2. В чем заключаются положительные стороны идеи повторного применения программного кода?  
Код уже готов, подвергнут отладке и тестированию.
3. Смысл префиксной, или венгерской, нотации заключается в разработке системы сокращенных названий типов, используемых для именования переменных. Верно ли это?  
Верно.
4. Термин *процедура* используется для обозначения и подпрограмм, и функций. Верно ли это?  
Верно.
5. Назовите важное полезное эмпирическое правило, которое обуславливает приемлемый размер кода процедуры или функции.  
Процедура должна быть достаточно краткой, чтобы выполнять только те операции, о которых свидетельствует ее название.

#### Упражнения

1. Придумайте имя функции, предназначенной для вычисления полной и частичной сумм.  
`GetTotalAndPartialSums`.
2. Предложите имена для пары функций, одна из которых просто закрывает файл, а другая выполняет проверку корректности выполнения этой операции.  
`DoCloseFile` и `CloseFile`.
3. Задайте названия переменной, предназначенной для хранения номера открытого файла, хранения номера ошибки открытия файла, а также для процедуры, вычисляющей сумму процента при известных сумме капитала и процентной ставке.  
`FileHandle`, `BAD_FILE_HANDLE` и `GetInterestAmount`.

## 15-й час. ADODB — ваш верный

### помощник

#### Тесты

1. Назовите синоним термина *провайдер*.  
Термин *провайдер* является новым, более широким. Он соответствует традиционному понятию *источник данных*.

2. С помощью какого предложения SQL могут быть отфильтрованы данные в запросе? WHERE.
3. Какое свойство Recordset ограничивает данные, доступные в записи? Свойство Filter объекта ADODB.Recordset соответствует назначению предложения команды WHERE на языке SQL.
4. Назовите имя свойства для представления данных, хранящихся в поле таблицы. Объекту какого класса оно принадлежит? Свойство Value принадлежит объекту Field.
5. Как называется коллекция в составе объекта Recordset, предназначенная для хранения данных? Fields.

## Упражнения

1. Исправьте текст запроса в листинге 15.6, чтобы предусмотреть возможность сортировки записей в порядке возрастания по значению поля, название которого задано переменной SearchFieldName. (Совет: воспользуйтесь предложением ORDER BY.)  

```
SQL = "SELECT * FROM MUSIC WHERE " & _
      SearchFieldName & "='" & _
      FindValue & "' ORDER BY " & SearchFieldName
```
2. Поясните, каким образом код листинга 15.7 можно еще более упростить и унифицировать.  
 Достаточно удачный способ состоит в том, чтобы передавать имя коллекции в качестве аргумента функции. В этом случае код легко приспособить для работы с любыми другими коллекциями.
3. Продемонстрируйте, как связать элементы произвольной коллекции с записями набора данных. (Совет: воспользуйтесь ключевым полем таблицы.)  
 Создавая элемент коллекции, добавьте в него значение ключевого поля записи:  

```
Call Target.Add( RecordSet( "ARTIST" ).Value, _
                  Str( RecordSet( "ID" ).Value ) )
```

 Значение ключевого поля сохраняется в коллекции в виде строки — не забудьте выполнить обратное преобразование (в длинное целое число) при необходимости использования значения ключа для поиска записи.

## 16-й час. Применение языка SQL

### Тесты

1. Что такое подчиненный (вложенный) запрос?  
 Вложенный запрос — это запрос, используемый в правой части одного из предикатов предложения WHERE, т.е. запрос, расположенный внутри другой команды SQL.
2. С какой целью употребляется служебное слово ALL вместе с UNION?  
 Служебное слово ALL в сочетании с UNION предписывает системе не удалять из результата выполнения объединенного запроса повторяющиеся строки. По умолчанию, т.е. в отсутствие слова ALL, такие строки изымаются.
3. Можно ли применять вложенный запрос в контексте команды DELETE? Если да, то зачем?

Да, в пределах предложения WHERE для уточнения содержимого набора удаляемых строк на основе данных другой таблицы или нескольких таблиц.

- Ограничиваются ли возможности хранимых процедур использованием команды SELECT?

Нет. В теле хранимой процедуры могут использоваться почти все возможные конструкции SQL — обращайтесь к документации. Другой вопрос — не во всех системах управления базами данных поддерживаются хранимые процедуры как таковые.

## Упражнения

- Напишите предложение WHERE с предикатом IN, позволяющее отобрать все записи таблицы MUSIC, в поле PUBLISHER которых содержатся значения Elektra или Empire.

```
WHERE Publisher IN ( 'Elektra', 'Empire' ).
```

- Исправьте команду UPDATE, приведенную в строке 2 листинга 16.6, таким образом, чтобы в верхний регистр были переведены первые символы имени и фамилии исполнителя.

```
UPDATE Music SET First_Name = ICap( [First_Name] ),  
                Second_Name = ICap( [Second_Name] );
```

- Создайте хранимую процедуру, позволяющую удалить из таблицы MUSIC запись по заданному значению поля LAST\_NAME.

```
PARAMETER [The_Artist] TEXT;  
DELETE FROM Music WHERE Last_Name = [The_Artist]
```

## 17-й час. Отладка кода

### Тесты

- Как называется класс, содержащий основные программные средства отладки? Debug.
- Какие, полезные во время тестирования методы предлагаются классом, упомянутым в предыдущем вопросе? Print И Assert.
- Какие цели преследует трассировка? Проверка кода.
- Следует ли удалять отладочный код из окончательной версии приложения с помощью редактора или условных директив компилятора?  
Никогда не удаляйте отладочный код, поскольку впоследствии, после внесения изменений и исправлений, он может понадобиться вновь, — пользуйтесь условными директивами компилятора, позволяющими быстро включать отладочные процедуры в окончательную версию приложения или изымать их.

## Упражнения

- Напишите функцию, возвращающую значение True или False в зависимости от того, существует ли файл с заданным именем.

```
Sub FileExists( ByVal FileName As String ) As Boolean  
    FileExists = Len( Dir( FileName ) ) > 0  
End Sub
```



2. Создайте тестовый код, использующий обращения к функции, построенной при выполнении п.1.

```
Sub TestFileExists( )  
    MsgBox FileExists( "C:\AUTOEXEC.BAT" )  
    MsgBox FileExists( 123 )  
End Sub
```

3. Разработайте версию процедуры Trace, предусматривающую вывод информации в текстовый файл.

```
Private Function GetTraceOutput( _  
    ByVal FileName As String, _  
    ByVal TraceNumber As Long, _  
    ByVal TraceMessage As Variant ) As String  
    GetTraceOutput = "Трассировка: " & FileName  
        & "(" & TraceNumber & ") в " & Now  
        & vbCrLf & TraceMessage & vbCrLf  
End Function  
Private Sub DoTraceFile( ByVal FileName As String, _  
    ByVal TraceNumber As Long, _  
    ByVal TraceMessage As Variant )  
    Dim Handle As Double  
    Handle = FreeFile  
    On Error GoTo FINALLY  
    Open FileName & ".log" For Append As #Handle  
    Print #Handle, GetTraceOutput( FileName, _  
        TraceNumber, TraceMessage )  
    FINALLY:  
    Close #Handle  
End Sub  
Sub TraceFile( ByVal FileName As String, _  
    ByVal TraceNumber As Long, _  
    ByVal TraceMessage As Variant )  
    #If DebuggingOn Then  
        Call DoTraceFile( FileName, TraceNumber, _  
TraceMessage)  
    #End If  
End Sub  
Sub TestTraceFile( )  
    Call TraceFile( "Module1", 1, "Test" )  
End Sub
```

Листинг содержит образец реализации полного набора процедур и функций. (Обратите внимание, что выражение форматирования строки результата трассировки вынесено в отдельную функцию.)

