Turbo Pascal 7.0

на примерах

Ю.А. Шпак

Дискета содержит программные коды примеров, рассматриваемых в книге

WWW.JUNIOR.COM.UA

Junior

Ю. А. Шпак

Turbo Pascal 7.0 на примерах

Под редакцией Ю. С. Ковтанюка

Киев "Издательство "ЮНИОР"
2003

ББК 32.973-01
Ш83
УДК 681.3.06

Шпак Ю. А.
Ш83 Turbo Pascal 7.0 на примерах/Под ред. Ю. С. Ковтанюка — К.: Издательство Юниор, 2003. — 496 с., ил.
ISBN 966-7323-30-7

Книга представляет собой учебное пособие, в котором материал излагается по схеме "от простого к сложному". Пособие рассчитанно на начинающих программистов в среде Turbo Pascal 7.0. Особое место в книге было уделено примерам, иллюстрирующим различные возможности языка Pascal и библиотечных программных модулей. В отдельную часть вынесены примеры более сложных программ, например, для работы с базами данных.

Особое место в книге занимают приложения, в которые были включены краткие справочники по командам языка ассемблера и по прерываниям. Одно из приложений представляет собой полный справочник по процедурам и функциям языка Pascal с примерами их использования.

ББК 32.973-01

Учебное издание

Шпак Юрий Алексеевич

Turbo Pascal 7.0 на примерах

Под редакцией Ю. С. Ковтанюка

Подписано в печать 05.09.2003. Формат 70 x 100 1/16.
Бумага газетная. Гарнитура Тайме. Печать офсетная.
Усл. печ. л. 19. Уч.-изд. л. 22,10.
Тираж 2000 экз. Заказ № 100903

Издательство "Юниор", Украина, 03142, г. Киев, ул. Совхозная, 35, оф. 111
тел./ф.: (044) 452-82-22; e-mail: office@junior.com.ua; http://www.junior.com.ua
Свидетельство о внесении субъекта издательского дела в Государственный реестр издателей, производителей и распространителей издательской продукции:
серия ДК, № 368 от 20 марта 2001 г.

Отпечатано с готовых диапозитивов в ЧП "Медиум"
88018, г. Ужгород, ул. Б. Хмельницкого, 2

ISBN 966-7323-30-7 ,

Краткое оглавление

Оглавление

Введение

Книга, которую вы держите в руках, представляет собой вводный курс по изучению методик программирования в среде Turbo Pascal 7.0. Первая версия этой системы была разработана компанией Borland в середине 1980-х годов, и хотя последняя версия Turbo Pascal была выпущена еще в 1997 году, эта система программирования пользуется неизменной популярностью как средство обучения основам программирования — как линейного программирования, так и объектно-ориентированного.

Материал книги, разбитой на 4 части, построен по принципу "от простого к сложному".

В первой части можно кратко познакомиться с интерфейсом интегрированной среды программирования Turbo Pascal 7.0, а также приступить к разработке первых несложных программ, демонстрирующих базовые конструкции языка Pascal.

Во второй части рассматриваются более сложные вопросы, связанные со структурированными типами данных.

В третьей части рассматриваются основы разработки приложений с применением таких мощных средств как встроенный ассемблер, прерывания и система программирования Turbo Vision.

Каждая глава сопровождается достаточно простыми примерами.

Четвертая часть книги полностью посвящена изучению примеров программ, в которых реализована работа с массивами, со звуком и мышью, а также создание базы данных.

Поскольку эта книга является, скорее, учебником для начинающих программистов, а не пособием для профессионалов, материал изложен таким образом, чтобы акцентировать внимание на главных моментах, необходимых для изучения языка программирования Pascal. Рассмотренные в книге программы оснащены достаточно подробными комментариями и имеют умышленно упрощенный интерфейс, такой подход облечает понимание примеров и повышает скорость изучения языка программирования.

Информация, изложенная в приложениях, позволяет освоить не только средства языка Pascal, и использовать эту книгу в качестве справочника по языку Pascal, но и изучить все "сильные стороны" языка ассемблера, что может значительно расширить "арсенал" читателя как программиста — справочники по основным прерываниям BIOS/DOS и командам процессоров 8x86.

Если это ваше первое путешествие в мир программирования, тогда не сомневайтесь — впереди вас ждет много интересного. Надеюсь, что эта книга станет для вас хорошим подспорьем в начале вашего пути к вершинам мастерства и профессионализма в области программирования.

ЧАСТЬ I

ОСНОВЫ ПРОГРАММИРОВАНИЯ В СРЕДЕ TURBO PASCAL

Терминология

Драйвер — специальная программа, управляющая работой некоторого внешнего устройства.

Интегрированная среда программирования — набор взаимодействующих средств, предназначенных для разработки программ. Доступ к таким средствам организован с помощью графических элементов пользовательского интерфейса (меню, окна, кнопки и т.д.).

Компиляция — процесс преобразования исходного текста программы в машинные коды.

Компоновка — процесс объединения отдельных откомпилированных модулей в единый выполняемый файл.

Модуль — файл с фрагментом кода программы. Исходный текст программных модулей хранится в файлах с расширением `.pas`, а откомпилированные модули — в файлах с расширением `.tpu`.

Модули рассматриваются в главе 14.

Отладка — исследование промежуточных результатов работы программы и поиск ошибок с использованием просмотра содержимого ячеек памяти компьютера.

Система программирования Turbo Pascal состоит из набора файлов, расположенных в нескольких каталогах. Рассмотрим четыре основных каталога и их содержимое (остальные каталоги содержат документацию и примеры программ).

- BGI. В этом каталоге расположены файлы, необходимые для организации работы в графическом режиме. Файлы с расширением `.bgi` — это драйверы различных графических компьютерных систем. Файлы с расширением `.chr` содержат шрифты.
- BIN. В этом каталоге расположены основные файлы пакета Turbo Pascal. Среди них ключевыми являются следующие файлы: `turbo.exe` — интегрированная среда программирования; `turbo.tpl` — библиотека стандартных модулей Turbo Pasca;

tpc.exe — компилятор программ Turbo Pascal в режиме командной строки (то есть вне интегрированной среды программирования).

» Компиляция в режиме командной строки рассматривается в главе 13.

- SOURCE. Этот каталог содержит файлы с расширением .pas — исходные тексты дополнительных программных модулей.
- UNITS. Каталог, в котором размещены откомпилированные дополнительные программные модули — файлы с расширением . tpu. Подобные откомпилированные модули *компонуются* в исполняемый файл с расширением . exe.

В общем случае процесс создания программ в среде Turbo Pascal состоит из следующих этапов.

- Написание текста программы. Исходный код программных модулей вводят либо в интегрированной среде программирования Turbo Pascal, либо при помощи обычного текстового редактора.
- Компиляция и компоновка программных модулей. Выполняется либо в интегрированной среде программирования, либо в режиме командной строки при помощи файла tpc.exe.
- **Отладка программы.**

Наиболее удобно создавать программы при помощи *интегрированной среды программирования* Turbo Pascal, Так как она предоставляет средства для всех этапов процесса разработки: текстовый редактор, компилятор, компоновщик и отладчик. Возможности интегрированной среды программирования рассматриваются в книге постепенно в процессе изложения материала.

В этой части рассматриваются основы программирования в среде Turbo Pascal, а также такие вопросы как отладка и компиляция программ.

Глава 1

Знакомство со средой Turbo Pascal

Переход в среду Turbo Pascal

Для перехода в среду Turbo Pascal используется файл `turbo.exe`. После запуска этой программы на экране компьютера раскрывается интегрированная среда программирования Turbo Pascal, как показано на рис. 1.1.

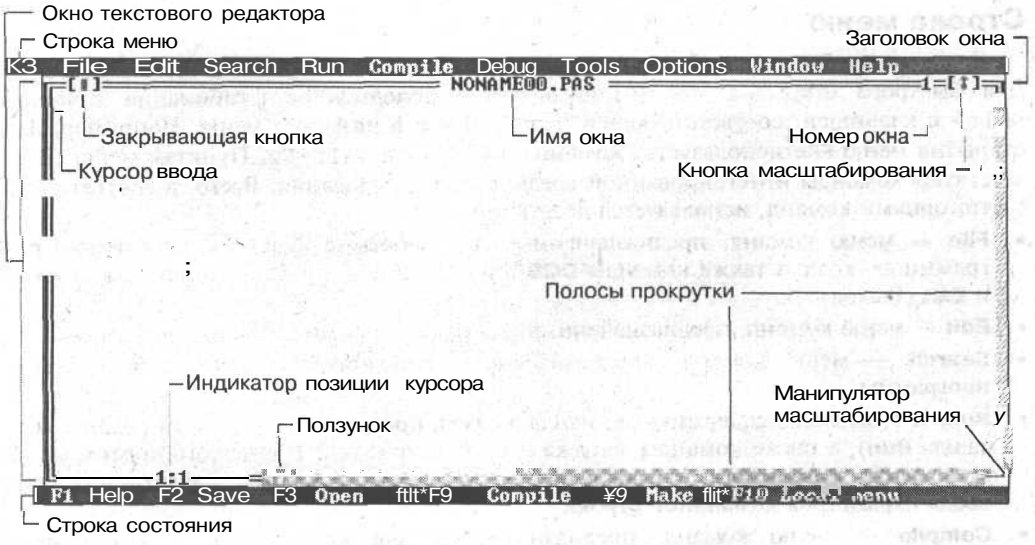


Рис. 1.1. Интегрированная среда программирования Turbo Pascal

ПРИМЕЧАНИЕ

Кроме интегрированной среды Turbo Pascal (файл `turbo.exe`) существует расширенная ее версия — среда Borland Pascal (файл `bp.exe`), которая позволяет создавать программы не только для операционной системы MS-DOS, но и для операционной системы Windows. Методы работы в среде Turbo Pascal полностью применимы для работы в среде Borland Pascal.

Интегрированная среда программирования Turbo Pascal работает в среде MS-DOS. Если требуется перейти в эту среду из операционной системы Windows, то запустить на выполнение программу `turbo.exe` можно одним из следующих способов.

- В режиме командной строки MS-DOS. Доступ к этому режиму осуществляется при помощи соответствующей команды системного меню **Пуск | Программы | Стандартные | Сеанс MS-DOS.**

- При помощи какой-либо оболочки управления файлами, например: Explorer, Windows Commander, Norton Commander, Volkov Commander, Far и т.п.
- Команды системного меню **Пуск | Выполнить.** В этом случае среда Turbo Pascal будет открыта в окне Windows.

Иногда для работы с кирилическим текстом в среде Turbo Pascal необходимо использование какой-либо русификатор клавиатуры, например, `keyrus.com`. Подобные программы запускаются перед открытием среды Turbo Pascal, файл `turbo.exe` следует запускать на выполнение только из режима командной строки MS-DOS или из DOS-ориентированных оболочек управления файлами.

Рассмотрим кратко основные элементы интегрированной среды программирования.

Элементы интегрированной среды программирования

Иинтегрированная среда программирования Turbo Pascal состоит из трех основных элементов: строка меню, окно текстового редактора, строка состояния.

Строка меню

Для того чтобы активизировать строку меню, достаточно нажать клавишу <F10>. Для быстрого открытия какого-нибудь меню используется комбинация клавиши `<Alt>` с клавишей, соответствующей первой букве в названии меню. Например, для открытия меню **File** используется комбинация клавиш <Alt+F>. Пунктам меню соответствуют команды интегрированной среды программирования. Всего, в соответствии с категориями команд, используется десять меню.

- **File** — меню команд, предназначенных для работы с файлами исходного программного кода, а также команды **DOS Shell** (Переход в режим командной строки) и **Exit** (Выход).
- **Edit** — меню команд, предназначенных для редактирования текста программы.
- **Search** — меню команд, предназначенных для поиска информации в тексте программы.
- Run. В этом меню содержится команда запуска программы в обычном режиме (команда Run), а также команды запуска в режиме отладки. Кроме того, в этом меню есть пункт **Parameters,** который используется для открытия диалогового окна ввода параметров командной строки.
- **Compile** — меню команд, предназначенных для компиляции и компоновки программы.
- **Debug** — меню команд, предназначенных для отладки программ.
- **Tools** — меню команд, предназначенных для работы с сообщениями, а также для вызова дополнительных средств, подобных Turbo Assembler.
- **Options** — меню команд, которые используются для настройки параметров интегрированной среды программирования.
- **Window** — меню команд, предназначенных для работы с окнами в интегрированной среде программирования.
- **Help** — меню команд, предназначенных для работы со справочной системой.

Некоторым командам меню поставлены в соответствие комбинации клавиш быстрого доступа. Например, команде **File | Save** соответствует клавиша `<F2>`.

▸▸ Клавиши быстрого доступа приведены в приложении Б.

Окно текстового редактора

Окно — это область экрана, которую можно перемещать, изменять ее размер, закрывать и открывать. В окнах текстового редактора среды Turbo Pascal выполняется ввод и редактирование исходного текста программ и модулей. Количество одновременно открытых окон в интегрированной среде Turbo Pascal ограничено только объемом свободной оперативной памяти, однако активным в любой момент времени может быть только одно окно. Ввод текста выполняется только в активном окне текстового редактора. Точно так же, все выполняемые команды применяются только к активному окну.

Элементы управления окнами в интегрированной среде Turbo Pascal

Окна в интегрированной среде Turbo Pascal имеют следующие элементы управления (см. рис. 1.1).

- **Закрывающая кнопка.** Кнопка расположена в левом верхнем углу окна и используется для закрытия окна.
- **Заголовок окна.** Заголовок расположен в самом верху окна — горизонтальная строка, содержащая имя и номер окна. Заголовок используют для перетаскивания окна мышью.
- **Кнопка масштабирования.** Кнопка расположена в правом верхнем углу окна и используется для увеличения окна до максимально возможного размера — [T], например, окно **Help** (рис. 1.2). В случае, если окно развернуто, кнопка масштабирования принимает вид двунаправленной стрелки, как показано на рис. 1.1.
- **Индикатор позиции курсора.** Индикатор используется только в окнах текстового редактора. Он расположен в левом нижнем углу окна: первая цифра означает номер текущей строки, а вторая — позицию курсора в строке.
- **Полосы прокрутки.** Полосы прокрутки расположены вдоль нижнего и правого края окна и используются для прокрутки содержимого окна. Обе полосы прокрутки работают одинаково. Прокрутку содержимого окна можно выполнить одним из следующих действий: *большими частями,* щелкая мышью на светло-серых участках полосы прокрутки; *небольшими частями,* щелкая мышью на кнопках прокрутки, расположенных в начале и конце полосы прокрутки; *на произвольное расстояние,* перетаскивая ползунок полосы прокрутки в нужном направлении.

Также перемещаться по тексту программы можно при помощи клавиш. В табл. 1.1 представлены комбинации клавиш для перемещения курсора ввода в окне текстового редактора и описание их действия.

Таблица 1.1. Клавиши перемещения курсора

Клавиши	Описание
<←>	Перемещает курсор на символ влево
<→>	Перемещает курсор на символ вправо
<↑>	Перемещает курсор на строку вверх
<↓>	Перемещает курсор на строку вниз
<Ctrl+←>	Перемещает курсор на слово влево
<Ctrl+→>	Перемещает курсор на слово вправо
<Ctrl+W>	Прокрутка текста на строку вверх
<Ctrl+Z>	Прокрутка текста на строку вниз
<Home>	Перемещает курсор в начало строки

Окончание таблицы 1.1

Клавиши	Описание
<End>	Перемещает курсор в конец строки
<PgUp>	Перемещает курсор на страницу вверх (отобразить предыдущую страницу)
<PgDn>	Перемещает курсор на страницу вниз (отобразить следующую страницу)
<Ctrl+Home>	Перемещает курсор на верхнюю строку текущей страницы (окна)
<Ctrl+End>	Перемещает курсор на нижнюю строку текущей страницы (окна)
<Ctrl+PgUp>	Перемещает курсор в начало файла
<Ctrl+PgDn>	Перемещает курсор в конец файла

- **Манипулятор масштабирования.** Манипулятор расположен в правом нижнем углу окна среды Turbo Pascal и используется для произвольного изменения размера окна. Для того чтобы изменить размер окна, необходимо перетащить мышью манипулятор масштабирования в другое положение на экране.

Управление окнами интегрированной среды Turbo Pascal

Кроме окон текстового редактора в интегрированной среде Turbo Pascal используются и другие окна. Например, если выполнить команду **Debug | Watch,** то откроется окно с заголовком **Watches,** которое используется для просмотра промежуточных значений переменных в процессе отладки программы (понятие "переменная" рассматривается в следующей главе). Если же нажать клавишу <F1>, то откроется окно справки с заголовком **Help** (рис. 1.2).

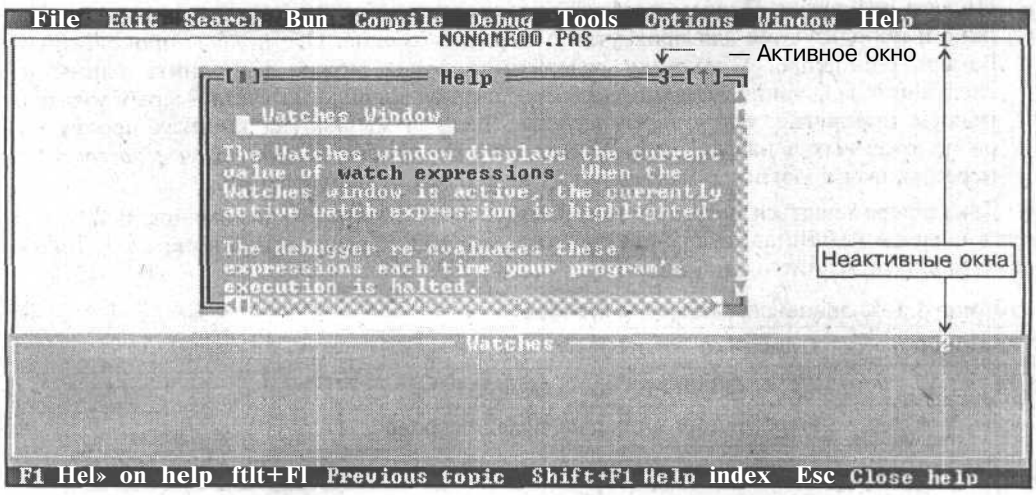


Рис. 1.2. Различные окна интегрированной среды программирования Turbo Pascal

Обратите внимание на то, что активное окно имеет двойную рамку, а неактивные окна — одинарную рамку. Следует также отметить, что в интегрированной среде может быть открыто любое количество окон. При этом каждому окну присваивается порядковый номер, отображаемый в правом верхнем углу (см. рис. 1.1). Для быстрого перехода к любому открытому окну по его порядковому номеру можно воспользо-

работы с интегрированной средой программирования Turbo Pascal открыты файлы с программами, которые не сохранялись после их изменения, то на экране появится диалоговое окно (см. рис. 1.9). Эта ситуация подобна попытке закрыть окно текстового редактора с измененной программой, которая описана в предыдущем разделе.

Открытие файлов с текстами программ

Автоматическое открытие файлов

1. Сохраните файл `Prog01.pas`, выполнив команду **File | Save** или нажав клавишу <F2>.
2. Выполните команду **Options | Save,** чтобы сохранить текущие значения параметров интегрированной среды. Если этого не сделать, то при следующем запуске Turbo Pascal будут установлены значения параметров, выбранные по умолчанию.
3. Теперь закройте Turbo Pascal, выполнив команду **File | Exit** или нажав комбинацию клавиш <Alt+X>.
4. Запустите Turbo Pascal еще раз. Те файлы, которые были открыты в интегрированной среде программирования в момент выполнения команды **Options | Save,** будут открыты автоматически. Например, файл ProgOl.pas из предыдущего раздела.

Открытие файла при помощи диалогового окна Open a File

Для того чтобы открыть любой файл с текстом программы, выполните команду **File | Open** или нажмите клавишу <F3>. В результате на экране проявится диалоговое окно выбора файла **Open a File** (рис. 1.10).

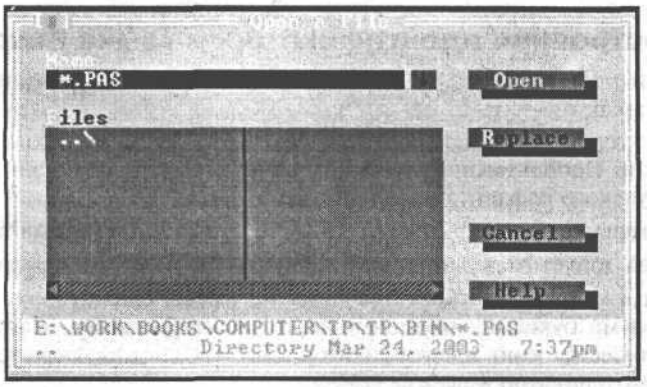


Рис. 1.10. Диалоговое окно для открытия файла

Поиск файла осуществляется при помощи списка **Files,** а имя выбранного файла отображается в поле **Name.** После того, как требуемый файл выбран, можно щелкнуть мышью на кнопке **Open** или нажать клавишу <Enter>. Как уже упоминалось, в интегрированной среде программирования Turbo Pascal можно открывать одновременно много файлов. Интегрированная среда "запоминает" список файлов, которые открывались в течение текущего сеанса работы. Этот список отображается в нижней части меню **File,** а также в окне, раскрываемом на экране при помощи кнопки с изображением направленной вниз стрелки, которая расположена справа от поля **Name** в диалоговом окне **Open a File.**

Глава 2

Идентификаторы, константы, переменные и операторы

Константы и переменные в программе обозначаются при помощи *идентификаторов*.

Идентификаторы

Идентификаторы — это имена, которые используются не только для обозначения констант и переменных, но также программ, процедур, функций и объектов. Идентификаторы бывают *стандартными* и *пользовательскими*. Стандартные идентификаторы используются для обозначения заранее определенных типов данных (типы данных рассматриваются в следующей главе), констант, процедур и функций. В качестве примера стандартного идентификатора можно привести имя процедуры `Writeln`, которая использовалась в программе Prog01`.pas` (см. листинг 1.1).

Для обозначения констант, переменных, процедур и функций, определенных самим программистом, используются пользовательские идентификаторы. При этом в одном и том же блоке программы один идентификатор не может использоваться для обозначения нескольких переменных, констант и т.п.

Правила построения идентификаторов языка Pascal

При создании программ следует соблюдать общие правила построения идентификаторов языка Pascal.

1. В идентификаторах могут использоваться только латинские буквы, цифры и знак подчеркивания. Использование пробелов, точек и других специальных символов, а также букв русского алфавита недопустимо.
2. Идентификаторы должны начинаться только с буквы или символа подчеркивания.
3. Максимальная допустимая длина идентификаторов — 127 символов.
4. Регистр букв в идентификаторах значения не имеет, однако рекомендуется выделять прописными буквами смысловые части идентификатора. Например, для обозначения количества книг идентификатор `NumberOfBooks` является более наглядным, чем идентификатор `numberofbooks`.

СОВЕТ

Идентификаторы рекомендуется выбирать таким образом, чтобы они передавали смысл той или иной величины. Очевидно, что имя идентификатора `NumberOf`Books несравненно более информативно, чем имя идентификатора N.

Константы

Константа — это элемент данных со строго определенным значением, которое в процессе выполнения программы не изменяется. Например, если в программе какая-

нибудь операция выполняется через строго определенные временные интервалы, то значение этого интервала лучше определить при помощи константы.

Объявления констант

Объявления констант размещаются в тексте программы после зарезервированного слова const в следующем формате:

```
I идентификатор = значение
```

Константы объявляются в программе, модуле, процедуре или функции **перед** первым словом begin.

Объявим несколько констант (листинг 2.1) в созданной ранее программе ProgOl (см. листинг 1.1) и используем их для вывода информации на экран (рис. 2.1).

Листинг 2.1. Использование переменных для вывода информации на экран

```
program ProgOl;
const
  MyName = 'Юрий Шпак';
  MyBirthDate = '7 августа 1974 года';
  MyBirthYear = 1974;
begin
  Writeln('Привет, мир!');
  Writeln('Меня Зовут ', MyName);
  Writeln('Я родился' , MyBirthDate) ;
end.
```

```
Привет, мир!
Меня зовут Юрий Шпак
Я родился 7 августа 1974 года
```

Рис. 2.1. В результате выполнения программы ProgOl.pas на экран будут выведены три строки

В языке Pascal есть несколько констант, к которым можно обращаться без предварительного объявления. Такие константы называют *зарезервированными*. Рассмотрим некоторые из них:

- True. Этой константе соответствует логическое значение "Истина".
- False. Этой константе соответствует логическое значение "Ложь".
- Maxint. Этой константе соответствует максимальное значение типа integer (целое число), то есть — 32767.

Переменные

Переменная — это величина, которая может изменять свое значение во время выполнения программы. Тип переменной определяется типом данных, которые будут храниться в этой переменной. Тип переменной обязательно должен быть определен до того, как над ней будут выполнены какие-либо действия.

Объявления переменных

Объявления переменных размещаются в тексте программы после зарезервированного слова var в формате

```
перечень_идентификаторов : тип
```

Так же как и константы, переменные объявляются **перед** первым словом begin программного блока.

Объявим в программе ProgOl (см. листинг 2.1) две переменных и используем их для вычисления возраста автора книги (листинг 2.2).

Листинг 2.2. Использование переменных для вычислений

```
program ProgOl;
const
  MyName = 'Юрий Шпак';
  MyBirthDate = '7 августа 1974 года';
  MyBirthYear = 1974;
var
 CurrentYear, MyAge: integer;
begin
  Writeln('Привет, мир!');
  Writeln('Меня зовут ', MyName);
  Writeln('Я родился', MyBirthDate);
  CurrentYear := 2003;
  MyAge := CurrentYear - MyBirthYear;
  Writeln('Мне ', MyAge, ' лет');
end.
```

Для сохранения в переменной значения используется так называемый *оператор присваивания.* Его формат следующий:

```
переменная := выражение
```

(Не путайте его с оператором сравнения =.) В левой части оператора присваивания находится идентификатор переменной, а в правой части — выражение соответствующего типа.

В программе ProgOl (см. листинг 2.2) переменной CurrentYear присваивается целочисленное значение, соответствующее типу integer.

Если попытаться присвоить этой переменной значение 200300, то при компиляции программы на экране появилось бы сообщение об ошибке, так как это число больше допустимого значения для типа integer, определенного диапазоном возможных значений от -32768 до 32767. Если присвоить переменной CurrentYear строковое значение '2003', то при компиляции программы на экране появится сообщение о несоответствии типов. Таким образом, переменной можно присвоить только то значение, которое соответствует ее типу, объявленному в разделе var.

Возможны случаи, когда переменным одного типа можно присваивать значения другого типа. Например, переменным типа real (числа с плавающей запятой) можно присваивать целочисленные значения. Подобные операции называются **приведением типов** (более подробно типы данных рассматриваются в следующей главе).

Типизированные константы

Каждая константа, так же как и переменная, принадлежит к определенному типу данных. При объявлении констант их тип распознается компилятором автоматически. Например, в программе ProgOl используется константа MyName типа string и константа MyBirthYear типа integer, хотя явно это не указано.

В языке Pascal существует возможность указывать тип констант явно. Подобные константы называются **типизированными.** Объявление подобных констант имеет следующий формат:

```
const идентификатор : тип = значение;
```

Например, в программе ProgO1 константу `MyBirthYear` можно было бы объявить как типизированную:

```
MyBirthYear: integer = 1974;
```

Операторы и выражения

Операторы выполняют некоторые предопределенные действия над переменными и константами. Например, оператор "+" выполняет сложение двух значений. Последовательность из нескольких операторов называется **выражением.** Например, рассмотрим две строки из программы ProgO1 (см. листинг 2.2).

```
CurrentYear := 2003;
MyAge := CurrentYear - MyBirthYear;
```

Первая строка — это пример оператора присваивания, а вторая — сочетание операторов вычитания и присваивания.

Назначение операторов

Все операторы можно разбить на несколько групп, соответствующих характеру выполняемых операторами действий над выражениями **(операндами).** Эти группы перечислены в табл. 2.1.

Таблица 2.1. Операторы языка Pascal

Категория	Оператор	Описание
Арифметические операторы	+	Сложение
	–	Вычитание или изменение знака
	*	Умножение
	/	Деление чисел с плавающей запятой
	div	Целочисленное деление
	mod	Получение остатка от деления
Логические операторы	not	Отрицание
	and	Логическое "И"
	or	Логическое "ИЛИ"
	xor	Логическое исключающее "ИЛИ"
Операторы сравнения	=	Равно
	<>	Не равно
	<	Меньше
	>	Больше
	<=	Меньше или равно
	>=	Больше или равно
Побитовые операторы (начало)	not	Побитовое отрицание
	and	Побитовое умножение

Окончание таблицы 2.1

Категория	Оператор	Описание
Побитовые операторы (окончание)	or	Побитовое сложение
	xor	Побитовое исключающее сложение
	shl	Побитовый сдвиг влево
	shr	Побитовый сдвиг вправо

Следует отметить, что оператор присваивания (:=) не относится ни к одному из представленных типов, и используется для сохранения в переменных результатов вычислений выражений.

Унарные и бинарные операторы

Операторы бывают унарными и бинарными. **Бинарные** операторы применяются к двум операндам, а **унарные** — к одному операнду. К унарным операторам относятся следующие операторы:

not — логическое или побитовое отрицание;

- (знак минус) — изменение знака;

in — используется для проверки принадлежности какого-нибудь значения указанному множеству (⏩ рассмотрен в следующей главе);

@ — используется для присвоения некоторого значения переменной ссылочного типа (⏩ рассмотрен в главе 11).

Все остальные операторы являются бинарными.

Порядок выполнения операторов

В сложных выражениях имеет большое значение порядок выполнения операторов. Распределение операторов в соответствии с приоритетом их выполнения представлено в табл. 2.2.

Таблица 2.2. Приоритет выполнения операторов языка Pascal

Операторы	Приоритет
Not	Самый высокий
*, /, div, mod, and, shl, shr	
+, -, or, xor	
=, <>, <, >, <=, >=, in	Самый низкий

В завершение этой главы рассмотрим логические операторы not, and, or и xor, а также операторы сравнения. Примеры использования арифметических и побитовых операторов будут рассмотрены в следующей главе, посвященной простым типам данных языка Pascal.

Логические операторы и операторы сравнения

Результатом выполнения логического выражения, состоящего из логических операторов (not, and, or и xor) является логическое (булево) значение True ("Истина") или False ("Ложь"). Операндами в таких выражениях могут быть только данные типа boolean, получаемые при помощи операторов сравнения (=, >, <, >=, <=, о) и круглых скобок.

Результатом применения унарного логического оператора not является инверсия текущего значения операнда.

Другими словами, если переменная A типа boolean имеет значение True, то в результате выполнения оператора

```
not A
```

будет получено значение False (и наоборот).

Результат выполнения остальных логических операторов представлен в табл. 2.3.

Таблица 2.3. Результат применения логических операторов and, or и `xor`

Первый операнд	Второй операнд	Оператор and	Оператор or	Оператор **xor**
False	False	False	False	False
False	True	False	True	True
True	False	False	True	True
True	True	True	True	False

Исходя из данных, приведенных в табл. 2.3, можно сформулировать следующие правила для логических операторов.

- В случае применения оператора and (логическое "И") результат получается истинным (значение `True`) только в том случае, если оба операнда имеют значение `True`.
- В случае применения оператора or (логическое "ИЛИ") результате получается истинным (значение True) только в том случае, если хотя бы один из операндов имеет значение `True`.
- В случае применения оператора `xor` (логическое исключающее "ИЛИ") результате получается истинным (значение True) только в том случае, если значение первого операнда не совпадает со значением второго операнда.

Рассмотрим пример использования логических операторов совместно с операторами сравнения. Выполните в интегрированной среде программирования Turbo Pascal команду **File | New,** чтобы создать новый файл, и сохраните его на диске под именем `BoolOp.pas`. Введите в одноименном окне текст программы, представленной в листинге 2.3, или откройте этот файл при помощи команды **File | Open** (<F3>). Файл BoolOp.pas находится на прилагаемой к книге дискете в папке 02.

Листинг 2.3. Программа `BoolOp.pas`

```
program BoolOp;
var
  A, B: integer;
begin
  Write('ВведитеA:');
  Readln(A);
  Write('ВведитеB:');
  Readln(B);
  WritelnCA = B (', A=B,')');
  Writeln('A>B (', A > B, ')');
  Writeln('A < B (', A<B,')');
  Writeln('A >= B (', A>=B,')');
  Writeln('A <= B (', A<= B,')');
  Writeln('A 0 B (', AOB,')');
end.
```

В программе BoolOp была использована функция Readln. Ее назначение противоположно назначению функции Writeln — функция Readln считывает значение,

введенное с клавиатуры, и сохраняет его в переменной, переданной в качестве параметра. В данном случае сначала программа будет ожидать ввода целого числа, которое будет присвоено переменной A, а затем — ввода целого числа, которое будет присвоено переменной B.

ВНИМАНИЕ

Для того чтобы введенное значение было присвоено переменной, необходимо нажать клавишу <Enter>.

Следует также отметить, что для ввода значений переменных A и B можно было бы воспользоваться и одной функцией Readln, например:

```
Write('Введите A и B:');
Readln(A,B);
```

В этом случае значения A и B вводятся через пробел, или символ табуляции, или при помощи клавиши <Enter>, для перехода на новую строку.

Запустите программу BoolOp на выполнение, нажав комбинацию клавиш <Alt+F5>. Введите какие-нибудь значения для переменных A и B, а затем посмотрите на результат работы программы. Например, если для переменной A было введено значение 1, а для переменной B — значение 2, то результат работы программы BooЮр будет выглядеть, как показано на рис. 2.2.

```
Введите A:1
Введите B:2,
A = B  (FALSE)
A > B  (FALSE)
A < B  (TRUE)
A >= B  (FALSE)
A <= B  (TRUE)
A 0 B  (TRUE)
```

Рис. 2.2. Результат выполнения программы BooЮр.pas

Если необходимо, сохраните файл BoolOp.pas на диске, а затем выполните в интегрированной среде программирования Turbo Pascal команду **File | New,** чтобы создать файл новой программы. Присвойте этому файлу имя, например, BoolOp2.pas, и введите в него текст, представленный в листинге 2.4, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 2.4. Программа BoolOp2.pas

```
program BoolOp2;
var
  A, B, C: integer;
begin
  Writeln('Введите A,B,C:');
  Readln(A,B,C);
  Write('A больше B и меньше C - ');
  Writeln ((A > B) and (A < C));
  Write('A равно B или C - ');
  Writeln((A = B) or (A = C));
  Write('A меньше B и больше или равно C - ');
  Writeln((A < B) and ( (A > C) or (A = C) ) );
  Write('A не равно B и не равно C - ' );
```

Окончание листинга 2.4

```
  Writeln(not ((A = B) or (A = C)));
end.
```

Запустите программу Boo10p2 на выполнение и в качестве значений переменных A, B и C введите 2, 1 и 3. Результат выполнения программы показан на рис. 2.3.

ПРИМЕЧАНИЕ

Напоминаем, чтобы введенное значение было присвоено переменной, необходимо после каждого введенного значения нажать одну из клавиш: <Пробел>, <Tab> или <Enter>. На рис. 2.3 показано, как ввод значений завершался нажатием клавиши <Enter>.

```
Введите A,B,C:
2
1
3
A больше B и меньше C - TRUE
A равно B или C - FALSE
A меньше B и больше или равно C - FALSE
A не равно B и не равно C - TRUE
```

Рис. 2.3. Результат выполнения программы BoolOp2.pas

Обратите внимание на использование скобок в программе Boo10p2. Эти скобки отделяют друг от друга операнды логических операторов. Их применение крайне необходимо. Например, в выражении A and B or C вначале будет выполнен оператор and, а затем — оператор or. Если же это выражение представить в виде A and (B or C), то вначале будет выполнен оператор or, а затем — оператор and. Таким образом, вначале выполняются операторы в скобках, а затем — остальные операторы. При этом учитывается вложенность операторов. Например, в выражении not ((A = B) or (A = C)) вначале выполняются операторы сравнения =, затем — оператор or, и в последнюю очередь — оператор not.

Глава 3

Простые типы данных

В предыдущих двух главах упоминалось такое понятие как "тип данных". Например, в программе `Prog01.pas` (см. листинг 2.2) объявлена переменная `MyAge` типа `integer`. По существу, тип — это название данных определенного вида. Например, при объявлении переменой обязательно указывается ее тип, определяющий набор допустимых значений этой переменной и операций, которые могут над ней выполняться. Любое выражение возвращает данные определенного типа. То же самое относится и к функциям.

Все типы данных языка Pascal делятся на *простые* и *структурированные* типы.

Простыми называются такие типы данных, значения которых не содержат составных частей. К ним относятся следующие типы:

- целочисленные типы;
- вещественные типы;
- символьный тип char;
- логический тип boolean;
- перечислимый тип;
- интервальные типы.

Структурированные типы данных определяют упорядоченную совокупность значений одного или нескольких простых типов. К ним относятся следующие типы:

- строки;
- массивы;
- множества;
- записи;
- файлы;
- указатели;
- объекты.

Кроме того, все типы данных делятся на *стандартные* и *пользовательские* типы.

Стандартные типы данных по отношению к системе Turbo Pascal являются встроенными. К таким типам относятся, например, целочисленные типы.

Пользовательские типы данных создаются самим программистом. К таким типам, например, относятся перечислимые типы и множества.

Для объявления пользовательских типов используется следующая конструкция:

```
type имя_типа = определение_типа;
```

Эта глава посвящена простым типам данных, а структурированные типы рассматриваются в части 2.

Целочисленные типы данных

Целочисленные типы данных языка Pascal перечислены в табл. 3.1.

Таблица 3.1. Целочисленные типы данных языка Pascal

Название типа	Диапазон значений
byte	**0..255**
shortint	**-128..127**
integer	-32768..32767

Окончание таблицы 3.1

Название типа	Диапазон значений
word	0..65535
longint	-2147483648..2147483647

Значения типов byte и shortint занимают в памяти компьютера один байт, значения типов integer и word — два байта, а значения типа longint — четыре байта.

Значения целочисленных типов данных могут быть представлены в программах в десятеричном и шестнадцатеричном виде. В десятичной системе каждому разряду числа соответствует значение от 0 до 9. В шестнадцатеричной системе каждому разряду числа соответствует значение от 0 до 15, при этом цифры от 10 до 15 обозначаются при помощи латинских букв от A до F. Для хранения одной шестнадцатеричной цифры в памяти отводится четыре двоичных разряда.

Двоичное представление шестнадцатеричных цифр, а также из соответствие десятичным числам представлено в табл. 3.2.

Таблица 3.2. Двоичное предстайление шестнадцатеричных цифр

Шестнадцатеричная цифра	Двоичное представление	Десятичное число
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Если число в программе на языке Pascal представлено в шестнадцатеричной системе, то перед ним без пробела ставится символ $. Примеры преобразования шестнадцатеричных чисел в десятичные представлены в табл. 3.3.

Таблица 3.3. Примеры преобразования шестнадцатеричных чисел в десятичные

Шестнадцатеричное число	Преобразование	Десятичное число
1	$1*16^0 = 1*1 = 1$	1
C	$C*16^0 = 12*16^0 = 12*1$	12
10	$1*16^1 + 0*16^0 = 1*16 + 0$	16
2F	$2*16^1 + F*16^0 = 2*16 + 15$	47

Окончание таблицы 3.3

Шестнадцатеричное число	Преобразование	Десятичное число
2A1	$2*16^2 + A*16^1 + 1*16^0 =$ 2*256 + 10*16 + 1*1 = 512 + 160 + 1	673
FFFF	$F*16^3 + F*16^2 + F*16^1 + F*16^0 =$ 15*4096 + 15*256 + 15*16 + 15 = 61440 + 3840 + 240 + 15	65535

К целочисленным Данным можно применять арифметические операторы, операторы сравнения, а также побитовые операторы. В результате применения арифметических и побитовых операторов получаются значения целочисленного типа, а в результате применения операторов сравнения — значения логического типа boolean.

Примеры использования целочисленных типов данных

Выполните в интегрированной среде программирования Turbo Pascal команду **File | New,** чтобы создать новый файл, и сохраните его на диске под именем IntTypes.pas. Введите в него текст, представленный в листинге 3.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 3.1. Программа IntTypes.pas

```
program IntTypes;
var
NumByte: byte;
  NumShort: shortint;
  NumInt: integer;
  NumWord: word;
  NumLong: longint;
begin
  NumLong := 100000;
  NumWord := NumLong div 3;
  NumInt := $FF + 1;
  NumShort := NumLong mod 3;
  NumByte := NumInt;
  writeln('NumLong = ',NumLong);
  writeln('NumWord = ',NumWord);
  writeln('NumInt = ',NumInt);
  writeln('NumShort = ',NumShort);
  writeln('NumByte = ',NumByte);
end.
```

Выполните эту программу, нажав комбинацию клавиш <Ctrl+F9>, а затем просмотрите результат ее работы, нажав комбинацию клавиш <Alt+F5>. На экране появятся сообщения, показанные на рис. 3.1.

В программе IntTypes используются арифметические операторы div и mod. Оператор div выполняет деление двух целых чисел — при этом остаток от деления отбрасывается. В данном случае в результате целочисленного деления 100000 на 3 было получено значение переменной NumWord = 33333. Остаток от деления 100000 на 3 (единица), полученный при помощи оператора mod, был сохранен в переменной

NumShort. Переменной NumInt был присвоен результат сложения шестнадцатеричного числа FF с десятичной единицей. Шестнадцатеричному числу FF соответствует десятичное число 255 ($F*16^1 + F*16^0 = 15*16 + 15*1 = 240 + 15 = 255$). После сложения 255 с единицей было получено число 256, которое было присвоено переменной NumByte. Но почему же в результате переменная NumByte оказалась равной не 256, а нулю? Дело в том, что переменная NumByte имеет тип byte, диапазон положительных значений которого ограничен числом 255. Это объясняется тем, что под переменные типа byte отводится один байт памяти, в котором может быть сохранено максимум восемь двоичных единиц. Двоичные представления некоторых чисел приведены в табл. 3.4.

```
NumLong = 100000
NumWord = 33333
Numlnt = 256
NumShort = 1
NumByte = 0
```

Рис. 3.1. Результат выполнения программы IntTypes.pas

Таблица 3.4. Двоичное представление чисел

Один байт - восемь двоичных разрядов	Схема перевода в десятичное число	Соответствующее десятичное число
00000000		0
00000001	$1*2^0$	1
00000010	$1*2^1+0*2^0 = 2 + 0$	2
00000011	$1*2^1 + 1*2^0 = 2 + 1$	3
00000100	$1*2^2+ 0*2^1 + 0*2^0 = 4 + 0 + 0$	4
...	...	...
11111110	$1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+0*2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 0$	254
11111111	$1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+1*2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$	255

ПРИМЕЧАНИЕ

Значения типа shortint также занимают в памяти один байт, однако восьмой (первый слева) разряд при этом отводится под знак (нулю соответствует знак "плюс", а единице — знак "минус"). Таким образом, максимальное число, типа shortint — это 01111111 = $1*2^6+ 1*2^5+ 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1+ 1*2^0=64+32+16+8+4+2+1=127$.

Для хранения десятичного числа 256 требуется девять двоичных разрядов: 100000000 = $1*2^8$. Таким образом, если переменной типа byte присвоить десятичное число 256, то девятый двоичный разряд теряется, и в результате останутся только первые восемь разрядов, заполненные нулями, то есть — число 0. Учитывая этот факт, следует быть внимательным при присвоении переменным значений с большей разрядностью, чем определено для этих переменных их типом (с меньшей разрядностью), так как это может привести к искажению результатов выражений.

Применение побитовых операторов

К целыми числами можно применять побитовые операторы (◄◄ см. раздел "Операторы и выражения" в гл. 2). Выполните в интегрированной среде программирования Turbo Pascal команду **File | New,** чтобы создать новый файл, и сохраните его на диске под именем `BitOp.pas`. Введите в него текст, представленный в листинге 3.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 3.2. Программа `BitOp.pas`

```
program BitOp;
var
  A, B: byte;
begin
  A := 11;                          {00001011}
  B := 6;                           {00000110}
  writeln('A=',A);
  writeln('B=',B);
  writeln('not A = ', not A);       {11110100 = 244}
  writeln('A and B = ', A and B);   {00000010 = 2}
  writeln('A or B = ', A or B);     {00001111 = 15}
  writeln('A xor B = ', A xor B);   {00001101 = 13}
  writeln('A shl 1 = ', A shl 1);   {00010110 = 22}
  writeln('B shr 2 = ', B shr 2);   {00000001 = 1}
end.
```

Запустите эту программу на выполнение, например, при помощи комбинации клавиш <Ctrl+F9>. Результат ее работы будет выглядеть, как показано на рис. 3.2.

```
A = 11
B = 6
not A = 244
A and B = 2
A or B = 15
A xor B = 13
A shl 1 = 22
B shr 2 = 1
```

Рис. 3.2. Результат выполнения программы `BitOp.pas`

Операнды, записанные в десятичной или шестнадцатеричной форме во время выполнения программы преобразуются в двоичную форму, а результат отображается в десятичной форме.

Арифметический оператор not выполняет побитовое отрицание. Это означает, что каждый двоичный разряд числа принимает противоположное значение. В данном случае двоичное число 00001011 (десятичное 11) превратилось в 11110100 (десятичное 244).

Арифметический оператор and выполняет побитовое умножение, арифметический оператор or — побитовое сложение, а арифметический оператор `xor` — побитовое исключающее сложение. Результат выполнения побитовых операций and, or и `xor` можно определить, руководствуясь табл. 3.5.

Исходя из данных, приведенных в табл. 3.5, можно сформулировать следующие правила для арифметических операторов.

- В случае применения арифметического оператора and, разряды результата будут равны 1 только в том случае, если соответствующие разряды обоих операндов равны 1.

Таблица 3.5. Результат применения арифметических операторов and, or и xor

Первый операнд	Второй операнд	Операция and	Операция or	Операция xor
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- В случае применения арифметического оператора or, разряды результата будут равны 1 только в том случае, если соответствующие разряды хотя бы одного из операндов равны 1.
- В случае применения арифметического оператора xor, разряды результата будут равны 1 только в том случае, если соответствующие разряды первого операнда не равны тем же разрядам второго операнда.

Оператор shl выполняет побитовый сдвиг влево, а оператор shr — побитовый сдвиг вправо на количество разрядов, указанное в качестве второго операнда. В программе BitOp исходное значение переменной A в двоичной форме представляется как 00001011. В результате побитового сдвига этого значения влево на один разряд (A shl 1) получается 00010110 (десятичное 22). Аналогичным образом, исходное значение переменной B в двоичной форме представляется как 00000110. В результате побитового сдвига этого значения вправо на два разряда (B shr 2) получается 00000001 (десятичное 1).

Вещественные типы данных

Вещественные типы данных языка Pascal представлены в табл. 3.6.

Таблица 3.6. Вещественные типы данных языка Pascal

Название типа	Диапазон значений	Количество цифр
real	$2{,}9*10^{-39}..1{,}7*10^{38}$	11-12
single	$1{,}5*10^{-45}..3{,}4*10^{38}$	7-8
double	$5*10^{-324}..1{,}7*10^{308}$	15-16
extended	$1{,}9*10^{-4951}..1{,}1*10^{4932}$	19-20
comp	$-2^{63}+1..2^{63}-1$	19-20

Значения типа real занимают в памяти компьютера шесть байтов, значения типа single — четыре байта, значения типа double и comp — восемь байтов, а значения типа extended — десять байтов.

Вещественные значения могут быть представлены в программах на языке Pascal в форме с фиксированной точкой, а также в экспоненциальном формате <мантисса>E<порядок>. Например, число 0.222 можно записать как 222E-03, а число 1000.12 — как 1.00012E+03.

К данным вещественного типа применяются арифметические операторы +(сложение), - (вычитание), * (умножение) и / (деление), а также операторы сравнения.

Результатом выражения, в котором в качестве операндов используются одновременно целочисленные и вещественные значения, будет вещественное число. Переменным вещественного типа можно присваивать целочисленные значения, но не наоборот.

Символьный тип данных

Значению символьного типа данных char соответствует один символ кодовой таблицы ASCII (American Standard Code for Information Interchange — американская стандартная кодировка для обмена информации). Каждому символу в этой таблице соответствует определенный целочисленный код в диапазоне от 0 до 255. Переменная символьного типа занимает в памяти компьютера один байт.

» Таблица кодировки ASCII представлена в приложении Е.

В тексте программы значения типа char заключаются в апострофы, например: 'A', '' (пустое значение), ' ' (пробел), '-'. К данным символьного типа применяют операторы сравнения.

Рассмотрим пример использования символьного типа данных. Создайте в интегрированной среде программирования Turbo Pascal новый файл, и сохраните его под именем CharType.pas. Введите в него текст, представленный в листинге 3.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг3.3. ПрограммаCharType.pas

```
program CharType;
var
  Letter: char;
  Code: byte;
begin
  Write('Введите символ:');
  Readln(Letter);
  Writeln('Код символа ', Letter, ' - ', ord(Letter));
  Write('Введите код символа:');
  Readln(Code);
  Writeln('Символ с кодом ', Code, ' - ', chr(Code));
  Read(Letter);
end.
```

Обратите внимание на вызов функции Read в конце программы CharType. Функции Read используется для того, чтобы остановить выполнение программы до тех пор, пока пользователь не нажмет какую-либо клавишу. Таким образом, результат работы программы, запущенной из интегрированной среды Turbo Pascal, не будет скрываться после выполнения последнего оператора этой программы.

В программе CharType сначала вводится символ, который затем сохраняется в переменной Letter. На основании значения этой переменной при помощи стандартной функции ord определяется код символа. Затем выполняется обратное действие — вводится код символа в переменную Code, на основании значения которой затем определяется сам символ при помощи стандартной функции chr. *Запомните эти две функции,* так как при работе с символьными значениями они используются очень часто.

Логический тип данных

Логический тип boolean уже упоминался в предыдущей главе в разделе "Логические операторы и операторы сравнения". Данные этого типа могут быть только двух значений: True ("Истина") и False ("Ложь"). Переменная типа boolean занимает в памяти компьютера один байт.

Перечислимый тип данных

Перечислимые типы данных используются для определения упорядоченных наборов значений в виде списка идентификаторов, соответствующих этим значениям. Для объявления перечислимого типа данных используется следующая конструкция:

```
type имя_типа = (значение1, ..., значениеN)
```

Например, можно объявить тип Digits, представляющий собой набор цифр:

```
type Digits = (Zero, One, Two, Three);
```

Затем можно объявить переменную типа Digits:

```
var CurDigit: Digits;
```

В данном случае можно также использовать упрощенный способ объявления переменной CurDigit, без объявления типа Digits:

```
var CurDigit: (Zero, One, Two, Three);
```

Каждому из значений перечислимого типа соответствует некоторое число. Первому значению соответствует число 0, второму — 1 и т.д. Для извлечения числа, соответствующего значению перечислимого типа, используется стандартная функция Ord (рассматривалась в разделе "Символьный тип данных"). Например, результатом выполнения функции Ord (One) будет 1.

Рассмотрим работу с перечислимыми типам данных на следующем примере. Если в программе, кроме вышеупомянутой переменной CurDigit объявить переменную Four типа integer, то ей можно присвоить значение при помощи следующего оператора:

```
Four := Ord(One) + Ord(Three);
```

К данным перечислимого типа можно применять операторы сравнения. Например, при вызове функции Writeln (One < Three); на экран будет выведено значение TRUE.

ПРИМЕЧАНИЕ

При попытке вызова функции вида Writeln (One), на этапе компиляции программы, будет выдано сообщение об ошибке. Дело в том, что язык Pascal не поддерживает непосредственный вывод значений перечислимого типа, поэтому за реализацию подобных задач отвечает сам программист.

Интервальные типы данных

Интервальный тип данных позволяет определить четкий диапазон целочисленных или символьных значений. При каждой операции с переменной интервального типа компилятор проверяет, находится ли ее значение внутри установленного для нее диапазона.

Примеры объявления переменных интервальных типов:

```
const
  Monday = 1; Tuesday = 2; Wednesday = 3; Thursday = 4;
  Friday = 5; Saturday = 6; Sunday = 7;
type
  ABC = 'A'..'Z';
  Digits = 0..9;
  WorkDays = Monday..Friday;
  Holidays = Saturday..Sunday;
```

Глава 4

Ввод и вывод данных

Любая программа, если в ней не реализован ввод и вывод данных, имеет мало смысла. Что пользы в игре, если она не реагирует на нажатие клавиш или не показывает никаких игровых действий; что пользы в текстовом редакторе, который не позволяет вводить текст или не отображает введенные символы на экране монитора?

Под вводом данных подразумевается передача программе на обработку какой-нибудь информации извне. Вывод данных — это обратный процесс, когда результаты обработки информации передаются программой на различные устройства компьютера (экран монитора, принтер, жесткий диск и др.).

В примерах из предыдущих глав уже использовались стандартные процедуры ввода (Write, Writeln) и вывода (Read, Readln) языка Pascal. В этой главе они рассмотрены подробнее. Кроме того, в разделе, посвященном выводу данных, затронут вопрос форматированного вывода. В конце главы рассмотрен пример программы, в которой организована запись и чтение данных с использованием текстового файла.

Процедуры ввода данных

Для чтения данных из файлов и ввода с клавиатуры используются процедуры Read и Readln. Они отличаются характером считывания данных. В случае чтения из файла, процедура Read считывает данные последовательно, значение за значением, а процедура Readln — построчно. В случае ввода с клавиатуры последовательные процедуры Read могут считывать данные как из одной строки значений, так и из нескольких, а для каждой процедуры Readln значения считываются с новой строки (после нажатия клавиши <Enter>). Проиллюстрируем это на примере.

Предположим необходимо ввести с клавиатуры значения двух переменных (назовем их A и B) типа integer с использованием двух процедур Read:

```
Read(A);
Read(B);
```

Например, рассмотрим следующую последовательность нажатия клавиш для ввода данных:

```
12 100<Enter>
12<Enter>100<Enter>
12<Tab>100<Enter>
```

Другими словами, для последней процедуры Read ввод данных обязательно должен завершаться нажатием клавиши <Enter>, а для первой — не обязательно, так как вторая процедура может продолжать считывать данные из той же строки введенных значений.

Заменим в этом примере процедуры Read на процедуры Readln:

```
Readln(A);
Readln(B);
```

В этом случае при вводе данных, после каждого введенного значения, обязательно необходимо нажимать клавишу <Enter>:

```
12<Enter>100<Enter>
```

Это объясняется тем, что каждая процедура Readln считывает значения, начиная с новой строки.

Процедуры Read и Readln имеет два формата записи. Первый используется при считывании данных из файлов:

```
Read(F, X1, X2, ..., Xn);
```

Где F — это переменная, связанная с файлом, а X1, X2, . . ., Xn — список переменных, для которых считываются данные.

» Подробно работа с файлами рассматривается в главе 10.

Второй формат используется для ввода данных с клавиатуры:

```
Read(X1, X2, . . ., Xn);
```

Где X1, X2, ..., Xn — список переменных, значения которых вводятся пользователем.

Значения переменных должны вводиться в строгом соответствии с их типами. Например, если переменная X1 имеет тип integer, а вводится символьное значение, то будет выдано сообщение об ошибке ввода-вывода.

Процедуры вывода данных

Для вывода данных в файл, на экран монитора или на принтер используются процедуры Write и Writeln. Процедура Writeln отличается от процедуры Write тем, что добавляет к выводимым данным символ перехода на новую строку.

Процедура Write (так же как и процедура Writeln) имеет два формата записи. Первый форма используется для вывода данных в файл:

```
Write(F, X1, X2, ..., Xn);
```

Где F — это переменная, связанная с файлом, а X1, X2, . . ., Xn — список выводимых значений.

Второй формат записи используется для вывода данных на экран монитора:

```
Write(X1, X2, ..., Xn);
```

Форматированный вывод

В процедурах Write и Writeln для каждого выводимого значения можно указать формат вывода.

Форматированный вывод данных невещественных типов

Для форматированного вывода данных невещественных типов используется следующая конструкция:

```
значение:ширина_поля_вывода
```

Если ширина поля вывода превышает количество цифр или символов выводимого значения, тогда это значение дополняется слева пробелами до указанной длины. Например, при вызове следующих процедур:

```
Writeln('Шпак', 'Юрий', 'Алексеевич');
Writeln('Шпак', 'Людмила', 'Владимировна');
```

на экран будут выведены следующие строки

```
ШпакЮрийАлексеевич
ШпакЛюдмилаВладимировна
```

Если же для вывода этих данных применить форматированный вывод:

```
Writeln('Шпак':6, 'Юрий':9, 'Алексеевич':14);
Writeln('Шпак':6, 'Людмила':9, 'Владимировна':14);
```

то фамилия, имя и отчество будут выведены в полях шириной 6, 9 и 11 символов соответственно:

```
  Шпак     Юрий     Алексеевич
  Шпак  Людмила  Владимировна
```

Подобный подход удобно использовать, например, при выводе данных в виде таблицы с выравниванием этих данных в столбцах по правому краю.

Форматированный вывод данных вещественных типов

В случае вывода вещественных данных, ширина поля определяет количество отображаемых разрядов чисел. При этом значение выводится в экспоненциальном формате. Например, в результате вызова следующей процедуры:

```
Writeln(5.2:10, 0.324:11);
```

данные на экран будут выведены в следующем виде:

```
 5.200E+00 3.2400E-01
```

Число 5.2 было отображено в поле шириной 10 символов: _5.200E+00 (символу подчеркивания соответствует пробел). Число 0.324 было отображено в поле шириной 11 символов: _3.2400E-01.

Если необходимо вывести вещественное значение в формате с фиксированной точкой, то используется еще одна, расширенная конструкция:

```
значение:ширина_поля_вывода:ширина_дробной_части
```

В этом случае представленный выше вывов процедуры Writeln можно было бы переписать следующим образом:

```
Writeln(5.2:10:4, 0.324:10:4);
```

В результате данные на экран будут выведены в следующем виде:

```
    5.2000    0.3240
```

Запись и чтение данных из текстового файла

Несмотря на то, что работа с файлами рассматривается в главе 10, здесь кратко будет затронуты следующие впоросы: запись данных в текстовый файл и чтение данных из текстового файла. Чтение и запись данных выполняется при помощи процедур Readln и Writeln.

Для доступа к текстовым файлам используются переменные стандартного типа Text. Создайте в интегрированной среде программирования Turbo Pascal новый файл

и сохраните его под именем TextIO.pas. Введите в него текст, представленный в листинге 4.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 4.1. Программа TextIO.pas

```
program TextIO;
var
  F: Text;
  FirstName, FatherName, LastName: string;
begin
  {Ввод имени}
  Write('Введите имя:');
  Readln(FirstName);
  Write('Введите отчество:');
  Readln(FatherName);
  Write('Введите фамилию:');
  Readln(LastName);

  {Запись в файл}
  Assign(F,'MyName.txt');
  Rewrite(F);
  Writeln(F,FirstName);
  Writeln(F,FatherName);
  Writeln(F,LastName);
  Close(F);

  {Считывание из файла}
  Reset(F);
  Readln(F,FirstName);
  Readln(F,FatherName);
  Readln(F,LastName);
  Close(F);

  {Вывод имени на экран}
  Writeln('Считано имя: ', FirstName);
  Writeln('Считано отчество: ',FatherName);
  Writeln('Считана фамилия: ',LastName);
end.
```

В программе TextIO, кроме типа Text, используется еще один структурированный тип — string. Этому типу соответствуют строки переменной длины от 1 до 255 символов.

▸▸ Подробно тип string рассматривается в главе 8.

Вначале программа сохраняет строки введенные пользователем в переменных FirstName, FatherName и LastName. Затем значение этих переменных записывается в текстовый файл, определяемый переменной F, при помощи процедуры Readln. Для назначения переменной F некоторого файла используется процедура Assign. Для открытия файла F на запись в него данных используется процедура Rewrite.

ПРЕДУПРЕЖДЕНИЕ

Если файл существует, то его старое содержимое будет утеряно, если же такого файла не существует, то он будет создан.

После записи значений переменных FirstName, FatherName и LastName в файл F, этот файл закрывается при помощи процедуры Close.

Затем тестовый файл открывается для чтения при помощи процедуры Reset, и из него в переменные FirstName, FatherName и LastName последовательно считываются три строки, после чего файл опять закрывается. В конце программы осуществляется тестовый вывод на экран содержимого переменных FirstName, FatherName и LastName.

Глава 5

Операторы ветвления

Операторы ветвления выполняют переход к какому-нибудь блоку программного кода на основании проверки некоторого условия. В качестве условия ветвления используется значение логического выражения. В языке Pascal используется два оператора ветвления: условный оператор if и оператор выбора case.

Условный оператор if

Оператор if (в переводе с английского "if означает "если") может быть записан двумя способами:

```
if выражение_условия then блок_операторов;
```

и

```
if выражение_условия then блок_операторов
                      else блок_операторов;
```

Если выражение условия возвращает значение True, тогда выполняется блок операторов, следующих после слова then, (в переводе с английского "then" означает "то") в противном случае выполняется блок операторов, следующий после слова else (в переводе с английского "else" означает "иначе").

ПРИМЕЧАНИЕ

Обратите внимание на то, что после блока операторов, расположенного между словами then и else, разделитель операторов (";") не ставится.

Блоки операторов

Определим, что такое "блок операторов". Блок операторов может состоять из одного оператора (в этом случае он ничем не отличается от обычного оператора) или из нескольких операторов. Блоки операторов, состоящие из нескольких операторов, также называют **составными операторами.** Составной оператор должен начинаться со слова begin и заканчиваться словом end. Эти два слова можно сравнить со "скобками", ограничивающими несколько операторов. Конечно же, один оператор тоже можно ограничить словами begin и end, однако это является излишним и снижает удобочитаемость исходного текста программы.

Если в ветви then или ветви else оператора if предполагается выполнение двух или более операторороров, то они должны быть объединены в блок. То же самое относится и к оператору выбора case, а также для операторов циклов, которые будут рассмотрены в следующей главе. Рассмотрим пример.

```
if A > B then
begin
  A := B;
```

```
  C := 100;
end;
B := 200;
```

Если бы операторы A := B и C := 100 не были объединены в блок, то при выполнении условия A > B был бы выполнен только оператор A ,:= B, а оператор C := 100 выполнялся бы в любом случае. Благодаря использованию слов begin и end, при несоблюдении условия A > в выполнение программы сразу переходит к оператору B := 200.

А теперь рассмотрим пример оператора if с ветвью else:

```
if A > B then
begin
  A := B;
  C := 100;
end else
begin
  A := C;
  C := 200;
end;
B := 200;
```

В этом случае при несоблюдении условия A > B выполняется блок операторов, следующий после слова else. Обратите внимание на то, что между блоком операторов ветви then и словом else разделитель ";" не ставится.

Слова begin и end также полезны при использовании вложенных операторов if, кода возникает неоднозначность принадлежности ветви else. Рассмотрим следующий пример:

```
if A > B then if B > C then A := B else A := C;
```

В данном случае ветвь else будет относиться к ближайшему оператору if с условием B > C. Если же по характеру выполнения программы эта ветвь должна относиться к первому оператору if, тогда вложенный оператор if необходимо выделить как блок:

```
if A > B then
begin
  if B > C then A := B;
end else A := C;
```

Пример использования оператора if

В качестве примера использования условного оператора if создадим программу, преобразовывающую число из шестнадцатеричного формата записи в десятичный формат/ Создайте в интегрированной среде Turbo Pascal новый файл и сохраните его под именем HexToDec.pas. Введите в него текст, представленный в листинге 5.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 5.1. Программа HexToDec.pas

```
program HexToDec;
var
  Hex: char;
```

Окончание листинга 5.1

```
begin
  Write('Введите шестнадцатеричную цифру:');
  Readln(Hex);
  if Hex in ['0'..'9']
  then Writeln(Hex,' = ',Hex)
  else if Hex in ['A'..'F']
       then Writeln(Hex,' = ', Ord(Hex)-55)
       else if Hex in ['a'..'f']
            then Writeln(Hex,' = ', Ord(Hex)-87)
            else Writelr('Цифра введена неверно!');
  Read(Hex);
end.
```

В программе HexToDec вводится некоторый символ, а затем проверяется его значение. Обратите внимание на оператор сравнения in. Этот оператор используется для того, чтобы проверять, входит ли значение первого операнда в набор значений, указанный в качестве второго операнда. В данном примере в первом операторе if проверяется, относится ли введенный символ к набору символов '0'..'9'. Если это условие выполняется, тогда введенная шестнадцатеричная цифра совпадает с десятичной цифрой, если же это условие не выполняется, тогда программа переходит ко второму оператору if. Во втором операторе if проверяется, относится ли введенный символ к набору символов 'A'..'F'. Если это условие выполняется, тогда при помощи функции Ord определяется код ASCII введенного символа, а затем из него вычитается число 55. Например, шестнадцатеричной цифре A соответствует десятичное число 10, а символ 'A' имеет код ASCII 65. Таким образом, для определения десятичного значения достаточно просто вычесть из кода символа число 55. Аналогичные действия выполняются и в том случае, если была введена не прописная, а строчная буква — только в этом случае из кода ASCII введенного символа вычитается число 87. В том случае, если введенный символ не попадает ни в интервал '0'..'9', ни в интервал 'A'..'F', ни в интервал 'a'..'f', то на экран выводится сообщение о том, что шестнадцатеричная цифра была введена неправильно.

Последний оператор Read(Hex) используется для остановки выполнения программы до нажатия клавиши <Enter>.

Оператор выбора case

Оператор case используется для ветвления, когда может существовать более двух возможных результатов выражения условия. В этом случае оператор case делает текст программы более наглядным. Например, в программе HexToDec (листинг 5.1) из предыдущего раздела удобнее использовать именно оператор case, так как проверка значения переменной Hex выполняется трижды. Так же как и оператор if, оператор case имеет две формы записи:

```
case выражение_условия of
  значение1: блок_операторов;
  ...
  значениеN: блок_операторов
end;
```

и

```
case выражение_условия of
  значение1: блок_операторов;
  ...
  значениеN: блок_операторов
else блок_операторов
end;
```

Блок операторов ветви else выполняется в том случае, если результат выражения условия не совпадает ни с одним из значений выбора. Если результат выражения условия не совпадает ни с одним из значений выбора, и ветвь else отсутствует, то выполнение программы продолжается с оператора, следующего за словом end, завершающим оператор case. Если результат выражения условия совпадает с одним из значений выбора, то выполняется только соответствующий блок кода, а затем выполнение программы продолжается с оператора, следующего за конструкцией case-end.

Результат выражения условия должен относиться к целочисленным типам byte, shortint или integer; к логическому типу boolean или символьному типу char. Рассмотрим примеры записи значений выбора оператора case, представленный в листинге 5.2.

Листинг 5.2. Примеры записи значений выбора оператора case

```
case A of
  1: Writeln('A=1');
  2, 4: Writeln('A=2 или A=4');
  3, 5..10: Writeln('A=4 или в интервале от 5 до 10');
  Ord('F'): Writeln('A=70');
end;
case Ch of
  'D': Writeln('Символ D');
  'E', 'F': Writeln('Символ E или F');
  'A'..'C': Writeln('Символ в интервале от A до C');
  Chr(71): Writeln('Символ G');
end;
```

Как видно в листинге 5.2, результат выражения условия может сопоставляться не только с отдельными значениями, но и со списками значений, интервалами и результатами, возвращаемыми функциями.

Примеры использования оператора case

Перепишем созданную в предыдущем разделе программу HexToDec (листинг 5.1), заменив операторы if на оператор case. Для этого откройте в интегрированной среде Turbo Pascal файл HexToDec.pas, выполните команду меню **File | Save As** и сохраните копию этого файла под именем HexToD_2.pas. Измените содержимое файла HexToD_2.pas в соответствии с программой, представленной в листинге 5.3.

Листинг 5.3. Программа HexToD_2.pas

```
program HexToD_2;
var
  Hex: char;
begin
  Write('Введите шестнадцатеричную цифру:');
  Readln(Hex);
```

Окончание листинга 5.3

```
  case Hex of
    '0'..'9': Writeln(Hex,' = ', Hex);
    'A'..'F': Writeln(Hex,' = ', ord(Hex)-55);
    'a'..'f': Writeln(Hex,' = ', ord(Hex)-87) ;
  else Writeln('Цифра введена неверно!');
  end;
  Read(Hex);
end.
```

Очевидно, что такая запись гораздо удобнее для чтения, чем в случае с использованием трех операторов if.

Оператор безусловного перехода goto

В программах подобных HexToDec, в которых существует вероятность ввода пользователем некорректного значения, можно реализовать возврат к началу программы для повторного вызова функции Readln. Это можно сделать при помощи операторов циклов, которые будут рассмотрены в следующей главе, или при помощи оператора безусловного перехода goto.

Оператор goto выполняет произвольный переход к специальной *метке,* расположенной где-то в тексте программы. Метки объявляются перед словом begin программы в разделе label, а затем расставляются в тексте программы, и после их имени ставится двоеточие (":").

Измените программу HexToD_2 (листинг 5.3), в соответствии с программой, представленной в листинге 5.4, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>). Новые строки выделены полужирным шрифтом.

Листинг 5.4. Новая версия программы HexToD_2.pas

```
program HexToD_2;
label
  M1;
var
  Hex: char;
begin
  M1:
  Write('Введите шестнадцатеричную цифру:');
  Readln(Hex);
  case Hex of
    '0'..'9': Writeln(Hex,' = ',Hex);
    'A'..'F': Writeln(Hex,' = ', ord(Hex)-55);
    'a'..'f': Writeln(Hex,' = ', ord(Hex)-87);
  else begin
         Writeln('Цифра введена неверно!');
         goto M1;
       end;
  end;
  Read(Hex);
end.
```

Теперь в случае ввода некорректной шестнадцатеричной цыфры будет выполнен переход в начало программы к метке M1 и весь процесс повторится сначала. При ис-

пользовании операторов goto следует помнить о том, что область действия метки распространяется только в том блоке операторов, в котором она описана (в данном случае — в одной программе). Переход в другие блоки запрещен.

СОВЕТ

Использовать операторы goto в программах следует только в том случае, если поставленную задачу невозможно реализовать никаким другим способом. Безусловные переходы значительно усложняют анализ и исправление программ и могут приводить к возникновению ошибок выполнения, поэтому старайтесь обходиться без оператора goto.

Еще один пример использования оператора case

Напишем еще одну программу, определяющую период суток на основании введенного часа. Создайте в интегрированной среде Turbo Pascal новый файл, назовите его TheDay.pas, и введите в него текст, представленный в листинге. 5.5, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 5.5. Программа TheDay.pas

```
program TheDay;
label
  M1;
var
  Hour: byte;
begin
  M1:
  Write('Введите час от 0 до 24:');
  Readln(Hour);
  if not (Hour in [0..24]) then goto M1;
  case Hour of
    22..24, 0..5: Writeln('Ночь');
    6..9: Writeln('Утро');
    10..13, 15..18: Writeln('Работа');
    14: Writeln('Обед');
    19..21: Writeln('Отдых');
  end;
end.
```

В этой программе показан пример сопоставления целочисленной переменной Hour с различными вариантами значений ветвления оператора case. В данном случае при проверке корректности введенного значения так же использовался оператор goto. В следующей главе будут рассмотрены операторы циклов, позволяющие избежать использования операторов безусловных переходов.

Глава 6

Операторы циклов

Если в программе необходимо многократно повторить ккакую-нибудь последовательность операторов, то в этом случае используются **операторы циклов.** В языке Pascal существует три таких оператора: for, while и repeat. Оператор for используется в тех случаях, когда количество повторов цикла заранее известно, а в остальных случаях используются операторы while и repeat.

Оператор for

В операторе for количество повторов задается при помощи переменной любого простого типа (кроме вещественного), называемой **параметром цикла.** Существует две формы записи оператора for. Первая форма используется в том случае, когда значение параметра цикла при каждом повторе возрастает:

```
for параметр_цикла := начальное_значение to конечное_значение do
блок_операторов;
```

Вторая форма используется в том случае, когда значение параметра цикла при каждом повторе уменьшается:

```
for параметр_цикла := начальное_значение to конечное_значение downto
блок_операторов;
```

Оператор for выполняется до тех пор, пока значение параметра цикла не достигнет конечного значения. Рассмотрим два примера использования оператора for. Создайте в интегрированной среде Turbo Pascal новый файл под именем For_1.pas и введите в него текст, представленный в листинге 6.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 6.1. Программа For_1.pas

```
program for_1;
var
  i, n: integer;
  factorial: longint;
begin
  Write('Введите число:');
  Readln(n);
  factorial := 1;
  for i := 1 to n do factorial := factorial * i;
  Writeln('Факториал ', n, ' = ', factorial);
end.
```

Для вычисления факториала целого числа используется формула 1*2*3*...*n, где n — целое число. В данном примере используется оператор for с увеличением значения

целочисленного параметра цикла i от 1 до n. При этом в каждом повторе цикла значение факториала (переменная factorial) умножается на текущее значение параметра цикла i. Предположим, для переменной n было введено значение 4. В этом случае ход выполнения цикла можно схематически представить в виде таблицы (табл. 6.1).

Таблица 6.1. Схематическое представление хода выполнения цикла из листинга 6.1

Значение параметра цикла i	Значение переменной **factorial** перед повтором цикла	Значение переменной factorial после повтора цикла
1	1	1*1=1
2	1	1*2=2
3	2	2*3=6
4	6	6*4=24

Теперь рассмотрим пример, в котором значение параметра цикла уменьшается при помощи ключевого слова downto. Создайте в среде Turbo Pascal новый файл, сохраните его под именем For_2.pas и введите в него текст, представленный в листинге 6.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 6.2. Программа For_2.pas

```
program for_2;
var
  c: char;
  s: string;
begin
  for c := 'Z' downto 'A' do s := s + c;
  writeln(s);
end.
```

В программе for_2 в качестве параметра цикла выступает переменная с типа char, значение которой уменьшается от 'Z' (код символа 90) до 'A' (код символа 65). В цикле формируется строка (переменная s типа string), представляющая собой набор букв английского алфавита от Z до A. Ход выполнения цикла можно схематически представить в виде таблицы (табл. 6.2).

Таблица 6.2. Схематическое представление хода выполнения цикла из листинга 6.2

Значение параметра цикла c	Значение переменной *s* перед повтором цикла	Значение переменной s после повтора цикла
Z		Z
Y	Z	ZY
X	ZY	ZYX
...	...	...
B	ZYXWVUTSRQPONMLKJIHGFEDC	ZYXWVUTSRQPONMLKJIHGFEDCB
A	ZYXWVUTSRQPONMLKJIHGFEDCB	ZYXWVUTSRQPONMLKJIHGFEDCBA

При использовании операторов for следует придерживаться определенных требований.

- В качестве параметра цикла может использоваться переменная любого простого типа, кроме вещественного типа.
- Начальные и конечные значения параметра цикла должны иметь тип, совместимый с типом самого параметра.
- В теле цикла нельзя явно изменять значение параметра-переменной.
- Если выполнение цикла не было прервано процедурой `break` (рассматривается в конце этой главы), то значение параметра-переменной после завершения оператора `for` будет неопределенным.

Оператор while

Оператор цикла while имеет следующую форму записи:

```
while выражение_условия_повтора do блок_операторов;
```

Блок операторов цикла while повторяется до тех пор, пока результатом логического выражения условия повтора является значение True. Выражения условия вычисляется перед каждым повтором цикла, и выход из цикла (то есть переход к оператору, следующему за оператором while) осуществляется в том случае, если выражение условия имеет значение False.

Рассмотрим пример использования оператора while. Создайте в интегрированной среде Turbo Pascal новый файл под именем `WhileEx.pas` и введите в него текст, представленный в листинге 6.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 6.3. Программа `WhileEx.pas`

```
program WhileEx;
var
  CharCode: integer;
begin
CharCode := 0;
 while CharCode <= 255 do
 begin
   if Chr(CharCode) in ['0'..'9'] then
     Writeln('Код символа "', Chr(CharCode), '" - ', harCode);
   Inc(CharCode);
 end;
end.
```

Эта программа выводит коды символов, соответствующих цифрам. В цикле while перебираются все коды символов в соответствии с таблицей ASCII (коды от 0 до 255). В качестве счетчика цикла используется переменная CharCode типа integer, в которой хранится текущее значение кода символа. Перед началом цикла переменная CharCode имеет значение 0, а выход из цикла происходит в том случае, если значение этой переменной превышает 255, т.е результат выражения CharCode <= 255 равен False.

Для увеличения значения переменной CharCode используется стандартная функция Inc. Эта процедура увеличивает значение переменной, указанной в качестве первого параметра, на число, указанное в качестве второго параметра. Если второй параметр не указан (как в программе WhileEx), то значение первого параметра увеличивается на единицу. Другими словами, оператор `Inc(N)` аналогичен оператору N := N + 1, а, например, оператор `Inc(N,2)` — оператору N := N + 2.

ПРИМЕЧАНИЕ

Для уменьшения значений переменных используется процедура Dec с теми же параметрами, что и процедура Inc.

Проблема "зацикливания"

Иногда при невнимательном использовании операторов while и repeat возникает так называемое "зацикливание" — бесконечное повторение тела цикла. Это может произойти по двум причинам.

1. Переменные, которые являются частью выражения условия цикла, не принимают внутри цикла корректных значений, в результате чего результатом выражения условия всегда будет значение True.
2. Некорректно само выражение условия цикла.

Для того чтобы проиллюстрировать первую причину, можно просто удалить из программы WhileEx оператор `Inc(CharCode)`. В результате переменная `CharCode` всегда будет равна 0, и выражение условия цикла CharCode <= 255 всегда будет возвращать значение True.

Вторая причина "зацикливания" подразумевает, что значение False не может быть результатом выражения условия даже теоретически. Например, если в программе WhileEx заменить объявление типа переменной CharCode с integer на byte, то эта переменная никогда не сможет принять значение, превышающее 255. Как следствие, результатом выражения условия цикла всегда было бы значение True, что привело бы к "зацикливанию".

С учетом вышеизложенного, следует быть очень внимательным к построению выражений условия циклов while и repeat, а также к изменению значений переменных, являющихся частью условия выхода из цикла.

СОВЕТ

Если в процессе выполнения программы произошло "зацикливание", то для прерывания ее работы можно воспользоваться комбинацией клавиш <Ctrl+C> или <Ctrl+Pause(Break)>. При этом, если программа была запущена из интегрированной среды Turbo Pascal, то она будет переведена в режим отладки.

Отладка программ подробно рассматривается в главе 13.

Оператор repeat

Оператор цикла repeat имеет следующую форму записи:

```
repeat блок_операторов until выражение_условия_повтора;
```

Оператор repeat подобен оператору while за исключением двух отличий.

- Условие выхода из цикла проверяется после выполнения тела цикла. Это означает, что блок операторов, расположенный между словами repeat и until выполняется как минимум один раз.
- Выход из цикла repeat происходит в том случае, когда в результате вычисления выражения условия получается значение True.

При помощи оператора repeat удобно организовывать проверку значений, вводимых пользователем. Например, изменим программу `TheDay` (листинг 5.5), которая

рассматривалась в предыдущей главе. Откройте в интегрированной среде Turbo Pascal файл TheDay.pas (или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>)), удалите в ней объявление метки M1:

```
label
  M1;
```

а также перепишите фрагмент

```
M1:
Write('Введите час от 0 до 24:');
Readln(Hour);
if not (Hour in [0..24]) then goto M1;
```

в новом виде с использованием цикла repeat:

```
repeat
  Write('Введите час от 0 до 24:');
  Readln(Hour);
until Hour in [0..24];
```

Представленный цикл будет выполняться до тех пор, пока пользователь не введет в переменную Hour значение, попадающее в диапазон от 0 до 24. Таким образом, при помощи циклических структур можно отказаться от использования в программах меток и операторов безусловного перехода goto.

ПРИМЕЧАНИЕ

Обратите внимание на то, что блок операторов цикла repeat ограничивать словами begin и end необязательно.

Процедуры управления циклом

Для управления циклами используется две дополнительные стандартные процедуры языка Pascal: Break и Continue. Процедура Break принудительно прерывает цикл, после чего выполнение программы продолжается с оператора, следующего за оператором for, while или repeat. Процедура Continue выполняет принудительный переход к следующему повтору цикла.