## 18-й час. Обработка ошибок во время выполнения программы

### Тесты

1. Какие аргументы необходимо передать методу Raise объекта класса Егг, чтобы связать сообщение об ошибке с файлами оперативной справки?

HelpFile и HelpContext.

2. Как называются объект и его метод, позволяющие выводить информацию в окно Immediate во время отладки приложения?

Объект Debug и метод Print.

3. Каково отличие обычного обработчика ошибок от того, который предназначен для решения задачи освобождения занятых ресурсов?

Тело обыкновенного обработчика выполняется только при возникновении ошибки, а во втором случае код работает всегда.

4. Можно ли сгенерировать стандартное сообщение об ошибке? Если да, то как и в каких случаях это целесообразно делать?

Да. Используйте метод Raise объекта Err в обработчике ошибок с целью отображения информации в тех проблемных ситуациях, для которых специальные действия вами не предусмотрены.

## Упражнения

1. Создайте обработчик, предусматривающий принудительное присвоение объекту класса Recordset значения Nothing.

```
FINALLY:  
Set RS = Nothing  
End Sub 'Или End Function
```

2. Исправьте код листинга 18.3 в предположении, что пользователь может удалять файл, помеченный атрибутом "только для чтения".

```
Sub DeleteFile( ByVal FileName As String )  
    On Error GoTo EXCEPT  
    Kill FileName  
    Exit Sub  
EXCEPT:  
    Const FILE_NOT_FOUND = 53  
    Const ACCESS_ERROR = 75  
    Select Case Err.Number  
    Case 53:  
        If (MsgBox(Err.Description & _  
            "(" & FileName & ")" _  
            & vbCrLf & "Попробуем снова?", _  
            vbRetryCancel ) = vbRetry) Then  
            FileName = InputBox( "Введите имя _  
                файла:", "Имя файла", "" )  
            If (FileExists( FileName )) Then Resume  
        End If  
    Case 75:  
        If (MsgBox( Err.Description & "(" _  
            & FileName & ")" & vbCrLf & _  
            "Файл только для чтения. Удалить?", _  
            vbYesNo ) = vbYes) Then  
            Call SetAttr( FileName, vbNormal )  
            Resume  
        End If  
    Case Else  
        Call RaiseError( Err )  
    EndSelect  
End Sub
```

3. Напишите команду генерации сообщения об ошибке отсутствия файла, предусматривающую задание пользовательского номера ошибки.

```
Call Err.Raise( vbObjectError + 100, "Module1", _  
    "Файл не найден" )
```

# 19-й час. Создание экранных форм

## Тесты

1. Каким общим термином описываются данные, принадлежащие объекту класса? Свойство.
2. При обращении к методу класса следует указывать имя объекта. Верно ли это? Верно.
3. Что такое событие в контексте лексики объектно-ориентированного программирования?  
Событие — термин, описывающий факт получения объектом приложения сообщения от операционной системы Windows.
4. Что представляет собой обработчик события?  
Обработчик события — это процедура, которая обеспечивает реакцию приложения на сообщение, полученное от операционной системы.
5. Каким образом можно разместить на форме новый управляющий элемент?  
Выберите в строке меню окна Access команду Вид⇒Панель элементов (View⇒Toolbox) и щелкните на кнопке, отвечающей нужному объекту. Щелчком кнопки мыши разместите объект на форме и настройте надлежащим образом его свойства.

## Упражнения

1. Опишите процедуру установления взаимосвязей между таблицами MUSIC и TRACKS, созданными на 15-м занятии.
  - Откройте базу данных, содержащую таблицы MUSIC и TRACKS.
  - Выберите в строке меню окна Access команду Сервис⇒Схема данных (Tools⇒Relationships).
  - Добавьте таблицы MUSIC и TRACKS в окно Схема данных.
  - Закройте диалоговое окно Добавление таблицы (Table).
  - Щелкните на элементе ID списка полей таблицы MUSIC и, не отпуская кнопку мыши, перетащите курсор к элементу MUSIC\_ID списка полей таблицы TRACKS. Система построит отрезок, соединяющий элементы списков (и отображающий таким образом связь полей таблиц).
  - Чтобы изменить или удалить созданную связь таблиц, щелкните на отрезке правой кнопкой мыши и выберите соответствующий элемент контекстного меню.
2. Используя схему данных, построенную при выполнении предыдущего упражнения, создайте с помощью мастера новую форму, которая будет представлять информацию из двух таблиц.
  - Выберите элемент Формы (Forms) в списке Объекты (Objects) окна База данных (Database) и щелкните на кнопке Создать (New) панели инструментов.
  - В диалоговом окне Новая форма (New Form) выберите режим Мастер форм (Form Wizard) и в качестве источника данных укажите таблицу MUSIC.
  - В первом диалоговом окне мастера выберите вначале таблицу MUSIC и добавьте в список Выбранные поля все ее поля, а затем ту же процедуру проделайте в отношении таблицы TRACKS.

- Щелкните на кнопке Готово (Finish).
  - Сохраните созданную форму (см. предшествующее упражнение) и обеспечьте ее автоматическое открытие при последующих обращениях к текущей базе данных.
3. Сохраните форму, построенную при выполнении предыдущего упражнения, под именем Главная форма, воспользовавшись командой меню **Файл⇒Сохранить как**.
- Выберите в строке меню команду **Сервис⇒Параметры запуска (Tools⇒Startup)**.
  - В раскрывающемся списке Вывод формы/страницы (Display Form/Page) диалогового окна Параметры запуска выберите элемент Главная форма (MAIN).
  - Щелкните на кнопке ОК.

## 20-й час. Как связывать Web-страницы с базами данных

### Тесты

1. Для каких целей применяется язык HTML?  
Hypertext Markup Language (HTML) — это первый из языков программирования, разработанных для создания Web-страниц.
2. Назовите другие языки программирования, применяемые в Web-дизайне.  
JavaScript, VBScript.
3. Для чего необходим URL?  
URL, или Uniform Resource Locator — это термин для обозначения адресов Internet.
4. Если существует необходимость в создании Web-сайта с **внутрикорпоративной** информацией, какого рода сайт вы постройте — intranet или Internet?  
Вероятно, более целесообразно построить сайт intranet.
5. При создании страниц доступа к данным можно ссылаться на те же таблицы и запросы, что и при проектировании форм. Верно ли это?  
Верно. Web-страницы и формы могут ссылаться на одни и те же источники данных.
6. Что означает фраза *привязка компонента к полю данных*?  
Речь идет об указании имени поля источника данных в свойствах компонента и обеспечении возможностей отображения значений поля средствами этого компонента.