Рассмотрим работу этих функций на примере. Создайте в интегрированной среде Turbo Pascal новый файл, присвойте ему имя BrConEx.pas, и введите в него текст, представленный в листинге 6.4, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 6.4. Программа BrConEx.pas

```
program BrConEx;
var
  i: integer;
begin
  for i := 1 to 20 do
  begin
    if (i mod 2) = 0 {Проверка на четность}
    then Continue
    else if i > 10 then Break;
    Writeln(i);
  end;
end.
```

Эта программа отображает на экране нечетные числа в диапазоне от 1 до 10. В цикле от 1 до 20 проверяется четность каждого значения параметра-переменной i. Если остаток от деления на 2 равен нулю, то это означает, что текущее значение четное, и при помощи процедуры Continue выполняется переход к следующему повтору цикла, а оператор `Writeln(i)` не выполняется. Если текущее значение переменной i нечетное, тогда оно сравнивается с числом 10. Если значение i больше 10, тогда цикл for прерывается при помощи процедуры break, в противном же случае значение i отображается на экране при помощи процедуры Writeln.

Глава 7

Процедуры и функции

Прежде, чем приступать к изучению процедур и функций, следует рассмотреть такое обобщенное понятие как "подпрограмма".

Подпрограмма — это отдельный фрагмент программы, представляющий собой самостоятельный программный блок. В подпрограммах обычно размещаются часто используемые совокупности операторов. Для идентификации отдельных подпрограмм им присваивают имена, при помощи которых к ним можно обращаться из различных мест программы. Согласно современным подходам к программированию, любой законченный функциональный фрагмент желательно оформлять в виде подпрограммы, так как это уменьшает количество ошибок и упрощает анализ исходного текста программы.

В языке Pascal механизм подпрограмм реализован при помощи *процедур* и *функций*.

Процедура — это подпрограмма, которую можно вызывать по имени для выполнения определенных в ней действий. В предыдущих главах уже рассматривались такие процедуры как Readln, Writeln, Break и др.

Функция аналогична процедуре, однако отличает от нее тем, что возвращает в точку вызова некоторое значение. Благодаря этому, функции, в отличие от процедур, можно использовать как составные части выражений. В предыдущих главах уже рассматривались такие функции, как Chr и Ord.

Передача данных в подпрограмму выполняется при помощи специальных переменных — **параметров.** Параметры, определенные в заголовке подпрограммы, называются **формальными.** Выражения, задающие конкретные значения при обращении к подпрограмме, называются **фактическими** параметрами. При обращении к подпрограмме ее формальные параметры замещаются фактическими параметрами.

В списке формальных параметров при объявлении процедуры или функции должны быть указаны их имена и типы. Имя параметра отделяется от типа двоеточием (":"), а параметры друг от друга — точкой с запятой (";"). Имена параметров одного типа могут объединяться в подсписки, в которых мена отделяются друг от друга запятой.

Между формальными и фактическими параметрами должно быть полное соответствие.

- Должно совпадать их количество.
- Должен совпадать порядок их следования.
- Тип каждого фактического параметра должен совпадать с типом соответствующего формального параметра.

Например, если заголовок процедуры выглядит так:

```
procedure Proc(a, b: integer; c: char);
```

то вызов этой функции может выглядеть следующим образом:

```
Num1 := 1;
Num2 := 2;
Ch := 'A';
```

```
Proc(Num1, Num2, Ch) ;
Proc(12, Num1 + Num2, 'D');
Proc(0, Ord('A'), Chr(Num1));
```

В то же время, представленные ниже вызовы являются ошибочными:

```
Proc(Num1, Num2); {не совпадает количество параметров}
Proc(Ch, Num1, Num2); {не совпадает порядок следования}
Proc(12.3, Num2, Ch); {не совпадает тип 1-го параметра}
```

Также следует отметить, что все процедуры и функции языка Pascal делятся на две категории: *стандартные* и *пользовательские*.

Стандартные процедуры и функции

Стандартные подпрограммы являются составной частью языка программирования и вызываются по строго определенному имени. Все стандартные процедуры и функции реализованы в специальных *библиотечных модулях,* которые подключаются к программе при помощи оператора uses, размещаемого сразу после заголовка программы.

» Подробнее модули рассматриваются в главе 14.

Например, для того чтобы можно было обращаться к процедурам и функциям стандартного библиотечного модуля Crt, в исходный текст программы должна быть включена следующая строка:

```
program SomeName;
uses Crt;
...
```

К стандартным модулям относятся следующие библиотечные модули.

- System — модуль, содержащий подпрограммы, обеспечивающие работу всей системы Turbo Pascal. Этот модуль подключается к программам автоматически, и потому его имя в разделе uses явно не указывается.
- Crt — модуль, содержащий средства управления дисплеем и клавиатурой.
- Dos — модуль, содержащий средства, позволяющие реализовать различные функции операционной системы MS-DOS.
- Graph — модуль, содержащий подпрограммы для работы с адаптерами мониторов CGA, EGA, VGA, HERC, IBM 3270, MCGA и ATT6330.
- Graph3 — модуль, содержащий стандартные подпрограммы, используемые при работе с графикой.
- Overlay — модуль; содержащий средства организации оверлейных программ.
- Printer — модуль, содержащий средства организации печати.
- Turbo3 — модуль, обеспечивающий совместимость с версией Turbo Pascal 3.0.
- Turbo Vision — модуль библиотеки объектно-ориентированных подпрограмм для разработки пользовательских интерфейсов.

В этой главе основное внимание уделено пользовательским процедурам и функциям.

» Вопросы разработки оверлейных программ, использования графики и функций DOS рассматриваются в главах частей 3 и 4.

» Использованию средств Turbo Vision посвящена глава 18.

» Полный перечень процедур и функций стандартных модулей можно найти в приложении Ж.

Пользовательские процедуры и функции

Пользовательские подпрограммы создаются и именуются самим пользователем. На время выполнения таких подпрограмм, вызываемых по их имени, работа главной программы приостанавливается. По завершении обработки данных подпрограммой продолжается выполнение главной программы. Если при этом в качестве подпрограммы выступала функция, то в главную программу возвращается результат выполнения подпрограммы.

Пользовательские процедуры и функции объявляются после объявления переменных, перед первым словом begin программы или другой процедуры или функции. Если пользовательская процедура или функция объявляется в отдельном модуле, тогда для обращения к ней необходимо указать ссылку на соответствующий модуль в разделе uses.

Процедуры

Для объявления процедуры используется следующая конструкция:

```
procedure имя_процедуры (параметры) ;
  const объявления_констант;
  type объявления_типов;
  var объявления_переменных;
  объявления_других_процедур_и_функций
begin
  операторы
end;
```

Константы, типы, переменные, процедуры и функции, объявленные внутри процедуры, доступны только внутри этой процедуры. В то же время, любые глобальные (то есть объявленные в заголовке программы) константы, типы, переменные, процедуры и функции доступны в любой точке программы, включая все подпрограммы. Если одно и то же имя используется и для глобального и для локального элемента, то внутри подпрограммы используется только локальный элемент, а глобальный — как бы "скрывается". Тем не менее, не стоит слишком увлекаться одинаковыми именами глобальных и локальных элементов, так как это может привести к возникновению логических ошибок, которые бывает тяжело обнаружить.

Для вызова процедуры необходимо указать ее имя, а затем — в круглых скобках перечень фактических параметров, соответствующих набору формальных параметров. Например, если объявлена процедура

```
procedure ShowMessage(s: string; len: integer);
begin
  Writeln(s:len);
end;
```

то для ее вызова должен использоваться оператор следующего вида:

```
(ShowMessage('Hello!',10);
```

Если процедура объявлена без списка формальных параметров, например:

```
procedure ShowHello;
begin
  Writeln('Hello!');
end;
```

тогда она вызывается только по имени, без использования круглых скобок:

```
ShowHello;
```

В качестве примера стандартных процедур, вызываемых без параметров, можно назвать, уже рассмотренные в предыдуще главе, процедуры Break и Continue.

Рассмотрим вызов процедуры на примере. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем ProcEx.pas и введите в него текст, представленный в листинге 7.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 7.1. Программа ProcEx.pas

```
program ProcEx;

procedure SortABC(AscOrder: boolean);
var
  c: char;
begin
  if AscOrder
  then for c := 'A' to 'Z' do Write(c)
  else for c := 'Z' downto 'A' do Write(c);
  Writeln('');
end;

begin
  SortABC(True);
  SortABC(False);
end.
```

В программе ProcEx объявлена процедура SortABC с одним формальным параметром AscOrder типа boolean. Если этому параметру передается значение True, тогда на экране отображается английский алфавит, отсортированный в прямом направлении, в противном случае алфавит будет отсортирован в обратном направлении. В теле программы расположены только два вызова процедуры SortABC. При первом вызове в процедуру передается фактический параметр True, а при втором — False. В результате выполнения программы ProcEx на экран будут выведены две строки, как кпоказано на рис. 7.1.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

Рис. 7.1. Результат выполнения программы ProcEx.pas

Функции

Для объявления функции используется следующая конструкция:

```
procedure имя_функции (параметры) : тип_результата;
  const объявления_констант;
  type объявления_типов;
  var объявления_переменных;
  объявления_других_процедур_и_функций
begin
  опера торы
end;
```

Как видите, объявление функции отличается от объявления процедуры только тем, что в ее заголовке указывается тип возвращаемого результата. В теле функции должен быть как минимум один оператор, в котором имени функции присваивается возвращаемое значение. В точку вызова функции возвращается результат последнего присваивания.

Обращение к функции ничем не отличается от обращения к процедуре. Единственное отличие состоит в том, что вызов функции может быть частью выражения, так как она возвращает некоторое значение.

Рассмотрим вызов функций на примере программы, вычисляющей десятичный эквивалент любого шестнадцатеричного числа. Напоминаем, что в главе 5 уже рассматривалась программа преобразования числа из шестнадцатеричного формата записи в его десятичное представление. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем HexToD_3.pas и введите в него текст, представленный в листинге 7.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 7.2. Программа HexToD_3.pas

```
program HexToD_3;
var
  HexNum: string;
  i: integer;
  DecDig, DecNum: longint;
  IsError: boolean;

{Функция преобразования шестнадцатеричной цифры
 в десячтичное число по формуле (цифра)*(16^позиция)}
function HexToDecimal(HexDig: char; pow: byte): longint;
  {Функция возведения в степень}
  function Power(Num, Pow: byte): longint;
  var
    i: integer;
    Res: longint;
  begin
    Res := 1;
    for i := 1 to Pow do Res := Res * Num;
    Power := Res;
  end;

begin
  case HexDig of
    '0'..'9': HexToDecimal := (Ord(HexDig) - 48) * Power(16, pow);
    'A'..'F': HexToDecimal := (Ord(HexDig) - 55) * Power(16, pow);
    'a'..'f': HexToDecimal := (Ord(HexDig) - 87) * Power(16, pow);
  else HexToDecimal := -1; {Некорректная цифра}
  end;
end;

begin
  repeat
    Write('Введите шестнадцатеричное число:');
    Readln(HexNum);
    DecNum := 0;
    isError := False; {Флажок, указывающий на ошибку}
```

Окончание листинга 7.2

```
    for i := 1 to Length(HexNum) do {Просматриваем
                                     шестнадцатеричное число}
    begin
      {Определяем десятичное число для
       шестнадцатеричной цифры в текущей позиции}
      DecDig:= HexToDecimal(HexNum[i],length(HexNum)-i);
      if DecDig >= 0 (Если введено корректное число,}
      then inc(DecNum, DecDig) (Увеличиваем текущее значение десятичного
      числа на число, соответствующее текущей шестнадцатеричной цифре}
      else begin
        {Если введено некорректное число,}
        isError := True; {Устанавливаем флажок,
                          указывающий на наличие ошибки}
        Writeln('Число введено некорректно!');
        break; (Принудительно завершаем цикл for}
      end;
    end;
  until not isError; (Если ошибки ввода не было,
                      то выходим из цикла repeat}
  Writeln('Десятичное число = ', DecNum);
end.
```

Программа HexToD_3 несколько сложнее тех, которые представлены в этой книге выше, поэтому рассмотрим ее поподробнее. Начнем с объявления функции HexToDecimal. Эта функция возвращает десятичное представление шестнадцатеричной цифры в определенной позиции шестнадцатеричного числа. Например, если было введено шестнадцатеричное число A1FD, то цифре A соответствует десятичное число 40960 ($10*16^3$), цифре 1 — десятичное число 256 ($1*16^2$), цифре F — десятичное число 240 ($15*16^1$), а цифре D — десятичное число 13 ($13*16^0$). Другими словами, десятичное число, соответствующее шестнадцатеричной цифре, определяется как произведение десятичного представления этой цифры и числа 16, возведенного в степень, соответствующую позиции цифры в шестнадцатеричном числе. Позиция считается от нуля, с права налево.

Представленные выше вычисления выполняет функция HexToDecimal. Шестнадцатеричная цифра передается в качестве параметра HexDig типа char, а позиция этой цифры в шестнадцатеричном числе — в качестве параметра pow типа byte. Для возведения 16 в степень pow была создана функция Power, локальная по отношению к функции HexToDecimal. Это означает, что функция Power доступна только из функции HexToDecimal. Функция Power просто выполняет умножение самого на себя числа, переданного в качестве параметра Num. Количество умножений числа самого на себя передается в функцию в качестве параметра Pow.

Итак, вернемся к функции HexToDecimal. В ней используется оператор case, уже рассмотренный в главе 5. При помощи оператора case определяется корректность введенной шестнадцатеричной цифры. Если шестнадцатеричная цифра является корректной, то главной программе возвращается ее десятичный эквивалент, в противном случае возвращается значение -1.

Перейдем к телу самой программы. Ввод шестнадцатеричного числа и все вычисления организованы внутри цикла repeat. При этом условием выхода из цикла является отсутствие ошибок в веденном шестнадцатеричном числе, что определяется при помощи логической переменой isError. Просмотр всех цифр шестнадцатеричного

числа организован при помощи цикла for, где конечное значение параметра-переменной определяется при помощи стандартной функции length. Эта функция принимает значение типа string и возвращает его длину в символах.

Перед началом анализа введенного числа переменная isError имеет значение False. Если во время анализа числа ошибок обнаружено не было, то значение переменной isError не изменяется, и по окончании вычислений происходит выход из цикла repeat. Если в процессе циклического вызова функции HexToDecimal встретится некорректная шестнадцатеричная цифра (HexToDecimal = -1), то переменной isError будет присвоено значение True, цикл for будет прерван при помощи процедуры break, и цикл repeat повторится сначала.

Для каждой шестнадцатеричной цифры соответствующее десятичное число, возвращаемое функцией HexToDecimal, сохраняется в переменной DecDig, значение которой затем прибавляется в цикле к значению переменной DecNum. При выходе из цикла repeat переменная DecNum будет содержать десятичный эквивалент введенного пользователем шестнадцатеричного числа.

Параметры процедур и функций

В процедурах и функциях могут использоваться:

- параметры, передаваемые по значению;
- параметры, передаваемые по ссылке;
- параметры-процедуры;
- параметры-функции;
- нетипизированные параметры.

Рассмотрим каждый из перечисленных случаев в отдельности.

Параметры, передаваемые по значению

Параметры, передаваемые по значению, используются только для передачи данных из основной программы в процедуру или функцию. Такие параметры использовались в рассмотренных выше примерах, например:

```
procedure SortABC(AscOrder: boolean);
function HexToDecimal(HexDig: char; pow: byte): longint;
function Power(Num, Pow: byte): longint;
```

Все эти параметры передаются по значению. В этом случае фактическими параметрами могут быть произвольные выражения. Например, формальному параметру AscOrder процедуры SortABC соответствует фактическое значение True или False, а формальному параметру pow функции HexToDecimal — выражение length(HexNum)-i.

В процедуру или функцию передается копия значения каждого формального параметра, даже если в их роли выступают переменные или константы. Это означает, что в процессе выполнения подпрограммы, значения этих параметров могут изменяться как угодно, но это никак не повлияет на значение соответствующих фактических параметров-переменных вызывающей программы — изменяются не они сами, а только их копия.

Параметры, передаваемые по ссылке

Параметры, передаваемые по ссылке, используются для хранения результатов выполнения процедуры или функции. Они перечисляются после зарезервированного слова var с обязательным указанием их типа. Каждому формальному параметру, объ-

явленного при помощи слова var, может соответствовать только фактический параметр в виде переменной того же типа.

Если формальный параметр передается по ссылке, то при вызове подпрограммы передается не копия, а сама переменная (или, говоря иначе, — ссылка на нее). В результате изменение значения параметра в подпрограмме приведет к изменению соответствующей переменной в вызывающей программе.

Рассмотрим программу, представленную в листинге 7.3. (Файл VarParam.pas можно открыть при помощи команды **File | Open** (<F3>) с прилагаемой к книге дискеты).

Листинг 7.3. Программа VarParam.pas

```
program VarParam;

var
  vMod, vDiv: integer;

function Divide(a,b: integer; var rMod: integer): integer;
begin
rMod := a mod b;
 divide := a div b;
end;

begin
  vDiv := Divide(10, 3, vMod);
  Writeln('vDiv=',vDiv);
  Writeln('vMod=',vMod);
end.
```

В функцию Divide программы VarParam параметры а и Ь передаются по значению, а параметр rMod — по ссылке. Таким образом, эта функция возвращает два значения: результат деления и остаток от деления в качестве значения параметра rMod. При вызове функции Divide из программы параметру rMod соответствует переменная vMod. После вызова функции эта переменная будет содержать значение 1 (10 mod 3).

Нетипизированные параметры

Нетепизированные параметры так же передаются по ссылке, однако их отличие состоит в том, что для них не указывается тип. В этом случае в подпрограмму можно передавать параметры-переменные любого типа, а на программиста возлагается ответственность за корректное **приведение типов** внутри подпрограммы. Для приведения нетипизированного параметра к какому-либо типу используется следующая конструкция:

```
имя_типа(имя_параметра)
```

В качестве примера, в представленной выше программе VarParam (см. листинг 7.3), сделаем параметр rMod нетипизированным:

```
function Divide(a,b: integer; var rMod):
```

В этом случае для того, чтобы присвоить параметру rMod результат выражения a mod b необходимо выполнить приведение к типу integer:

```
integer(rMod) := a mod b;
```

С таким же успехом параметр rMod можно было бы привести и к типу char, например:

```
char(rMod) := 'A';
```

В этом случае в вызывающей программе переменной vMod будет присвоено значение кода символа 'A' (то есть 65).

Параметры-процедуры и параметры-функции

В языке Pascal процедуры и функции могут выступать в качестве параметров. Это возможно благодаря использованию *процедурных типов*.

Процедурный тип — это шаблон объявления функции или процедуры. Например, можно объявить следующий процедурный тип:

```
type
  Operation = function(A,B: integer): integer;
```

Затем можно создать две функции, объявление которых соответствует шаблону, приведенному в листинге 7.4.

Листинг 7.4. Пример объявления функций

```
{$F+}
function Addition(A,B: integer): integer;
begin
  Addition := A + B;
end;
function Subtraction(A,B: integer): integer;
begin
  Subtraction := A - B;
end;
{$F-}
```

ПРИМЕЧАНИЕ

{$F+} — это директива (указание) для компилятора использовать дальний (far) тип вызова подпрограмм. В случае объявления процедур или функций, которые будут переданы в качестве параметров-процедур или параметров-функций, использование директивы {$F+} является обязательным условием.

Подробнее директивы компилятора рассматриваются в главе 13.

Далее в программе можно объявить еще одну функцию, например:

```
function SomeOp(A,B: integer; Op: Operation): integer;
begin
  SomeOp := Op(A, B);
end;
```

Эта функция возвращает результат в зависимости о того, как функция будет передана параметру Op. Если будет передана функция Addition, то будет возвращен результат сложения A и B, если же будет передана функция Subtraction, то будет возвращен результат вычитания B из A.

Теперь в теле программы можно вызвать функцию SomeOp:

```
Writeln('10 + 5', SomeOp(10,5,Addition);
Writeln('10 - 5', SomeOp(10,5,Subtraction);
```

В результате выполнения этого фрагмента программы на экране будет отображено две строки:

```
10 + 5 = 15
10 - 5 = 5
```

В заключение этого раздела следует отметить, что процедуры и функции, которые используются в качестве параметров, не могут быть стандартными и вложенными, а также не могут обозначаться как inline и interrupt.

» Подробнее о функциях этого вида читайте ниже в этой главе в разделе "Внешние и опережающие объявления процедур и функций".

Рекурсия

Рекурсия — это способность программы вызывать саму себя. Классическим примером использования рекурсии является вычисление факториала целого числа. В предыдущей главе уже рассматривался пример вычисления факториала с использованием цикла for. Для решения этой же задачи применим другую методику.

Создайте в интегрированной среде Turbo Pascal новый файл, назовите его FactorEx.pas и введите в него текст, представленный в листинге 7.5, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 7.5. Программа FactorEx.pas

```
program FactorEx;
var
  Num: integer;

  function Factorial(N: integer): longint;
  begin
    if N = 1
    then Factorial := 1
    else Factorial := N * Factorial(N - 1);
  end;

begin
  Write('Введите число:');
  Readln(Num);
  Writeln('Факториал числа ',Num,' = ', Factorial(Num));
end.
```

Предположим, что было введено значение Num=4. В этом случае функция Factorial будет вызывать сама себя 3 раза (N - 1). В последний, третий рекурсивный вызов значение параметра N будет равно 1 и функция Factorial(1) вернет в вызвавшую ее функцию Factorial(2) значение 1. Функция Factorial(2) умножит это значение на 2 и возвратит результат (число 2) в функцию Factortial(3). В свою очередь, функция Factorial(3) умножит это значение на 3 и возвратит результат (число 6) в функцию Factorial(4). Функция Factorial(4), расположенная "на вершине" рекурсивной пирамиды, умножит это значение на 4 и в результате в вызывающую программу будет возвращено значение 24.

Внешние и опережающие объявления процедур и функций

В языке Pascal используются формы объявления процедур и функций, отличные от обычных. Рассмотрим два из них: внешние и опережающие объявления.

Объявление с директивой external

Пример подобного объявления:

```
procedure SetMode(Mode: Word); external; {$L CURSOR.OBJ}
```

Объявление с директивой external называется **внешним.** Подобные объявления позволяют связывать отдельно скомпилированные процедуры и функции, написанные, например, на языке ассемблера (» язык ассемблера рассматривается в главе 15). С помощью директивы компилятора {$L} подобные внешние подпрограммы можно связывать с программами или модулями, написанными на языке Pascal. Например, в представленном выше объявлении, процедура SetMode, реализованная в файле cursor.obj, объявляется в программе как внешняя.

Объявление с директивой forward

По правилам языка Pascal подпрограмма должна быть обязательно объявлена до своего вызова. Тем не менее, бывают случаи, когда одна из подпрограмм вызывает другую подпрограмму, которая в свою очередь вызывает первую подпрограмму. В такой ситуации используется **опережающее объявление** при помощи директивы forward. При объявлении подпрограммы указывается только ее заголовок с набором параметров и директивой forward, а ее тело создается далее в тексте программы без повторения описания списка формальных параметров.

Пример опережающего объявления процедуры:

```
procedure Proc(A, B: integer); forward;
```

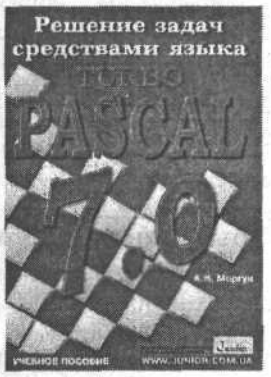
и реализации ее тела далее в тексте программы:

```
procedure Proc;
begin
  ...
end;
```

Если вы только начинаете изучать программирование и чувствуете, что книга, которую вы держите в руках, пока еще непонятна вам, попробуйте начать изучение языка программирования Pascal по оригинальной методике Александра Николаевича Моргуна, которую предлагает вам издательство "Юниор" в книге

Решение задач средствами языка Turbo Pascal 7.0

Моргун Александр Николаевич



ISBN 966-7323-22-6
Объем: 216 с., ил.
Формат: 170x240 мм (70x100/16)
Брошюра
Примеры по Internet

Тема. В этой книге очень подробно и последовательно изложены основы алгоритмизации и программирования на языке Паскаль. Обучение программированию, в отличие от традиционных подходов, построено по принципу "от задачи к программе". Автор показывает, как целесообразно строить программы, начиная с решения простейших задач. Средства языка Паскаль вводятся в сферу их применения по мере появления необходимости в них для решения все усложняющихся задач

Читательская аудитория. Книга предназначена, в первую очередь, для учащихся средней школы и студентов начальных курсов ВУЗов, а также для всех, кто желает научиться правильному программированию, не ограничиваясь просто записью операторов на Паскале. Книга может быть полезна особенно тем студентам, которым трудно преодолеть вузовские барьеры на базе существующего школьного уровня подготовки по информатике. Никаких специальных знаний при изучении материала книги не требуется

Автор. Моргун Александр Николаевич — кандидат технических наук, доцент, заведующий кафедрой "Информатики и современных информационных технологий" Черкасского государственного университета. В сферу его научных интересов входит не только методика преподавания информатики, но и целый ряд вопросов, связанных с автоматизированными системами обработки информации и управления. Преподавательский стаж более 17 лет. Моргун А.М. поддерживает постоянную связь со школой. Он ведет работу с учащимися физико-математического лицея, регулярно приглашается для проведения занятий в областном институте последипломного образования педагогических работников, является председателем жюри школьных областных олимпиад по информатике. За последние три года им создано около десяти учебно-методических пособий для студентов и школьников на базе преподаваемых им дисциплин, в том числе: "Основы информатики и вычислительной техники", "Информатика и программирование", "Школьный курс информатики и методика его преподавания", "Высшая математика", "Теория информации и кодирования", "Теория защиты данных в информационных системах" и т.д.

Содержание

ЧАСТЬ II
СТРУКТУРИРОВАННЫЕ типы ДАННЫХ

Все простые данные, которые рассматривались в предыдущих главах, имеют два характерных признака: неделимость и упорядоченность их значений. В отличие от них, структурированные типы данных определяют множества сложных значений с одним общим именем. Это означает, что каждая переменная или константа структурированного типа всегда имеет несколько компонентов. Можно условно сказать, что структурированные типы образуются из нескольких простых типов. Например, тип string — это совокупность значений простого типа char.

Как уже упоминалось в главе 3, существуют следующие структурированные типы языка Pascal: строки, массивы, множества, записи, файлы, указатели и объекты. Напоминаем, что структурированные типы данных определяют упорядоченную совокупность значений одного или нескольких простых типов.

В этой части будет рассмотрены каждый из перечисленных выше типов:

- в главе 8 — строки и массивы;
- в главе 9 — записи и множества;
- в главе 10 — файлы;
- в главе 11 — указатели;
- в главе 12 — объекты.

Глава 8

Строки и массивы

Наверное, самыми популярными структурированными типами являются *строки* и *массивы,* без которых не обходится ни одна более-менее сложная программа. Строковый тип `string` кратко рассматривается в главе 4 в разделе "Запись и чтение данных из текстового файла", а также в предыдущей главе в разделе "Функции". Рассмотрим его более подробно, а во второй части главы перейдем к изучению массивов.

Строки

Строка — это последовательность значений типа char длиной от 0 до 255 символов. При использовании в выражениях, строки выделяются апострофами. Для определения данных строкового типа используется идентификатор `string`, за которым следует, заключенное в квадратные скобки, значение максимально допустимой длины строки. Если это значение не указано, то по умолчанию максимальная длина строки равна 255 символам.

Примеры объявления и использования строковых констант и переменных:

```
...
const
  MyS = 'Первая строка';
var
  s1: string;       {Строка длиной 255 символов}
  s2 : string[20];  {Строка длиной 20 символов}
  s3: string[100]; {Строка длиной 100 символов}
...
begin
  s1 := MyS;
  s2 := 'Вторая строка';
  s3 := s1 + ' + ' + s2; {'Первая строка + вторая строка'};
...
```

Для хранения строки отводится на один байт больше, чем указанная длина строки. Этот дополнительный байт находится в начале строки (то есть является нулевым байтом) и предназначен для хранения ее длины. В программах к каждому символу строки можно обращаться отдельно. Например:

```
var
  s1, s2: string;
  len: byte;
...
s1[1] := 'A';       {Первым символом в s1 становится 'A'}
s2 := s1[1];        {Строка s2 = 'A'}
s2[2]:=s1[1];       {Теперь строка s2 = 'AA'}
len := Ord(s2[0]); {Значение len = 2 — длина строки s2}
```

Если длина строкового значения, расположенного справа от оператора присваивания, превышает максимально допустимую длину переменной, указанной слева от оператора присваивания, то лишние символы отбрасываются. Например:

```
var
  s1, s2: string[5];
...
s1 := 'ABC';
s2 := s1 + 'DEF';  {В результате s2 = 'ABCDE'}
```

К строкам можно применять оператор конкатенации (соединения) +, а также операторы сравнения =, о, >, <, >=, <=. Сравнение строк выполняется посимвольно слева направо до первого несовпадающего символа. Большей считается та строка, в которой больше код первого несовпадающего символа в соответствии с таблицей ASCII (⏩ все коды символов ASCII перечислены в приложении Е). Например, выражение `'ABBA' = 'ABBA'` вернет значение True, так как все символы этих двух строк совпадают. В то же время, выражение `'ABBA' < 'AABB'` вернет значение False, так как код первого несовпадающего символа первой строки (`'B'`) больше кода соответствующего символа второй строки (`'A'`), а это означает, что строка `'ABBA'` больше строки `'AABB'`.

⏩ При работе со строками используется ряд стандартных процедур и функций, подробное описание которых можно найти в приложении Ж.

Массивы

Массив — это совокупность фиксированного числа элементов одинакового типа. Например, можно сказать, что строка — это массив элементов типа char. Элементы массива могут иметь любой тип, включая структурированный. Массивы бывают *одномерными* и *многомерными*. **Многомерными** называются массивы, элементы которых — это также массивы.

Каждому элементу массива соответствует совокупность номеров (индексов), определяющих его положение в общей последовательности. **Индексы** — это выражения любого простого типа, кроме вещественного типа. Количество индексов, необходимых для определения положения каждого элемента массива, совпадает с его размерностью. Так, в одномерных массивах, для доступа к его элементам используется один индекс, в двухмерных — два индекса, в трехмерных — три и т.д.

Массив определяют при помощи следующей конструкции:

```
array[тип_индекса] of имя_типа;
```

Например, переменную А можно объявить как одномерный массив целых чисел, состоящий из 10 элементов, двумя способами:

```
type
  MyArr = array[1..10] of integer;
var
  A: MyArr;
```

или просто

```
var
  A: array[1..10] of integer;
```

Правила доступа к элементам одномерных массивов такие же, как и при доступе к элементам строк:

```
A[1] := 2;
A[2] := A[1] + 3;
A[3] := A[2] - A[1];
```

Многомерные массивы также можно объявлять двумя способами. Рассмотрим, например, объявление двухмерного массива B:

```
type
  ArrA: array[1..10] of integer;
  ArrB: array[1..5] of ArrA;
var
  B: ArrB;
```

То же самое можно сделать проще:

```
var
  B: array[1..5,1..10] of integer;
```

Двухмерный массив — это аналог таблицы или матрицы, поэтому представленное выше объявление массива B можно пояснить следующим образом: "Объявлен массив целых чисел B, состоящий из 5 строк и 10 столбцов". В этом случае доступ к элементам массива B выполняется при помощи двух индексов:

```
B[1,1] := 3;
B[1,2] := B[1,1] * 5;
B[5,10] := B[1,2] + B[1,1];
```

Подобным же образом определяются массивы с большей размерностью:

```
C: array[1..2,1..5,1..10] of integer; {трехмерный}
D: array[1..2,1..2,1..3,1..4] of char; {четырехмерный}
```

Элементы многомерных массивов располагаются в памяти последовательно. Другими словами, схема размещения элементов представленного выше двухмерного массива B будет выглядеть так:

```
B[1,1], B[1,2], ... B[1,10], B[2,1], B[2,2] ... B[5,10]
```

При написании программ необходимо следить за тем, чтобы значения индексов не превышали границ, указанных при объявлении массива, так как выход индекса за границы массива приводит к сбою в работе программы. Контроль значений индексов массивов можно организовать при помощи директивы компилятора {$R+}, которая приводит к проверке всех индексных выражений на соответствие их значений диапазону индекса.

» Директивы компилятора рассматриваются в главе 13.

Примеры использования строк и массивов

Рассмотрим два примера использования строк и массивов. В первом из них выполняется отображение введенных пользователем строк в отсортированном порядке. В этом примере непосредственная сортировка введенного массива не используется, а формируется дополнительный массив, в который сохраняются строки в порядке их возрастания.

» Различные методы сортировки массивов рассматриваются в главе 19.

Создайте в интегрированной среде Turbo Pascal файл с именем SortStrs.pas и введите в него текст, представленный в листинге 8.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 8.1. Программа SortStrs.pas

```
program SortStrs;
var
  ArrOfS: array[1..5] of string;
  ArrOfR: array[1..5] of string;
  i, j, P: byte;
  s: string;
begin
  for i := 1 to 5 do
  begin
    Write('Введите ',i,'-ю строку: ');
    Readln(ArrOfS[i]);
  end;
  for i := 5 downto 1 do
  begin
    s:='';
    for j := 1 to 5 do
      if ArrOfS[j] > s then
      begin
        s := ArrOfS[j]; {Определяем "наибольшую" строку}
        p := j;         {Запоминаем ее позицию в массиве}
      end;
    ArrOfS[p] := ''; {Очищаем элемент массива для "наибольшей" строки}
    ArrOfR[i] := s;  {Сохраняем "наибольшую" строку в
                      результирующий массив}
  end;
  for i := 1 to 5 do writeln(ArrOfR[i]);
end.
```

В программе SortStrs вначале вводятся значения в массив строк ArrOfS. Затем этот массив просматривается пять раз, и при каждом проходе в строковой переменной s сохраняется "наибольшая" строка. При этом позиция этой строки в массиве ArrOfS сохраняется в переменной p. В конце каждого из пяти проходов элемент массива ArrOfS с индексом p ("наибольшая" строка) очищается, а его значение, хранимое в переменной s, сохраняется в массиве ArrOfR. В результате, при следующем просмотре массива та строка, которая была "наибольшей" в предыдущий раз, будет пропущена.

Рассмотрим еще один пример: вычисление среднего арифметического значений, хранимых в строках двухмерного массива. Создайте в интегрированной среде Turbo Pascal файл с именем Average.pas и введите в него текст, представленный в листинге 8.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 8.2. Программа Average.pas

```
program Average;
var
  A: array[1..5,1..5] of integer;
  i, j: byte;
```

Окончание листинга 8.2

```
  sum: longint;
begin
  Writeln('Введите значения массива:');
  for i := 1 to 5 do
    for j := 1 to 5 do Read(A[i,j]);
  Writeln('Средние арифметические:');
  for i := 1 to 5 do
  begin
    sum := 0;
    for j := 1 to 5 do inc(sum, A[i,j]);
Writeln(sum/5:8:2);
  end;
end.
```

Программа Average достаточно проста и дополнительных пояснений не требует.

▸▸ Более сложные примеры использования массивов будут рассмотрены в главе 19.

Глава 9

Множества и записи

Множества и записи — это два структурированных типа данных, которые, возможно, используются не так часто как строки и массивы, но, тем не менее, при решении некоторых программных задач просто незаменимы.

Стоит также отметить, что записи являются переходным типом на пути к изучению такого понятия как "объекты", о которые рассматриваются в главе 12.

Множества

Множество — это набор взаимосвязанных объектов, которые можно рассматривать как единое целое. Каждый такой объект, называемый **элементом множества,** должен принадлежать одному из простых типов, кроме вещественного типа.

В выражениях языка Pascal элементы множества указываются в квадратных скобках, например, `[1,2,3,4/5]`, `['A'..'Z']`, `[0..10, 100..110]`. Если множество не содержит никаких элементов, то оно называется **пустым** и обозначается как `[]`.

Для объявления множества используется следующая конструкция:

```
set of имя_типа;
```

Рассмотрим пример объявления нескольких переменных множественного типа:

```
type
  ABC = set of 'A'..'Z';
  Digits = set of 0..9;
var
  Caps: ABC;
  Key: Digits;
  Odds: set of byte;
```

В данном случае Caps — это множество символов английского алфавита, Key — множество цифр от 0 до 9, Odds — пустое множество целочисленных значений типа byte. Количество элементов множества не может превышать 256, а номера значений должны находиться в диапазоне от 0 до 255, поэтому попытка объявить пустое множество типа integer приведет к ошибке компиляции.

Операторы, применяемые к множествам

При работе со множествами используются операторы сравнения (=, o, >=, <=), объединения (+), пересечения (*), вычитания (-), а также оператор in.

Описание этих операторов представлено в табл. 9.1.

Пример использования множеств

Напишем программу, которая формирует два множества значений типа char на основе двух введенных пользователем строк, а затем отображает те символы, которые

имеются в обеих строках. Создайте в интегрированной среде Turbo Pascal файл с именем SetsEx.pas и введите в него текст, представленный в листинге 8.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Таблица 9.1. Операторы, применяемые к множествам

Оператор	Описание
=	Равно. Два множества считаются равными, если они состоят из одних и тех же элементов. При этом порядок следования элементов значения не имеет
<>	Неравно. Два множества считаются неравными, если отличаются хотя бы одним элементом или их количеством
>=	Больше или равно. Если все элементы множества B содержатся в множестве A, тогда результатом выражения A >= в будет значение True
<=	Меньше или равно. Если все элементы множества A содержатся в множестве B, тогда результатом выражения A <= в будет значение True
+	Объединением двух множеств является третье множество, содержащее элементы обоих множеств. [1,2,3] + [3,4,5] = [1,2,3,4,5]
*	Пересечением двух множеств является третье множество, которое содержит элементы, входящие одновременно в оба множества. [1,2,3] * [3,4,5] = [3]
-	Разностью двух множеств является третье множество, которое содержит элементы первого множества, не входящие во второе множество. [1,2,3] – [3,4,5] = [1,2]

in	Используется для проверки принадлежности какого-нибудь значения указанному множеству. Оператор in часто позволяет значительно упростить сложные условные выражения. Например, следующий оператор if
	`if ((i = 0) or (i = 1) or (i = 2)) and`
	`((j = 3) or (j = 4) or (j = 5)) then ...`
	можно переписать в виде
	`if (i in [0..2]) and (j in [3..5]) then ...`

Листинг 9.1. Программа SetsEx.pas

```
program SetsEx;
var
  s1, s2: string;
  i: byte;
  Letters1, Letters2: set of char;
begin
  Write('Введите 1-ю строку: ');
  Readln(s1);
  Write('Введите 2-ю строку: ');
  Readln(s2);
  for i := 1 to Length(s1) do Letters1 := Letters1 + [s1[i]];
  for i := 1 to Ord(s2[0]) do Letters2 := Letters2 + [s2[i]];
  Writeln('Символы, входящие в обе строки:');
  Letters1 := Letters1 * Letters2;
  for i := 0 to 255 do
```

Окончание листинга 9.1

```
    if Chr(i) in Letters1 then Write(Chr(i):2);
end.
```

Предположим, пользователь ввел следующие две строки:

```
s1 = ABABCDEEF
s2 = WEAAAFXR
```

В первом цикле for просматривается посимвольно строка s1, и при помощи оператора объединения формируется множество Letters1. Обратите внимание на то, что при включении значений в множества они должны быть заключены в квадратные скобки. То есть оператор

```
Letters1 := Letters1 + s[i];
```

привел бы к ошибке компиляции, так как здесь происходит попытка объединения множества Letters1 со значением типа char, в то время как множества могут объединяться только с множествами. Поэтому значение типа char должно быть преобразовано в множество, состоящее из одного элемента, при помощи квадратных скобок:

```
Letters1 := Letters1 + [s[i]];
```

Аналогичным образом формируется множество Letters2. Обратите внимание на два способа определения длины строк s1 и s2. В первом случае используется стандартная функция Length, а во втором — преобразование нулевого элемента строки, содержащего длину, в числовое значение при помощи функции Ord.

После выполнения обоих операторов for множества Letters1 и Letters2 будут состоять из следующих элементов:

```
Letters1 = 'A','B','C','D','E','F'
Letters2 = 'W','E','A','F','X','R'
```

Обратите внимание на то, что операция объединения добавляет в множество только те элементы, которых в нем еще нет. Благодаря этому, элементы множества не дублируются.

После выполнения оператора Letters1 := Letters1 * Letters2; множество Letters1 будет содержать пересечение множеств Letters1 и Letters2, то есть — те элементы, которые входят в оба множества:

```
Letters1 = 'A','E','F'
```

Затем выполняется проверка каждого символа таблицы ASCII на его принадлежность множеству Letters1. Если символ принадлежит этому множеству, то он выводится на экран при помощи процедуры Writeln.

Записи

В предыдущей главе речь шла о том, что для хранения совокупности однотипных данных можно использовать массивы. Но что делать, если необходимо сохранить совокупность разнотипных данных? Именно для этой цели и служат *записи*. Они позволяют обращаться к компонентам различного типа при помощи одного идентификатора записи. Например, в одной записи может храниться строковая информация о фамилии, имени и отчестве, а также целочисленное значение возраста. Итак, что же такое запись?

Запись — это структурированный тип данных, состоящий из фиксированного количества разнотипных компонентов, называемых **полями.** Тип запись объявляется при помощи следующей конструкции:

```
type
  имя_типа = record
               идентификатор_поля: тип_поля;
               ...
               идентификатор_поля: тип_поля;
             end;
```

Например, можно объявить тип Person, который будет использоваться для хранения информации о человеке:

```
type
  Person = record
             LastName: string[20];
             FirstName: string[20];
             BirthYear: integer;
           end;
```

Далее объявляются переменные типа Person для хранения информации о двух людях:

```
var
  Man1, Man2: Person;
```

Для того чтобы обратиться к какому-либо полю записи используется следующая конструкция:

```
идентификатор_записи.имя_поля
```

В листинге 9.2 представлены примеры обращения к полям записей Man1 и Man2.

Листинг 9.2. Примеры обращения к полям записей

```
Man1.LastName := 'Шпак';
Man1.FirstName := 'Юрий';
Man1.BirthYear := 1974;
Man2.LastName := Man1.LastName;
Man2.FirstName := 'Богдан';
Man2.BirthYear := Man1.BirthYear + 27;
```

Если в программе необходимо обрабатывать некоторое количество однотипных записей, то можно объявить массив записей:

```
var
  Family: array[1..3] of Person;
...
Family[1].FirstName := 'Юрий';
Family[2].FirstName := 'Людмила';
Family[3].FirstName := 'Богдан';
```

В языке Pascal не определен тип для хранения комплексных чисел, и для реализации этой задачи удобно использовать записи:

```
type
  Complex = record
              RealPart: real;   {Действительная часть}
```

```
    ImagePart: real; {Мнимая часть}
  end;
```

Использование оператора with

Если используются длинные идентификаторы записей и полей, то обращение к ним имеет достаточно громоздкий вид. Для решения этой проблемы в языке Pascal используется специальный оператор with, который имеет следующий формат:

```
with идентификатор_записи do блок_операторов;
```

Например, к полям представленных выше записей Man1 и Man2 (листинг 9.2) можно было бы обратиться, как показано в листинге 9.3.

Листинг 9.3. Примеры обращения к полям записей при помощи оператора with

```
with Man1 do
begin
  LastName := 'Шпак';
  FirstName := 'Юрий';
  BirthYear := 1974;
end;
with Man2 do
begin
  LastName := Man1.LastName;
  FirstName := 'Богдан';
  BirthYear := Man1.BirthYear + 27;
end;
```

В языке Pascal допускается вложение записей друг в друга (не более 9 уровней), поэтому оператор with также может быть вложенным:

```
with Rec1, Rec2, ..., RecN do ...;
```

Записи с вариантами

В некоторых случаях возникает необходимость использовать записи, для которых заданы различные варианты их структуры. Они называются **записями с вариантами,** которые используются для объединения записей, имеющих похожую, но не одинаковую структуру.

Такие записи имеют фиксированную и вариантную часть. Использование фиксированной части аналогично использованию обычных записей, а для определения вариантной части применяется оператор case:

```
type
  имя_типа =
    record
      {фиксированная часть}
      ...
      {вариантная часть}
      case поле_признака: тип_поля_признака of
        константа_выбора1: (поле: тип);
        ...
        константа_выбораN: (поле: тип);
    end;
```

Рассмотрим пример объявления записи с вариантами:

```
type
  Car = record
          Brand: string[20];
          Year: integer;
          case Kind: string[8] of
            'грузовая': (Tonnage: real);
            'легковая': (Passengers: byte);
        end;
```

Тип Car используется для хранения информации об автомобиле. В поле Brand хранится производитель автомобиля, а в поле Year — год выпуска. Поле Tonnage (грузоподъемность) типа real доступно только в том случае, если поле признака Kind имеет значение 'грузовая'. Соответственно, поле Passengers (пассажиры) типа byte доступно только в том случае, если поле признака Kind имеет значение 'легковая'.

При использовании записей с вариантами необходимо придерживаться следующих правил.

- Все имена полей должны отличаться друг от друга, даже если они встречаются в разных вариантах.
- Запись может иметь только одну вариантную часть, объявленную в конце.

Пример использования записей

Разработаем программу учета музыкальных компакт-дисков. В ней пока будет реализован только ввод информации о дисках. В следующей главе эта программа будет доработана с тем, чтобы введенная пользователем информации могла сохраняться в файле. Создайте в интегрированной среде Turbo Pascal файл с именем Audio.pas и введите в него текст, представленный в листинге 9.4, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 9.4. Программа Audio.pas

```
program Audio;
type
  {Структура для хранения данных об одной дорожке}
  Track = record
            Name: string[50]; {Название дорожки}
            Hours: 0..3;      {Часы}
            Minutes: 0..59;   {Минуты}
            Seconds: 0..59;   {Секунды}
          end;
  {Структура для хранения информации о диске}
  CD = record
         Author: string[20];   {Автор}
         Title: string[50];    {Название}
         Year: integer;        {Год}
         Kind: string[10];     {Жанр}
         case Tracks: byte of  {Количество дорожек}
           1: (Track1: Track); {Если дорожек больше одной, тогда
                                информация хранится в виде массива}
           [2..20]: (ATracks: array[1..20] of Track);
       end;
```

Окончание листинга 9.4

```
var
  c: char;
  CDData: CD;
  i, j : byte;

{Процедура ввода данных об одной дорожке}
procedure DataForTrack(var TrackData: Track);
begin
  Write('Название: ');
  Readln(TrackData.Name);
  Write('Длительность (часы): ');
  Readln(TrackData.Hours);
  Write('Длительность (минуты): ');
  Readln(TrackData.Minutes);
  Write('Длительность (секунды): ');
  Readln(TrackData.Seconds);
end;

begin
  i := 0;
  repeat
    inc(i);
    Writeln('Ввод данных о ',i,'-м диске:');
\   Write('Автор, исполнитель: ');
    Readln(CDData.Author);
    Write('Название: ');
    Readln(CDData.Title);
    Write('Год: ');
    Readln(CDData.Year);
    Write('Жанр (Рок,Поп,Классика,Джаз,Блюз,Реп,Фанк): ');
    Readln(CDData.Kind);
    Write('Количество дорожек: ');
    Readln(CDData.Tracks);
    for j := 1 to CDData.Tracks do
    begin
      Writeln('Данные о ',j,'-й дорожке:');
      if CDData.Tracks = 1
      then DataForTrack(CDData.Track1)
      else DataForTrack(CDData.ATracks[j]);
    end;
    Write('Ввести данные еще об одном диске (Да/Нет)? ');
    Readln(c);
  until c in ['N','n','Н','н'];
end.
```

В программе Audio используется два типа записей: Track и CD. Тип CD является записью с вариантами и предназначен для хранения информации о компакт-диске. Тип Track используется типом CD для хранения информации о звуковых дорожках диска. В программе объявлена переменная CDData типа CD. Ввод данных о дисках организован в цикле repeat, выход из которого происходит в том случае, если после ввода информации об очередном альбоме пользователь нажмет клавишу <Н>, <н>, <N> или <п>.

Если на диске есть только одна звуковая дорожка, тогда поле CDData.Tracks имеет значение 1, и данные об этой дорожке будут сохраняться в поле Track1 — записи типа Track. Если на диске есть несколько звуковых дорожек, тогда данные о них будут сохраняться в массиве записей ATracks типа Track.

Для ввода данных об одной дорожке в программе была создана процедура DataForTrack, в которую по значению передается параметр типа Track. Если на диске только одна дорожка, то данные вводятся в поле CDData.Track1, в противном случае — в текущий элемент массива CDData.ATracks.

Глава 10

Файлы

Файл — это совокупность данных, сохраненная на диске под определенным именем. Сохранение информации в файле уже кратко затрагивалось в разделе "Запись и чтение данных из текстового файла" главы 4, где была рассмотрена организация обмена информацией с текстовыми файлами при помощи процедур Readln и Writeln. Кроме ***текстовых*** файлов, содержимое которых пользователь может прочитать, отредактировать или распечатать на принтере, существуют также *двоичные* файлы. В соответствии со своим названием, двоичные файлы содержат данные в двоичном формате, и интерпретация их содержимого возлагается на специализированные программные средства. Например, интерпретацию файлов, содержащих графические изображения, выполняют графические редакторы, архивных файлов — программы архивации, а выполняемых файлов — операционная система.

В операционной системе MS-DOS каждый файл на диске обозначается при помощи имени и расширения. Имя, длина которого может составлять от 1 до 8 символов, отделяется от расширения точкой. Расширение можно не указывать вообще. В этом случае точка после имени файла не ставится, или же оно может состоять из одного, двух или трех символов. Например, выполняемый файл интегрированной среды программирования Turbo Pascal называется turbo.exe. В данном случае turbo — это имя, а exe — расширение. Имена файлов могут содержать буквы, цифры и следующие символы: !, @, #, %, ^, &, (,), ', ~, -, _.

Расширение имени файла, как правило, определяет содержимое файла. В табл. 10.1 перечислены некоторые общепринятые расширения и их назначение.

Таблица 10.1. Общепринятые расширения имен файлов

Расширение	Назначение
arj	Файл архива, созданного архиватором arj
asm	Файл программы на языке ассемблера
bak	Резервная копия файла
bat	Командный (пакетный) файл
com	Выполняемый файл размером до 64 Кбайт
dat	Файл данных
doc	Текстовый документ
exe	Выполняемый файл произвольного размера
gif	Графический файл в формате GIF
hlp	Файл справочной системы
ini	Файл с параметрами инициализации программ
pas	Файл программы на языке Pascal
pcx	Графический файл в формате PCX
pic	Графический файл в формате PIC

Окончание таблицы 10.1

Расширение	Назначение
rar	Файл архива, созданного архиватором rar
,sys	Системный файл
tif	Графический файл в формате TIFF
txt	Текстовый файл
zip	Файл архива, созданного архиватором pkzip

Неотъемлемыми характеристиками каждого файла являются его размер, а также дата и время его Создания. Размер файла определяется количеством входящих в него байтов информации, а дата и время создания изменяются при изменении файла.

Имена файлов регистрируются на магнитных дисках в **каталогах,** имена для которых выбирают по тем же правилам, что и для имен файлов. При этом, как правило, расширения имен каталогов не используются. Если каталог А зарегистрирован в каталоге В, то говорят, что А — это родительский каталог, а В — это **подкаталог.** Каждый магнитный диск обязательно имеет главный каталог, так называемый корневой каталог, в котором зарегистрированы все каталоги и файлы первого уровня. В свою очередь, в каталогах первого уровня зарегистрированы файлы и каталоги второго уровня и т.д. Таким образом, файлы и каталоги в операционной системе MS-DOS размещаются в виде древовидной структуры.

Последовательность имен от корневого каталога к файлу называется **путем.** Путь к файлу в сочетании с его именем называется **полным именем.** Например, полное имя файла `turbo.exe` может выглядеть следующим образом: `e:\work\books\computer\tp\bin\turbo.exe`, где `e:` — обозначение магнитного диска.

ПРИМЕЧАНИЕ

Магнитные диски обозначаются при помощи латинских букв с двоеточием (от `a:` до `z:`). При этом накопителям на гибких магнитных дисках соответствуют обозначения `a:` и `b:`, а все остальные буквы — прочим носителям. Как правило, первыми, начиная с диска `c:`, обозначаются логические диски жестких дисков.

Работа с файлами в языке Pascal

В языке Pascal можно объявлять переменные для работы с тремя типами файлов: текстовыми, типизированными и нетипизированными.

Текстовый файл — это последовательность символов, разбитая на строки длиной от О до 256 символов,

К типизированным относятся файлы, содержащие данные строго определенного типа. Обычно такие файлы представляются собой наборы записей.

К нетипизированным относятся двоичные файлы, которые могут содержать любые совокупности байтов данных без привязки к какому-нибудь одному типу.

Каждый из этих файловых типов рассматривается ниже в соответствующих разделах этой главы. Здесь же рассмотрим средства языка Pascal, предназначенные для создания, открытия и закрытия файлов, независимо от их типа.

Создание, открытие и закрытие файлов

Каждому файлу в программе ставится в соответствие файловая переменная определенного типа (предположим, что эта переменная обозначается при помощи идентификатора F). Для установления связи между этой переменной и реальным файлом используется процедура `Assign`, например:

```
Assign(F, 'c:\docs\doc1.txt');
```

В данном случае переменной F ставится в соответствие файл doc1.txt, расположенный на диске c: в каталоге docs.

Процедура Assign всегда предшествует всем остальным процедурам, предназначенным для работы с файлами. Кроме того, недопустимо использование этой процедуры для уже открытого файла. Для открытия файла в языке Pascal используется две процедуры: Reset и Rewrite. Процедура Reset открывает уже существующий файл, а процедура Rewrite создает и открывает новый файл. В обе процедуры передается параметр файлового типа. Если при вызове процедуры Reset, отсутствует файл, связанный с файловой переменной, то это приводит к возникновению ошибки в программе. Также следует быть внимательным и при вызове процедуры Rewrite. Если на диске уже существует файл с именем, связанным с файловой переменной, передаваемой в процедуру Rewrite, то все его данные будут *удалены*.

Для закрытия файла используется процедура Close, в которую в качестве параметра передается файловая переменная. Эта процедура удаляет связь файловой переменной с файлом на диске, установленную процедурой Assign. Таким образом, при работе с файлами обязательно должна соблюдаться следующая последовательность действий:

```
Assign — Reset/Rewrite — (чтение/запись) — Close
```

Переименование и удаление файлов

К общим средствам языка Pascal, предназначенных для работы с файлами можно отнести еще две процедуры: Rename и Erase. Процедура Rename переименовывает неоткрытый файл любого типа. В нее передается два параметра: первый — это файловая переменная, а второй — это новое имя соответствующего файла. Процедура Erase принимает только один параметр — файловую переменную, и удаляет соответствующий неоткрытый файл любого типа. В случае применения этих двух процедур к открытым или не существующим файлам возникает ошибка выполнения программы.

Обработка ошибок при работе с файлами

Для обработки ошибок ввода-вывода в языке Pascal используется специальная функция IOResult. Она не принимает никаких параметров и возвращает значение типа integer, соответствующее состоянию последней выполненной операции ввода-вывода. Нормальному выполнению операции соответствует значение 0. Функцию lOResult можно вызывать только в том случае, если на время выполнения файловых операций отключена стандартная проверка операций ввода-вывода. Для этого в программе можно воспользоваться директивой компилятора {$I}. Использование функции lOResult будет рассмотрено в следующем разделе на примере программы, в которой реализовано копирование текстового файла.

» Подробно директивы компилятора рассматриваются в главе 13.

Текстовые файлы

Для объявления файловой переменной, связанной с текстовым файлом используется стандартный тип Text:

```
var
  F: Text;
```

Каждая строка в текстовом файле оканчивается двумя символами, обозначающими конец строки — Chr(13) и Chr(10). При вызове процедуры Reset текстовый файл открывает только для чтения, а при вызове процедуры Rewrite — для чтения и записи.

Для открытия текстовых файлов используется еще одна процедура: Append. Она отличается от процедуры Rewrite тем, что не удаляет содержимое существующего файла, а открывает его для дополнения данных в конец файла.

Чтение данных

Чтение данных из файла осуществляется при помощи процедур Read и Readln. Процедура Read считывает данные общим потоком, а Readln приводит к переходу к следующей строке текстового файла. При считывании нетекстовых данных (например, целого числа) в переменную некоторого простого типа выполняется автоматическое преобразование считанного значения в соответствующий тип:

```
Assign(F, '1.txt');
Reset(F);
Readln(F, s);
Readln(F, i);
Readln(F, r);
Close(F);
```

Запись данных

Запись данных в текстовый файл осуществляется при помощи процедур Write и Writeln. При этом в случае вызова процедуры Writeln после записываемых данных, в файл автоматически добавляются символы перехода на новую строку. Сохранять в текстовом файле можно значения любого простого типа — они автоматически преобразуются в текст (листинг 10.1).

Листинг 10.1. Сохранение данных в текстовом файле

```
var
  F: Text;
  s: string;
  i: integer;
  r: real;
...
Assign(F, '1.txt');
Rewrite(F);
Writeln(F, s);
Writeln(F, i);
Writeln(F, r);
Close(F);
```

Перемещение по файлу

После открытия текстового файла при помощи процедуры Reset файловая позиция устанавливается на начало файла, и после каждого вызова процедур Read и Readln перемещается по направлению к концу файла. В языке Pascal есть специальные функции, позволяющие определить текущую позицию в текстовом файле: Eoln, Eof, SeekEoln и SeekEof (для использования этих функций файл должен быть открытым).

- Функция Eoln возвращает значение True, если текущая файловая позиция находится на символах конца строки или в конце файла.
- Функция Eof возвращает значение True, если достигнут конец файла.
- Функция SeekEoln возвращает значение True при достижении символов конца строки.
- Функция SeekEof возвращает значение True при достижении конца файла.

Функцию Eof обычно используют в цикле while для последовательно считывания всех строк текстового файла, а функцию Eoln — для последовательного считывания всех символов текущей строки:

```
while not Eof(F) do readln(F,s);
while not Edln(F) do read(F,c);
```

Копирование текстового файла

Рассмотрим работу описанных процедур на примере программы, выполняющей копирование текстового файла. Создайте в интегрированной среде Turbo Pascal файл с именем FileCopy.pas и введите в него текст, представленный в листинге 10.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 10.2. Программа FileCopy.pas

```
program FileCopy;
var
  FFrom, FTo: Text;
  NameFrom,NameTo,s: string;
  isError: boolean;
begin
  isError := True;
  while isError do
  begin
    Writeln('Укажите имя файла, который необходимо скопировать:');
    Readln(NameFrom);
    Assign(FFrom, NameFrom);
    {$I-}          {Отключается стандартная обработка ошибок}
    Reset(FFrom);  {Открывается файл}
    {$I+}          {Включается стандартная обработка ошибок}
    isError := IOResult > 0;
    if isError then Writeln('Файла с таким именем не существует! ');
  end;
  Writeln('Укажите имя файла назначения:');
  Readln(NameTo);
  Assign(FTo, NameTo);
  Rewrite(FTo);
  while not Eof(FFrom) do
  begin
    Readln(FFrom,s); {Считывание строки из файла источника}
    Writeln(FTo,s);  {Запись строки в файл назначения}
  end;
  Close(FTo);
  Close(FFrom);
end.
```

В первой части этой программы в цикле while организовано открытие текстового файла источника. Имя этого файла вводится пользователем, после чего оно ставится в соответствие файловой переменной FFrom. Перед попыткой открыть этот файл на чтение выполняется отключение стандартной проверки операций ввода-вывода при помощи директивы компилятора {$I-}. После вызова процедуры Reset стандартная проверка операций ввода-вывода опять включается при помощи директивы {$I+}. Затем, если функция IOResult возвращает некоторое положительное значение, что свидетельствует о наличии ошибки ввода-вывода, логической переменной isError присваивается значение True, и цикл while повторяется сначала. Если же ошибок обнаружено не было, переменной isError присваивается значение False и цикл whileзавершается.

Во второй части программы создается и открывается для записи файл назначения, соответствующий файловой переменной FTo, а затем в цикле while строки последовательно считываются из файла FFrom и записываются в файл FTo до тех пор, пока не будет достигнут конец файла источника FFrom, т.е. функция Eof(FFrom) не вернет значение True. В конце программы оба файла закрываются при помощи функции Close.

Вывод содержимого файлов на внешние устройства компьютера

При помощи программы FileCopy можно копировать содержимое текстовых файлов не только в другой текстовый файл, но и на внешние устройства компьютера. Для этого в качестве имени файла назначения достаточно указать одно из стандартных имен устройств.

- com1, com2, cot3 — устройства, подключенные к последовательным портам 1, 2 и 3, соответственно.
- con — консоль. При выводе этим устройством является экран монитора, а при вводе клавиатура.
- lpt1, lpt2, lpt3 — устройства, подключенные к параллельным портам 1, 2 и 3, соответственно.
- prn — принтер.

Например, для того чтобы вывести содержимое некоторого файла на экран монитора, необходимо указать в качестве имени файла назначения con. Если же файл необходимо распечатать на принтере, его можно скопировать в файл с именем prn или lpt1 (если принтер подключен к порту LPT1).

Типизированные файлы

Типизированные файлы используются для хранения однотипных данных, в качестве которых обычно выступают записи. Такие файлы применяются для создания баз данных — наборов данных, имеющих определенную структуру.

» Подробнее базы данных рассматриваются в главе 22.

Для объявления файловых переменных, используемых для обращения к типизированным файлам, используется конструкция file of *имя_типа*:

```
type
  Rec = record
        ...
        end;
```

```
var
  F1: file of FileRec;
  F2: file of real;
```

В данном случае файл Fl используется для хранения записей типа Rec, а файл F2 — значений типа real.

Если содержимое текстового файла рассматривается как набор символов, то содержимое типизированного файла рассматривается как последовательность записей определенной структуры. При этом длина одной записи определяется при помощи функции SizeOf, которой в качестве параметра передается имя типа записи (например, SizeOf(Rec)). Так как длина каждой записи в типизированном файле строго постоянна, это позволяет организовать доступ к каждому элементу по его порядковому номеру при помощи процедуры Seek:

```
Seek(F, 9); {переход к десятой записи в файле F}
```

Первая запись в файле имеет номер 0, таким образом, физический номер записи на единицу меньше ее логического номера. Для определения номера текущей записи используется функция FilePos, а для определения общего количества записей в файле — функция FileSize. Например, для перехода к концу типизированного файла F можно воспользоваться следующим оператором:

```
Seek(F, FileSize(F));
```

а для перехода к последней записи оператором

```
Seek(F, FileSize(F) - 1);
```

Примеры использования типизированных файлов

Усовершенствуем программу Audio, разработанную в предыдущей главе (см. листинг 9.4). Откройте в интегрированной среде программирования Turbo Pascal файл Audio.pas, выполните команду меню **File | Save As,** и сохраните его под именем AudioIn.pas. Внесите в него изменения, выделенные полужирным шрифтом в листинге 10.3. (Файл Audioln.pas можно открыть при помощи команды **File | Open** (<F3>) с прилагаемой к книге дискеты).

Листинг 10.3. Программа AudioIn.pas

```
program AudioIn;
type
  {Структура для хранения данных об одной дорожке}
  Track = record
            Name: string[50]; {Название дорожки}
            Hours: 0..3;      {Часы}
            Minutes: 0..59;   {Минуты}
            Seconds: 0..59;   {Секунды}
          end;
  {Структура для хранения информации о диске}
  CD = record
         Author: string[20];   {Автор}
         Title: string[50];    {Название}
         Year: integer;        {Год}
         Kind: string[10];     {Жанр}
         case Tracks: byte of  {Количество дорожек}
```

Окончание листинга 10.3

```
            1: (Track1: Track); {Если дорожек больше одной, тогда
                                 информация хранится в виде массива}
          [2..20]: (ATracks: array[1..20] of Track);
      end;
var
  c: char;
  CDData: CD;
  i, j: byte;
  FName: string;
  F: file of CD;

{Процедура ввода данных об одной дорожке}
procedure DataForTrack(var TrackData: Track);
begin
  ...
end;

begin
  Writeln('Введите имя файла с данными о музыкальных дисках:');
  Readln(FName);
  Assign(F, FName);
  {$I-}
  Reset(F);
  {$I+}
  if IOResult = 0
  then Seek(F, FileSize(F))  {Если такой файл уже существует,}
                             {перемещаемся в его конец}
  else Rewrite(F);   {В противном случае создаем новый файл}
  i := FileSize(F);  {Порядковый номер следующего диска}
  repeat
    inc(i);
    Writeln('Ввод данных о ',i,'-м диске:');
    ...
    Write(F, CDData); {Сохранение записи в файле F}
    Write('Ввести данные еще об одном диске (Да/Нет)? ');
    Readln(c);
  until c in ['N','n','Н','н'];
  Close(F);
end.
```

Программа AudioIn записывает данные о компакт-дисках, вводимые пользователем, в типизированный файл, имя которого указывается в самом начале программы. Этому файлу ставится в соответствие переменная F типа file of CD. Если файл с указанным именем уже существует, то с помощью процедуры Seek выполняется перемещение в его конец, чтобы к нему можно было добавить новые записи. Если файла с указанным именем не существует, то он создается и открывается для записи при помощи процедуры Rewrite.

Данные об очередном диске и его звуковых дорожках, введенные пользователем и сохраненные в записи CDData, записываются в файл F при помощи процедуры Write. (Внимание! При работе с типизированными файлами процедура Writeln не используется.) После ввода данных обо всех дисках файл F закрывается вызовом процедуры Close.

Запустите эту программу на выполнение и введите сведения о нескольких компакт-дисках. В следующем примере будет рассмотрена программа просмотра данных, введенных при помощи программы Audio In.

Сохраните при помощи команды **File | Save As** интегрированной среды программирования Turbo Pascal файл AudioIn.pas (см. листинг 10.3) под именем AudioOut.pas. Удалите в нем все, кроме раздела объявления типов и переменных, а затем внесите в него изменения, выделенные полужирным шрифтом в листинге 10.4. (Файл AudioOut.pas можно открыть при помощи команды **File | Open** (<F3>) с прилагаемой к книге дискеты).

Листинг 10.4. Программа AudioOut.pas

```
program AudioOut;
type
  {Структура для хранения данных об одной дорожке}
  Track = record
            Name: string[50]; {Название дорожки}
            Hours: 0..3;      {Часы}
            Minutes: 0..59;   {Минуты}
            Seconds: 0..59;   {Секунды}
          end;
  {Структура для хранения информации о диске}
  CD = record
         Author: string[20];   {Автор}
         Title: string[50];    {Название}
         Year: integer;        {Год}
         Kind: string[10];     {Жанр}
         case Tracks: byte of  {Количество дорожек}
           1: (Track1: Track); {Если дорожек больше одной, тогда
                                информация хранится в виде массива}
           [2..20]: (ATracks: array[1..20] of Track);
       end;
var
  c: char;
  CDData: CD;
  i, num: byte;
  FName: string;
  F: file of CD;
  isError: boolean;

{Процедура вывода данных об одной дорожке}
procedure DataForTrack(var TrackData: Track; num: byte);
var
  hh, mm, ss: string[2];
  Timing: string[8];
begin
  with TrackData do
  begin
    {Преобразовываем в строки значения часов, минут и секунд}
    Str(Hours, hh);
    Str(Minutes, mm);
    Str(Seconds, ss);
    {Формируем строку времени звучания}
    if Hours = 0 then Timing := '' else Timing := hh + ':';
    Timing := Timing + mm + '"' + ss + Chr(39);
```

Окончание листинга 10.4

```
    Writeln(num, ' - ',Name, ' (',Timing, ')');
  end;
end;

begin
  isError := True;
  while isError do
  begin
    Writeln('Введите имя файла с данными о музыкальных дисках:');
    Readln(FName);
    Assign(F, FName);
    {$I-}
    Reset(F);
    {$I+}
    isError := (IOResult > 0);
    if isError then Writeln('Такой файл не существует!');
  end;
  repeat
    Writeln('Файл содержит данные о ',FileSize(F),' дисках:');
    Seek(F, 0);  {Перемещаемся к первой записи}
    for i := 0 to FileSize(F) - 1 do
    begin
      Read(F, CDData);
      Writeln(i + 1, ' - ', CDData.Author, ', "', CDData.Title,'"');
      {Отображаем список дисков по 20 на экране}
      if (((i + 1) mod 20) = 0) and
         (i < (FileSize(F) - 1))
      then begin
        Writeln('Для промотра остальных дисков нажмите <Enter>...');
        Read(c);
      end;
    end;
    Writeln('Введите порядковый номер диска для просмотра: ');
    Readln(num);
    Seek(F, num - 1);  {Перемещаемся к указанной записи}
    Read(F, CDData);
    Writeln('>>> ', CDData.Author, ', "', CDData.Title,
            '" - ', CDData.Year, 'г. (', CDData.Kind, ') <<<');
    {Выводим данные о дорожках}
    if CDData.Tracks = 1
    then DataForTrack(CDData.Track1, 1)
    else for i := 1 to CDData.Tracks do
           DataForTrack(CDData.ATracks[i], i);
    Write('Показать данные еще об одном диске (Да/Нет)? ');
    Readln(c);
  until c in ['N','n','Н','н'];
  Close(F);
end.
```

В начале программы AudioOut выполняется уже знакомая проверка наличия файла с указанным именем. Затем на экран в цикле for выводится информация обо всех дисках в виде списка по 20 наименований на экране. Если номер диска кратен 20 и

не является последним в списке, то вывод списка приостанавливается до нажатия пользователем клавиши <Enter>.

После вывода списка дисков выполняется перемещение в файле к записи с номером, указанным пользователем. При помощи процедуры Read данные о диске считываются из файла в переменную-запись CDData, а затем отображаются на экране. Отображение данных о каждой звуковой дорожке возложено на процедуру DataForTrack. В ней сначала формируется строка, обозначающая время звучания текущей звуковой дорожки. При этом для преобразования целочисленных значений компонентов времени в их строковое представление используется функция str.

Нетипизированные файлы

Нетипизированные файлы предназначены для хранения данных, не имеющих строго определенного типа, и представляют собой просто некоторую совокупность байтов. Фактически, любой файл, созданный как текстовый или типизированный, можно открыть как нетипизированный и работать с ним как с произвольным набором байтов. Для объявления в программе переменных, предназначенных для доступа к нетипизированным файлам, используется зарезервированной слово File:

```
var
  F: File;
```

При работе с нетипизированными файлами их содержимое считывается в определенную область памяти компьютера. Для таких файлов самым важным параметром является длина считываемой/записываемой записи в байтах, которая указывается при открытии файла:

```
Rewrite(F, 1);
Reset (F, 10);
```

По умолчанию длина записи нетипизированного файла составляет 128 байт, поэтому после открытия файла при помощи вызова процедуры Rewrite или Reset без передачи второго параметра все процедуры и функции, предназначенные для работы с нетипизированными файлами, будут работать с записями длиной 128 байтов.

Для данного типа файлов в языке Pascal, кроме процедур Read и Write, используются процедуры BlockRead и BlockWrite, поддерживающие операции ввода-вывода с более высокой скоростью. Они имеют следующий синтаксис:

```
BlockRead(var F: file; var Buf; Count: word {; Result: word});
BlockWrite(var F: file; var Buf; Count: word {; Result: word});
```

Параметру Buf соответствует любая переменная, используемая для хранения данных считываемых или записываемых данных; параметр Count — это количество считываемых или записываемых блоков. Размер блока данных вычисляется по формуле Count * RecSize, где RecSize — размер записи файла, указанный при его открытии. Необязательный параметр Result после вызова процедуры содержит количество фактически считанных или записанных записей.

Рассмотрим работу с нетипизированными файлами на примере программы копирования файлов. Откройте в интегрированной среде программирования Turbo Pascal файл FileCopy.pas (листинг 10.2), сохраните его под именем CopyFile.pas, и внесите в него изменения, выделенные полужирным шрифтом в листинге 10.5, или откройте этот файл *с* дискеты при помощи команды **File | Open** (<F3>).

Листинг 10.5. Программа CopyFile.pas

```
program CopyFile;
var
  FFrom, FTo: File;
  NameFrom,NameTo,s: string;
  isError: boolean;
  CurPos, FSize: longint;
  Blocks: integer;
begin
  isError := True;
  while isError do
  begin
    Writeln('Укажите имя файла, который необходимо скопировать:');
    Readln(NameFrom);
    Assign(FFrom, NameFrom);
    {$I-}
    Reset(FFrom, 1); {Открывается файл с размером записи в 1 байт}
    {$I+}
    isError := IOResult > 0;
    if isError then Writeln('Файла с таким именем не существует!');
  end;
  Writeln('Укажите имя файла назначения:');
  Readln(NameTo);
  Assign(FTo, NameTo);
  Rewrite(FTo, 1); {Создается файл с размером записи в 1 байт}
  CurPos := 0;     {Счетчик текущей позиции в файле}
  Blocks := 256;   {Количество считываемых блоков}
  FSize := FileSize(FFrom); {Размер файла источника}
  Write('Процесс... ');
  while not Eof(FFrom) do
  begin
    BlockRead(FFrom,s,Blocks);
    BlockWrite(FTo,s,Blocks);
    inc(CurPos, Blocks);
    {Проверка выхода эа границы файла}
    if (CurPos + Blocks) > FSize then
      Blocks := FSize - CurPos;
  end;
  Close(FTo);
  Close(FFrom);
  Writeln('Готово!');
end.
```

В программе CopyFile размер записи был установлен, равным 1 байту, а количество считываемых за один раз блоков — 256. Таким образом, за один вызов процедуры BlockRead в переменную s типа string считывается 256 байтов. В том случае, когда размер последнего считываемого из файла фрагмента данных меньше 256 байтов, последний проход цикла приведет к возникновению ошибки ввода-вывода, так как будет осуществлена попытка считать данных больше, чем их есть на самом деле. Для решения этой проблемы в программе была реализована проверка выхода за границы файла. Если сумма уже считанных байтов и количества байтов, считываемых за один раз, превышает размер файла (if (CurPos + Blocks) > FSize then), выполняется корректировка (уменьшение) количества считываемых блоков: Blocks := FSize - CurPos;

Глава 11

Указатели

Фактически, объявление любой переменной или константы подразумевает выделение определенного участка оперативной памяти для хранения ее значения. Процесс выделения области памяти называется **распределением памяти.** В предыдущих главах использовалось так называемое **статическое распределение памяти,** при котором компилятор может выделить необходимую память для объявленных переменных и констант только на основании текста программы, без ее выполнения. Статические переменные можно использовать в тех случаях, когда можно контролировать процесс управления памятью в момент написания программы.

Длина сегмента данных, используемого процессорами 8x86, составляет 65536 байтов (64 Кбайта), чего бывает недостаточно для обработки больших массивов данных. Для решения этой проблемы используется механизм **динамического распределения памяти,** когда данные могут размещаться во всей доступной оперативной памяти компьютера. При таком подходе память выделятся не на этапе компиляции, а во время выполнения программы по мере необходимости обработки данных. После того как значение переменной обработано, отведенную под него память можно освободить. Таким образом, обеспечивается рациональное использование памяти — особенно при работе с большими массивами данных. Например, если объявить статический массив

```
var A: array[1..300] of byte;
```

то при компиляции для него будет жестко выделено 300 байтов памяти, хотя фактически в процессе выполнения программы из них могут использоваться не больше половины. При статическом распределении память под массивы приходится выделять с запасом, чтобы предусмотреть все возможные варианты. Это приводит к возникновению избыточности, когда большие объемы оперативной памяти во время выполнения программы вообще не используются.

Переменные, которые создаются и уничтожаются в процессе выполнения программы называются **динамическими.** Их используют в тех случаях, когда необходимый размер памяти в момент написания программы предсказать невозможно. Динамическое распределение памяти организовано при помощи **указателей** — особых переменных, указывающих расположение какой-то другой переменной заранее определенного типа. Другими словами, указатель содержит адрес первого байта области памяти, в которой хранятся данные. Сам по себе указатель занимает в памяти четыре байта. К переменным, на которые указывает указатель, можно обращаться через имя указателя, и по этой причине их называют **ссылочными переменными.**

Адреса, хранимые в указателях, состоят из двух частей — *сегмента* и *смещения* длиной 16 разрядов каждый (тип word). **Сегмент** — это участок памяти длиной 65536 байтов (64 Кбайта), начинающийся с физического адреса, кратного 16 (то есть 0, 16, 32 и т.д.). **Смещение** указывает расстояние в байтах от начала сегмента до требуемого адреса памяти. Таким образом, фактический (абсолютный) адрес образуется по следующей формуле:

```
сегмент * 16 + смещение
```

Типизированные указатели

Типизированными называются указатели, содержащие адрес области памяти, в которой хранится значение переменной заранее определенного типа. Для объявления типизированного указателя используется знак "^", расположенный перед соответствующим типом. Можно привести следующие примеры объявления типизированных указателей:

```
type
  pInt = ^integer;  {указатель на целочисленные значения}
  pReal = ^real;    {указатель на вещественные значения}
var
  P1, P2: pInt;
  P3: pReal;
  P4: ^byte;  {указатель на значение типа byte}
```

Для того чтобы присвоить переменной ссылочного типа некоторое значение, используют оператор @, который можно читать как "адрес". Например, если объявлена целочисленная переменная i, то запись @i можно прочитать как "адрес переменной i" или "указатель на i":

```
...
var
  P: ^integer;
  i: integer;
begin
  i := 2;
  P := @i; {указатель P содержит адрес области памяти,
            в которой хранится значение 2}
  ...
end.
```

К значению переменной можно получить доступ при помощи указателя соответствующего типа, после имени которого расположен знак "^". В представленном выше примере, для отображения на экране значения переменной i можно воспользоваться одним из двух операторов:

```
Writeln(i);   {непосредственный вывод значения i}
Writeln(P^);  {вывод содержимого области памяти, на
               которую ссылается указатель P}
```

Такой механизм называется **разыменованием** указателя.

ПРИМЕЧАНИЕ

В языке Pascal используется один специальный указатель, который, фактически ни на что не указывает. Для обозначения такого указателя используется слово Nil. Указатель Nil считается константой, совместимой с любым ссылочным типом, поэтому его значение можно присваивать любому указателю.

Если ссылочная переменная имеет значение Nil (то есть не указывает ни на какую область памяти), то ее разыменование приведет к некорректному выполнению программы.

К указателям можно применять операторы сравнения о и =. Два указателя равны в том случае, если они указывают на один и тот же элемент данных.

Рассмотрим использование типизированных указателей на примере ввода и вывода значений массива целых чисел. Создайте в интегрированной среде Turbo Pascal новый

файл, присвойте ему имя Pointer1.pas и введите в него текст из листинга 11.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 11.1. Программа Pointer1.pas

```
program Pointer1;
type
  TArr = array[1..5] of integer;
var
  PArr: ^TArr;  {Указательна массив}
  Arr: TArr;
  i: integer;
begin
  PArr := @Arr; {PArr содержит адрес первого элемента массива Arr}
  for i := 1 to 5 do
  begin
    Write(i,'-й элемент массива = ');
    Readln(PArr^[i]); {Учитывая то, что PArr указывает
                       на первый элемент массива Arr,
                       PArr^ - это аналог идентификатора Arr}
  end;
  for i := 1 to 5 do
  begin
    PArr := @Arr[i];   {PArr содержит адрес i-го элемента массива Arr}
    Writeln(PArr^[1]); {PArr^ - значение i-го элемента массива}
  end;
end.
```

Подробно программу Pointer1 рассматривать не будем, так как все очевидно из приведенных в ней комментариев.

Нетипизированные указатели

В языке Pascal можно объявлять указатели, не связанные с конкретным типом данных. Для этого служит стандартный тип pointer. С помощью нетипизированных указателей удобно динамически размещать данные, у которых меняется структура и тип в процессе выполнения программы.