### Упражнения

1. Установите для построенной страницы доступа к данным одну из тем — предопределенных наборов параметров внешнего оформления.  
Выберите в строке меню команду **Формат⇒Тема (Format⇒Theme)**. Укажите в списке диалогового окна Тема нужный элемент и щелкните на кнопке ОК.
2. Добавьте в построенную страницу новый элемент управления — Номер подразделения.  
Выберите команду **Вид⇒Панель элементов (View⇒Toolbox)**. Поместите на страницу элемент управления Поле (TextBox), сместив другие элементы управления. Откройте диалоговое окно Свойства (Properties) и укажите в качестве источника данных Номер подразделения.

3. Измените цвет фона текстового поля **Наименование** подразделения, чтобы визуально подчеркнуть, что оно допускает только операцию чтения.  
Щелкните в пределах текстового поля, выберите в строке меню команду **Вид⇒Свойства (View⇒Properties)** перейдите на вкладку **Формат (Format)** диалогового окна свойств и задайте нужное значение для свойства **BackgroundColor**.
4. Измените параметр **TabIndex** текстового поля **Наименование** подразделения таким образом, чтобы оно оказалось в начале списка компонентов, управляющего порядком перемещения фокуса при нажатии пользователем клавиши <Tab>.  
Щелкните в пределах текстового поля, выберите в строке меню команду **Вид⇒Свойства**, перейдите на вкладку **Другие (Other)** диалогового окна свойств и задайте значение 1 для свойства **TabIndex**.

## 21-й час. Основы программирования классов

### Тесты

1. Тип и переменная находятся в такой же взаимосвязи, как класс и ... (вставьте пропущенный термин).  
Объект.
2. Класс может содержать свойства, для которых не определены методы **Property**. Верно ли это?  
Верно. Но правильнее снабдить все (или многие) переменные-свойства класса квалификатором **Private** и создать для них соответствующие методы **Property**.
3. Каким обобщенным термином обозначаются процедуры и функции-члены класса?  
Метод.
4. Перечислите действия, необходимые для создания нового модуля класса.  
Откройте окно редактора **Microsoft Visual Basic** и выберите команду **Insert⇒Class Module**.
5. С помощью каких служебных слов обозначаются **Property**-методы, предназначенные для чтения переменных и присвоения им значений?  
Словом **Get** обозначаются методы, обеспечивающие возможность чтения данных, **Let** используется для присвоения значений переменным простых типов, а **Set** — для инициализации объектов-членов класса.

### Упражнения

1. Определите класс, содержащий переменные-свойства для хранения данных о телефонных номерах и именах их владельцев.

Необходимо создать в базе данных модуль класса и ввести в него следующий код:

```
Private FName As String
Private FPhoneNumber As String
Property Get PhoneNumber( ) As String
    PhoneNumber = FPhoneNumber
End Property
Property Let PhoneNumber( ByVal Value As String )
    FPhoneNumber = Value
End Property
```

```

Property Get Name( ) As String
    Name = FName
End Property
Property Let Name( ByVal Value As String )
    FName = Value
End Property

```

2. Наберите код класса `FileStream` в окне редактора. Постройте тестовую процедуру, которая в целях проверки корректности кода класса должна записывать в файл строку текста, возвращать указатель в начало файла и считывать ту же строку.

```

Sub TestFileStream( )
    Const TEXT = "Это тест"
    Dim Stream As New FileStream
    Call Stream.OpenStream( "Test.txt" )
    Call Stream.WriteStream( TEXT )
    Stream.Position = Stream.Position - Len( TEXT )
    Dim V As String
    Call Stream.ReadStream( V, Len( TEXT ) )
    MsgBox V
    Call Stream.CloseStream
    Set Stream = Nothing
End Sub

```

3. Примените объект `Scripting.FileSystemObject` для чтения файла с данными, сохраненными при выполнении предыдущего упражнения.

Выберите в строке меню окна редактора Microsoft Visual Basic команду **Tools** → **References**, установите флажок **Microsoft Scripting Runtime** списка библиотек диалогового окна **References** и щелкните на кнопке **OK**. Затем введите следующий код:

```

Sub TestScriptingObject( )
    Dim FileSystemObject As New Scripting.FileSystemObject
    Dim File As Scripting.TextStream
    Set File = FileSystemObject.OpenTextFile( _
        "Test.txt", ForReading, True )
    MsgBox File.ReadLine
    File.Close
    Set File = Nothing
    Set FileSystemObject = Nothing
End Sub

```

## 22-й час. Совершенствование типов данных

### Тесты

1. В чем заключаются преимущества агрегации кода?

Если коротко, вы основываете новые решения на существующих, проверенных и надежных.

2. Сколько атрибутов внешнего интерфейса целесообразно определить при создании нового класса?

Во многих случаях целесообразно ограничить количество членов внешнего интерфейса десятком или даже менее. Чем больше открытых атрибутов, тем более трудным становится восприятие и использование класса и тем меньшей — вероятность его повторного использования.

3. Как отличить атрибуты внешнего интерфейса класса от остальных?  
Члены внешнего интерфейса класса снабжаются квалификатором **Public**.
4. Назовите стандартные процедуры, которые вызываются при создании объекта класса и его уничтожении.  
**Class\_Initialize** и **Class\_Terminate** соответственно.

## Упражнения

1. Добавьте в состав класса **strings** атрибут **Text**, позволяющий возвратить содержимое всех элементов коллекции в виде единой строки, в которой частные значения разделяются константой **vbNewLine**.

```
Public Property Get Text ( ) As String
    Dim I As Integer
    For I = 1 To FStrings.Count
        Text = Text & FStrings( I ) & vbNewLine
    Next I
End Property
```

2. Добавьте в интерфейс класса **Strings** Property-метод **Let Text**, который дает возможность расчленив общую строку **Text** на отдельные сегменты и сохранить их в элементах коллекции.

```
Public Property Let Text( ByVal Value As String )
    Clear
    Call ParseStrings( Value )
End Property
```

3. Дополните текст методов **ReadFromFile** и **WriteToFile** класса **Strings** таким образом, чтобы обеспечить поддержку свойства **Text** (см. предыдущие упражнения).

```
Public Sub ReadFromFile( ByVal FileName As String )
    Clear
    Dim F As New FileStream
    Call F.OpenStream( FileName )
    Dim S As String
    Call F.ReadStream( S, F.Count )
    Text = S
    Call F.CloseStream
    Set F = Nothing
End Sub
Public Sub WriteToFile( ByVal FileName As String )
    Dim F As New FileStream
    Call F.OpenStream( FileName )
    Call F.WriteStream( Text )
    Call F.CloseStream
    Set F = Nothing
End Sub
```

## 23-й час. Настройки Access

### Тесты

1. С какой целью файлы баз данных настроек Access обозначаются расширением имени **.MDA**?