Переменные типа pointer не могут быть разыменованы. Если после имени нетипизированного указателя ввести символ "^", то при компиляции программы будет выдано сообщение об ошибке. Как и указатели Nil, указатели pointer совместимы со всеми остальными типами указателей. Одним из примеров использования нетипизированых указателей является передача нетипизированных параметров в функции и процедуры.

◀◀ Рассматриваются в разделе "Параметры процедур и функций" главы 7.

Связанные списки

Обычно указатели используются при работе с записями. Если при этом сама запись содержит в себе поле-указатель, указывающий на следующую за ней запись, то таким образом формируется **связанный список** — структура, в которой отдельные записи последовательно связаны друг с другом.

Рассмотрим работу со связанными списками на примере программы, в которой реализовано создание перечня фамилий и имен, и удаление из этого списка записи, указанной пользователем.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Links.pas и введите в него текст из листинга 11.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 11.2. Программа Links.pas

```
program Links;
type
  Link = ^Person;
  Person = record
             ID: integer;            {Идентификатор}
             Lastname: string[20];   {Фамилия}
             FirstName: string[20]; {Имя}
             NextRec: Link;          {Указатель на следующую запись}
           end;
var
  i, c, j: integer;
  Last, First, Curr: Link; {Last - указатель на последнюю запись
                            First - указатель на первую запись
                            Curr - указатель на текущую запись}

procedure ShowList;
begin
  Writeln('Полный список:');
  Curr := First; {Переходим в начало списка}
  while Curr <> Last do {Повторять до тех пор, пока не
                         будет достигнут конец списка}
  begin
    Writeln(Curr^.ID:3,' - ',Curr^.LastName,' ',Curr^.FirstName);
    Curr := Curr^.NextRec; {Перемещаемся к следующей записи}
  end;
end;

begin
  Write('Укажите длину списка: ');
  Readln(c);
  Last := New(Link);  {Создаем первую запись типа Person}
  First := Last;      {Запоминаем начало списка}
  for i := 1 to c do
  begin
    Writeln('Запись ',i,':');
    Last^.ID := i;
    Write('Фамилия: ');
    Readln(Last^.LastName);
    Write('Имя: ');
    Readln(Last^.FirstName);
    Last^.NextRec := New(Link); {Создаем новый элемент списка
                                 и сохраняем указатель на него}
    Last := Last^.NextRec;      {Перемещаемся к новому элементу}
  end;
  ShowList; {Отображаем список на экране}
  Write('Укажите номер записи для удаления: ');
```

Окончание листинга 11.1

```
  readln(j);
  Curr := First; {Перемещаемся в начало списка}
  {Находим запись перед удаляемой из списка}
  for i := 1 to j - 2 do Curr := Curr^.NextRec;
  if j = 1
  then First := First^.NextRec {Если удаляется первая запись,
                                тогда смещаем указатель First}
  else if j = c     {Если удаляется последняя запись...}
       then begin
         Curr^.NextRec := Nil;  {В записи, расположенной перед
                                 удаляемой, очищается поле NextRec}
         Last := Curr^.NextRec; {Перемещаем указатель Last}
       end else  {В остальных случаях указатель должен ссылаться
                  на запись, следующую после удаляемой}
         Curr^.NextRec := Curr^.NextRec^.NextRec;
  ShowList;
  Dispose(First);  {Освобождает область памяти, на которую
                    указывает переменная First}
end.
```

В программе Links для хранения данных об одном человеке используется запись типа Person. Кроме того, объявлен тип Link, являющийся указателем на тип Person.

ПРИМЕЧАНИЕ

Обратите внимание на то, что тип указателя на запись объявляется до объявления типа самой записи.

В записи типа Person есть поле NextRec типа Link, которое предназначено для хранения указателя на следующую запись связанного списка. Для перемещения по списку используются три переменных-указателя: First — указывает на первую запись в списке, Last — указывает на последнюю запись в списке, Curr — указывает на текущую запись в списке.

После того как указана длина списка, при помощи стандартной функции New создается первый объект типа Person. Функция New отводит область памяти для хранения данных, указанных в качестве параметра, и возвращает адрес этой области. Таким образом, при выполнении оператора

```
Last := New(Link);
```

вначале распределяется память для хранения объекта, на который ссылается указатель типа Link (то есть — записи типа Person), а затем адрес этой области памяти сохраняется в переменной-указателе Last. Поскольку на данный момент указатель Last содержит адрес первой записи в списке, то его содержимое присваивается переменной-указателю First.

Затем в цикле for выполняется сохранение данных в текущей записи списка, создание следующей записи с сохранением ее адреса в поле NextRec и переход к новой созданной записи для сохранения данных. Для отображения содержимого связанного списка используется процедура ShowList. В ней, в цикле while просматриваются все записи от первой, на которую указывает переменная First, до последней записи, на которую указывает переменная Last.

Во второй части программы реализовано удаление записи с номером, указанным пользователем. При этом, если удаляется первая запись, то указатель First просто смещается ко второму элементу списка. Если удаляется последняя запись, то в поле NextRec предпоследней записи сохраняется пустой указатель Nil, а указатель Last смещается назад. В том случае, если удаляется одна из промежуточных записей, в поле NextRec предыдущей записи просто сохраняется адрес записи, следующей после удаляемой.

В конце программы вызывается процедура Dispose, освобождающая область памяти, отведенную для хранения данных, на которые ссылается переменная First (то есть — связанный список).

⏩ Остальные процедуры и функции, используемые при динамическом распределении памяти, рассматриваются в приложении Ж.

Глава 12

Объекты

Указатели, рассматриваемые в предыдущей главе, являются основой *объектно-ориентированного программирования* (ООП). ООП построено на использовании структурных единиц под названием "объекты", которые по своей сути напоминают записи. Можно определить, что **объект** — это некоторая совокупность данных, которая рассматривается как единое целое. Объекты отличаются от записей тем, что к ним применимы такие базовые понятия ООП как *инкапсуляция, наследование* и *полиморфизм*. Эти понятия позволяют применять в ООП подходы, используемые при описании объектов в реальном мире.

Инкапсуляция — это объединение внутри объекта его данных с процедурами и функциями, обрабатывающими эти данные. Рассмотрим это на примере объекта Автомобиль. Этот объект характеризуется определенными свойствами, например: цвет, марка, количество горючего в баке и т.д. Внутри этого объекта реализованы механизмы, позволяющие выполнять те или иные действия на основании определенных свойств. Например, для действия Завести двигатель должны выполняться определенные условия: есть горючее, есть зажигание, исправен карбюратор и т.д.

Наследование — это механизм, позволяющий создавать иерархии объектов. При этом свойства объектов, расположенных на более высоких уровнях иерархии, автоматически относятся ко всем порожденным от них объектам. Рассмотрим все тот же пример с объектом Автомобиль. Понятие "автомобиль" достаточно абстрактно. О нем можно с уверенностью сказать, что он обладает только свойствами, общими для всех автомобилей: вес, цвет, объем двигателя и т.п. Для объекта Автомобиль можно создать порожденные объекты, например: Грузовой и Легковой. Оба эти объекта обладают всеми свойствами объекта Автомобиль, а также собственными характерными свойствами. Например, объект Грузовой дополнительно характеризуется грузоподъемностью, а объект Легковой — количеством дверей. В свою очередь, для объекта Легковой можно создать порожденные объекты Седан, Пикап и Лимузин, каждый из которых будет характеризоваться собственными свойствами, а также свойствами объектов Автомобиль и Легковой.

Полиморфизм — это механизм, при котором действие с одним и тем же названием выполняется каждым объектом иерархии по-своему. В примере с автомобилем таким действием может служить пуск двигателя. В дизельных двигателях воспламенение горючего осуществляет посредством сжатия, а в бензинных — при помощи искры. Таким образом, действие Пуск для различных объектов, производных от объекта Автомобиль , может быть реализовано по-разному.

Для описания объектов в языке Pascal используется специальный тип Object:

```
type
  имя_типа = Object
    поле_1: тип_поля;
    ...
```

```
    поле_N: тип_поля;
    метод_1;
    ...
    метод_N;
  end;
```

Поля объектов аналогичны полям записей.

Методы — это процедуры и функции, обрабатывающие поля объектов. Например, для объявления типа, соответствующего объекту Автомобиль, может использоваться следующая конструкция:

```
type
  TCar = Object
    Weight: real;   {Вес в тоннах}
    Color: string[20];    {Цвет}
    MotorV: byte;   {Объем двигателя в куб.дм.}
    procedure Start;      {Действие пуска двигателя}
    procedure Stop; {Действие останова двигателя}
  end;
```

В данном случае Weight, Color и MotorV — это поля (свойства), а процедуры Start и Stop — методы объекта типа TCar ("car" в переводе с английского — "автомобиль").

ПРИМЕЧАНИЕ

Обратите внимание на то, что название типа TCar начинается с заглавной буквы т — это соглашение языка Pascal о присвоении имен типам объектов (хотя, конечно же, при желании можно использовать любые имена).

В результате в программе можно объявить переменную типа TCar:

```
var
  MyCar: TCar;
```

Для обращения к полям и методам такой переменной используется тот же механизм, что и для обращения к полям записей:

```
MyCar.Weight := 2.5;
MyCar.Color := 'Белый';
MyCar.Start;
```

Для динамического создания экземпляров объекта можно использовать указатели:

```
type
  PCar = ^TCar;
var
  MyCar: PCar;
```

В этом случае для выделения памяти под новый объект следует использовать функцию New:

```
MyCar := New(Pcar);
MyCar^.Weight := 1;
MyCar^.Start;
```

Наследование типов

Как уже упоминалось выше, производный тип наследует все свойства и методы "родительского" типа. Это означает, что при определении производного типа унаследованные свойства и методы не описываются. Для создания производных типов используется следующая конструкция:

```
производный_тип = Object(родительский_тип)
```

Например, можно определить тип TSedan, производный от типа TCar:

```
type
  TSedan = Object(TCar)
    Doors: byte;
  end;
```

Все поля и методы типа TCar унаследованы типом TSedan, поэтому следующее их использование корректно:

```
var
  MySedan: TSedan;
...
MySedan.Color := 'Красный';
MySedan.Start;
```

Здесь следует упомянуть о совместимости объектных типов. Под совместимостью подразумевается возможность использования данных одного типа вместо данных другого типа. Это часто используется при передаче параметров в методы. Например, если определена процедура

```
procedure TypeInfo(T: TCar);
```

то в нее можно передать как переменную типа TCar, так и переменную типа TSedan. Если же определена процедура

```
procedure TypeInfo(T: TSedan);
```

то в нее можно пережать только переменную типа TSedan или типа, производного от него. Таким образом, можно сказать, что в языке Pascal допускается *обратная совместимость объектных типов*.

Методы объектов

Как уже упоминалось выше, **метод** — это процедура или функция, включенная в объект. В определение типа включается только заголовок метода, а его реализация создается вне определения объекта. Если метод, определенный в типе объекта, не реализован, то при компиляции будет выдано соответствующее сообщение об ошибке. Учитывая это, все методы, заголовки которых расположены в определении типа объекта, обязательно должны быть реализованы в тексте программы с указанием типа, к которому они принадлежат.

Например, в типе TCar можно объявить метод Filling, соответствующий заправке автомобиля горючим. В этот метод должен передаваться параметр, содержащий объем заливаемого топлива. Кроме того, он должен возвращать объем бензобака, незанятый горючим. Это означает, что метод Filling является функцией. Вот как это будет выглядеть в тексте программы, листинг 12.1.

Листинг 12.1. Пример методов объектов

```
type
  TCar = Object
    ...
    TankV: byte;  {Объем бензобака в литрах}
    Fuel: byte;   {Объем залитого горючего в литрах}
    function Filling(Volume: byte): byte;
  end;
  {Реализация метода Filling}
  function TCar.Filling(Volume: byte): byte;
  begin
    if Volume > TankV then Volume := TankV;
    Fuel := Volume;
    Filling := TankV - Fuel;
  end;
var
  MyCar: TCar;
...
{Вызов метода Filling в программе}
MyCar.TankV := 100;
Writeln(MyCar.Filling(60)); {На экран будет выведено 40}
```

Обратите внимание на то, что в реализации метода Filling обращения к полям объекта типа TCar выполняется без явного указания имени типа (то есть не TCar.TankV, а просто TankV, и не TCar.Fuel, а просто Fuel). Это объясняется тем, что поля TankV и Fuel относятся к тому же типу, в котором определен метод Filling, то есть — к TCar.

Переопределение методов

В производных типах можно переопределять методы, унаследованные от "родительского" типа. Это означает, что в обоих типах используется метод с одним и тем же названием, но с различной реализацией. Например, в типах TCar и TSedan (см. листинг 12.1) можно по-разному определить метод IsFuel, позволяющий определить наличие горючего в бензобаке, листинг 12.2.

Листинг 12.2. Примеры разного определения метода

```
type
  TCar = Object
    ...
    function isFuel: boolean;
  end;

  TSedan = Object(TCar)
    ...
    function IsFuel: integer;
  end;

  function TCar.IsFuel: boolean;
  begin
    IsFuel := (Fuel > 0);
  end;
```

Окончание листинга 11.2

```
  function TSedan.IsFuel: integer;
  begin
    IsFuel := Fuel;
  end;
```

Если в бензобаке есть горючее, то метод, определенный в типе TCar, возвращает значение True. Метод, определенный в типе TSedan, возвращает фактическое количество горючего. Таким образом, метод IsFuel можно использовать одним из двух способов:

```
var
  MyCar: TSedan;
...
Writeln('В бензобаке осталось ',MyCar.IsFuel,' литров');
if TCar(MyCar).IsFuel
then Writeln('Горючее есть')
else Writeln('Горючего нет');
```

Во втором случае переменная MyCar приводится к типу TCar, в результате чего можно вызывать метод IsFuel, определенный в этом типе.

Можно, например, также переопределить в типе TSedan (см. листинг 12.2) метод Filling, изменив набор формальных параметров и внутреннюю реализацию, листинг 12.3.

Листинг 12.3. Пример переопределения метода

```
function TSedan.Filling(Volume, Limit: byte): byte;
begin
  if Fuel > Limit then
  begin
    Writeln('Заправка не требуется');
    Filling := TankV - Fuel;
  end else TCar.Filling(Volume);
end;
```

Параметр Limit определяет минимально допустимый объем горючего, при котором не требуется дозаправка. Если заправка требуется, то вызывается метод Filling "родительского" типа TCar.

Статические и виртуальные методы

Все методы, которые рассматривались до сих пор, были статическими. Это означает, что в том случае, если в производном типе не был переопределен некоторый метод, он будет работать с данными объекта "родительского" типа. Другими словами, в случае вызова унаследованного (не переопределенного) статического метода на самом деле вызывается метод, который может не иметь доступа к некоторым данным вызывающего объекта, что может привести к нежелательным последствиям в ходе выполнения программы. Это объясняется тем, что ссылки на статические методы определяются во время компиляции программы, и компилятор относит их к определенному типу. Решением этой проблемы является использование динамических или виртуальных методов, ссылки на которые определяются во время выполнения программы, благодаря чему метод сопоставляется именно с типом вызывающего объекта.

Для определения виртуальных методов используется зарезервированное слово virtual:

```
procedure Stop; virtual;
```

При этом следует помнить, что если метод объявлен в "родительском" типе как virtual, то все методы с таким же именем в производных типах также должны быть объявлены как виртуальные.

ВНИМАНИЕ

В отличие от статических методов, заголовки всех одноименных виртуальных методов, определенных в производных типах, должны быть идентичны, включая набор параметров и их типы.

Конструктор

Для каждого типа объекта, содержащего виртуальные методы, в оперативной памяти создается так называемая таблица виртуальных методов. Эта таблица содержит указатели на код, соответствующий каждому виртуальному методу, определенному в типе. Связь между экземпляром объекта и его таблицей виртуальных методов устанавливается при помощи конструктора.

Для каждого типа объекта создается только одна таблица виртуальных методов, а отдельные экземпляры объекта содержат только адрес этой таблицы. Значение этого адреса устанавливается специальной процедурой, называемой **конструктором.** В этой процедуре вместо слова procedure используется слово constructor.

Вызов виртуального метода до вызова конструктора приведет к ошибке, так как к этому моменту поле адреса таблицы виртуальных методов еще не инициализировано и содержит неопределенный адрес. Кроме того, в конструкторе выполняются действия по инициализации данных объекта. Если этого не сделать, то при вызове виртуального метода может возникнуть сбой в программе, так как некоторые данные, используемые объектом, могут быть не определены.

Например, в типе TCar (см. листинг 12.2) можно определить конструктор, представленный в листинге 12.4.

Листинг 12.4. Пример определения конструктора

```
type
  TCar = Object
    ...
    constructor Init(tv,f: byte);
  end;

  constructor TCar.Init(tv,f: byte);
  begin
    TankV := tv;
    Fuel := f;
  end;
```

Конструктор должен быть определен в каждом объектном типе, имеющем виртуальные методы. Кроме того, конструктор должен вызываться для каждого экземпляра объекта. Пример **неправильной** инициализации:

```
var
  MyCar, YourCar: TCar;
begin
  MyCar.Init(100, 0);
  YourCar := MyCar; {Некорректная инициализация}
```

Вместо этого следует вызвать конструктор так же и для переменной YourCar:

```
YourCar.Init(80, 80);
```

Сокрытие полей и методов объектов

Одним из важнейших принципов ООП является то, что при разработке программы необходимо ломнить о взаимосвязи данных и обрабатывающих их методов. Если данные и методы не взаимосвязаны, то всегда существует опасность вызова правильного метода с неверными данными или некорректного метода с правильными данными. Доступ к критически важным полям объектных типов рекомендуется осуществлять не напрямую, а только при помощи методов. Примером этого является использование метода TSedan.IsFuel вместо обращения к полю TSedan.Fuel. Присвоение значений полям также должно быть реализовано при помощи методов. Такой подход гарантирует корректное использование данных объекта и их защиту от нежелательного использования.

Кроме того, можно запретить доступ к полям и методам объектов из других модулей. Для этого в языке Pascal используется механизм сокрытия. Скрытые поля и методы доступны только внутри того модуля, в котором определен объект. Те поля и методы, которые следуют непосредственно за заголовком объектного типа или за служебным словом public не имеют никаких ограничений доступа. В отличие от них, поля и методы, объявленные после служебного слова private считаются скрытыми и ограничены использованием только в пределах модуля, например:

```
type
  TMyType = Object
    Field1: integer;       {общедоступное поле}
    procedure Prod;        {общедоступный метод}
    private
      Field2: string;      {скрытое поле}
      procedure Proc2;     {скрытый метод}
    public
      Field3: real;        {общедоступное поле}
      procedure Proc1;     {общедоступный метод}
  end;
```

Динамические объекты

Как и любые другие типы данных, с объектами можно работать динамически при помощи указателей. Одним из самых простых способов размещения объектов в памяти является использование процедуры New:

```
var
  P: ^TCar;
...
New(TCar);
```

Процедура New выделяет в динамической памяти область, достаточную для хранения экземпляра типа, определяемого указателем, и возвращает адрес этой области в указателе.

В случае использования динамических объектов, обращение к их полям и методам осуществляется точно так же, как и обращение к полям динамических записей:

```
P^.Color := 'Черный';
P^.Init(100,100);
```

В языке Pascal допускается также использование в качестве второго параметра процедуры New — вызов конструктора, например:

```
New(P, Init(100,100));
```

Кроме того, как и в случае с другими типами данных, вместо процедуры New может использоваться функция New, возвращающая значение указателя. При этом параметр, передаваемый функции New, должен быть не самим указателем, а его типом:

```
type
  PCar = ^TCar;
var
  P: PCar;
...
P := New(Pcar);
```

Подобно другим типам данных, динамические объекты могут удаляться из памяти при помощи процедуры Dispose:

```
Dispose(P);
```

Деструкторы

Объекты могут содержать указатели на динамические структуры или объекты, которые необходимо удалить из памяти в определенном порядке. В этом случае простого освобождения памяти при помощи процедуры Dispose недостаточно. В языке Pascal эту задачу выполняют специальные методы, называемые деструкторами, в определении которых вместо слова procedure используется слово Destructor. В деструкторах процесс удаления объектов объединяется с другими действиями, которые необходимо выполнить для данного типа объектов. Для одного объектного типа может быть определено несколько деструкторов.

```
type
  TCar = Object
    ...
    destructor Done;
  end;
```

Деструкторы можно *наследовать, и* они могут быть как *статическими,* так и *виртуальными.* Рекомендуется объявлять деструкторы как виртуальные, чтобы обеспечить корректность их вызова для каждого объектного типа. Деструктор, кроме реализованных в нем операций, перед удалением объекта всегда выполняет проверку объема занимаемой этим объектом области памяти при помощи таблицы виртуальных методов. Благодаря этому, при вызове процедуры вида

```
Dispose (P, Done) ;
```

будет гарантировано освобождено корректное количество байтов памяти.

Следует также отметить, что сам по себе деструктор может быть пустым, так как его функции выполняются, благодаря использованию в заголовке слова destructor. Кроме того, деструктор производного типа (например, TSedan) должен последним действием вызывать соответствующий деструктор своего непосредственного "родительского" типа, чтобы освободить поля всех наследуемых указателей объекта. Для

ссылки на все "родительские" типы, в языке Pascal используется специальное зарезервированное слово inherited, после которого через пробел следует имя метода или поля:

```
destructor TSedan.Done;
begin
  inherited Done;
end;
```

Пример использования объектов

Рассмотрим использование объектов на примере программы, преобразующей десятичное число в его двоичное, шестнадцатеричное и восьмеричное представления. Алгоритм преобразования десятичного числа в другую систему счисления одинаков для всех систем. Десятичное число делится на основание системы счисления (для двоичной — 2, для шестнадцатеричной — 16, для восьмеричной — 8). Результату целочисленного деления соответствует старший разряд числа в новом представлении. Затем, если остаток от деления меньше основания системы, то он опять делится на основание системы, и результату деления соответствует предпоследний разряд числа в новом представлении. Этот цикл повторяет до тех пор, пока остаток от деления не станет меньше системы счисления. Этому последнему остатку соответствует первый разряд числа в новом представлении.

Функцию преобразования имеет смысл реализовать в некотором базовом объекте, назовем его DecTo. Затем от объекта DecTo можно создавать порожденные объекты DecToBin (для преобразования в двоичное число), DecToHex (для преобразования в шестнадцатеричное число) и DecToOct (для преобразования в восьмеричное число), в которых будет достаточно только переопределить конструктор для инициализации набора символов, входящих в данную систему счисления. При этом нет необходимости реализовывать процедуру преобразования Trans для каждого объекта, так как она является унаследованной от родительского типа DecTo.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем TransTo.pas и введите в него текст из листинга 12.5, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 12.5. Программа TransTo.pas

```
program TransTo;
uses Crt;
type
  PDecTo = ^DecTo;
  DecTo = Object
    constructor Init(s: string);

    function Trans(Num: word): string; virtual;
    private
      Symbols: string;
      Base: byte;
  end;

  PDecToBin = ^DecToBin;
  DecToBin = Object(DecTo)
    constructor Init;
  end;
```

Продолжение листинга 12.5

```
  PDecToHex = ^DecToHex;
  DecToHex = Object(,DecTo)
    constructor Init;
  end;

  PDecToOct = ^DecToOct;
  DecToOct = Object(DecTo)
    constructor Init;
  end;

  constructor DecTo.Init(s: string);
  begin
    Symbols := s;
    Base := Length(s);
  end;

  constructor DecToBin.Init;
  begin
    inherited Init('01');
  end;

  constructor DecToHex.Init;
  begin
    inherited Init('0123456789ABCDEF');
  end;

  constructor DecToOct.Init;
  begin
    inherited Init ('01234567');
  end;

  function DecTo.Trans(Num: word): string;
  var
    i: integer;
    s: string;
  begin
    s := '',                              ,
    while Num >= Base do
    begin
      s := copy(Symbols,(Num mod Base)+1,1) + s;
      Num := Num div Base;
    end;
    Trans := copy(Symbols,Num+1,1) + s;
  end;

var ,
  P: PDecTo;
  DecN: word;
begin
  ClrScr;
  Write('Введите десятичное число: ');
  Readln(DecN);
  P := New(PDecToBin,Init);
  Writeln('В двоичном представлении ',
          DecN, ' = ', P^.Trans(DecN));
```

Окончание листинга 12.5

```
  Dispose(P);
  P := New(PDecToHex,Init);
  Writeln('В шестнадцатеричном представлении ',
          DecN, ' = ', P^.Trans(DecN));
  Dispose(P);
  P := New(PDecToOct,Init);
  Writeln('В восьмеричном представлении ',
          DecN, ' = ', P^.Trans(DecN));
  Dispose(P);
end.
```

Преобразование выполняется в методе DecTo.Trans. Конструктор Init каждого унаследованного метода вызывает родительский конструктор, передавая в него набор символов своей системы счисления. Набор символов сохраняется в скрытом поле Symbols объекта, производного от DecTo, а основание системы счисления (длина поля Symbols) — в скрытом поле Base.

Для динамического создания и удаления объектов, производных от типа DecTo, для каждого типа объявлен указатель, который затем используется при вызове функции New. Благодаря наследованию, переменная P, объявленная в программе, как указатель на объект типа DecTo, в результате вызова функции New, может использоваться как указатель на объект одного из производных типов.

ЧАСТЬ III
Основы разработки приложений

Приложениями обычно называют сложные программы, которые могут состоять из нескольких программных модулей. С большими приложениями связаны некоторые особенности компиляции, и зачастую они требуют тщательной отладки. Эти вопросы рассматриваются в главе 13, а глава 14 посвящена использованию программных модулей.

В большинстве приложений реализован пользовательский графический интерфейс, поэтому действительно сложные программы немыслимы без использования графических средств модулей `Crt.tpu` или `Graph.tpu`. Оба эти модуля рассматриваются в главе 15.

Глава 16 посвящена такому мощному средству, как встроенный ассемблер, а глава 17 описывает возможности работы с устройствами компьютера при помощи прерываний.

Последняя глава этой части — это краткий вводный курс в средства Turbo Vision, позволяющие разрабатывать приложения с графическим пользовательским интерфейсом на базе объектно-ориентированных библиотек.

Глава 13

Компиляция и отладка программ

Базовый материал о компиляции программ в интегрированной среде Turbo Pascal уже был рассмотрен в главе 1. Кратко напомним, что в этом случае используется команда **Compile | Compile** или комбинация клавиш <Alt+F9>. Кроме того, при выборе команды меню Run | Run или нажатии комбинации клавиш <Ctrl+F9> выполняется компиляция программы с последующим ее выполнением. Если на этапе компиляции будут обнаружены ошибки, то в редакторе интегрированной среды Turbo Pascal отобразится соответствующее сообщение, а курсор будет установлен в строке с ошибкой.

Еще одна команда, **Compile | Information** используется для вывода на экран окна, в котором указывается информация о последней откомпилированной программе, а также о текущем состоянии памяти и окружения. Остальные команды меню **Compile** описаны в следующей главе, посвященной модулям.

В данной главе будут рассмотрены директивы компилятора, а также процесс компиляции программ не в интегрированной среде Turbo Pascal, а в режиме командной строки MS-DOS. В завершение главы будет рассмотрена отладка программ средствами встроенного отладчика интегрированной среды Turbo Pascal.

Директивы компилятора

В тексте программы могут содержаться *директивы компилятора,* которые используются программистом для управления процессом компиляции (например, в главе 7 использовались директивы компилятора `{$F}` и `{$L}`).

Директива — это комментарий специального вида, управляющий ходом выполнения компиляции. Директивы компилятора, как и любые другие комментарии, заключаются в фигурные скобки, однако в них используется специальный символ $.

Все директивы делятся на три категории.

- **Директивы-переключатели.** Включают или отключают определенные свойства компилятора при помощи знаков "+" и "–", указанных после имени директивы (например, `{A+}` или `{$F-}`).
- **Директивы-параметры.** Влияют на параметры компиляции.
- **Условные директивы.** Управляют компиляцией отдельных частей исходного текста программы в соответствии с некоторым условием.

Директивы-переключатели

Директивы-переключатели бывают локальными и глобальными. Локальные директивы оказывают влияние на компиляцию только того программного блока, в котором они указаны. Глобальные директивы указываются в разделе объявлений программы и влияют на весь процесс компиляции.

Директива {$A}

Глобальная директива. Значение по умолчанию — `{$A+}`. Используется для выравнивания данных. Состояние директивы `{$A+}` приводит к тому, что все перемен-

ные и типизированные константы длиной больше одного байта выравниваются по границам машинного слова.

Директива {$B}

Локальная директива. Значение по умолчанию — {$B-}. Используется для оценки логических выражений. Состояние директивы {$B+} приводит к тому, что компилятор создает код для полной оценки логических выражений. Это означает, что всегда выполняется оценка операндов, входящих в логические выражения, составленные при помощи операторов and и or (даже если результат всего выражения уже известен). Если эта директива находится в состоянии {$B-}, то оценка прерывается, как только становится очевидным результат всего выражения.

Директива {$D}

Глобальная директива. Значение по умолчанию — {$D+}. Используется для активизации или отмены генерирования отладочной информации. К отладочной информации относится таблица с пронумерованными строками для каждой процедуры. В этой таблице каждой строке исходного кода ставится в соответствие адрес объектного кода. Если эта директива находится в состоянии {$D-}, то при работе с программой можно использовать независимые или встроенные в интегрированную среду Turbo Pascal средства отладки. Отладочная информация увеличивает размер откомпилированных модулей .tpu, однако никак не влияет на размер и скорость выполнения результирующего файла .exe.

Директива {$E}

Глобальная директива. Значение по умолчанию — {$E+}. Используется для связи программы с библиотекой функций, эмулирующих работу числового сопроцессора 80x87. Эта директива используется совместно с директивой {$N} (смотри ниже).

Директива {$F}

Локальная директива. Значение по умолчанию — {$F-}. Используется для активизации дальних вызовов внешних процедур и функций. Процедуры и функции, откомпилированные при состоянии директивы {$F+}, всегда используют *дальнюю модель вызова* (far). Это означает, что эти откомпилированные процедуры могут находиться во внешних модулях.

Директива {$I}

Глобальная директива. Значение по умолчанию — {$I+}. Используется для активизации проверки операций ввода-вывода. Если эта директива находится в состоянии {$I+}, и в результате вызова процедур ввода-вывода возвращается ненулевой результат, то работа программы прерывается и на экране отображается сообщение об ошибке. Если эта директива находится в состоянии {$I-}, то для проверки ошибок ввода-вывода необходимо использовать функцию IOResult.

Директива {$L}

Глобальная директива. Значение по умолчанию — {$L+}. Используется для генерирования информации о локальных символах. К локальным символам относятся имена и типы всех локальных переменных и констант в модуле, а также символы внутри процедур и функций. Если при помощи директивы {$L+} для программы или модуля генерируются локальные символы, то для проверки значений локальных пе-

ременных можно использовать независимые или встроенные в интегрированную среду Turbo Pascal средства отладки. Отладочная информация увеличивает размер откомпилированных модулей, однако никак не влияет на размер и скорость выполнения результирующего файла .exe.

Директива {$N}

Глобальная директива. Значение по умолчанию — {$N-}. Используется для активизации числового сопроцессора. Если эта директива находится в состоянии {$N-}, компилятор генерирует код для программного выполнения операций с вещественными числами при помощи вызовов специальных функций. В состоянии директивы {$N+} компилятор генерирует код для выполнения операций с вещественными числами при помощи числового сопроцессора 80x87. Это позволяет использовать четыре дополнительных вещественных типа: Single, Double, Extended и Comp.

Если в программе используется сочетание директив {$N+, $E+}, то при компиляции будет выполнена полная программная эмуляция сопроцессора 80x87. В результате программа может выполняться на любом компьютере, независимо от того, используется в нем сопроцессор 80x87 или нет. Если в программе используется сочетание директив {$N+, $E-}, то компилятор обращается к специальной библиотеке функций, которые могут использоваться только на компьютерах с сопроцессором 80x87.

Директива {$O}

Глобальная директива. Значение по умолчанию — {$O-}. Используется для генерирования оверлейного кода.

ПРИМЕЧАНИЕ

Оверлеи — это части программы, которые совместно используют общую область оперативной памяти. В один и тот же момент времени в памяти может находиться тот или иной оверлей в зависимости от выполняемых функций. В процессе выполнения программы эти части могут замещать друг друга в памяти. Оверлеи используются крайне редко, и потому в этой книге не рассматриваются.

Если эта директива находится в состоянии {$O+}, то во время компиляции программы выполняется генерирование специального кода при передаче строковых и постоянных параметров из одной оверлейной процедуры или функции в другую.

Директива {$P}

Глобальная директива. Значение по умолчанию — {$P-}. Используется для контроля над параметрами-переменными типа string. Если эта директива находится в состоянии {$P+}, то параметры-переменные типа string считаются **открытыми.** Это означает, что фактическим параметрам может соответствовать переменная любого строкового типа, а внутри процедуры или функции максимальная длина формальных параметров совпадает с максимальной длиной фактических параметров.

Директива {$Q}

Локальная директива. Значение по умолчанию — {$Q-}. Используется для контроля над переполнением разрядности при выполнении операций над целыми числами. Если эта директива находится в состоянии {$Q+}, то результат каждой операции проверяется на принадлежность допустимому диапазону значений. Директива {$Q+} замедляет выполнение программы и увеличивает ее размер, поэтому ее рекомендуется использовать только на этапе отладки.

Директива {$R}

Локальная директива. Значение по умолчанию — {$R-}. Используется для генерирования кода контроля над интервалом значений. Если эта директива находится в состоянии {$R+}, то выполняется проверка значений индексов при выполнении всех операций с массивами и строками, а также проверяется корректность значений переменных простых и интервальных типов.

Директива {$S}

Локальная директива. Значение по умолчанию — {$S+}. Используется для генерирования кода контроля над переполнением стека. **Стек** — это область памяти, в которой последовательно сохраняются для временного хранения значения локальных переменных при вызове процедур и функций. При этом используется характерный для стека принцип "зашел первым, вышел последним".

Если эта директива находится в состоянии {$S+}, то компилятор в начале каждой процедуры и функции генерирует код для проверки объема свободного места в стеке. Если места недостаточно, то вызов процедуры или функции прервет выполнение программы, а на экране отобразится сообщение об ошибке. Если программа была откомпилирована с директивой {$S-}, то переполнение стека при вызове процедуры или функции может привести к отказу в работе операционной системы.

Директива {$T}

Локальная директива. Значение по умолчанию — {$T-}. Используется для контроля над типом значения указателя, полученного при помощи оператора @. Если эта директива находится в состоянии {$T-}, то результатом применения оператора @ всегда будет нетипизированный указатель. Если оператор @ применяется при включенной директиве {$T+}, то его результатом будет указатель, совместимый только с другими указателями, указывающими на тип соответствующих переменных.

Директива {$V}

Локальная директива. Значение по умолчанию — {$V+}. Используется для проверки типа строк, передаваемых в качестве параметров-переменных. Если эта директива находится в состоянии {$V+}, то тип фактических и формальных строковых параметров должен совпадать. Если эта директива находится в состоянии {$V-}, то в качестве фактического параметра может выступать любая строковая переменная, даже если ее максимальная длина не совпадает с максимальной длиной соответствующего формального параметра.

Директива {$X}

Глобальная директива. Значение по умолчанию — {$X+}. Используется для активизации или отключения расширенного синтаксиса Turbo Pascal. Если в программе указана директива {$X+}, то с функциями можно работать как с процедурами. Это означает, что возвращаемый ими результат можно игнорировать. Директива {$X+} не влияет на работу встроенных функций, реализованных в модуле System. Кроме того/ переключатель {$X+} обеспечивает поддержку строк типа PChar.

Директива {$Y}

Глобальная директива. Значение по умолчанию — {$Y+}. Используется для генерирования информации о ссылках на символы. Информация о ссылках на символы представляет собой таблицы с номерами строк всех объявлений символов в модуле и

ссылок на них. Переключатель {$Y+} приводит к генерированию информации о ссылках на символы, в результате чего увеличивается размер откомпилированных модулей. Однако на размере и скорости выполнения результирующего файла .exe это никак не отражается. Директива {$Y+} не имеет никакого эффекта до тех пор, пока не будут установлены переключатели {$D+} и {$L+}.

Каждой директиве-переключателю соответствует флажок диалогового окна **Compiler Options** (рис. 13.1), раскрываемого при помощи команды **Options | Compiler.** Эти соответствия перечислены в табл. 13.1.

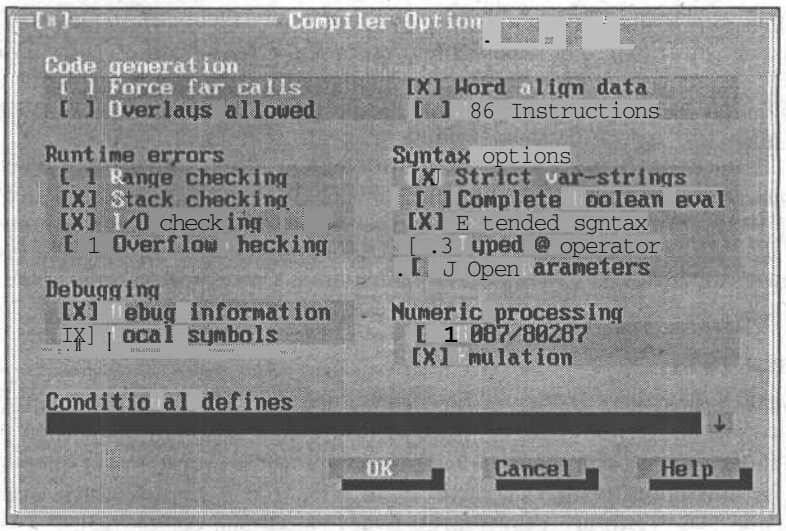


Рис. 13.1. Диалоговое окно **Compiler Options**

Таблица 13.1. Соответствие директив-переключателей и флажков диалогового окна **Compiler Options**

Директива	Флажок окна Compiler Options	Директива	Флажок окна Compiler Options
{$A}	Word Align Data	{$O}	Overlays allowed
{$B}	Complete boolean **eval**	{$P}	Open parameters
{$D}	Debug information	{$Q}	Overflow checking
{$F}	Force far calls	{$R}	Range checking
{$E}	Emulation	{$S}	Stack checking
{$G}	286 instructions	{$T}	Typed @ operator
{$I}	I/O checking	{$V}	Strict var-strings
{$L}	Local symbols	{$X}	Extended syntax
{$N}	8087/80287		

Директивы-параметры

Директивы-параметры определяют характер выполнения компиляции программы. Так же как и директивы-переключатели, они могут быть локальными и глобальными. Их особенность заключается в том, что после имени директивы указываются значения соответствующих параметров компилятора. Рассмотрим некоторые из них.

Директива {$C}

Глобальная директива. Используется для управления атрибутами сегмента кода. **Сегмент кода** — это область оперативной памяти, в которой размещается откомпилированный код программы. Каждый сегмент кода обладает рядом атрибутов (флажков), определяющих его поведение при загрузке программы. Перечислим эти атрибуты.

- MOVEABLE. Операционная система может изменять размещение сегмента кода в оперативной памяти.
- FIXED. Операционная система не может изменять размещение сегмента кода в оперативной памяти.
- PRELOAD. Сегмент кода загружается при запуске программы.
- DEMANDLOAD. Сегмент кода загружается только по мере необходимости.
- PERMANENT. Сегмент кода после загрузки остается в памяти.
- DISCRDABLE. Сегмент кода, если он больше не нужен, может быть освобожден.

По умолчанию директива {$C} имеет следующий синтаксис:

```
{$C MOVEABLE DEMANDLOAD DISCARDABLE}
```

Директива {$D}

Глобальная директива. Используется для вставки текста в заголовок выполняемого файла .exe. Пример использования:

```
{$D Myprogram}
```

Директива {$I}

Локальная директива. Указывает компилятору имя внешнего файла для включения этого файла в процесс компиляции. По умолчанию включаемые файлы имеют расширение .pas. Пример использования:

```
{$I InFile1.pas}
```

Если для включаемого файла не указан каталог его размещения, то интегрированная среда Turbo Pascal ищет его сначала в текущем каталоге, а затем — в каталогах, указанных в поле **Include Directories** диалогового окна **Directories** (см. рис. 1.4).

ПРИМЕЧАНИЕ

Включаемые файлы — это не модули, поэтому в них не используются слова program или unit. Они используются для разбиения текста больших программ на несколько частей и содержат фрагмент программного кода (возможно, даже один оператор), который вставляется в программу в том месте, где указана соответствующая им директива {$I}.

Директива {$L}

Локальная директива. Указывает компилятору о необходимости связи подпрограмм, объявленных как внешние при помощи слова external, с файлом, содержащим программный код на языке ассемблера. В качестве подобных внешних файлов используются объектные файлы с расширением .obj. Пример использования:

```
procedure SetMode(Mode: Word); external; {$L CURSOR.OBJ}
```

Если для объектного файла не указан каталог его размещения, то интегрированная среда Turbo Pascal ищет его сначала в текущем каталоге, а затем — в каталогах, указанных в поле **Object Directories** диалогового окна **Directories** (см. рис. 1.4).

Директива {$O}

Локальная директива. Указывает, какие" модули программы должны быть превращены в оверлеи (то есть размещены не в файле .exe, а в файле .ovr). Пример использования:

```
{ ($0 Unit1.pas Unit2.pas}
```

Эта директива должна находиться в программе сразу же после раздела uses. При этом модули, размещаемые в файле .ovr, должны быть откомпилированы с директивой-переключателем {$O+}.

Условные директивы

Выборочная компиляция частей программы на основании некоторого условия основана на оценке выражений, указанных в специальных условных директивах. Использование условных директив напоминает обычный условный оператор if.

- {$DEFINE}. *Определяет* условный символ с указанным именем. Затем на основании этого символа, при компиляции осуществляется *выбор* того или иного фрагмента текста программы.
- {$IFDEF}. Приводит к компиляции расположенного после нее фрагмента программы, если *определен* указанный условный символ.
- {$IFNDEF}. Приводит к компиляции расположенного после нее фрагмента программы, если указанный условный символ *не определен*.
- {$ELSE}. Приводит к компиляции или пропуску фрагмента программа, аналогично блоку операторов else в конструкции if-else.
- {$ENDIF}. *Определяет* завершение фрагмента программы, к которой применяется условная компиляция.
- {$UNDEF}. *Отменяет* ранее определенный условный символ с указанным именем.

Рассмотрим использование условных директив на примере. Создайте в интегрированной среде Turbo Pascal файл, сохраните его под именем MultiLan.pas и введите в него текст из листинга 13.1, или откройте этот файл с дискеты при, помощи команды **File | Open** (<F3>).

Листинг 13.1. Программа Multilan.pas

```
program MultiLan;
begin
  {$DEFINE R};
  {$IFDEF R}
    Writeln('Выбран русский язык');
  {$ELSE}
    {$IFDEF U}
      Writeln('Вибрано українську мову');
    {$ELSE}
      Writeln('The English is selected');
    {$ENDIF}
  {$ENDIF}
end.
```

Запустите эту программу на выполнение. На экране должна отобразится строка "Выбран русский язык". Это объясняется тем, что при помощи директивы {$DEFINE} определен условный символ с именем R. Измените эту директиву на

{$DEFINE U} и вновь запустите программу. На экране появится строка, соответствующая украинскому языку. Если в директиве {$DEFINE} определить условный символ с именем E, то будет отображена строка на английском языке.

Компиляция в режиме командной строки MS-DOS

Компилировать программы можно не только, находясь в интегрированной среде Turbo Pascal, но и в режиме командной строки MS-DOS. Для этого используется независимая программа-компилятор tpc.exe, которая размещается в том же каталоге, что и файл запуска интегрированной среды Turbo Pascal turbo.exe. Выйдите из интегрированной среды Turbo Pascal и перейдите в каталог, где расположен файл tpc.exe. Выполните в командной строке MS-DOS команду tpc. В результате на экран будет выведена информация об использовании внешнего компилятора Turbo Pascal (рис. 13.2).

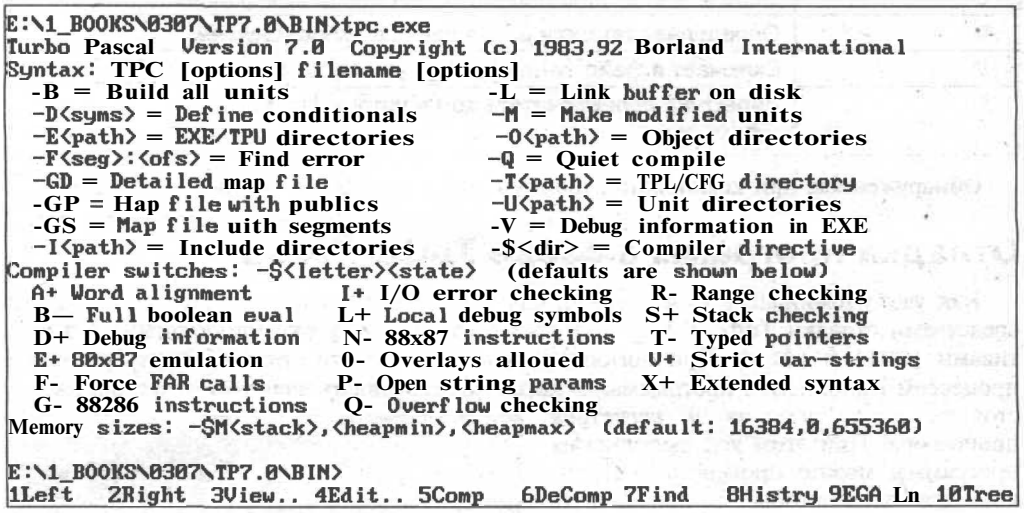


Рис. 13.2. Подсказка для запуска программы-компилятора tpc.exe

Обычно, при запуске программы tpc.exe используется следующий синтаксис:

```
tpc имя_файла параметры
```

Параметры компиляции отделяются друг от друга пробелами. Предположим, необходимо откомпилировать программу, текст которой находится в файле E:\p01.pas, с полной оценкой логических выражений. При этом результирующий файл .exe должен располагаться в каталоге e:\books. Для этого используется следующая команда MS-DOS:

```
tpc e:\p01.pas -Ee:\books -$B+
```

При помощи параметра -E указывает каталог размещения результирующего файл, а при помощи параметра $B — состояние соответствующей директивы компилятора. Некоторые параметры компилятора tpc.exe перечислены в табл. 13.2.

Таблица 13.2. Параметры компилятора tpc. exe

Параметр	Описание
-D	Определяет условные символы. Например, в рассмотренной в предыдущем разделе программе `MultiLan` (листитнг 13.1) можно было бы удалить условную директиву `{$DEFINE}`, а затем откомпилировать ее при помощи следующей команды MS-DOS: `tpc MultiLan.pas -DR` В результате был бы создан файл . exe, при запуске которого на экране отобразилась бы строка на русском языке
-E	Определяет каталоги размещения файлов . exe и . tpu
-I	Определяет каталоги размещения файлов, включаемых при помощи директивы `{$I}`
-O	Определяет каталоги размещения объектных файлов
-U	Определяет каталоги размещения файлов модулей
-V	Включает в файл . exe отладочную информацию
-$	Директива-переключатель компилятора. Например: -$B+ -$A-

Обнаруженные при компиляции ошибки отображаются на экране.

Отладка программ в среде Turbo Pascal

Как уже упоминалось в этой главе, для того чтобы воспользоваться встроенными средствами отладки Turbo Pascal, программа должна быть откомпилирована с директивами `{$D+}` и `{$L+}`. При помощи встроенных средств отладки можно управлять процессом выполнения программы, а также просматривать значения переменных, состояние стека вызовов и регистров процессора. При этом ход выполнения программы можно проверять построчно. Для того чтобы активизировать встроенные средства отладки, выполните команду **Options | Debugger,** в появившемся на экране окне (рис. 13.3) установите флажок **Integrated** (Встроенный) и щелкните мышью на кнопке OK.

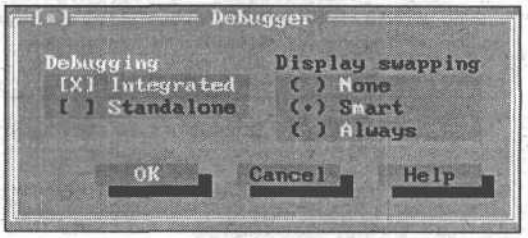


Рис. 13.3. Диалоговое окно **Debugger**

Переход в режим отладки

Для того чтобы перейти в режим отладки, можно воспользоваться одной из команд меню Run, перечисленных в табл. 13.3.

Таблица 13.3. Команды перехода в режим отладки

Команда	Комбинация клавиш	Описание
Run	<Ctrl+F9>	Если в исходном коде установлена точка прерывания (рассматривается в следующем разделе), то выполнение программы останавливается на соответствующей строке

Окончание таблицы 13.3

Команда	Комбинация клавиш	Описание
Step over	<F8>	Построчная отладка без захода в процедуры
Trace into	<F7>	Построчная отладка с заходом в процедуры
Go to cursor	<F4>	Выполнение программы останавливается на строке, в которой до ее запуска был расположен курсор

Для того чтобы выйти из режима отладки можно либо убрать все точки прерывания и нажать комбинацию клавиш <Ctrl+F9> (команда **Run | Run),** либо нажать комбинацию клавиш <Ctrl+F2> (команда **Run | Program reset).** В первом случае работа приложения продолжится, а во втором — прервется.

Точки прерывания

Точка прерывания — это строка исходного кода, на которой выполнение программы останавливается и происходит переход в режим отладки. Точек прерывания может быть столько же, сколько строк исходного текста программы. Если после остановки на точке прерывания нажать комбинацию клавиш <Ctrl+F9>, то программа будет выполняться до следующей точки прерывания или до своего завершения, если же нажать клавишу <F8> или <F7>, то начнется построчная отладка программы.

Установка точки прерывания

Установить точку прерывания в текущей строке можно одним из трех способов.

1. Выполнить команду меню **Debug | Add breakpoint** и щелкнуть мышью на кнопке OK в раскрывшемся диалоговом окне **Add Breakpoint** (рис. 13.4).
2. Щелкнуть правой кнопкой мыши и выбрать в контекстном меню команду **Toggle breakpoint.**
3. Нажать комбинацию клавиш <Ctrl+F8>.

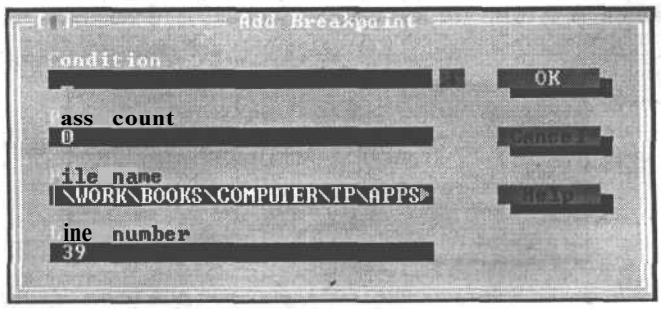


Рис. 13.4. Диалоговое окно **Add Breakpoint**

В поле **Condition** указывается условие, при котором программа останавливает выполнение в данной точке прерывания. В поле **Pass count** указывается, на какой раз должно быть остановлено выполнение программы в данной точке прерывания. В поле **File name** указывается имя файла с исходным текстом программы, а в поле **Line number** — номер строки этого файла, в которой будет установлена точка прерывания. Последние два поля заполняются автоматически: в поле **File name** указывается имя активного файла, а в поле **Line number** — строка, соответствующая текущей позиции курсора.

Использование параметра Condition

Например, установим точку прерывания в программе TransTo, которая была рассмотрена в предыдущей главе (см. листинг 12.5).

- Откройте файл TransTo.pas в интегрированной среде Turbo Pascal.
- Переместите курсор в строку 70 (оператор P := New(PDecToBin,Init);).
- Выполните команду меню **Debug | Add breakpoint.**
- В появившемся на экране диалоговом окне **Add breakpoint,** укажите в поле **Condition** условие **DecN > 9** (рис. 13.5).

Это будет означать, что программа должна остановиться в этой точке прерывания в том случае, если введенное пользователем значение переменной DecN будет больше девяти.

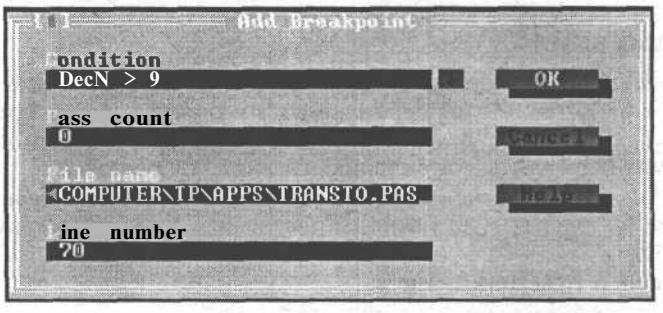


Рис. 13.5. Добавление точки прерывания с условием

Запустите программу TransTo на выполнение и введите для DecN значение меньше 10. Как видите, выполнение программы остановлено не было, так как значение DecN не удовлетворяет условию, определенному для точки прерывания. Запустите программу TransTo на выполнение еще раз, но на этот раз введите для DecN значение больше 9. Теперь выполнение программы будет остановлено и на экране появится окно с сообщением о том, что выполнено условие для текущей точки прерывания (рис. 13.6).

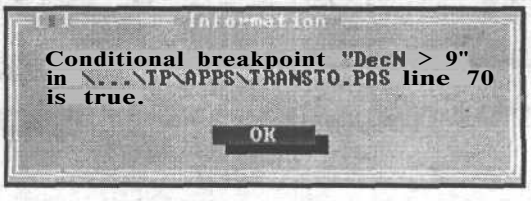


Рис. 13.6. Выполнено условие, определенное для точки прерывания

Текущая выполняемая строка будет выделена светло-зеленым цветом. Если при останове нажать клавишу <F8>, то произойдет переход к следующей строке программы. Если нажать комбинацию клавиш <Ctrl+F2>, то выполнение программы прервется досрочно. Для обычного продолжения выполнения программы нажмите комбинацию клавиш <Ctrl+F9>.

По умолчанию строка с точкой прерывания выделяется красным цветом, однако цветовые настройки можно изменить в диалоговом окне **Colors,** раскрываемом при помощи команды меню **Options | Environment | Colors.**

Удаление точек прерывания

Для того чтобы быстро убрать точку прерывания в текущей строке, можно воспользоваться командой контекстного меню или комбинацией клавиш <Ctrl+F8>. Для удаления и редактирования точек прерываний в интегрированной среде Turbo Pascal также используется специальное диалоговое окно **Breakpoints** (рис. 13.7), которое раскрывается на экране при помощи команды **Debug | Breakpoints.**

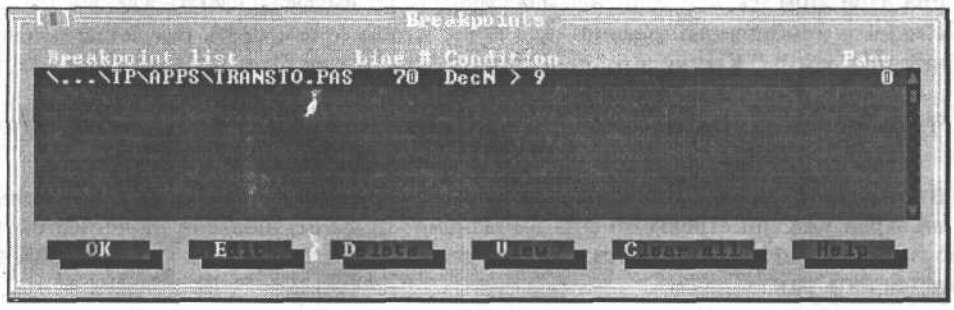


Рис. 13.7. Диалоговое окно **Breakpoints**

В этом окне можно отредактировать одну из точек прерывания (кнопка Edit), удалить ее (кнопка **Delete**), перейти к соответствующей строке в тексте программы (кнопка **View**), а также удалить все точки прерывания (кнопка **Clear all).**

Редактирование точки прерывания

Редактирование точки прерывания выполняется в диалоговом окне **Edit Breakpoint** (рис. 13.8).

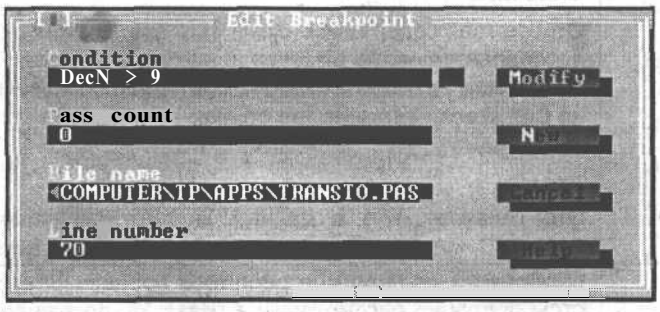


Рис. 13.8. Диалоговое окно **Edit Breakpoint**

Это окно подобно окну **Add Breakpoint** (см. рис. 13.5) за тем исключением, что в нем вместо кнопки OK используются две кнопки: **Modify** (Изменить) и New (Новая). Если щелкнуть мышью на кнопке **Modify,** изменения параметров будут применены к текущей точке прерывания, если же щелкнуть мышью на кнопке New, то будет создана новая точка прерывания с указанными параметрами.

Использование параметра Pass count

В качестве примера использования параметра **Pass count** установим точку прерывания в программе WhileEx, разработанной в главе 6 (см. листинг 6.3).

- Откройте файл `WhileEx.pas` в редакторе интегрированной среды Turbo Pascal.

- Расположите курсор в строке с процедурой Writeln и установите в ней точку прерывания, выполнив команду меню **Debug | Add breakpoint.**
- Введите в поле **Pass count** диалогового окна **Add breakpoint** значение 2. Это означает, что остановка в этой точке прерывания будет выполняться каждый второй раз при достижении программой оператора с процедурой Writeln.
- Запустите программу на выполнение. Ее работа будет остановлена в точке прерывания. При этом текущая выполняемая строка будет выделена светло-зеленым цветом.
- Нажмите комбинацию клавиш <Alt+F5>, чтобы просмотреть результат выполнения программы. Как видите, процедура Writeln уже была вызвана один раз, однако выполнение программы прервано не было.
- Чтобы продолжить выполнение программы, нажмите комбинацию клавиш <Ctrl+F9>.

В следующий раз после остановки на точке прерывания программой будет выведено на экран уже три строки, то есть выполнение будет остановлено на четвертом вызове процедуры Writeln. Таким образом, выполнение программы WhileEx останавливается пять раз.

Окна отладчика

Всего в интегрированной среде Turbo Pascal используется пять окон отладчика: **Breakpoints** (Точки прерывания), см. рис. 13.7; **Call Stack** (Стек вызовов); **Register** (Регистры процессора); **Watches** (Наблюдения); **Output** (Выходные данные). Каждое из этих окон вызывается при помощи соответствующей команды меню **Debug.** Описание окон отладчика (кроме окна **Breakpoints)** представлено в табл. 13.4.

Таблица 13.4. Окна встроенного отладчика Turbo Pascal

Окно	Описание
Call stack	Отображает содержимое стека вызовов. В этом окне указывается последовательность вызовов процедур и функций. Например, для программы `ProcEx` при первом останове на точке прерывания окно **Call stack** содержит две строки: `SortABC(False)` ProcEx Это означает, что в данный момент выполняется процедура SortABC, которая была вызвана из программы ProcEx
Register	Отображает содержимое регистров процессора и флажков. Регистры подробно рассмотрены в главе 15, посвященной встроенному ассемблеру
Watches	В этом окне можно просматривать текущие значения переменных и констант. Для того чтобы добавить какое-либо значение в список окна **Watches** можно прямо в текстовом редакторе среды Turbo Pascal, установить курсор на какой-нибудь переменной или константе и нажать комбинацию клавиш <Ctrl+F7>, или выполнить команду **Add Watch** контекстного меню. В результате отобразится диалоговое окно **Add Watch,** в котором можно просто щелкнуть на кнопке **OK.** Если при вызове окна **Add Watch** курсор в тексте программы не установлен на переменной или константе, тогда требуемый идентификатор придется ввести вручную

Окончание таблицы 13.4

Окно	Описание
Output	Отображает результаты выполнения программы в отдельном окне, которое можно перемещать и масштабировать как любое другое окно интегрированной среды Turbo Pascal

Напоследок, следует упомянуть еще об одной возможности встроенного отладчика Turbo Pascal. Во время отладки программы можно проверять или изменять значения отдельных переменных и выражений. Для этого используется диалоговое окно **Evaluate and Modify.** Чтобы раскрыть на экране это окно можно выполнить команду меню **Debug | Evaluiate/modify,** нажать комбинацию клавиш <Ctrl+F4> или щелкнуть в тексте программы правой кнопкой мыши и выполнить одноименную команду контекстного меню.

Если в окне **Evaluate and Modify** щелкнуть мышью на кнопке **Evaluate,** то в поле **Result** отобразится результат выражения, указанного в поле **Expression.** Для того чтобы изменить текущее значение переменной, указанной в поле **Expression,** необходимо ввести новое значение в поле **New value** и щелкнуть мышью на кнопке **Modify.**

Глава 14

Модули

Максимальный размер программы ограничен. Компилятор позволяет обрабатывать программы, в которых объем данных и генерируемый машинный код не превышают 64 Кбайт. Если программа требует большего количества памяти, то ее фрагменты следует выносить в модули или оверлейные структуры. При этом объем данных и код одного модуля так же не может превышать 64 Кбайт. В главе 7 уже упоминались стандартные библиотечные модули, на которые можно ссылаться в программах при помощи зарезервированного слова uses. В этой главе понятие модулей будет рассмотрено более подробно, включая их структуру, а также компиляцию многомодульных программных проектов.

Модуль — это набор констант, типов данных, переменных, процедур и функций. По сути, — это небольшая программа на языке Pascal, реализованная в отдельном файле, в заголовке которой вместо слова program используется слово unit. Откомпилированные модули сами по себе не являются выполняемыми файлами, а только — фрагментами программного кода, подключаемыми к главной части программы или к другому модулю. Например, после подключения стандартного библиотечного модуля Graph можно использовать реализованные в нем процедуры и* функции, предназначенные для работы с графикой. Таким образом, использование модулей позволяет разбить программу на отдельные логически взаимосвязанные фрагменты, что упрощает анализ сложных программных проектов, а также избежать повторной реализации в различных программах часто используемых процедур и функций.

Исходный текст модулей хранится в файлах с расширением .pas, а откомпилированный код — в файлах с расширением .tpu. Любой программный проект обязательно состоит из одного файла, в заголовке которого указано слово program, и любого количества взаимосвязанных модулей, в заголовке которых указано слово unit. Для того чтобы получить доступ к какому-нибудь модулю, необходимо указать его имя в специальном разделе программы или модуля, обозначаемом словом uses, например:

```
program Prog1;
uses Unit1;  {Ссылка из главного файла программы на
              модуль, исходный текст которого хранится
              в файле Unit1.pas, а откомпилированный
              код — в файле Unit1.tpu}
...

unit Unit1;
uses Unit2;  {Ссылка из модуля Unit1 на модуль Unit2}
...
```

Если в разделе uses указаны ссылки на пользовательские модули, то соответствующие им файлы . tpu должны быть расположены либо в текущем каталоге, либо в

одном из каталогов, указанных в поле **Unit directories** диалогового окна **Directories** (см. рис. 1.4), раскрываемого на экране по команде **Options | Directories.**

Структура модулей

Структуру модуля можно схематически представить следующим образом:

```
unit Unit1; {заголовок модуля}
interface
  {интерфейсный раздел}
implementation
  {раздел реализации}
begin
  {раздел инициализации}
end.
```

В интерфейсном разделе, обозначенном служебным словом interface, указываются элементы модуля, доступные для других модулей и программ. Здесь объявляются функции, процедуры, переменные, константы и типы. Таким образом, для того чтобы получить доступ извне к элементам модуля, необходимо поместить соответствующие объявления в интерфейсный раздел этого модуля.

В разделе реализации, обозначенном служебным словом implementation, указываются элементы, доступные только внутри данного модуля. Здесь также размещается реализация процедур и функций, объявленных в интерфейсном разделе. Таким образом, в интерфейсном разделе располагаются только объявления (заголовки) процедур и функций, а их фактический программный код — в разделе реализации.

В разделе инициализации, расположенном после раздела implementation между служебными словами begin и end указываются операторы, выполняющие начальные установки, необходимые для корректной работы модуля. Эти операторы выполняются при запуске программы в том же порядке, в котором имена модулей указаны в разделах uses. Если операторов инициализации в модуле нет, то слово begin можно опустить.

Рассмотрим использование модулей на примере. Создайте в интегрированной среде Turbo Pascal файл, сохраните его под именем Unitl.pas и введите в него текст из листинга 14.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 14.1. Программа Unit1.pas

```
unit Unitl;
interface
  function Sum(a,b: integer): longint;
implementation
  function Sum(a,b: integer): longint;
  begin
    Sum := a + b;
  end;

  function Avg(a,b: integer): real;
  begin
    Avg := Sum(a,b)/2;
  end;
end.
```

В интерфейсном разделе модуля Unit1 объявлена функция Sum, возвращающая сумму двух целых чисел. Реализация этой функции находится в разделе implementation. В этом же разделе находится реализация еще одной функции Avg, возвращающей среднее арифметическое двух целых чисел. Обратите внимание на то, что слово begin после раздела реализации отсутствует, так как в этом модуле операторы инициализации не используются.

Теперь создайте в интегрированной среде Turbo Pascal главный файл программы, который будет обращаться к модулю Unitl. Сохраните его под именем Units.pas и ведите в него текст из листинга 14.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 14.2. Программа Units.pas

```
program Units;
uses Unitl;
var
  x, y: integer;
begin
  x := 10;
  y := 5;
  Writeln('x + y = ', Sum(x,y));
 end.
```

В разделе uses указана ссылка на модуль Unitl, поэтому в программе можно использовать функцию Sum, объявленную в интерфейсном разделе этого модуля. Запустите программу, чтобы убедиться, что все работает нормально. Теперь добавьте в программу Units следующий оператор:

```
  Writeln('x + y = ', Sum(x,y));
  Writeln('Среднее арифметическое x + y = ', Avg(x,y));
end.
```

Теперь откомпилируйте программу. Компилятор выдаст сообщение об ошибке "Unknown identifier" ("Неизвестный идентификатор") по отношению к имени функции Avg. Это объясняется тем, что эта функция объявлена в модуле Unitl в разделе реализации, а значит, может использоваться только внутри этого модуля. Добавьте в интерфейсный раздел модуля Unitl заголовок функции Avg, и программа будет откомпилирована без сообщения об ошибке.

Теперь создайте еще один модуль — файл с именем Unit2.pas — и введите в него текст из листинга 14.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 14,3. Программа Unit2.pas

```
unit Unit2;
interface
  uses Unit1;
  function Sum3(a,b,c: integer): longint;
implementation
  function Sum3(a,b,c: integer): longint;
  begin
    Sum3 := Sum(Sum(a,b),c);
  end;
end.
```

В этом модуле реализована функция Sum3, возвращающая сумму трех целых чисел. При этом внутри ее используется функция Sum, реализованная в модуле Unit1. Поэтому, для того чтобы иметь доступ к функции Sum, необходимо указать ссылку на модуль Unitl при помощи слова uses в интерфейсном разделе или в разделе реализации (в данном случае это было сделано в разделе реализации).

Внесите в главный файл программы (см. листинг 14.2) следующие дополнения:

```
program Units;
uses Unitl, Unit2;
...
  Writeln('x + y + y = ', Sum3(x,y,y));
end.
```

Теперь, благодаря тому, что в главном файле программы (Units.pas) указана ссылка на модуль Unit2, в нем можно вызывать функцию Sum3.

Проблема циклических ссылок

Предположим, что в созданном в предыдущем разделе модуле Unitl необходимо использовать функцию Sum3, реализованную в модуле Unit2. Для этого, как уже отмечалось, в модуле Unit1 необходимо указать имя модуля в разделе uses. Однако, если это сделать в интерфейсном разделе:

```
unit Unitl;
interface
  uses Unit2;
...
```

ТО при попытке откомпилировать программу будет выдано сообщение об ошибке "Circular unit reference (Unit1)" ("Циклическая ссылка на модуль"). Это означает, что два модуля ссылаются друг на друга в одном и том же разделе (интерфейсном или реализации), что по правилам языка Pascal не допустимо.

Для решения этой проблемы достаточно просто сослаться в модуле Unitl на модуль Unit2 не в интерфейсном разделе, а в разделе реализации:

```
unit Unitl;
interface
  function Sum(a,b: integer): longint;
  function Avg(a,b: integer): real;
implementation
  uses Unit2;
...
```

Теперь компиляция программы будет выполнена без ошибок, и в модуле Unitl можно будет вызывать функцию Sum3, реализованную в модуле Unit2, например, как показано в листинге 14.4.

Листинг 14.4. Пример корректного вызова в модуле функции, которая реализована в другом модуле

```
unit Unitl;
interface
  function Sum(a,b: integer): longint;
  function Avg(a,b: integer): real;
  function Avg3(a,b,c: integer): real;
```

Окончание листинга 14.4

```
implementation
  uses Unit2;
  function Avg3(a,b,c; integer): real;
  begin
    Avg3 := Sum3(a,b,c)/3;
  end;
...
```

Теперь можно добавить еще один оператор в главный файл программы Units:

```
program Units;
...
  Writeln('Среднее арифметическое x + y + y = ', Avg3(x,y,y));
end;
```

В завершение этого раздела разработаем модуль, содержащий функцию проверки существования файла. В дальнейшем этот модуль можно использовать в программах, где перед открытием файла требуется проверить, существует ли он на самом деле. Создайте в интегрированной среде Turbo Pascal новый файл, присвойте ему имя IsFile.pas и введите в него текст из листинга 14.5, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 14.5. Программа IsFile.pas

```
unit IsFile;
interface
  function FileExists(FName: string): boolean;
implementation
  function FileExists(FName: string): boolean;
  var
    F: Text;
    isError: boolean;
  begin
    Assign(F,Fname);
    {$I-}
    Reset(F);
    {$I+}
    isError := IOResult > 0;
    if isError then Writeln('Файла с таким именем не существует!');
    FileExists := not isError;
  end;
end.
```

Откомпилируйте этот модуль. Теперь в программах можно использовать функцию FileExists, указывая в разделе uses ссылку на модуль IsFile.

Компиляция многомодульных программных проектов

В предыдущих главах уже рассматривалась команда меню **Compile | Compile,** а также пункт **Compile | Destination.** В меню **Compile** есть еще четыре команды, используемых для различных вариантов компиляции: **Make, Build, Primary File и Clear primary file,** — а также команда **Information,** которая открывает окно с информацией

о последней откомпилированной программе и текущем состоянии памяти. Рассмотрим команды **Make, Build, Primary file и Clear primary file.**

Команды Compile | Make и Compile | Build

Команда **Compile | Make,** которой соответствует нажатие клавиши <F9>, используется для, так называемой, *сборки* программного проекта. Сборка проекта — это процесс создания выполняемого файла .exe в результате компиляции программы, состоящей из нескольких модулей. При этом соблюдаются следующие условия.

- Если при помощи команды меню **Compile | Primary file** был задан первичный файл проекта, то будет перекомпилирован именно он, в противном случае компилируется файл, котоый активный в текущий момент времени в интегрированной среде Turbo Pascal.
- Если исходный тест данного модуля, хранимый в файле .pas, был изменен со времени создания файла .tpu, то модуль перекомпилируется.
- Если для данного модуля был изменен интерфейсный раздел, то все модули, связанные с ним, также перекомпилируются.
- Если в модуль при помощи директивы {$I} включен внешний файл, который новее, чем файл .tpu этого модуля, то модуль перекомпилируется.
- Если для файла .tpu не найдено соответствующего ему файла .pas, то такой модуль подключается без перекомпиляции.

Команде **Compile | Build** соответствует *полная сборка,* при которой перекомпилируются все файлы программы, независимо от того, когда в них вносились изменения. Эта команда аналогична команде **Compile | Make** за тем исключением, что она не имеет каких-либо условий.

Команды Compile | Primary file и Compile | Clear primary file

Как уже упоминалось выше, команда **Primary File** используется для того, чтобы указать первичный файл проекта — файл .pas, который будет компилироваться при использовании команд **Make или Build.** Если такой файл задан, то он будет всегда перекомпилироваться, независимо от того, в каком файле программы были внесены изменения.

Команда **Clear primary file** отменяет выбор первичного файла проекта, после чего команды **Make и Build** будут применяться к файлу в активном окне редактора интегрированной среды Turbo Pascal.

Глава **15**

Графика

Работа с графикой в программах на языке Pascal возможна в двух режимах: текстовом и графическом. В текстовом режиме экран монитора условно разбит на строки и столбцы, то есть представляет собой сетку, в каждой ячейке которой может быть отображен один символ. Размер этой таблицы зависит от выбранного режима работы монитора. Таким образом, в текстовом режиме графические изображения на экране формируются при помощи сочетаний символов. Подобные изображения еще называют "псевдографикой".

В отличие от текстового, в графическом режиме экран монитора представляет собой совокупность большого количества точек (пикселей). Различные сочетания точек разного цвета и яркости формируют графические фигуры — линии, символы, фигуры и т.д. Размерность экрана, выражаемая в количестве точек по горизонтали и вертикали, зависит от выбранного режима видеоадаптера. ;

В этой главе будет рассмотрена работа с графикой в обоих режимах.

Графика в текстовом режиме

Процедуры и функции, позволяющие расширить возможности отображения информации при помощи управления монитором в текстовом режиме, реализованы в стандартном библиотечном модуле Crt. Таким образом, для того чтобы использовать возможности этого модуля в программе, необходимо указать ссылку на него в разделе uses.

Вывод на экран содержимого текстового файла

Для ускоренного вывода на экран текстовой информации в модуле Crt реализована процедура AssignCrt. По своей сути она напоминает процедуру Assign, однако вывод данных происходит не в текстовый файл на диске, а в файл, связанный с экраном монитора. Рассмотрим это на примере, отображающем на экране содержимое текстового файла. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем FToScr.pas и введите в него текст из листинга 15.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.1. Программа FToScr.pas

```
program FToScr;
uses Crt, IsFile;
var
  FDisk,FScr:. Text;
  s,FName: string;
begin
  ClrScr;  {Очистка экрана}
  repeat
    Write ('Укажите имя файла: ');
```

Окончание листинга 15.1

```
    Readln(FName);
  until FileExists(FName); {Функция FileExists
                            реализована в модуле IsFile}
  Assign(FDisk,FName);
  Reset(FDisk);
  AssignCrt(FScr); {Назначаем вывод текста на экран}
  Rewrite(FScr);
  while not EOF(FDisk) do
  begin
    Readln(FDisk,s);
    Writeln(FScr,s);
  end;
  Close(FScr);
  Close(FDisk);
end.
```

В этой программе подключаются два модуля: Crt и IsFile. Модуль IsFile был разработан в предыдущей главе и содержит только одну функцию — FileExists, которая используется для проверки существования файла на диске. В самом начале программы вызывается процедура очистки экрана ClrScr, реализованная в модуле Crt. Эта процедура после очистки экрана закрашивает его текущим цветом фона (по умолчанию — черным). Затем открывается файл, имя которого введено пользователем, и его содержимое в цикле while переписывается построчно в файл, связанный с экраном монитора.

Текстовые режимы монитора

Возможные текстовые режимы перечислены в табл. 15.1.

Для выбора текстового режима монитора в модуле Crt реализована процедура TextMode, в которую передается целочисленный параметр, соответствующий номеру режима. Например, для выбора режима вывода в цвете на экране размером 80x25 символов используется оператор

```
TextMode(CO80);
```

или

```
TextMode(3);
```

Кроме того, в модуле Crt определена переменная LastMode, хранящая номер последнего выбранного текстового режима.

Таблица 15.1. Текстовые режимы монитора

Номер режима	Константа Crt	Размер экрана	Количество цветов символа	Количество цветов фона
0	BW40	40x25, черно-белый монитор на цветном адаптере	16	8
1	C040	40x25, цветной монитор на цветном адаптере	16	8
2	BW80	80x25, черно-белый монитор на цветном адаптере	16	8
3	C080	80x25, цветной монитор на цветном адаптере	16	8

Окончание таблицы 15.1

Номер режима	Константа Crt	Размер экрана	Количество цветов символа	Количество цветов фона
7	Mono	80x25, черно-белый монитор на монохромном адаптере	3	
256	Font8x8	43 или 50 строк для адаптеров EGA/VGA	16	8

Управление цветом и яркостью символов

Область памяти компьютера, в которой хранится информация об отображаемой на экране информации, называется видеопамятью. Для монохромного режима первый байт видеопамяти расположен по адресу $B000, а в остальных случаях — $B800. Для каждого символа в видеопамяти отводится два байта. В первом байте хранится сам символ (то есть его код по таблице ASCII), а во втором — атрибуты его цвета. Структура байта атрибутов представлена в табл. 15.2.

Таблица 15.2. Структура байта атрибутов цвета символа

Бит	Категория	Описание
0	Бит голубого цвета	Цвет символа
1	Бит зеленого цвета	Цвет символа
2	Бит красного цвета «	Цвет символа
3	Бит яркости	Цвет символа
4	Бит голубого цвета	Цвет фона
5	Бит зеленого цвета	Цвет фона
6	Бит красного цвета	Цвет фона
7	Бит мерцания	Цвет фона

Таким образом, для формирования цвета символа используются четыре бита (с 0 по 3), при помощи которых можно определить 16 (2^4) различных цветов. Для цвета фона используется три биты (с 4 по 6), при помощи которых можно определить 8 (2^3) различных цветов. Если в седьмом бите байта атрибутов хранится двоичная единица, то соответствующий символ на экране будет мерцать.

Каждому цвету в модуле Crt соответствует константа. Перечень цветовых констант и соответствующие им битовые значения представлены в табл. 15.3.

Таблица 15.3. Цветовые константы модуля Crt

Константа	Значение (набор битов)	Цвет
Black	0 (0000)	Черный
Blue	1 (0001)	Синий
Green	2 (0010)	Зеленый
Cyan	3(0011)	Бирюзовый
Red	4 (0100)	Красный
Magenta	5 (0101)	Малиновый
Brown	6(0110)	Коричневый
LightGray	7(0111)	Светло-серый
DarkGray	8 (1000)	Темно-серый

Окончание таблицы 15.3

Константа	Значение (набор битов)	Цвет
LightBlue	9 (1001)	Светло-синий
LightGreen	10(1010)	Светло-зеленый
LightCyan	11 (1011)	Светло-бирюзовый
LightRed	12 (1100)	Светло-красный
LightMagenta	13(1101)	Светло-малиновый
Yellow	14 (1110)	Желтый
White	15(1111)	Белый

Еще одна константа — Blink обозначает мерцание символа, ее значение равно 128 (10000000).

Для установки цвета символов используется процедура TextColor, а для установки цвета фона — процедура TextBackGround. В обе процедуры передается единственный параметр, содержащий значение цвета. Например:

```
TextColor(White); {Цвет символов — белый}
TextBackGround(Blue); {Цвет фона — синий}
```

Для установки мерцания символов можно к значению цвета прибавить значение константы Blink. Например, для получения белого мерцающего цвета используется оператор `TextColor(White+Blink)`. Выражение `White+Blink` равнозначно 00001111 + 1000000 = 10001111. Таким образом, в последний бит записывается единица, что соответствует мерцанию символа.