Речь идет о соглашении, принятом с целью различать файлы настроек и обычных баз данных Access.

2. Как называется таблица, которую необходимо создать в базе данных надстройки для обеспечения возможности ее регистрации?  
USysRegInfo.
3. Методы Property могут **быть** созданы в обычном, не "классовом", модуле. Верно ли это?  
Верно.
4. Использование **квалификаторов** Public и Private допустимо в пределах обычного модуля. Верно ли это?  
Верно.
5. Что означает константа IACCDIR?  
Указанная константа служит для ссылки на стандартную папку, предназначенную для хранения надстроек Access. Если установка Windows проводилась в режиме, предлагаемом по умолчанию, полный путь к упомянутой папке c:\Windows\Application таков — Data\Microsoft\AddIns.

## Упражнения

1. Создайте условную конструкцию, которая проверяет факт существования таблицы LOG и в случае необходимости вызывает процедуру CreateTable.  

```
If (TableExists( "LOG" ) = False) Then
    Call CreateTable( LOG_QUERY )
End If
```
2. Приведите выражение для тестирования процедуры WriteEntry.  

```
Call WriteEntry(0, "Module", "Test", "Helpfile.hlp", 0, _
    "admin")
```
3. Создайте пустую базу данных, определите ссылку на базу Log.mda и вызовите процедуру WriteEntry. Будет ли в результате этого обращения построена таблица LOG?  
Да. При выполнении строки 20 листинга 23.3 будет вызван метод CreateTable.

## 24-й час. Управление информацией о контактах Outlook

### Тесты

1. Как называется объект верхнего уровня, предоставляющий средства доступа к папкам Outlook?  
NameSpace.
2. Какая из разновидностей (пока единственная) объектов NameSpace существует в настоящее время?  
MAPI.
3. Назовите тип универсального объекта для хранения коллекций сообщений всех видов.  
MAPIFolder.
4. Как называется метод, позволяющий перемещать документы Outlook из одной папки (коллекции) в другую?  
Move.



5. Каково наименование объекта для хранения отдельных почтовых сообщений?  
`MailItem`.

## Упражнения

1. Напишите фрагмент кода, позволяющий создать новый объект типа `JournalItem`.  
`Dim Journal As JournalItem`  
`Set Journal = Outlook.CreateItem( olJournalItem )`
2. Сохраните созданный объект (см. предыдущий пример) в папке Outlook.  
`Journal.Save`
3. Укажите выражение, позволяющее передать фокус объекту Дневник (`Journal`) в списке папок окна Microsoft Outlook.  
`Journal.Display`

# Предметный указатель

## A

ActiveX, 123; 395  
ActiveX Data Objects — см. ADO, 34  
ActiveX Data Objects Extensions 2.1 for  
DDL and Security — CM. ADOX, 35  
ADO, 121; 123; 124; 126; 154  
    общие сведения, 34; 251  
ADODB, 228  
ADOX, 112; 133; 228  
    общие сведения, 35  
ANSI, 274

## C

COM, 34; 57; 124; 396; 413; 425  
Component Object Model — CM.  
COM, 124

## D

DAO, 121; 123; 126; 154; 258  
Data Access Page, 356  
Direct Access Objects — CM. DAO, 121  
Dynamic Data Exchange — DDE, 123  
Dynamic Link Library — DLL, 123

## E

Extended Markup Language — XML, 376  
Extranet, 36

## H

Hypertext Markup Language —  
HTML, 356

## I

Internet, 356  
Internet Information Services, 34  
Intranet, 36; 37; 356

## M

Markup Language — XML, 356  
Microsoft Access 2000  
    мастера Data Pages Wizards, 36  
    общие сведения, 25  
Microsoft Access 2002  
    Microsoft Access Project, 36  
    мастера Data Pages Wizards, 362  
Microsoft Developer Network —  
MSDN, 136

## O

Object Linking and Embedding —  
OLE, 123  
ODBC, 123; 171; 251; 254  
OLE Automation, 124; 195; 396; 427  
OLE DB, 252; 254  
OLE-клиент, 124  
OLE-сервер, 124  
Open DataBase Connectivity — CM.  
ODBC, 123

## P

Proxy-сервер, 357

## R

RDBMS, 281  
RDO, 121; 123  
Remote Data Objects — CM. RDO, 121

## U

Unified Modeling Language — UML, 401  
Uniform Resource Locator — URL, 359  
Universal Naming Convention —  
UNC, 374

## W

Windows Application Programming Interface - API, 83; 202  
Wireless Markup Language — WML, 356  
World Wide Web, 270; 355

## A

Автосумма, 367  
Агрегация, 142; 199; 382; 395; 404  
Адрес IP, 356  
Алгоритм, 297  
    вычислительная сложность, 215; 222;  
    223; 226  
Аргумент, 55; 143  
    макрокоманды, 161  
Атрибут, 132; 228

## Б

База данных, 106; 114; 151  
    создание, 106; 160  
Блок-схема, 382

## В

Верификатор условий, 301; 304;  
310; 324  
Всплывающая подсказка, 58; 156; 370  
Выражение, 44; 45; 141  
    арифметическое, 43  
    условное, 88  
    циклическое, 94

## Г

Гиперссылка, 359  
    назначение интерфейсному  
    объекту, 29  
Гипертекст, 356

## Д

Декомпозиция задачи, 141  
Диалоговое окно  
    Add Reference, 419  
    Add Watch, 63; 64  
    Call Stack, 307

Edit Watch, 64  
Export File, 408  
GroupLevel, 372  
Quick Watch, 64  
References, 112; 129; 133; 345;  
419; 430  
Выбор базы данных (в Windows), 256  
Выбор источника данных, 376  
Выбор рисунка, 345  
Вызов, 33  
Диспетчер надстроек, 424  
Добавление таблицы, 160; 261; 275  
Запрос на выборку, 160  
Мастер диаграмм Microsoft  
Office, 369  
Назначить гиперссылку —  
Открыть, 29  
Настройка, 28; 29; 351  
Новая страница доступа к  
данным, 362  
Новая таблица, 109; 172  
Новая форма, 330; 334  
Новый запрос, 160; 275  
Орфография, 30  
Открытие надстройки, 424  
Параметры  
    вкладка  
        Вид, 162  
Построитель выражений, 164; 339  
Пошаговое исполнение макроса, 166  
Свойства, 169  
Свойства подключения для  
страницы, 375  
Создание нового источника данных  
(в Windows), 255  
Создание форм, 334  
Сортировка и группировка, 368  
Сохранение, 111  
Сохранение спецификации импорта  
и экспорта, 172; 173  
Спецификация импорта, 172  
Список полей, 368  
Тема, 369  
Установка драйвера ODBC для  
Microsoft Access (в Windows), 256  
Файл новой базы данных, 107; 415  
Фильтр, 338  
Форма, 342  
Дизъюнкция, 77  
Динамический обмен данными, 123

Директива  
условная, компилятора, 306

## З

Заповеди программирования, 148

## И

Идиома, 203  
Импликация, 78; 79  
Инкапсуляция, 228  
Интегрированная среда разработки, 58  
Интерфейс  
    класса, 385  
    пользователя, 329  
    программный, 125; 224; 253  
Исключительная ситуация, 314  
Источник данных, 252; 255; 257

## К

Квалификатор

    ByRef, 146  
    ByVal, 145  
    Optional, 146

Клавиша

    <Enter>, 66  
    <F1>, 135; 193; 235; 321; 343  
    <F5>, 64; 419  
    <F8>, 60; 64; 89; 156; 419  
    <F9>, 166

Класс, 202; 203

    ADODB, 125; 251; 258; 273  
    Application, 125; 151; 157; 429  
    Catalog, 114; 121; 133; 289  
    Collection, 228; 263; 341; 383;  
    400; 403  
    Command, 260; 289; 290  
    Connection, 114; 129; 133; 252;  
    258; 266  
    CurrentDB, 125  
    DAO, 416  
    Database, 127  
    Debug, 301; 304; 310; 324  
    DoCmd, 121; 125; 151; 174  
    Err, 164; 253; 320; 321; 415  
    Field, 115; 131; 132; 264; 269  
    FileSystemObject, 395  
    Form, 330; 340; 343

Key, 133  
Label, 336; 340  
PivotTable, 334  
Procedure, 35; 289  
Recordset, 114; 121; 127; 130; 203;  
258; 268; 291; 340  
Table, 133; 135  
TextBox, 336; 340  
общие сведения, 35; 228; 341; 381  
создание, 392

Ключ таблицы

    внешний, 279  
    первичный, 279

Код, 40

Коллекция, 97; 99; 103; 134; 227; 268

    Columns, 135  
    Fields, 263; 268  
    Groups, 135  
    Indexes, 135  
    Keys, 135  
    Procedures, 289  
    Properties, 135  
    Tables, 135  
    Users, 135  
    Views, 135  
объявление, 229

Команда

    Close, 187  
    Debug⇒Add Watch, 63  
    Debug⇒Compile, 422  
    Debug⇒Edit Watch, 64  
    Debug⇒Quick Watch, 74  
    Edit⇒Find, 303  
    F7, 347  
    File⇒Export File, 304; 408  
    File⇒Import File, 304  
    Get, 192  
    Input, 190  
    Insert⇒Class Module, 383; 384  
    Kill, 319  
    Line Input, 190  
    Open, 187  
    Option Base, 101; 213  
    Option Compare, 184  
    Option Explicit, 57  
    Print, 66; 186; 190  
    Put, 190; 192  
    Resume, 318  
    Resume Next, 320  
    Tools⇒References, 112; 129; 133; 251;  
    345; 419  
    View⇒Call Stack, 307  
    View⇒Immediate Window, 65

View⇒Locals Window, 61  
 Write, 190  
 Вид⇒Имена макросов, 162  
 Вид⇒Источник HTML, 368  
 Вид⇒Конструктор, 336; 343; 368  
 Вид⇒Панель элементов, 344; 368  
 Вид⇒Программа, 346  
 Вид⇒Просмотр страницы, 365; 368  
 Вид⇒Режим SQL, 160; 275  
 Вид⇒Режим таблицы, 336  
 Вид⇒Режим формы, 336; 349  
 Вид⇒Свойства, 342; 343; 368; 371  
 Вид⇒Свойства уровня группы, 370  
 Вид⇒Список полей, 368  
 Вид⇒Структура данных, 368  
 Вид⇒Условия, 162  
 Вставка⇒Гиперссылка, 369  
 Вставка⇒Диаграмма Office, 369  
 Вставка⇒Несвязанный раздел, 369  
 Вставка⇒Новая запись, 337  
 Вставка⇒Рисунок, 368  
 Вставка⇒Сводная таблица Office, 369; 374  
 Вставка⇒Фильм из файла, 368  
 Вставка⇒Электронная таблица Office, 369  
 Вставка⇒Элемент ActiveX, 369  
 Записи⇒Фильтр, 337  
 Записи⇒Фильтр⇒Изменить фильтр, 338  
 Записи⇒Фильтр⇒Исключить выделенное, 338  
 Записи⇒Фильтр⇒Расширенный фильтр, 339  
 Записи⇒Фильтр⇒Фильтр по выделенному, 338  
 Запрос⇒Запуск, 160; 275  
 Запуск⇒Запуск, 162; 165; 166  
 Настройка⇒Панель управления (в Windows), 255  
 Правка⇒Выделить все записи, 337  
 Правка⇒Выделить запись, 337  
 Правка⇒Найти, 337  
 Правка⇒Перейти, 337  
 Правка⇒Удалить запись, 337  
 Предварительный просмотр веб-страницы, 374  
 Сервис⇒Макрос⇒Редактор Visual Basic, 419  
 Сервис⇒Настройки, 419  
 Сервис⇒Настройки⇒Диспетчер надстроек, 423; 424

Сервис⇒Настройка, 28; 29; 350  
 Сервис⇒Общие приложения, 34  
 Сервис⇒Орфография, 30  
 Сервис⇒Параметры, 162  
 Сервис⇒Параметры запуска, 352  
 Сервис⇒Совместная работа⇒Начать собрание, 32; 33  
 Сервис⇒Схема данных, 354  
 Файл⇒Предварительный просмотр Web-страницы, 368  
 Файл⇒Создать, 59; 414  
 Файл⇒Сохранить, 275  
 Файл⇒Сохранить как, 275  
 Фильтр⇒Применить фильтр, 338  
 Формат⇒Тема, 369  
 Формат⇒Фон⇒Звук, 369  
 Формат⇒Фон⇒Рисунок, 369  
 Формат⇒Фон⇒Цвет, 369  
 Команда SQL  
 CREATE TABLE, 160; 163; 285; 415  
 DELETE, 264; 275; 288; 294  
 INSERT, 284  
 SELECT, 260; 264; 271; 274; 286  
     вложенная, 278  
 UPDATE, 275; 287; 294  
 оператор  
 AND, 276; 277  
 AS, 283  
 ASC, 279  
 BETWEEN, 277  
 DESC, 279  
 IN, 277; 294  
 INNER JOIN, 282  
 JOIN, 282  
 LEFT JOIN, 282  
 NOT, 276; 277  
 OR, 276; 277  
 RIGHT JOIN, 283  
 UNION, 283  
 UNION ALL, 283; 294  
 предложение  
 FROM, 274  
 GROUP BY, 280  
 HAVING, 280  
 INTO, 284  
 ORDER BY, 271; 272; 279; 280; 457  
 PARAMETERS, 290  
 SET, 287  
 VALUES, 284  
 WHERE, 264; 267; 268; 275; 280; 287; 294; 337  
 функция