Текущее значение байта атрибута цвета хранится в специальной переменной модуля Crt — TextAttr. Эту переменную можно использовать вместо процедур TextColor и TextBackGround. Например, для того чтобы установить белый мерцающий цвет символов на синем фоне, можно выполнить один из следующих операторов:

```
TextAttr := $9F; {10011111}
TextAttr := White + Blink + (Blue shl 4);
```

Рассмотрим последний оператор. Как было показано выше, результат выражения White + Blink в двоичной форме равен 10001111. Константе Blue соответствует двоичное значение 0001. Однако, для того чтобы указывать цвет фона, оно должно быть смещено в байте атрибута цветности влево на 4 бита. В результате вычисления выражения Blue shl 4 будет получено двоичное значение 00010000 (рис. 15.1). В результате в байт атрибута цвета будет записано значение 10001111 + 00010000 = 10011111 (шестнадцатеричное число 9F).

```
+ 00001111 — White        00000001 — Blue          + 10001111 — White+Blink
  10000000 — Blink          0000 — Сдвиг             00010000 — Blue shl 4
  10001111 — White+Blink  00010000 — Blue shl 4      10011111 — 9F
```

Рис. 15.1. Схематическое представление вычисления байта атрибута цвета

Разработаем в качестве примера простую программу, отображающую на экране все символы таблицы ASCII с различными значениями атрибута цветности. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем `ClrChars.pas` и введите в него текст из листинга 15.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.2. Программа ClrChars.pas

```
program ClrChars;
uses Crt;
var
  c, StartAttr: byte;
begin
  ClrScr;
  StartAttr := TextAttr;  {Запоминаем начальные цветовые установки}
  for c := 33 to 255 do
  begin
    if (c mod 16) = 0 then Writeln('');
    TextAttr := c;
    Write(Chr(c));
  end;
  TextAttr := StartAttr; {Восстанавливаем цветовые установки}
end.
```

Результат работы программы ClrChars представлен на рис. 15.2.

Рис. 15.2. Результат работы программы ClrChars

Для управления яркостью символов в модуле Crt определены три процедуры: LowVideo, NormVideo и HighVideo. Процедура LowVideo устанавливает режим минимальной яркости, записывая в бит яркости двоичный нуль. Процедура NormVideo устанавливает режим нормальной яркости символов. Процедура HighVideo устанавливает режим максимальной яркости свечения. Некоторые модели мониторов не распознают сигналы интенсивности, поэтому для них применять процедуры LowVideo, NormVideo и HighVideo бесполезно.

Текстовые окна

Модуль Crt позволяет выводить информацию только в определенной части экрана, которая называется **окном.** Величина окна определяется при помощи процедуры Window и не может превышать размер экрана. Процедура Windows принимает четыре параметра: X1, Y1 — координаты левого верхнего угла окна, и X2, У2 — координаты правого нижнего угла окна. Например:

```
Window(1,1,80,25); {Окно во весь экран}
Window(1,1,8,25) ;  {Окно на десятую часть ширины экрана}
```

На экране может отображаться одновременно несколько окон, однако в каждый отдельный момент активным может быть только одно окно. Все процедуры и функции, предназначенные для вывода информации на экран или для определения текущих координат, выполняются только для активного окна.

После вызова процедуры Window координаты текущего окна хранятся в двух переменных модуля Crt типа word — WindMin и WindMax. Переменной WindMin соответствует координата левого верхнего угла окна, а переменной WindMax — координата правого нижнего угла окна. Координата X хранится в младшем байте переменной, а координата Y — в старшем байте. Для извлечения координат верхнего левого угла необходимо применить функции Lo и Hi:

```
X := Lo(WindMin);
Y := Hi(WindMin);
```

Следует помнить, что при работе с переменными WindMin и WindMax отсчет координат начинается не с 1, а с 0.

Для того чтобы изучить на примере рассмотренные выше и другие процедуры, реализованные в модуле Crt, разработаем игровую программу под названием "Сборщик мусора". Создайте в интегрированной среде программирования Turbo Pascal новый файл, сохраните его под именем Gatherer.pas и введите в него текст, представленный в листинге 15.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.3. Программа Gatherer.pas

```
program Gatherer;
uses Crt;
var
  CurX, CurY, CInLine, StartAttr, i,j: byte;
  Total,Gathered: integer;
  'AField: array[1..22,1..80] of char;
  PressedKey: char;

  procedure ShowCount; {Вывод количества собранного
                        мусора в окне заголовка}
  begin
    Window(1,1,80,3);
    TextColor(Yellow);
    GotoXY(40,1);
    Write('Собрано мусора: ',Gathered,' из ',Total);
  end;

  procedure DrawPos;  {Прорисовка сборщика}
  var
    CurChar: Char;
  begin
    {Рисуем сборщика мусора}
    Window(1,3,80,25);
    TextBackGround(Black);
    TextColor(LightRed);
    GotoXY(CurX,CurY);
    Write ('O');
    if AField[CurY,CurX] = '*' then {Если в этой позиции
                                     находится "мусор"...}
    begin
      Inc(Gathered); {Увеличиваем количество собранного мусора на 1}
      AField[CurY,CurX] := ' '; {В массиве "мусора"
                                 затираем текущую позицию}
    end;
    ShowCount; {Обновляем информацию о собранном "мусоре"}
  end;

  {Процедура прорисовки строки подсказки}
  procedure DrawStatus(FColor, BColor: byte; Si string);
  begin
    Window(1,25,80,25);
```

Продолжение листинга 15.3

```
    TextBackGround(BColor);
    TextColor(FColor);
    ClrScr;
    GotoXY(1,25);
    Write(s);
  end;
begin
  ClrScr;
  StartAttr := TextAttr; {Запоминаем текущие цветовые настройки}
  {Рисуем заголовок}
  Window(1,1,80,3);
  TextBackGround(Black); {Цвет заливки}
  TextColor(White);      {Цвет символов}
  ClrScr; {Заливка окна}
  Write(' Игра "Сборщик мусора" ');
  ShowCount; {Показываем количество собранного мусора}
  GotoXY(1,2);
  for i := 1 to 80 do Write('-');
  {Рисуем игровое поле}
  Window(1,3,80,25);
  TextBackGround(Black);
  TextColor(LightGreen);
  ClrScr; {Заполняем поле черным цветом}
  GotoXY(1,1);
  Randomize;  {Активизируем генерацию случайных чисел}
  {Наполняем поле "мусором"}
  for i := 3 to 24 do
  begin
    InsLine; {Вставляем строку}
    CInLine := Random(9) + 1; {Количество "мусора" в строке выбираем
                               случайным образом - от 0 до 10}
    CurX := 1;
    CurY := 1;
    for j := 1 to CInLine do {Заполняем строку "мусором"}
    begin
      CurX := CurX + Random(8) + 1;
      if CurX > 80 then CurX := 79; {Проверка выхода за границу экрана}
      GotoXY(CurX,1);
      Write('*');
      AField[25-i,CurX] := '*'; {Запоминаем позицию "мусора" в массиве}
      {Распределяем "мусор" по равных отрезкам}
      while (CurX mod 8) > 0 do Inc(CurX);
    end;
    Inc(Total,CInLine); {Увеличиваем общее количество "мусора"}
  end;
  {Прорисовка строки состояния}
  DrawStatus(Blue,LightGray,
             ' Для окончания игры -Esc.' +
             ' Перемещение сборщика - клавиши со стрелками. ');
  {Начало игры}
  GotoXY(1,1);
  CurX := 1;
```

Окончание листинга 15.3

```
  CurY := 1;
  repeat
    DrawPos; {Прорисовка позиции сборщика}
    PressedKey := ReadKey; {Считываем клавишу с клавиатуры}
    Window(1,3,80,25);
    if Ord(PressedKey) then break
      {Если нажата клавиша Esc, то выходим из цикла}
    else
    begin
      {Затираем предыдущую позицию сборщика}
      GotoXY(CurX,CurY);
      TextBackGround(Black);
      Write(' ');
      {Изменяем позицию сборщика в зависимости от нажатой клавиши}
      case Ord(PressedKey) of
        72: if CurY > 1 then Dec(CurY); {Стрелка вверх}
        75: if CurX > 1 then Dec(CurX); {Стрелка влево}
        77: if CurX < 80 then Inc(CurX); {Стрелка вправо}
        80: if CurY < 22 then Inc(CurY); {Стрелка вниз}
      end;
    end;
  until Gathered = Total;
  DrawStatus(White+Blink,Red,' Игра окончена!');
  ReadKey;
  TextAttr := StartAttr; {Восстанавливаем текущие цветовые настройки}
  Window(1,1,80,25);
  ClrScr;
end.
```

Результат работы программы Gatherer представлен на рис. 15.3.

Рис. 15.3. Игра "Сборщик мусора"

Эта программа достаточно сложная, поэтому рассмотрим ее по частям. Начнем с описания переменных. Переменные CurX и CurY служат для хранения координат X и

Y текущей позиции "сборщика мусора" в игровом поле. Переменная CInLine используется при построчном заполнении игрового поля "мусором" — в ней сохраняется количество мусора в текущей строке. Переменная StartAttr используется для сохранения атрибутов цвета в начале игры с последующим их восстановлением в конце игры. Переменные i и j — счетчики циклов. Переменная Total используется для хранения общего количества "мусора" в игровом поле, а переменная Gathered — количества собранного "мусора". Массив AField предназначен для хранения данных игрового поля — в ячейках массива сохраняются символы в соответствии с расположением "мусора" на экране. Если в некоторой позиции на экране находится "мусор", то в соответствующую ячейку массива будет записан символ "*". Переменная PressedKey используется для хранения символа, нажатого пользователем во время игры. На основании значения этой переменной определяется дальнейшее действие.

Переходим к самой программе. Она начинается с очистки экрана и сохранения текущих атрибутов цветности в переменной StartAttr. Затем при помощи процедуры Window определяется текстовое окно, занимающее верхние три строки экрана. Для этого окна, при помощи процедур TextBackGround и TextColor, устанавливаются цвет фона и цвет символа (синий и белый соответственно). После заливки окна цветом фона в нем отображается название игры, а затем вызывается процедура ShowCount.

Процедура ShowCount используется для отображения символами желтого цвета в верхней строке экрана количества уже собранного "мусора" и его общее количество. Обратите внимание на процедуру GotoXY. Эта процедура модуля Crt перемещает курсор в позицию с координатами X и Y относительно левого верхнего края текущего окна, определенного при помощи процедуры Window. Если текстовое окно не определено, то позиция определяется относительно левого верхнего угла всего экрана. После вызова процедуры ShowCount курсор перемещается к первому символу второй строки, и эта строка заполняется символами "-".

Теперь наступила очередь прорисовки игрового поля. Для него определено окно Window(1, 3, 80, 25), с черным цветом фона и светло-зеленым цветом символов "мусора" (для обозначения "мусора" использовался символ "*"). Позиции "мусора" должны выбираться случайным образом, поэтому при помощи стандартной процедуры Randomize активизируется генератор случайных чисел. После вызова этой процедуры для получения случайного числа используется функция Random, параметр которой указывает верхний предел диапазона случайных чисел (нижний предел — 0).

В цикле, формирующем игровое поле, в позиции первой строки при помощи процедуры InsLine последовательно вставляются 22 пустые строки. При этом сформированные ранее строки сдвигаются вниз. Количество "мусора" в каждой выбирается случайным образом в диапазоне от 0 до 10. Функция Random может возвращать 0, а в каждой строке должен быть как минимум один символ "мусора", поэтому используется выражение Random(9) + 1. Заполнение строки "мусором" начинается с первой позиции (CurX := 1; CurY := 1). Смещение каждого следующего символа "*" определяется случайным образом в диапазоне от 0 до 9. Для сохранения позиции символа "*" в массиве AField используется оператор AField[25-i, CurX] := '*';. Использование для определения номера строки массива выражения 25-i обусловлено тем, что после вставки очередной строки игрового поля текущая строка сдвигается вниз. Таким образом, если "мусором" заполняется первая строка (i = 1), то в конце концов она окажется последней (25 - 1 = 24); если заполняется вторая строка (i = 2), то в конце концов она окажется предпоследней (25 - 2 = 23) и т.д. Для равномерного распределения "мусора" каждая строка была разбита на отрезки по 8 символов, в каждом из которых может быть только один символ "*". Поэтому, после вывода очередного символа "мусора" на экран и сохранения его позиции в массиве AField, ко-

ордината X текущей строки смещается до начала следующего отрезка. В конце заполнения каждой строки игрового поля общее количество "мусора" увеличивается на количество "мусора" в текущей строке.

После заполнения игрового поля вызывается процедура прорисовки строки состояния (самой нижней строки экрана) DrawStatus. Этой процедуре передается три параметра: цвета фона и символов, а также строка, которую необходимо отобразить внизу экрана.

Теперь наступила очередь рассмотреть организацию самой игры. В начале позиция курсора устанавливается в игровом поле с координатами (1,1). Затем в цикле repeat при помощи вызова процедуры DrawPos прорисовывается "сборщик мусора". Рассмотрим эту процедуру подробнее. В начале этой процедуры определяется окно, соответствующее игровому полю, в качестве цвета фона устанавливается черный, а цвета символа — красный. После этого курсор перемещается в текущую позицию "сборщика мусора", хранимую в переменных CurX и CurY и на экран выводится символ "O" (при желании, для обозначения "сборщика мусора" можете использовать какой-либо другой символ). Затем проверяется текущая позиция "сборщика" по массиву AField. Если в этой позиции находится символ "*" (то есть "мусор"), то количество собранного мусора увеличивается на единицу, а в соответствующую ячейку массива вносится пробел (или какой-либо другой символ, кроме "*"). В завершение процедуры DrawPos вызывается процедура ShowCount.

После отображения "сборщика" в его текущей позиции на экране, программа ожидает нажатия пользователем клавиши на клавиатуре. Для этого используется функция ReadKey, реализованная в модуле Crt. Эта функция возвращает значение типа Char, соответствующее нажатой клавише. Это значение сохраняется в переменной PressedKey, а затем на его основании при помощи оператора if определяется дальнейшее действие. Если была нажата клавиша <Esc> (код 27), то при помощи процедуры break выполняется выход из цикла repeat, и программа завершается. Если нажата какая-либо другая клавиша, тогда текущая позиция "сборщика" затирается пробелом черного цвета, и, в зависимости от того, какая клавиша с изображением стрелки была нажата, изменяется текущая координата CurX или CurY. Для того чтобы курсор не выходил за пределы игрового поля перед каждым изменением значений переменных CurX и CurY выполняется сопоставление с граничными координатами игрового поля. После этого цикл повторяется — "сборщик мусора" отображается в новой позиции, программа ожидает нажатия клавиши и т.д. Если пользователь в процессе игры не нажмет клавишу <Esc>, то игра продолжается до тех пор, пока количество собранного "мусора" не совпадет с его общим количеством.

В завершение программы в нижней строке экрана мерцающими белыми символами на красном фоне отображается строка "Игра окончена!", и экран очищается с восстановленными атрибутами цветности.

Работа в графическом режиме

Возможности текстового режима, реализованные в модуле Crt, достаточно ограничены и не позволяют создавать сложных графических фигур. Поэтому для формирования разнообразных изображений в графическом режиме используется модуль Graph. В этом модуле реализованы десятки процедур и функций, предназначенных для работы с графикой. Как уже упоминалось в начале главы, в графическом режиме экран монитора представляет собой набор точек (пикселей), количество которых по вертикали и горизонтали зависит от выбранного в данный момент видеорежима.

Видеорежим определяется типом видеоадаптера компьютера, основной характеристикой которого является объем **видеопамяти** — памяти, в которой хранятся данные

об экранном изображении. Для работы с каждым из типов видеоадаптеров модуль Graph использует так называемые **графические драйверы** — специальные программы, организующие взаимодействие с видеоадаптером. В среде Turbo Pascal эти драйверы хранятся в каталоге BGI в файлах с расширением .bgi. Прежде, чем приступить к работе в графическом режиме, в программе на языке Pascal обязательно должен быть активизирован тот или иной графический драйвер, что будет рассмотрено чуть ниже.

Каждый видеорежим характеризуется **разрешением** (количеством пикселей по горизонтали и вертикали), а также количеством цветов, которыми может отображаться на экране каждая точка. Существует и еще одна характеристика — видеостраница. **Видеостраница** — это фрагмент видеопамяти фиксированного размера, достаточного для хранения информации о полном экранном изображении. Чем больше объем видеопамяти, тем больше в ней может храниться видеостраниц. На экране всегда отображается только одна видеостраница, которая называется **текущей.** В то время, когда на экране выводится текущая видеостраница, остальные видеостраницы могут заполняться информацией об изображении, которое будет отображено в следующий момент времени. Такой подход позволяет избежать мерцания экрана при медленной прорисовке изображения, и отображать последовательно подготовленные "за кулисами" страницы одну за другой. Страница, на которой в данный момент формируется изображение, называется **активной.**

Информация о видеорежимах представлена в табл. 15.4.

Таблица 15.4. Видеорежимы

Драйвер	Видео-адаптер	Видео-режим	Разре-шение	Количество цветов	Страниц
ATT.BGI	AT&T 6300	ATT400CO	320x200	3	1
		ATT400C1	320x200	3	1
		ATT400C2	320x200	3	1
		ATT400MED	640x200	2	1
		ATT400HI •	640x400	2	1
CGA.BGI	IBM CGA	CGACO	320x200	3	1
		CGAC1	320x200	3	1
		CGAC2	320x200	3	1
		CGAC3	320x200	3	1
		CGAHI	640x200	2	1
	IBM MCGA	MCGAC0	320x200	3	1
		MCGAC1	320x200	3	1
		MCGAC2	320x200	3	1
		MCGAC3	320x200	3	1
		MCGAMED	640x200	2	1
		MCGAHi	640x480	2	1
EGAVGA.BGI	IBM EGA	EGALO	640x200	16	4
		EGAHI	640x350	16	2
		EGA64LO	640x200	16	1
		EGA64HI	640x350	4	1
		EGAMONOHI	720x348	На плате: 64/ 256 Кбайт	1 2

Окончание таблицы 15.4

Драйвер	Видео-адаптер	Видео-режим	Разре-шение	Количество цветов	Стра-ниц
EGAVGA.BGI	IBM VGA	VGALO	640x200	16	4
		VGAMED	640x350	16	2
		VGAHI	640x480	16	1
HERC.BGI	Hercules	HERCMONOHI	720x348	2	1
IBM8514.BGI	IBM 8514	IBM8514LO	640x480	256	
			1024x768	256	
PC3270.BGI	IBM 3270 PC	PC3270HI	720x350	2	1

Инициализация графического режима

Для инициализации графического режима в модуле Graph реализована процедура InitGraph, в которую передается три параметра: номер графического драйвера, номер режима и каталог размещения графических драйверов `.bgi`. Значения констант, соответствующих графическим драйверам и видеорежимам, представлены в табл. 15.5.

Таблица 15.5. Константы, соответствующие графическим драйверам и видеорежимам

Константа драйвера	Значение	Константа видеорежима	Значение
CurrentDriver	-128	Текущий выбранный драйвер	
Detect	0	Автоматическое обнаружение	
CGA	1	CGACO	0
		CGAC1	1
		CGAC2	2
		CGAC3	3
		CGAHI	4
MCGA	2	MCGAC0	0
		MCGAC1	1
		MCGAC2	2
		MCGAC3	3
		MCGAMED	4
MCGA	2	MCGAHI	5
EGA	3	EGALO	0
		EGAHI	1
EGA64	4	EGA64LO	0
		EGA64HI	1
EGAMONO	5	EGAMONOHI	0
IBM8514	6	IBM8514LO	0
		IBM8514HI	1
HercMono	7	HERCMONOHI	0
ATT400 (начало)	8	ATT400CO	0
		ATT400C1	1
		ATT400C2 -.	2

Окончание таблицы 15.5

Константа драйвера	Значение	Константа видеорежима	Значение
ATT400 (окончание)	8	ATT400MED	4
		ATT400HI	5
VGA	9	VGALO	0
		VGAMED	1
		VGAHI	2
PC3270	10	PC3270HI	0

Например, для инициализации 16-цветного видеорежима VGAHI с разрешением 640x480 используется один из двух вариантов вызова процедуры InitGraph:

```
InitGraph(VGA, VGAHI, 'd:\tp\bin\bgi');
InitGraph(9, 2, 'd:\tp\bin\bgi');
```

Если в качестве третьего параметра передается пустая строка (''), то графические драйверы должны быть расположены в текущем каталоге. В случае использования константы Detect (0) инициализация текущего графического драйвера выполняется автоматически. Рассмотрим инициализацию графического режима и выход из него на примере простой программы, отображающей параметры текущего видеорежима.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Graph_01.pas и введите в него текст из листинга 15.4, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.4. Программа Graph_01.pas

```
program Graph_01;
uses Graph, Crt;
var
  DriverVar, ModeVar, ErrorCode: integer;
  DriverName, ModeName: string;
begin
  DriverVar := Detect;  {Драйвер определяется автоматически}
  InitGraph(DriverVar,ModeVar,'\tp\bgi');
    {Проверяем корректность перехода в графический режим}
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln('Ошибка инициализации графического ' +
            'режима: ', ErrorCode);
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);  {Прерываем выполнение программы}
   end;
  DriverName := GetDriverName; {Извлекаем имя выбранного драйвера}
  ModeName := GetModeName(ModeVar); {Извлекаем имя выбранного режима}
  RestoreCRTMode; {Временно переключаемся в текстовый режим}
  Writeln('Вывод в текстовом режиме:');
  Writeln('Выбран драйвер ', DriverName,
          ' (', DriverVar, ')');
  Writeln('Выбран режим ', ModeName, ' (', ModeVar, ')');
  ReadKey; {Ожидаем нажатия клавиши}
  SetGraphMode(ModeVar); {Опять устанавливаем графический режим}
  OutTextXY(10,10,'Вывод в графическом  режиме');
```

Окончание листинга 15.4

```
  OutTextXY(10,30,'Выбран драйвер ' + DriverName);
  OutTextXY(10,50,'Выбран режим ' + ModeName);
  ReadKey;
  CloseGraph;  {Выходим из графического режима}
end.
```

ПРИМЕЧАНИЕ

В примерах, представленных в этом разделе, в качестве пути к каталогу с файлами графических драйверов указано \tp\bgi. Если эти файлы расположены на компьютере в другом каталоге, внесите в программы соответствующие изменения.

Обратите внимание на то, что в разделе uses указаны имена двух подключаемых модулей — Graph и Crt. В данной программе модуль Crt необходим для доступа к процедуре ReadKey, которая ожидает нажатия пользователем какой-либо клавиши.

После автоматической инициализации видеорежима в этой программе вызывается функция GraphResult, возвращающая число 0, если последняя графическая операция была выполнена без ошибки, или число в диапазоне от −14 до −1, если была обнаружена ошибка. Каждому коду ошибки в модуле Graph соответствует определенная константа (табл. 15.6).

Таблица **15.6.** Константы ошибок графического режима

Константа	Значение	Описание
grOK	0	Нет ошибок
grNoInitGraph	-1	Графика не инициализирована
grNotDetected	-2	Графическое устройство не обнаружено
grFileNotFound	-3	Файл драйвера устройства не найден
grInvalidDriver	-4	Неправильный файл драйвера устройства
grNoLoadMem	-5	Недостаточно памяти для загрузки драйвера
grNoScanMem	-6	Выход за пределы Памяти при заполнении
grNoFloodMem	-7	Выход за пределы памяти при заполнении
grFontNotFound	-8	Файл шрифта не найден
grNoFontMem	-9	Недостаточно памяти для загрузки шрифта
grInvalidMode	-10	Неверный графический режим для этого драйвера
grError	-11	Графическая ошибка
grIOerror	-12	Ошибка графического ввода-вывода
grInvalidFont	-13	Неверный файл шрифта
grInvalidFontNum	-14	Неверный номер шрифта

В случае обнаружения ошибки вначале выводится соответствующее сообщение при помощи специальной функции GraphErrorMsg, а затем выполнение программы прерывается при помощи процедуры Halt. Если ошибок обнаружено не было, то программа определяет информацию о видеорежиме при помощи двух функций модуля Graph: GetDriverName и GetModeName. Первая возвращает имя текущего графического драйвера, а вторая — имя текущего видеорежима. Вывод на экран информации, о выбранном в данный момент драйвере и режиме, осуществляется сначала в текстовом режиме при помощи процедур Writeln, а затем — в графическом при помощи процедур OutTextXY. Обратите внимание на использование процедуры

RestoreCRTMode. Она предназначена для временного переключения из графического режима в текстовый. Для обратного перехода используется процедура SetGraphMode, при этом информация, выведенная на экран в текстовом режиме, при переключении в графический режим исчезает. С учетом этого, для просмотра результата вызова процедур Write ln необходимо приостановить выполнение программы при помощи процедуры ReadKey.

Для вывода текста в графическом режиме используется процедура OutTextXY, реализованная в модуле Graph. В качестве первых двух параметров этой функции указываются координаты X и Y (количество пикселей), которые откладываются от верхнего левого угла экрана.

Для выхода из графического режима вызывается процедура CloseGraph.

Построение изображений в графическом режиме

Для построения изображений на экране используется система координат, отсчет в которой начинается от верхнего левого угла экрана. По аналогии с текстовым режимом, графический экран может рассматриваться как одно окно или совокупность нескольких окон. В каждый отдельный момент может быть активно только одно окно. При инициализации графического режима автоматически создается окно, занимающее весь экран.

Для создания окна в графическом режиме используется процедура SetViewPort. Так, например, чтобы определить окно размером 100x100 пикселей, необходимо выполнить следующий оператор:

```
SetViewPort(0,0,100,100,True);
```

Пятый параметр процедуры SetViewPort указывает, будет ли изображение обрезаться по границам окна. Если этот параметр имеет значение True, то изображение не будет выходить за пределы окна. Для получения атрибутов окна используется процедура GetViewSettings, в которую передается переменная-параметр стандартного типа ViewPortType, например:

```
type
  ViewPortType = record
                   x1, y1, x2, y2: integer;
                   Clip: boolean;
                 end;
```

Для очистки всего экрана используется процедура ClearDevice, а для очистки текущего окна — ClearViewPort.

Как уже упоминалось выше, изображение, фактически, формируется на видеостранице, а затем отображается на экране. Для выбора активной видеостраницы используется процедура SetActivePage, а для выбора отображаемой страницы — процедура SetVisualPage.

Палитра

Палитра — это соответствие между кодами цветов и цветами, отображаемыми на экране монитора. Для различных видеоадаптеров могут использоваться различные палитры, то есть одному и тому же коду в различных палитрах могут соответствовать разные цвета. Цветовая палитра для адаптера CGA аналогична таблице кодировке цвета в текстовом режиме (см. табл. 15.3). Для кодирования цвета в адаптерах EGA/VGA используется не четыре бита, как для адаптеров CGA, а шесть. Благодаря этому значительно расширяется гамма доступных цветов, 64 (2^6) вместо 16 (2^4). В

табл. 15.7 представлены кодировки 16 базовых цветов при работе с адаптерами EGA/VGA, а также перечислены имена соответствующих им констант.

Таблица 15.7. Кодировка базовых 16 цветов для адаптеров EGA/VGA

Константа	Значение (набор битов)	Цвет
EGABlack	0 (000000)	Черный
EGABlue	1 (000001)	Синий
EGAGreen	2 (000010)	Зеленый
EGACyan	3 (000011)	Бирюзовый
EGARed	4 (000100)	Красный
EGAMagenta	5 (000101)	Малиновый
EGABrown	6 (000110)	Коричневый
EGALightGray	7(000111)	Светло-серый
EGADarkGray	56 (111000)	Темно-серый
EGALightBlue	57 (111001)	Светло-синий
EGALightGreen	58(111010)	Светло-зеленый
EGALightCyan	59(111011)	Светло-бирюзовый
EGALightRed	60 (111100)	Светло-красный
EGALightMagenta	61 (111101.)	Светло-малиновый
EGAYellow	62(111110)	Желтый
EGAWhite	63(111111)	Белый

Для рисования фигур и вывода текста устанавливается один из цветов, поддерживаемых текущей палитрой, при помощи процедуры SetColor. Для установки цвета фона используется процедура SetBkColor. Если при помощи этих процедур цвета не определены особо, то для рисования фигур и вывода текста используется цвет с максимальным номером в палитре, а для рисования фона — цвет с минимальным номером в палитре.

Для изменения порядка следования цветов в палитре EGA/VGA (только в 16-цветных графических режимах EGA, EGA64 или VGA) используются процедуры SetPalette и SetAllPalette. Процедура SetPalette изменяет только какой-нибудь один цвет в палитре. Например, для того чтобы поменять местами в палитре белый и черный цвета, необходимо выполнить следующие операторы:

```
SetPalette(0,EGAWhite);
SetPalette(63,EGABlack);
```

Процедура SetAllPalette полностью изменяет порядок следования цветов во всей палитре. Ей, в качестве параметра, передается значение стандартного типа PaletteType:

```
type
  PaletteType = record
                  Size: byte;
                  Colors: array[0..MaxColors] of shortint;
                end;
```

Полю Size соответствует размер палитры, а полю Colors — массив цветов. После установки новой палитры все фигуры, отображенные ранее на экране, соответствующим образом изменят свой цвет.

В модуле Graph реализована еще одна процедура — SetRGBPalette, предназначенная для модификации палитр в режимах VGA и IBM8514. В эту процедуру передаются четыре параметра. Первый — это номер изменяемого цвета (от 0 до 255 для режима IMB8514, и от 0 до 15 для режима VGA), а оставшиеся три — это составляющие красного (R), зеленого (G) и синего (B) цветов). Например, для замены черного цвета в палитре чистым зеленым цветом можно воспользоваться следующим оператором:

```
SetRGBPalette(Black,63,0,0);
```

Рассмотрим работу с палитрой на примере программы, в которой выполняется инвертирование цветов.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Graph_02.pas и введите в него текст из листинга 15.5, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.5. Программа Graph_02.pas

```
program Graph_02;
uses Graph, Crt;
var
  i, DriverVar, ModeVar, ErrorCode: integer;
  CurPal: PaletteType;
begin
  DriverVar := EGA;
  ModeVar := EGAHI;
  InitGraph(DriverVar,ModeVar,'\tp\bgi');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  {Рисуем цветные прямоугольники}
  for i := 0 to 15 do
  begin
    SetFillStyle(1,i);  {Выбираем стиль и цвет заливки}
    Bar(i*20, 0, (i+1)*20, 100); {Рисуем заполненный прямоугольник}
  end;
  ReadKey;
  {Инвертируем палитру}
  for i := 0 to 15 do SetPalette(i, 15-i);
  ReadKey;
  {Инвертируем еще раз}
  CurPal.Size := 16;
  for i := 0 to 15 do CurPal.Colors[i] := i;
  SetAllPalette(CurPal);
  ReadKey;
  CloseGraph;
end.
```

Примеры построения изображений

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Graph_03.pas и введите в него текст, представленный в листинге 15.6, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.6. Программа Graph_03.pas

```
program Graph_03;
uses Graph, Crt;
var
  i, j, DriverVar, ModeVar, ErrorCode: integer;
  PressedKey: Char;

  {Процедура формирования многоугольника}
  procedure ShowPolygon;
  var
    PolyPoints: arra'y[l. .10] of PointType;
    i: byte;
  begin
    {Формируем массив с координатами вершин}
    PolyPoints[1].X := 300;
    PolyPoints[1].Y := 50;
    PolyPoints[10].X := 300; {Последняя точка}
    PolyPoints[10].Y := 50;  {совпадает с первой}
    for i := 2 to 5 do begin
      PolyPoints[i].X := PolyPoints[i-1].X + Random(50);
      PolyPoints[i].Y := PolyPoints[i-1].Y + Random(50);
    end;
    for i := 6 to 9 do begin
      PolyPoints[i].X := PolyPoints[i-1].X - Random(50);
      PolyPoints[i].Y := PolyPoints[i-1].Y + Random(50);
    end;
    DrawPoly(10,PolyPoints); {Рисуем многоугольник}
    ReadKey;
    FillPoly(7,PolyPoints); {Заполняем многоугольник по
                             первые 7 вершин}
  end;

begin
  DriverVar := EGA; {Работаем в 16-цветном режиме EGA}
  ModeVar := EGAHI;  {с разрешением 640x350}
  InitGraph(DriverVar,ModeVar,'\tp\bgi');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  Randomize; {Активизируем генератор случайных чисел}

  {Рисуем ряд разноцветных точек}
  OutTextXY(10,10,'Рисование точек');
  for i := 0 to 15 do PutPixel(i*10+10, 30, i);
  ReadKey;

  ClearDevice; {Очищаем экран}
  {Рисуем хаотический набор линий }
  OutTextXY(10,10,'Рисование линий');
  {Определяем графическое окно для рисования линий}
  SetViewPort(10,30,GetMaxX-10,GetMaxY-10,True);
```

Продолжение листинга 15.6

```
  for i := 0 to 15 do
  begin
    SetColor(i); {Устанавливаем цвет линии}
    for j := 0 to 4 do
    begin
      SetLineStyle(j,0,1); {Устанавливаем стиль линии}
      Line(Random(GetMaxX div 2),Random(GetmaxY div 2),
           Random(GetMaxX),Random(GetMaxY));
    end;
  end;
  ReadKey;

  ClearDevice;
  GraphDefaults; {Переустанавливаем графическую систему}
  OutTextXY(10,10,'Последовательное рисование линий');
  OutTextXY(10,30,'Пользуйтесь клавишами с изображением стрелок');
  MoveTo(10,50); {Перемещение в позицию экрана}
  SetColor(White);
  SetLineStyle(SolidLn,0,3);
  {Повторяем цикл до тех пор, пока пользователь
   нажимает клавиши с изображением стрелок}
  repeat
    PressedKey := ReadKey;
    {Рисуем линию от предыдущей точки до точки,
     координата которой смещена на 10 пикселей}
    case Ord(PressedKey) of
      72: LineTo(GetX, GetY - 10); {Стрелка вверх}
      75: LineTo(GetX - 10, GetY); {Стрелка влево}
      77: LineTo(GetX + 10, GetY); {Стрелка вправо}
      80: LineTo(GetX, GetY + 10); {Стрелка вниз}
    end;
  until not (Ord(PressedKey) in [0,72,75,77,80]);

  ClearDevice;
  {Рисуем геометрические фигуры}
  OutTextXY(10,10,'Геометрические фигуры');
  SetLineStyle(SolidLn,0,1);
  OutTextXY(10,60,'Дуга');
  Arc(50,80,0,100,50);
  OutTextXY(150,60,'Полоса');
  SetFillStyle(HatchFill,Green); {Стиль заполнения}
  Bar(200,40,220,80);
  OutTextXY(270,60,'Трехмерная полоса');
  SetFillStyle(SolidFill,Red);
  Bar3D(410,40,430,80,2,TopOn);
  OutTextXY(10,100,'Окружность');
  Circle(50,150,50);
  OutTextXY(150,100,'Эллипс');
  Ellipse(200,150,0,360,20,40);
  OutTextXY(270,100,'Заполненный эллипс');
  SetFillStyle(XHatchFill,Blue);
  FillEllipse(340,150,20,40);
  OutTextXY(450,100,'Прямоугольник');
```

Окончание листинга 15.6

```
  Rectangle(470,110,530,160);
  OutTextXY(10,210,'Сектор');
  SetFillStyle(SolidFill,Yellow);
  PieSlice(50,260,30,100,50);
  OutTextXY(150,210,'Эллиптический сектор');
  Sector(200,270,0,120,20,50);
  ReadKey;

  SetFillStyle(SolidFill,LightGray);
  FloodFill(50,150,White);   {Заполняем окружность}
  Delay(2000);               {Задержка на 2 секунды}
  FloodFill(200,150,White);  {Заполняем эллипс}
  Delay(2000);
  FloodFill(500,150,White);  {Заполняем прямоугольник}
  ReadKey;

  ClearDevice;
  OutTextXY(10,10,'Рисование многоугольника');
  ShowPolygon;
  ReadKey;

  ClearDevice;
  {Работа с текстом в графическом режиме}
  OutTextXY(10,10,'Отображение текста');
  SetTextStyle(TriplexFont,VertDir,1);
  OutTextXY(10,50,'Вертикальный текст со шрифтом 1');
  SetTextStyle(TriplexFont,HorizDir,3);
  OutTextXY(10,30,'Горизонтальный текст со шрифтом 3');
  OutTextXY(200,100,'Текст в рамке');
  Rectangle(190, 90, 200 + TextWidth('Текст в рамке'),
            110 + TextHeight('Текст в рамке'));
  SetTextStyle(DefaultFont,HorizDir,3);
  {Выравнивание текста по горизонтали и вертикали}
  SetTextJustify(CenterText,CenterText);
  OutTextXY(200, 200, 'A');
  SetTextJustify(CenterText,TopText);
  OutTextXY(230, 200, 'A');
  SetTextJustify(CenterText,BottomText);
  OutTextXY(260, 200, 'A');
  SetTextJustify(CenterText,CenterText);
  OutTextXY(400, 200, 'B');
  SetTextJustify(LeftText,CenterText);
  OutTextXY(400, 230, 'B');
  SetTextJustify(RightText,CenterText);
  OutTextXY(400, 260, 'B');
  ReadKey;
  CloseGraph;
end.
```

Некоторые результаты работы программы **Graph_03.pas** представлены на рис. 15.4 и рис. 15.5.

Как **видно,** демонстрационная программа **Graph_03** разбита на несколько частей, отделенных друг от друга процедурой ReadKey. В начале каждой из этих частей вызывается процедура ClearDevice для очистки экрана. Благодаря комментариям,

смысл используемых операторов достаточно очевиден, поэтому затронем только те моменты, которые могут быть не совсем понятны.

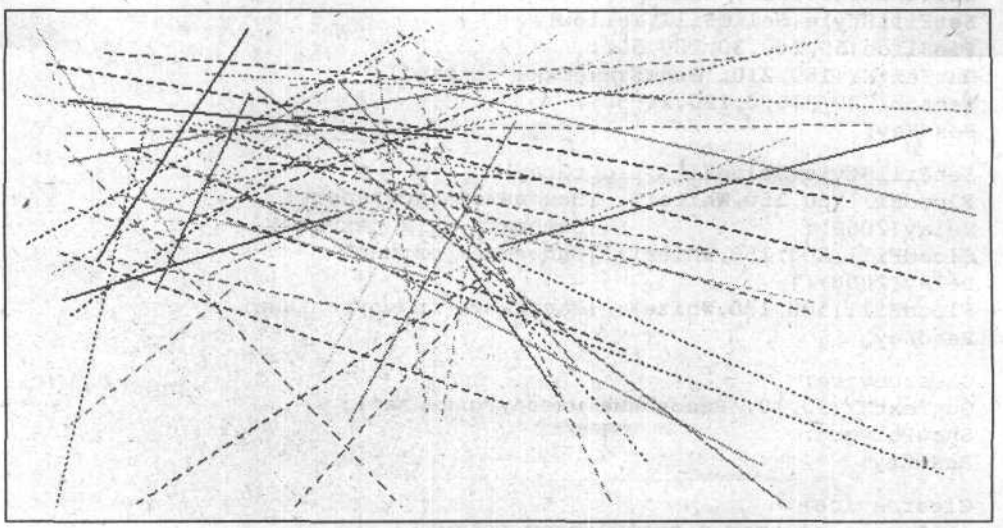


Рис. 15.5. Хаотическое рисование линий при помощи программы Graph_03.pas

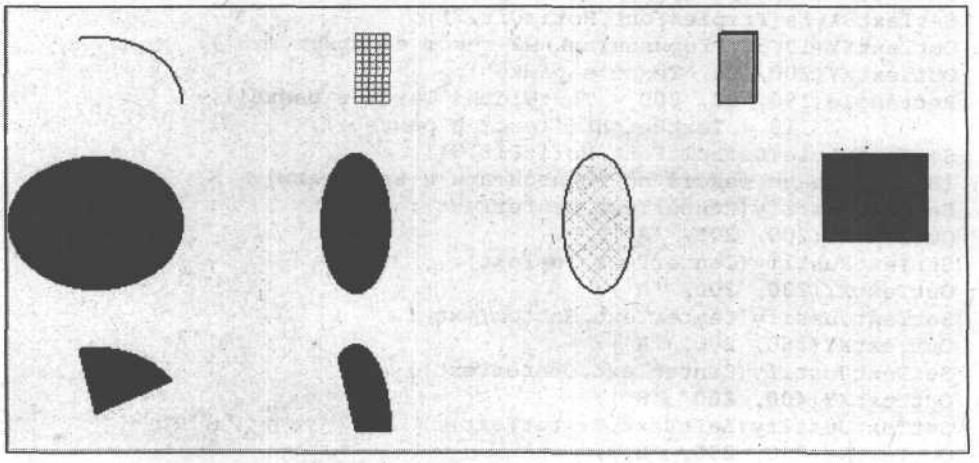


Рис. 15.5. Рисование геометрических фигур при помощи программы Graph_03.pas

- Функции `GetMaxX` и GetMaxY возвращают координаты правого нижнего угла экрана для текущего видеорежима.
- Процедура `GraphDefaults` используется в данной программе для того, чтобы удалить активное графическое окно, определенное ранее при помощи процедуры SetViewPort.
- Процедура `LineTo` отличается от процедуры Line тем, что рисует линию, начиная от текущей точки экрана, а затем перемещает текущую точку в конец нарисованной линии.