CDATE, 281; 289

COUNT, 280; 289

SUM, 280; 289

## КомандаSQL

SELECT, 267

Комментарий, 247

Компилятор, 52; 143; 232; 233; 306

конкатенация, 48

Константа, 67

    глобальная, 71

    именованная, 71; 73; 205; 249

    литеральная, 70; 73

Конструктор, 219

Контекст, 63; 67; 72; 143; 188; 204

Контроллер автоматизации, 428

Конъюнкция, 77; 78

## Л

Литерал, 70

Ловушка, 301

## М

Макрос, 159

Массив, 53; 97; 99; 103; 209

    ассоциативный, 399

    динамический, 68; 211

    задание точки отсчета индекса, 213

    многомерный, 210

    объявление, 210

    статический, 213

    фиксированного размера, 211

Мастер подключения данных, 376

Матрица, 214

Меню

    индивидуальная настройка, 26

    элемент Развернуть, 26

Метка, 316

Метод, 125; 131; 194; 228; 341; 386

    Add, 229; 231; 342; 406

    AddNew, 131; 262; 263

    Append, 135

    Assert, 304; 310; 324

    BOF, 131

    Class\_Initialize, 391; 404

    Class\_Terminate, 391; 404

    Clear, 321

    Close, 116

    Delete, 264

    Edit, 131

EOF, 120; 131; 262; 269

Execute, 291

Item, 229; 234

Move, 132; 348

MoveFirst, 120; 132; 267

MoveLast, 132

MoveNext, 132; 267

MovePrevious, 132

Open, 253; 257; 259

Print, 304; 310; 324

Raise, 320; 321; 326

Refresh, 116

Remove, 229; 233

Repaint, 343

TransferText, 152

Update, 131; 262; 263

Visible, 396

Модель

    объектная, 270; 271; 428

Модуль, 57; ПО; 156

    класса, 383

## Н

Надстройка, 413

Наследование, 235

## О

Обработка ошибок, 313

Обработчик событий, 341; 391

Объединение

    запросов, 283

    таблиц, 279; 281

Объект, 125; 204; 228

    Form, 348

    Printer, 350

    Report, 349

    наблюдения, 62; 303

    общие сведения, 35; 341; 381

Окно

    Immediate, 65; 67; 84; 88; 116; 181;

    186; 211; 216; 269; 302; 324; 432

    Locals, 61; 98

    Project Explorer, 420

    Watches, 62; 64; 301

Администратор источников данных

ODBC (в Windows), 255

База данных, 59; 108; 160; 161; 261;

330; 383

Запрос на выборку, 261; 275; 276  
 Импорт текста, 172  
 Макрос, 161; 162  
 модальное, 193  
 Панель управления (в Windows), 255  
 редактора Microsoft Visual Basic, 60;  
 63; 89; ПО; 133; 156  
 Схема данных, 354  
 Операнд, 43; 69; 70  
 Оперативное запоминающее  
 устройство — ОЗУ, 51  
 Оператор  
   Not, 47  
   Or, 47  
 Оператор, 42; 46; 69  
   ', 91  
   ", 43; 74  
   #, 70  
   &, 48; 81  
   O, 46; 267  
   \*, 74  
   ., 203; 384  
   A 43; 74  
   ^, 74  
   +, 42; 43; 48; 74; 81  
   <, 46; 75  
   <=, 46; 75; 95  
   O, 46; 75  
   =, 45; 46; 55; 75; 79; 267  
   >, 46; 75; 95  
   >=, 46; 75  
   AddressOf, 73; 79; 83  
   And, 47; 78; 79; 267  
   Eqv, 78; 79  
   Imp, 79  
   Is, 82  
   Like, 82; 120  
   Mod, 74  
   Not, 73; 79  
   Or, 79; 267  
   TypeOf, 437  
   Xor, 79  
   арифметический, 45; 74  
   бинарный, 43; 73  
   логический, 47; 76; 267  
   побитовый, 78; 79  
   сравнения, 46; 74; 95; 267  
   тернарный, 43; 73  
   унарный, 43; 73  
 Отступ, 243

Отчет, 349  
   печать, 350

## П

Панель задач, 26  
 Панель задач Windows, 255  
 Панель инструментов  
   База данных, 28  
   индивидуальная настройка, 26  
   Конструктор макросов, 162; 165; 166  
   Панель элементов, 344; 368; 370  
   Собрание по сети, 38  
 Парадигма, 228  
 Параметр, 55  
 Переменная, 44; 61; 62; 70; 73; 229  
   callback-, 83  
   глобальная, 71; 73; 204; 249  
   локальная, 61; 204  
   объявление, 52  
   правила именования, 44  
 Перечислимый тип, 205; 207  
   CursorTypeEnum, 131  
   LockTypeEnum, 131  
 Подпрограмма, 55  
 Подстановка, 367  
 Поток, или нить, 257  
 Предварительный просмотр веб-  
 страницы, 374  
 Префиксная нотация, 240  
 Приоритет выполнения операций, 81  
 Проблема неразрешимости, 297  
 Провайдер OLE DB, 252; 257  
 Проверка правописания, 30  
 Программирование  
   объектно-ориентированное, 381  
   структурное, 382  
 Программные продукты  
   JUnit, 298  
 Программный продукт  
   Borland Delphi, 125; 127; 301  
   Dreamweaver, 358  
   Microsoft Developers Tools for Office  
   XP, 353  
   Microsoft Excel, 59; 125; 127  
   Microsoft FrontPage, 359  
   Microsoft Internet Explorer, 36;  
   356; 357  
   Microsoft Internet Information  
   Services, 357  
   Microsoft Jet, 36; 37  
   Microsoft NetMeeting, 32; 33

- Microsoft NetMessenger, 32
- Microsoft NetShow Player, 369
- Microsoft Office 2000, 179
- Microsoft Office XP, 37; 107; 427
- Microsoft Outlook, 195; 356; 427
- Microsoft Personal Web Server, 357
- Microsoft SQL Server, 36; 37; 273
- Microsoft Visual Basic, 125; 425
- Microsoft Visual C++, 301
- Microsoft Visual Studio, 425
- Microsoft Windows NT, 40
- Microsoft Windows 98, 40
- Microsoft WinProxy, 32
- Microsoft Word, 127; 195; 356
- Netscape Communicator, 36
- Netscape Navigator, 356; 377
- Rational Rose, 401
- Процедура, 55; 141
- MsgBox, 72; 80; 93; 104; 184; 192
- правила именования, 143

## P

- Разложение на элементарные операции, 245
- Регистр символов, 76; 93
- Реестр Windows, 255
- Режим
  - отладки, 59; 156; 249; 297
  - соединения, 257
- Рекурсия, 223

## C

- Сборка мусора, 232; 233; 236
- Сводная таблица, 374
- Свойство, 152; 194; 228; 229; 341; 388
  - ActiveConnection, 135
  - CommandText, 290
  - ConnectionFile, 376
  - Count, 229; 233
  - Description, 321
  - Fields, 263
  - Filter, 268
  - HelpContext, 321
  - HelpFile, 321
  - LastDLLError, 321
  - Name, 264
  - Number, 321
  - Order, 271
  - OriginalValue, 264

- SortOrder, 271
- Source, 321
- Value, 132; 264
  - статическое, 390
- Связывание
  - динамическое, 430
  - статическое, 430
- Связывание и внедрение объектов, 123
- Сервер автоматизации, 427
- Сервер каталогов, 33
- Синтаксис, 40; 143
- Системы счисления, 80
- Служебное слово, 40
  - #Const, 306
  - =, 97
  - Append, 188
  - As, 45; 52; 129; 144; 210
  - Binary, 188
  - ByRef, 146; 249
  - ByVal, 145; 249
  - Call, 231
  - Case, 91; 209
  - Const, 53; 54; 71; 202
  - Dim, 44; 45; 52; 129; 210
  - Do, 95
  - Each, 100
  - Else, 90
  - Empty, 216
  - End, 88; 92; 143; 200; 205
  - Enum, 205
  - Exit, 101; 317
  - For, 97; 100
  - Function, 141; 147
  - Get, 389; 406
  - Global, 53; 55; 67; 71; 202; 204
  - If, 88
  - Input, 188
  - Len, 189
  - Let, 389; 406
  - Lock, 188
  - Loop, 95
  - New, 112; 129; 229; 253; 258
  - Next, 97
  - Nothing, 116; 216; 229; 391
  - null, 211; 216
  - Optional, 146; 249
  - Output, 188
  - ParamArray, 218
  - Preserve, 212
  - Private, 309; 385
  - Property, 317; 388; 397; 406
  - Public, 385



Random, 188  
 Read, 188; 192  
 ReDim, 53; 68; 212  
 Select, 92; 209  
 Set, 230; 258; 390  
 Shared, 188  
 Static, 202; 204; 205; 213  
 Step, 99  
 Sub, 141  
 Then, 88  
 To, 97  
 Type, 199; 200  
 Until, 95; 96  
 Wend, 94  
 While, 94; 95; 96  
 Write, 188; 192

Событие, 341

OnDirty, 348  
 OnRecordExit, 348  
 OnUndo, 348

Совместная работа, 32; 305

Сообщение, 341

Сортировка данных, 220

Стек, 307

вызовов, 303

Стиль программирования, 239; 258;  
 309; 316; 387; 402

Страница доступа к данным, 356

Строка

соединения, 129; 253; 257

Схема базы данных, 361

## Т

Таблица

Microsoft Excel, 157; 177  
 базы данных, 107; 151; 160  
 USysRegInfo, 422  
 истинности, 48; 77

Тип данных, 44

Boolean, 46; 132  
 Currency, 203  
 Date, 45; 49; 203  
 Double, 49; 53; 70  
 Integer, 45; 53; 70  
 Long, 53  
 String, 45; 48; 53; 70  
 Variant, 57; 68; 132; 210; 216; 219  
 встроенный, 48  
 пользовательский, 61; 62; 189;  
 199; 382

Точка останова, 156; 166; 303

Трассировка, 301; 303; 310

## У

Управляющая структура, 87

#If ... #End If, 306

Do ... Loop, 95; 101; 103; 118;  
 189; 262

For ... Next, 97; 103; 117

For Each, 100; 104; 116; 213; 234

If... Then ... Else ... End If, 90

If... Then ... End If, 88

On Error GoTo, 316

On Error Resume Next, 318

Select Case ... End Case, 92; 209; 245

While ... Wend, 94; 101; 103

вложенная, 90; 103; 245

Уравнение, 69

## Ф

Файл

ADP, 36

.MDA, 414

.MDB, 36; 37

Msado15.dll, 34

Msadox.dll, 35

Форма, 330

Формат

Active Server Pages — ASP, 36; 360

Extensible Markup Language, 356

Hypertext Markup Language —

HTML, 36; 172; 356

Office Data Connection — ODC, 375

Wireless Markup Language, 356

Функция, 55; 72; 141; 147

Array, 216

CDate, 50; 322

Chr, 182

CreateObject, 395

Date, 50; 186

Dir, 150

Erase, 216

Format, 49; 185

FreeFile, 187; 188; 189; 192

Iff, 104

InputBox, 89; 93; 96; 189; 193

InStr, 184

IsArray, 217

IsDate, 320; 323

Lbound, 99; 213

LCase, 185

Len, 150

MsgBox, 193; 288

Size, 189

Space, 185

Str, 180  
String, 185  
Switch, 104  
Time, 50; 186  
Ubound, 99; 213  
UCase, 93; 185; 287  
Val, 90; 180

## Х

Хранимая процедура, 35; 260; 289

## Ц

Цикл, 94  
    итерационный, 97  
    прерывание, 101  
    условный, 94

## Э

Эквивалентность, 78  
Элемент  
    управляющий, 336

## Я

Язык программирования  
    язык ассемблера, 40

Язык программирования  
    С, 241

Язык программирования  
    С, 73; 202; 210; 213; 220; 230; 307;  
    382  
    C++, 43; 73; 179; 225; 230; 240; 301;  
    307; 397  
    COBOL, 257; 381  
    GW-BASIC, 179  
    Java, 232; 233; 247  
    JavaScript, 360  
    Microsoft Visual Basic for Applications  
    (VBA), 179  
    Object Pascal, 128; 301; 397  
    PDP-11 B.A.S.I.C., 179  
    PL/SQL, 289  
    ROM BASIC, 179  
    SQL, 111; 122; 151; 153; 260; 265;  
    271; 273  
    VBScript, 360  
    Visual Basic, 35; 179; 220; 301; 425  
    Visual Basic for Applications (VBA),  
    35; 39; 52; 107  
    Visual Basic for DOS, 179  
    слабо типизированный, 240  
    строго типизированный, 240  
    язык ассемблера, 179; 210  
    язык макросов, 159

*Научно-популярное издание*

**Пол Киммел**

# **Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа**

Литературный редактор	<i>О. В. Ожигова</i>
Верстка	<i>О. В. Мишутина</i>
Художественный редактор	<i>В. Г. Павлютин</i>
Корректоры	<i>Л. А. Гордиенко, О. В. Мишутина</i>

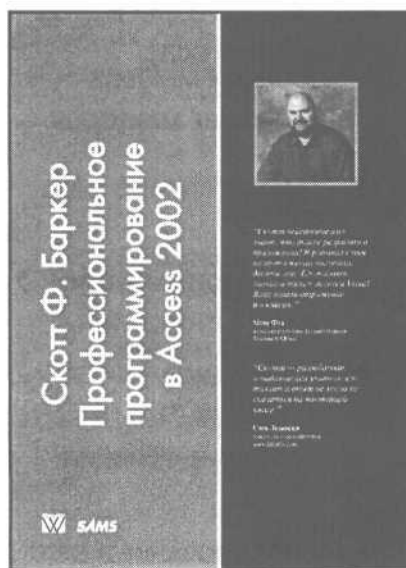
Издательский дом "Вильямс".  
101509, Москва, ул. Лесная, д. 43, стр. 480.  
Изд. лиц. ЛР № 090230 от 23.06.99  
Госкомитета РФ по печати.

Подписано в печать 21.04.2003. Формат 70×100/16.  
Гарнитура Times. Печать офсетная.  
Усл. печ. л. 36,37. Уч.-изд. л. 28,2.  
Тираж 4000 экз. Заказ № 2850.

Отпечатано с диапозитивов в ФГУП "Печатный двор"  
Министерства РФ по делам печати,  
телерадиовещания и средств массовых коммуникаций.  
197110, Санкт-Петербург, Чкаловский пр., 15.

# ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ В MICROSOFT ACCESS 2002

**Скотт Ф. Баркер**



[www.williamspublishing.com](http://www.williamspublishing.com)

**в продаже**

Книга предназначена для пользователей, которым стал узок круг стандартных функций Access и макросов. Рассматриваемое приложение предоставляет широчайший круг средств и функций для эффективного создания баз данных и управления ими. Однако реальная жизнь богата непредвиденными проблемами и часто выдвигает требования, которые невозможно предусмотреть заранее. Но разработчики компании Microsoft предоставили вам мощное средство создания и настройки собственной уникальной среды разработки — универсальный для всех приложений Microsoft Office язык программирования Visual Basic for Applications (VBA). Собственно, все уже сделано. Вам осталось лишь овладеть в совершенстве этим языком. Вы научитесь создавать не-большие пользовательские программы для автоматизации базы данных, созданной с помощью Access 2002, и узнаете, что многие рутинные функции по поддержанию базы данных могут выполняться автоматически, почти без вашего участия. В книге приведены примеры программных кодов, тщательно подобранные автором. Книга рассчитана главным образом на пользователей Access с различным уровнем подготовки, желающих расширить свой профессиональный кругозор и навыки программирования на VBA. Особенно полезной книга будет для разработчиков баз данных корпораций, доступ к которым необходимо обеспечить для группы пользователей локальной сети. Легкий и доступный стиль изложения поможет пользователям даже начинающим быстро разобраться со всеми возможностями написания программ для Access и эффективно использовать их в своей повседневной работе.

# ИНЖЕНЕРНЫЕ РАСЧЕТЫ В EXCEL

**Рональд У. Ларсен**



[www.williamspublishing.com](http://www.williamspublishing.com)

**в продаже**

- ВВЕДЕНИЕ В EXCEL
- ПОСТРОЕНИЕ ДИАГРАММ В EXCEL
- ФУНКЦИИ В EXCEL
- МАТРИЧНЫЕ ОПЕРАЦИИ В EXCEL
- ЛИНЕЙНАЯ РЕГРЕССИЯ В EXCEL
- РЕАЛИЗАЦИЯ В EXCEL ИТЕРАЦИОННЫХ МЕТОДОВ
- ИСПОЛЬЗОВАНИЕ МАКРОСОВ В EXCEL
- ПРОГРАММИРОВАНИЕ В EXCEL НА ЯЗЫКЕ VBA
- ВЗАИМОДЕЙСТВИЕ EXCEL С ДРУГИМИ ПРОГРАММАМИ
- ОЦЕНКА ЭФФЕКТИВНОСТИ ДЕНЕЖНЫХ ВЛОЖЕНИЙ С ПОМОЩЬЮ EXCEL
- ФИНАНСОВЫЕ РАСЧЕТЫ В EXCEL
  - СТАТИСТИЧЕСКИЕ ФУНКЦИИ EXCEL
  - ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ В EXCEL
  - ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ В EXCEL
  - МЕТОДЫ ЧИСЛЕННОГО РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С ПОМОЩЬЮ EXCEL

# Мир книг по базам данных от издательской группы “ДИАЛЕКТИКА-ВИЛЬЯМС”



ISBN 5-8459-0138-3



ISBN 5-8459-0109-X



ISBN 5-8459-0384-X



ISBN 5-8459-0297-5



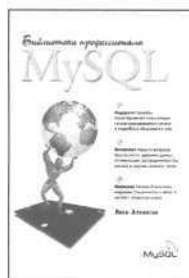
ISBN 5-8459-0355-6



ISBN 5-8459-0270-3



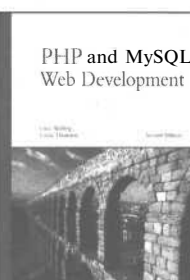
ISBN 5-8459-0158-8



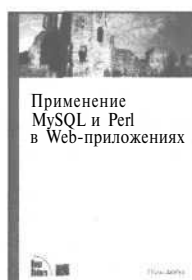
ISBN 5-8459-0291-6



ISBN 5-8459-0276-2



ISBN 5-8459-0302-5



ISBN 5-8459-0179-0



[www.dialektika.com](http://www.dialektika.com)



[www.williamspublishing.com](http://www.williamspublishing.com)



[www.ciscopress.ru](http://www.ciscopress.ru)

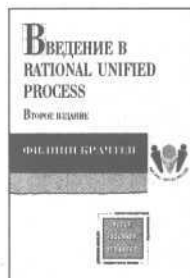
# Мир книг по UML от издательской группы "ДИАЛЕКТИКА • ВИЛЬЯМС"



ISBN 5-8459-0276-2



ISBN 5-8459-0250-9



ISBN 5-8459-0239-8



ISBN 5-8459-0203-7



ISBN 5-8459-0265-7



ISBN 5-8459-0368-8



ISBN 5-8459-0346-7



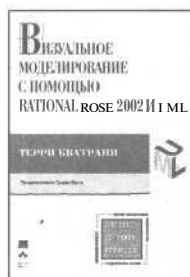
ISBN 5-8459-0301-7



ISBN 5-8459-0393-9



ISBN 5-8459-0299-1



ISBN 5-8459-0425-0



ISBN 5-8459-0355-6



[www.dialektika.com](http://www.dialektika.com)



[www.williamspublishing.com](http://www.williamspublishing.com)



[www.ciscopress.ru](http://www.ciscopress.ru)

**SAMS****Освой самостоятельно****Программирование**

для Microsoft

**Access 2002****за 24  
часа**

## Узнайте, как:

- Создавать программы на Visual Basic for Applications (VBA)
- Создавать профессиональные приложения
- Создавать макросы и преобразовывать их в VBA-код
- Создавать и использовать классы
- Управлять базами данных с помощью программного кода, используя ADO
- Создавать страницы доступа к данным
- Использовать сводные таблицы и сводные диаграммы для получения динамически настраиваемого представления данных

Категория: Программирование

Уровень: для начинающих программистов и пользователей среднего и высокого уровня

**SAMS**

www.williamspublishing.com  
www.sampublishing.com

## Начните прямо сейчас!

### Двадцать четыре практических занятия

Всего за 24 урока вы приобретете навыки, которые повысят производительность и надежность приложений Access. Доступные пошаговые инструкции помогут разобраться в основах программирования в Access.

**Советы** указывают кратчайший путь к выполнению задачи

**Замечания** просто и доходчиво объясняют понятия и процедуры

**Предостережения** помогают избежать наиболее частых недоразумений

Пол **Киммел** — основатель компании Software Conceptions, которая занимается созданием программного обеспечения и предоставлением консультационных услуг для самых разных организаций и фирм, разбросанных по всему миру. Пол более 10 лет руководил проектами на основе Microsoft Access, он автор многих книг по программированию в Access и Visual Basic. Пол также автор в еженедельной рассылке *Code Guru Visual Basic Tech Notes of Internet.com*.

ISBN 5-8459-0453-6



9 785845 904539