- Процедура FloodFill начинает заполнение, цветом рисования, установленным в данный момент, от точки, координаты которой передаются в качестве первого и второго параметра. Заполнение выполняется до тех пор, пока на его пути не будет встречена линия цвета, указанного в качестве третьего параметра.
- В процедуре ShowPolygon используется стандартный тип PointType, представляющий собой запись из двух полей (X и Y) типа Integer. Этот тип используется для хранения координат вершин многоугольника, отображаемого на экране. В данном случае многоугольнику, состоящему из 10 вершин, соответствует массив из 10 значений типа PointType.
- Процедура FillPoly заполняет **многоугольник** текущим цветом заливки. При этом, если в качестве первого параметра этой процедуре передается число, меньше количества вершин, то заполняется только часть многоугольника по соответствующую вершину.
- Первый параметр процедуры SetTextStyle — это тип шрифта. По умолчанию используется тип DefaultFont', которому соответствует шрифт, используемый драйвером клавиатуры. Кроме этого шрифта, русские символы присутствуют также в шрифте TriplexFont, который определен в файле Trip.chr. Для установки внешних шрифтов используется функция InstallUserFont. Например, для установки пользовательского шрифта можно воспользоваться следующими операторами:

```
var
  Fnt: integer;
...
Fnt := InstallUserFont('MyFont.chr');
SetTextStyle(Fnt, HorizDir, 1);
```

Работа с видеостраницами

Как было показано в табл. 15.4, каждый видеорежим характеризуется определенным количеством видеостраниц. Например, в видеорежиме EGAHI используется 2 видеостраницы. В то время как на экране отображается содержимое одной видеостраницы, можно заполнять вторую **видеостраницу**, а затем выполнить переключение на нее. Это позволяет избежать мерцания экрана при реализации движения графических элементов, а также ускоряет процесс прорисовки.

Для работы с видеостраницами используется две процедуры: SetActivePage и SetVisualPage. При помощи процедуры SetActivePage выбирается **активная** страница — то еть такая страница, на которой выполняются все последующие операции рисования. При этом активная страница не обязательно совпадает с текущей (видимой) страницей, выбранной при помощи процедуры SetVisualPage. Проиллюстрируем использование этих процедур на примере программы, реализующей перемещение по экрану круга.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Graph_04 .pas и введите в него текст из листинга 15.7, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 15.7. Программа Graph_04.pas

```
program Graph_04;
uses Graph, Crt;
var
  CurX, CurY, DriverVar, ModeVar, ErrorCode: integer;
```

Окончание листинга 15.7

```
  PressedKey: Char;

  procedure DrawCircle(X, Y, Color: integer);
  begin
    SetColor(Color);
    SetFillStyle(SolidFill,Color);
    FillEllipse(X, Y, 150, 110);
  end;

begin
  DriverVar := EGA;
  ModeVar := EGAHI;
  InitGraph(DriverVar,ModeVar,'\tp\bgi');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  ClearDevice;
  CurX := 200;
  CurY := 200;
  repeat
    DrawCircle(CurX, CurY, Red); {Рисуем круг в новой позиции}
    PressedKey := ReadKey;
    DrawCircle(CurX,CurY,GetBkColor); {Стираем круг в старой позиции}
    case Ord(PressedKey) of
      72: Dec(CurY,10); {Стрелка вверх}
      75: Dec(CurX,10); {Стрелка влево}
      77: Inc(CurX,10); {Стрелка вправо}
      80: Inc(CurY,10); {Стрелка вниз}
    end;
  until not (Ord(PressedKey) in [0,72,75,77,80]);
  CloseGraph;
end.
```

Запустите программу `Graph_04` на выполнение и понажимайте на клавиатуре клавиши с изображением стрелок. Все должно работать правильно, однако у этой программы есть один недостаток — сильное мерцание круга во время его прорисовке. Для решения этой проблемы внесите в текст программы дополнения, выделенные полужирным шрифтом в листинге 15.8.

Листинг 15.8. Новая версия программы `Graph_04.pas`

```
program Graph_04;
uses Graph, Crt;
var
  ...
  CurPage: byte;

  procedure DrawCircle(X, Y, Color: integer);
  begin
    ...
    if Color = GetBkColor
```

Окончание листинга 15.7

```
    then FillEllipse(X, Y, 160, 120)
    else FillEllipse(X, Y, 150, 110);
  end;
begin
  ...
  CurX := 200;
  CurY := 200;
  CurPage := 0;
  SetActivePage(0);
  repeat
    DrawCircle(CurX, CurY, Red); {Рисуем круг в новой позиции}
    SetVisualPage(CurPage);
    PressedKey := ReadKey;
    CurPage := 1 - CurPage; {Переключение между страницами}
    SetActivePage(CurPage);
    DrawCircle(CurX,CurY,GetBkColor); {Стираем круг в старой позиции}
    ...
  until not (Ord(PressedKey) in [0,72,75,77,80]);
  CloseGraph;
end.
```

Запустите новую версию программы Graph_04 на выполнение, и опять понажимайте клавиши с изображениями стрелок. В этом случае будет видно, что мерцание круга значительно меньше, так как процесс удаления старого круга и прорисовки нового происходит как бы "за кулисами" — на невидимой видеостранице.

ПРИМЕЧАНИЕ

Изменения, внесенные в процедуру DrawCircle, обусловлены тем, что прорисовка на каждой из страниц выполняется через раз, поэтому радиус затирания цветом фона должен быть больше на длину одного шага, то есть на 10 пикселей.

Глава 16

Встроенный ассемблер

Ассемблер — это язык программирования низкого уровня. Это означает, что программы, написанные на этом языке, напрямую работают с регистрами процессора. Благодаря средствам Turbo Pascal, поддерживающих встроенный ассемблер, в программы, написанные на языке Pascal, можно вставлять фрагменты программ на языке ассемблера для процессоров 8086 и 80286. В этой книге синтаксис языка ассемблера будет рассмотрен кратко. За дополнительной информацией по этому вопросу можно обратиться к специализированным изданиям.

Для включения ассемблерных фрагментов в программу на языке Pascal можно воспользоваться одним из перечисленных способов.

- При помощи оператора `asm`.
- При помощи процедур и функций с директивой assembler.
- При помощи директивы компилятора `{$I}` и процедуры, объявленной как external.

Оператор asm

Оператор asm может располагаться в любом месте блока операторов и имеет следующую структуру:

```
asm
  ассемблерная_команда
  ассемблерная_команда
  ...
end;
```

Если несколько ассемблерных команд размещены в одной строке, то они отделяются друг от друга символом ";". Кроме того, в строке с ассемблерной командой можно размещать комментарии в стиле языка Pascal.

Ассемблерная команда

Ассемблерная команда имеет следующий синтаксис. Для операций с двумя операндами:

```
1 метка: префикс_операции код_операции операнд, операнд
```

Для операций с одним операндом:

```
метка: префикс_операции код_операции операнд
```

При этом метка и префикс кода операции не являются обязательными элементами ассемблерной команды.

Метки ассемблерных команд — это обычные метки языка Pascal, объявленные в программе в разделе Label, или же особые метки, начинающиеся с символа "@", которые в разделе Label не объявляются.

Префикс операции — это специальная ассемблерная директива, расширяющая смысл используемой операции. Средства встроенного ассемблера Turbo Pascal поддерживают следующие префиксы операций: LOCK, REP, REPE, REPZ, REPNE, REPNZ, SEGCS, SEGDS, SEGES, SEGSS.

Код операции — это обозначение ассемблерной команды или директивы.

ПРИМЕЧАНИЕ

Детальное изучение языка ассемблера выходит за тематические рамки этой книги, однако краткий справочник по командам процессоров 8x86 представлен в приложении Д.

Операнд. Операндом может быть обозначение регистра, адрес области памяти или некоторое значение.

В ассемблерных командах для обозначения регистров используются соответствующие зарезервированные идентификаторы, перечисленные в табл. 16.1.

Таблица 16.1. Идентификаторы для обозначения регистров процессора

Идентификаторы	Описание
AX, BX, CX, DX	16-битовые регистры общего назначения
AL, BL, CL, DL	Младшие байты регистров
AH, BH, CH, DH	Старшие байты регистров
SP, BP, SI, DI	16-битовые указатели или индексы
CS, DS, SS, ES	16-битовые сегментные регистры
ST	Регистр стека сопроцессора 8087

Кроме типов, используемых в языке Pascal, в командах встроенного ассемблера можно использовать дополнительные типы, перечисленные в табл. 16.2.

Таблица 16.2. Дополнительные типы, используемые в командах встроенного ассемблера

Идентификаторы	Описание
BYTE	Значение длиной 1 байт
WORD	Значение длиной 2 байта
DWORD	Значение длиной 4 байта
QWORD	Значение длиной 8 байтов
TBYTE	Значение длиной 10 байтов

NEAR	Используется для ближнего вызова процедур и функций (при ближнем вызове процедура или функция находится в том же модуле, что и вызывающая ее ассемблерная команда). Например:
	`...`
	`procedure MyProc; far;`
	`...`
	`asm`
	`  PUSH CS`
	`  CALL NEAR PTR MyProc;`
	`end;`

FAR	Используется для дальнего вызова процедур и функций

В ассемблерных командах могут также использоваться различные операторы, перечисленные в табл. 16.3.

Таблица 16.3. Операторы, используемые в командах встроенного ассемблера

Оператор	Описание
&	Идентификатор, следующий сразу же за символом "&", рассматривается как определенный пользователем, даже если он совпадает с одним из зарезервированных слов встроенного ассемблера. Синтаксис: & *идентификатор*
[...]	Выражение, заключенное в квадратные скобки, исполняет роль адреса области памяти
HIGH	Возвращает старший байт. Синтаксис: HIGH *выражение*
LOW	Возвращает младший байт. Синтаксис: LOW *выражение*
:	Перед двоеточием ставится идентификатор сегментного регистра, после двоеточия — выражение, результат которого исполняет роль адреса в соответствующем сегменте. Синтаксис для сегмента данных: DS: *выражение* ,
OFFSET	Возвращает смещение (младшие два байта) результата выражения. Синтаксис: OFFSET *выражение*
SEG	Возвращает сегмент (старшие два байта) результата выражения. Синтаксис: SEG *выражение*
TYPE	Возвращает размер результата выражения в байтах. Синтаксис: TYPE *выражение*
PTR	Приводит тип выражения, указанного после него, к типу выражения, указанного перед ним. Синтаксис: *выражение1* PTR *выражение2*
+, -, *, /, MOD	Арифметические операторы
SHL, SHR	Побитовый сдвиг влево и вправо
NOT,AND,OR, XOR	Побитовые операторы

Пример использования оператора asm

Рассмотрим пример использования оператора asm, разработав программу, преобразующую строчные буквы введенной строки в прописные буквы. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем AsmEx_1.pas и введите в него текст из листинга 16.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 16.1. Программа AsmEx_1.pas

```
program AsmEx_1;
uses Crt;
var
  s: string;
  Len: integer;
begin
  ClrScr;
  Write('Введите строку: ');
  Readln(s);
  Len := Length(s);
  asm
```

Окончание листинга 16.1

```
    LEA BX,s+1      {Записываем в BX адрес первого символа в строке}
    MOV CX,Len      {Инициализируем счетчик цикла}
@1: MOV AH,[BX]     {Записываем в AH текущий символ}
    CMP AH,97       {Сравниваем символ с английской 'a'}
    JB @e           {Если < 'a', то на повтор цикла}
    CMP AH,127      {Сравниваем символ с русской 'A'}
    JA @2           {Если >= 'A', то - на метку @2}
    SOB AH,32       {Значит английская строчная - вычитаем из кода 32}
    JMP @m          {Безусловный переход к точке сохранения
                     измененного символа в строке}
@2: CMP AH,160      {Сравниваем символ с русской 'a'}
    JB @e           {Если < 'a', то на повтор цикла}
    CMP AH, 223     {Сравниваем символ с русской 'p'}
    JA @3           {Если >= 'p', то - на метку @3}
    SUB AH,32       {Значит русская от 'a' до 'п' - вычитаем из кода 32}
    JMP @m          {Безусловный переход к точке сохранения
                     измененного символа в строке}
@3: SOB AH,80       {Значит русская от 'p' до 'я' - вычитаем из кода 80}
@m: MOV [BX],AH     {Сохраняем изменения в строке}
@e: INC BX          {Переходим к следующему символу}
    LOOP @1         {Повтор цикла}
  end;
  Writeln(s);
end.
```

Директива assembler

Директиву assembler можно применять к процедуре или функции в тех случаях, когда весь ее блок операторов состоит только из ассемблерных команд. Например, можно реализовать функции умножения и деления на два при помощи побитового сдвига влево и вправо. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните храните его под именем `AsmEx_2.pas` и введите в него текст из листинга 16.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 16.2. Программа `AsmEx_2.pas`

```
program AsmEx_2;
uses Crt;
var
  n: integer;

  function Mul2(n: integer): integer; assembler;
  asm
    MOV AX,n
    SHL AX,1
  end;

  function Div2(n: integer): integer; "assembler;
  asm
    MOV AX,n
    SHR AX,1
  end;
```

Окончание листинга 16.2

```
begin
  ClrScr;
  Write('Введите число: ');
  Readln(n);
  Writeln(n, ' * 2 = ', Mul2(n));
  Writeln(n, ' / 2 = ', Div2(n));
end.
```

Функции, реализованные с директивой assembler, возвращают результат следующим образом.

- Результаты целочисленных и перечислимых типов, а также типов Char и Boolean возвращаются в регистре AL (для 1-байтных значений), в регистре AX (для 2-байтных значений) или в паре регистров DX, AX (для 4-байтных значений).
- Результаты типа real возвращаются в регистрах DX, BX, AX.
- Результаты типов Single, Double, Extended и Comp возвращаются в регистре стека сопроцессора ST.
- Результаты указательного типа возвращаются в паре регистров DX, AX.

Директива {$L}

Как было отмечено в главе 13, директива компилятора {$L} служит для включения в программу на языке Pascal подпрограммы, хранимой в виде объектного кода во внешнем файле с расширением .obj. При этом включаемая процедура должна быть объявлена в программе на языке Pascal с директивой external. Например, если бы функции Mul2 и Div2, реализованные в программе AsmEx_2 (см. листинг 16.2), были ассемблированы в отдельный объектный файл (например, с именем Ops2.obj), то программа AsmEx_2 превратилась бы в программу, приведенную в листинге 16.3.

Листинг 16.3. Версия программы AsmEx_2.pas с использованием объектного файла

```
program AsmEx_2;
uses Crt;
var
  n: integer;
  {$L Ops2.obj}
  function Mul2(n: integer): integer; external;
  function Div2(n: integer): integer; external;

begin
  ClrScr;
  Write('Введите число: ');
  Readln(n);
  Writeln(n, ' * 2 = ', Mul2(n));
  Writeln(n, ' / 2 = ', Div2(n));
end.
```

Глава 17

Доступ к устройствам через прерывания

В основе организации доступа к устройствам компьютера (экран, динамик, принтер, жесткий диск и т.д.) лежат ассемблерные команды, регистры процессора и *прерывания*.

Прерывание — это особая операция (функция), которая приостанавливает работу программы для выполнения специальных системных действий. Управление некоторыми прерываниями выполняется базовой системой ввода-вывода BIOS (Basic Input/Output System), которая хранится в постоянном запоминающем устройстве компьютера ROM (Read Only Memory), а некоторыми — непосредственно операционной системой MS-DOS. Перед вызовом ряда прерываний должны быть записаны данные в определенные регистры процессора. Результат вызова прерываний также может сохраняться в определенных регистрах. Перечень первых сорока прерываний представлен в табл. 17.1.

Таблица 17.1. Прерывания

Номер прерывания	Категория	Описание
00	BIOS	Деление на нуль
01	BIOS	Переход в пошаговый режим при отладке
02	BIOS	Немаскированное прерывание
03	BIOS	Точка прерывания при отладке программы
04	BIOS	Переполнение регистра
05	BIOS	Печать экрана
06	BIOS	Неверный код команды
07	BIOS	Не найдена микросхема для обработки операций с вещественными числами
08	BIOS	Запрос от таймера
09	BIOS	Запрос от клавиатуры
0A	BIOS	Используется в компьютерах AT для каскадирования запросов на прерывание.
0B	BIOS	Запрос от портов COM2 и COM4
0C	BIOS	Запрос от портов COM1 и COM3
0D	BIOS	Запрос от порта LPT2
0E	BIOS	Запрос от дисковода
0F	BIOS	Сигнал от порта LPT1 о том, что он готов принять символ (используется при выводе на принтер)
10	BIOS	Управление дисплеем
11	BIOS	Запрос информации о подключенных устройствах
12	BIOS	Запрос информации об объеме памяти
13	BIOS	Дисковые операции ввода-вывода

Окончание таблицы 17.1

Номер прерывания	Категория	Описание
14	BIOS	Операции ввода-вывода через последовательный порт
15	BIOS	Кассетные операции и специальные функции компьютеров AT
16	BIOS	Взаимодействие с клавиатурой
17	BIOS	Взаимодействие с принтером
18	BIOS	Обращение к интерпретатору языка BASIC, хранящемуся в постоянном запоминающем устройстве
19	BIOS	Перезапуск системы
1A	BIOS	Запрос и установка времени и даты
1B	BIOS	Прерывание от клавиатуры
1C	BIOS	Прерывание от таймера
1D	BIOS	Получение параметров дисплея
1E	BIOS	Получение параметров дисковода
1F	BIOS	Получение таблицы символов, используемых в графическом режиме
20	MS-DOS	Нормальное завершение программы
21	MS-DOS	Обращение к функциям MS-DOS
22	MS-DOS	Адрес, к которому происходит переход после завершения программы
23	MS-DOS	Адрес, к которому происходит переход после нажатия комбинации клавиш <Ctrl+Break>
24	MS-DOS	Адрес, к которому происходит переход в случае возникновения фатальной ошибки
25	MS-DOS	Прямая запись на диск
26	MS-DOS	Прямое чтение с диска
27	MS-DOS	Создание резидентной программы

Большинство прерываний с большими номерами либо зарезервированы для дальнейшего использования, либо исполняют какие-нибудь специализированные функции, предназначенные для обслуживания определенного вида оборудования. В этой главе будут рассмотрены некоторые примеры взаимодействия с устройствами компьютера при помощи базовых прерываний.

▸▸ Описание основных программных прерываний находится в приложении Г.

Прерывания в программах на языке Pascal

Для использования прерываний в программах, написанных на языке Pascal, можно воспользоваться одним из двух способов.

1. При помощи специального типа данных Registers, предназначенного для обращения к регистрам процессора, и процедуры Intr, вызывающей прерывание с указанным номером. Как тип данных Registers, так и процедура Intr реализованы в стандартном библиотечном модуле Dos.
2. При помощи команд встроенного ассемблера. В этом случае для вызова прерывания используется операция INT.

Рассмотрим работу прерываний на примере использования прерывания с номером $05, которое служит для вывода содержимого экрана на печать. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int05.pas и введите в него текст из листинга 17.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.1. Программа Int05.pas

```
program Int05;
uses Crt, Dos;
var
  Regs: Registers;

begin
  Writeln('Печать при помощи функции Intr...');
  Intr($05,Regs);
  Writeln('По окончании печати нажмите любую клавишу');
  ReadKey;
  Writeln('Печать при помощи команды INT...');

  asm
    INT 05H
  end;
end.
```

Как видно в листинге 17.1, в процедуру **Intr**, реализованную в модуле Dos, передается два параметра: номер прерывания в шестнадцатеричной форме и переменная типа **Registers**. Этот тип объявлен в модуле Dos следующим образом:

```
type
  Registers = record
  case Integer of
    0: (AX, BX, CX, Dk, BP, SI, DI, DS, ES, Flags: Word);
    1: (AL, AH, DL, DH, CL, CH, DL, DH: Byte);
end;
```

Для вызова прерывания с номером $05 не требуется какая-либо предварительная установка регистров процессора, поэтому в программе Int05 переменная Regs используется только для соответствия набора формальных и фактических параметров. Однако при вызове многих других прерываний требуется либо записывать данные в определенные регистры, либо считывать из определенных регистров. Например, при помощи прерывания с номером $17 можно работать с принтером. При этом содержимое регистра AH определят функцию, выполняемую прерыванием. В частности, если регистр AH содержит 0, то прерывание $17 передает на принтер один символ, номер которого указан в регистре DX (начиная с 0). Код передаваемого символа хранится в регистре AL. Рассмотрим это на примере программы. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int17.pas и введите в него текст из листинга 17.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.2. Программа Int17.pas

```
program Int17;
uses Crt, Dos;
var
  Regs: Registers;
```

Окончание листинга 17.2

```
  s: string;
  i: byte;
begin
  ClrScr;
  Writeln('Введите строку: ');
  Readln(s);
  for i := 1 to Length(s) do
  begin
    Regs.AH := $00;          {AH — номер функции}
    Regs.AL := Ord(s[i]);    {AL - код символа}
    Regs.DX := $00;          {DX - номер принтера}
    Intr($17,Regs);          {Передача символа на принтер}
  end;
end.
```

Программа `Int17` передает посимвольно на принтер, введенную строку с клавиатуры (программная реализация печатной машинки). Некоторые прерывания (например, `$17`) могут выполнять различные функции в зависимости от номера, записываемого в регистр AH перед вызовом прерывания. Так, прерывание с номером `$17`, кроме печати символа, выполняет еще две функции: инициализацию порта принтера (AH = 1) и чтение состояния порта принтера (AH = 2).

Более подробная информация об этом приведена в приложении Г.

Прерывания BIOS

Система BIOS выполняет обработку прерываний с номерами от $00 до `$1F`. Эти прерывания имеют отношения к работе таких устройств компьютера как таймер, монитор, клавиатура, гибкий диск, принтер и др. В этом разделе будут рассмотрены некоторые практические примеры использования прерываний BIOS. Во всех программах для краткости при вызове прерываний будут использоваться команды встроенного ассемблера.

Управление дисплеем в текстовом режиме

В этом разделе будут рассмотрены конкретные примеры использования прерывания с номером `$10` для прямого доступа к экрану компьютера.

Очистка экрана и установка курсора

Для очистки экрана прерывание `$10` использует функцию под номером `$06`. Эта функция очищает область экрана, высота которой (в строках) указывается в регистре AL. Если регистр AL содержит 0, то очищается весь экран. В регистр BH записывается байт атрибутов цветности (атрибуты цветности рассматриваются в главе 15) для очищаемой области, в регистры CL и CH — координаты X и Y верхнего левого угла, а в регистры DL и DH — координаты X и Y правого нижнего угла очищаемой области (начиная с 0).

Для изменения позиции курсора на экране компьютера используется функция прерывания `$10` с номером $02. При этом номер видеостраницы (начиная с 0) записывается в регистр BH, а координаты курсора — в регистры DL и DH.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем `Int10_1.pas` и введите в него текст из листинга 17.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.3. Программа Int10_1.pas

```
program Int10_1;
  procedure SetCursor (x, y: byte); assembler;
  asm
     {Установка курсора в позицию на экране}
    MOV AH,02H    {Функция установки курсора}
    MOV BH,00H    {На первой видеостранице}
    MOV DL,x
    MOV DH,y
     INT ЮН
  end;

  procedure ClearArea(Rows,Colors,LeftTopX,LeftTopY,
                      RightBottomX,RightBottomY: byte); assembler;
  asm
    MOV AH,06H           {Функция очистки}
    MOV AL,Rows          {Количество строк}
    MOV BH,Colors        {Цветовые атрибуты очищаемой области}
    MOV CL,LeftTopX      {Координаты верхнего левого угла}
    MOV CH,LeftTopY
    MOV DL,RightBottomX  {Координаты правого нижнего угла}
    MOV DH,RightBottomY
     INT 10H             {Вызов прерывания 10H}
  end;

begin
  ClearArea(0,$07,0,0,79,24); {Очистка всего экрана}
  SetCursor(0,0);
  Writeln('Светло-серые символы на черном фоне');
  ClearArea(3,$1E,0,1,35,3); {Очистка области экрана}
  SetCursor(0,2);
  Writeln('Желтые символы на синем фоне');
end.
```

В программе IntlO_l очистка области экрана выполняется в ассемблерной процедуре ClearArea, в которую передаются следующие параметры: количество очищаемых строк (Rows), байт атрибутов цветности (Colors), и координаты верхнего левого и нижнего правого углов очищаемой области. Младшим четырем битам байта атрибутов цветности соответствует цвет символов, а старшим четырем — цвет фона. Так, в первом случае байт $07 означает, что очищенная область будет заполнена черным цветом (0), а символы будут отображаться светло-серым цветом (7). Во втором случае очищаемая область заполняется синим цветом (1), а символы отображаются желтым цветом ($E = 14).

Вывод на экран символов и строк

Для вывода на экран символов в прерывании $ 10 используются функции под номером $09 и $0A. Функция с номером $09 отличается от функции с номером $OA тем, что она дополнительно устанавливает атрибуты цветности выводимого символа. Выводимый символ сохраняется в регистре AL, номер видеостраницы — в регистре BH, байт атрибутов цветности (для функции $09) — в регистре BL, количество повторений символа — в регистре CX.

Для вывода строк используется функция с номером $13. При этом в регистр AL сохраняется тип вывода: 0 — использовать атрибут цветности и не перемещать кур-

сор; 1 — использовать атрибут цветности и переместить курсор; 2 — выводится сначала символ, затем атрибут, и курсор не перемещается; 3 — выводится сначала символ, затем атрибут, и курсор перемещается. В регистре BH сохраняется номер видеостраницы, в регистре BP — адрес строки, в регистре CX — длина строки, а в регистре DX — координаты позиции строки на экране.

ПРИМЕЧАНИЕ

Функция с номером $13 прерывания $10 используется, начиная с компьютеров типа AT. Для любых компьютеров ту же самую операцию выполняет функция $09 прерывания $21 (рассмотрена в разделе "Прерывания MS-DOS").

Рассмотрим пример вывода на экран дисплея отдельных символов и целых строк. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int10_2.pas и введите в него текст из листинга 17.4, или откройте этот файл с дискеты при помощи команды File | Open (<F3>).

Листинг 17.4. Программа Int10_2.pas

```
program Int10_2;
uses Crt;
var
  i,a: byte;
  c: integer;
  s: string;

  procedure SetCursor(x,y: byte); assembler;
  asm
    MOV AH,02H
    MOV BH,00H
    MOV DL,x
    MOV DH,y
    INT 10H
  end;

begin
  ClrScr;
  {Вывод посимвольно}
  for i := 49 to 57 do
  begin
    SetCursor(0,i-49);
    c := i - 48;  {Количество повторений}
    a := i - 42;  {Байт атрибутов цветности}
    asm
      MOV AH,09H  {Функция вывода символа}
      MOV AL,i    {Код выводимого символа}
      MOV BH,0    {Первая видеостраница}
      MOV BL,a
      MOV CX,c
      INT 10H
    end;
  end;
  {Вывод строки}
  s := 'Вывод строки функцией $13';
  for i := 0 to 24 do
  begin
```

Окончание листинга 17.4

```
    c := i+1;
    asm
      MOV AH,13H    {Функция вывода строки}
      MOV AL,1      {Использовать атрибут цветности и переместить курсор}
      MOV BH,0      {Первая видеостраница}
      MOV BL,7      {Байт арибута цветности}
      LEA BP,[s+1]  {Адрес строки s}
      MOV CX,c      {Длина выводимой строки}
      MOV DL,10     {Координата X}
      MOV DH,i      {Координата Y}
       INT 10H
    end;
  end;
end.
```

В программе IntIO_2 используется ассемблерная процедура SetCursor (см. листинг 17.3), разработанная в предыдущем разделе. Вначале на экран выводятся различными цветами цифры от 1 до 9, причем количество символов в каждой строке соответствует значению цифры. Затем в каждой строке, начиная с одиннадцатого столбца, на экран выводятся части строки длиной, соответствующей номеру строки (рис. 17.1).

Рис. 17.1. Результат работы программы IntIO_2

Информация о текущей позиции курсора

Для извлечения информации о текущей позиции курсора используются следующие функции прерывания $10.

- $01 — установка размера курсора. Размер курсора колеблется от 0 до 13 для мониторов EGA или от 0 до 7 для большинства цветных мониторов.
- $03 — определение текущих координат курсора. После вызова прерывания номер строки содержится в регистре DH, а номер столбца — в регистре DL.

- $08 — чтение символа и байта атрибутов цветности. После вызова прерывания символ хранится в регистре AL, а байт атрибутов цветности — в регистре AH.

Рассмотрим использование этих функций прерывания $10 на примере демонстрационной программы. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int10_3.pas и введите в него текст из листинга 17.5, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.5. Программа Int10_3.pas

```
program Int10_3;
uses Crt;
var
  x,y,a,c: byte;
  PressedKey: char;

  procedure SetCursor(x,y: byte); assembler;
  asm
    MOV AH,02H
    MOV BH,00H
    MOV DL,x
    MOV DH,y
    INT 10H
  end;

begin
  ClrScr;
  {Вывод столбцов цифр от 0 до 7 с различными атрибутами цвета}
  for y := 0 to 7 do
  begin
    for x := 1 to 15 do
    begin
      TextColor(x);        {Цвет символа}
      TextBackGround(y);   {Цвет фона}
      Write(Chr(y+48));    {Вывод цифры}
    end;
    Writeln;
  end;
  x := 0;
  y := 0;
  repeat
    SetCursor(x,y);
    asm
      {Определяем цифру в текущей позиции}
      MOV AH,08H   {Функция чтения символа}
      MOV BH,0     {Первая видеостраница}
      INT 10H
      MOV c,AL     {Сохраняем символ}
      MOV a,AH     {Сохраняем атрибут}
    end;
    c := c - 48; {Значение текущей цифры}

    asm
      {Определяем размер курсора в
       соответствии с расположенной под ним цифрой}
```

Окончание листинга 17.5

```
      MOV AH,01H   {Функция установки размера курсора}
      MOV CH,c     {Верхняя граница курсора - в
                    соответствии текущему символу}
      MOV CL,07    {Нижняя граница курсора}
       INT 10H
       {Определяем координаты курсора}
      MOV AH,03H
      MOV BH,00
      • INT 10H
      MOV x,DL
      MOV y,DH
    end;
    {Отображаем информацию о позиции
     курсора с учетом текущих атрибутов цветности}
    TextAttr := a;
    SetCursor(20,0);
    Writeln('Позиция курсора: x=',x:2,'/y=',y:2);
    SetCursor(x,y); {Возвращаем курсор}
    PressedKey := ReadKey;  {Ожидаем нажатия клавиши}
    {Изменяем координату в зависимости от клавиши}
    case PressedKey of
      #72: if y>0 then Dec(y);  {Стрелка вверх}
      #75: if x>0 then Dec(x);  {Стрелка влево}
      #77: if x<14 then Inc(x); {Стрелка вправо}
      #80: if y<7 then Inc(y);  {Стрелка вниз}
    end;
  until PressedKey = #27;
end.
```

Результат работы этой программы представлен на рис. 17.2.

Для установки позиции курсора используется ассемблерная процедура SetCursor, разработанная в листинге 17.3. Поскольку программа IntlO_3 демонстрирует методы получения информации о текущем состоянии курсора, то для установки атрибутов цветности и вывода строк были использованы не рассмотренные ранее функции прерывания $10, а процедуры модулей Crt и System.

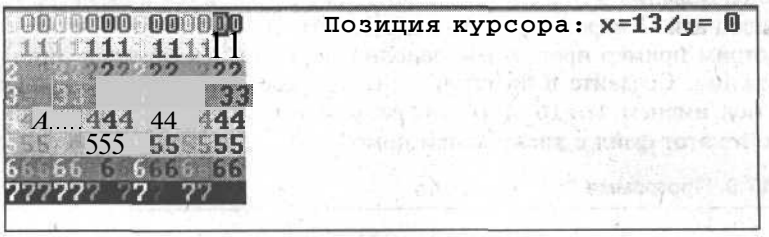


Рис. 17.2. Результат работы программы IntlO_3

Первая часть программы IntlO_3 выводит на экран компьютера 15 столбцов цифр с различными атрибутами цветности. В цикле repeat из второй части программы выполняются следующие действия.

1. Курсор устанавливается в позицию, соответствующую значениям переменных x и y, которые хранят текущие координаты курсора.

2. При помощи функции $08 прерывания $10 извлекается символ (сохраняется в переменной c) из текущей позиции курсора и его атрибуты цветности (сохраняются в переменной a).
3. Для получения абсолютного значения цифры, из кода соответствующего символа вычитается число 48 (код символа "0").
4. При помощи функции $01 прерывания $10 устанавливается размер курсора. Размер курсора определяется высотой его верхней границы (содержимое регистра CH) и высотой его нижней границы (содержимое регистра CL). Для стандартного курсора в виде символа подчеркивания нижняя и верхняя границы равны $07. Для полного курсора в виде прямоугольника верхняя граница равна $00, а нижняя — $07. Таким образом, в позиции символов "0" курсор должен быть полным, в позиции символов "7" — стандартным, а в позиции остальных символов иметь промежуточный размер.
5. При помощи функции $01 прерывания $10 определяются текущие координаты курсора (сохраняются в переменных x и y).
6. Стандартной переменной атрибутов цветности TextAtrr присваивается значение переменной *a,* хранящей атрибуты текущего символа. Курсор устанавливается в позицию справа от столбцов цифр. При помощи процедуры Writeln выводится информация о курсоре с текущими атрибутами цвета. Курсор возвращается в позицию, соответствующую координатам x и y.
7. В зависимости от того, какая нажата клавиша с изображением стрелки, изменяется значение одной из координат курсора.
8. Выход из цикла — по клавише <Esc>.

Управление дисплеем в графическом режиме

Рассмотрим работу в графическом режиме на примере использования двух функций прерывания $10: $00 — выбор режима (текстовый или графический) и $0C — вывод на экран цветной точки.

- Остальные функции, используемые при работе с графикой описаны в приложении Г.

В случае выбора функции $00, вид режима указывается в регистре AL. Например, обычному текстовому 16-цветному режиму 80x25 соответствует значение AL=$03, а графическому 16-цветному режиму с разрешением 320x200 — значение AL=$0D. В случае использования функции $0C, цвет точки сохраняется в регистре AL, координата по горизонтали — в регистре CX, а координата по вертикали — в регистре DX.

Рассмотрим пример программы, заполняющей экран цветными линиями в графическом режиме. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int10_4.pas и введите в него текст из листинга 17.6, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.6. Программа Int10_4.pas

```
program Int10_4;
uses Crt;
begin
  asm
    {Переход в графический режим}
    MOV AH,00   {Функция перехода в графич. режим}
    MOV AL,0DH  {Режим 320x200, 16 цветов}
    INT 10H
    {Выводим ряд цветных точек}
```

Окончание листинга 17.6

```
    MOV BX,00    {Начальный цвет}
    MOV CX,00    {Начальный столбец}
    MOV DX,00    {Начальная строка}
@1: MOV AH,0CH   {Функция вывода точки}
    MOV AL,BL    {Установить цвет}
    INT 10H
    INC CX       {Увеличить номер столбца}
    CMP CX,320   {Достигнут столбец 320?}
    JNE 01       {Если нет, то повторяем цикл}
    MOV CX,00    {Если да, то сбрасываем счетчик}
    INC BL       {Переходим к следующему цвету}
    INC DX       {Переходим к следующей строке}
    CMP DX,200   {Достигнута ли строка 200?}
    JNE @1       {Если нет, то повторяем цикл}
  end;
  ReadKey;

  asm
    {Возврат в текстовый режим}
    MOV AH,00
    MOV Al,03    {Текстовый цветной режим}
    INT 10H
  end;
end.
```

Результат работы этой программы представлен на рис. 17.3.

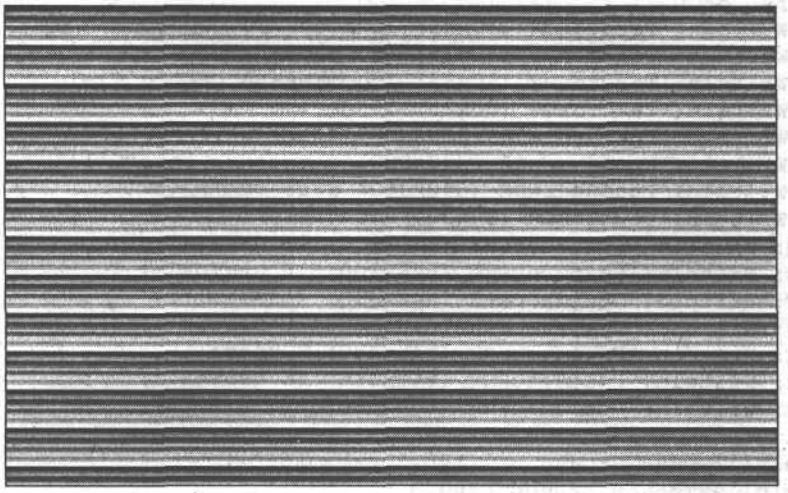


Рис. 17.3. Результат работы программы IntIO_4

Запрос информации об устройствах и о состоянии системы

Запрос информации об устройствах и о состоянии системы выполняется при помощи следующих прерываний BIOS.

- $10 (функция $OF) — запрос о видеорежиме.

- $11 — запрос списка подключенного оборудования.
- $12 — запрос объема базовой оперативной памяти.
- $13 (функция $08) — запрос информации о диске

Реализуем демонстрацию этих прерываний в одной программе. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int_Qry.pas и введите в него текст из листинга 17.7, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.7. Программа Int_Qry.pas

```
program Int_Qry;
uses Crt;
var
  rAL,rAH,rBH,rDL: byte;
  rAX,rCX: word;

begin
  ClrScr;
  {Определяем параметры видеорежима}
  asm
    MOV AH,0FH
    INT 10H
    MOV rAL,AL
    MOV rAH,AH
    MOV rBH,BH
  end;
  Writeln('Параметры видеорежима:');
  case rAL of
    0: Writeln('Текстовый (40x25) 16 оттенков серого');
    1: Writeln('Текстовый (40x25) 16 цветов');
    2: Writeln('Текстовый (80x25) 16 оттенков серого');
    3: Writeln('Текстовый (80x25) 16 цветов');
    4: Writeln('Графический (320x200) 4 цвета');
    5: Writeln('Графический (320x200) 4 оттенка серого');
    6: Writeln('Графический (640x200) 2 цвета');
    7: Writeln('Текстовый (80x25) 3 цвета');
    $0D: Writeln('Графический (320x200) 16 цветов');
    $0E: Writeln('Графический (640x200) 16 цветов');
    $0F: Writeln('Графический (640x350) 3 цвета');
    $10: Writeln('Графический (640x350) 16 цветов');
    $11: Writeln('Графический (640x480) 2 цвета');
    $12: Writeln('Графический (640x480) 16 цветов');
    $13: Writeln('Графический (640x480) 256 цветов');
  end;
  Writeln('Ширина экрана - ',rAH,' символов');
  Writeln('Активная видеостраница - ',rBH);
  ReadKey;

  {Определяем список подключенного оборудования}
  asm
    MOV AH,11H
    INT 11H
    MOV rAX,AX
  end;
```

Окончание листинга 17.7

```
  Writeln('');
  Writeln('Информация о подключенном оборудовании:');
  if rAX and $01 = 0 then {Если 1-й бит AX = 0}
    Writeln('Нет дисковода');
  if rAX and $02 = 2  {Если 2-й бит AX = 1}
    then Writeln('Есть сопроцессор')
    else Writeln('Сопроцессора нет');
  Write('Объем памяти на материнской плате: ');
  case rAX and $OC shr 2 of {Проверка 3 и 4 битов AX}
    1: Writeln('16K');
    2: Writeln('32K');
    3: Writeln('64K и больше');
  else Writeln('Не определен');
  end;
  Write('Начальный видеорежим:  ');
  case rAX and $30 shr 4 of {Проверка 5 и 6 битов AX}
    1: Writeln('Цветной 40x25');
    2: Writeln('Цветной 80x25');
    3: Writeln('Черно-белый 80x25');
  else
  end;
  Writeln('Количество дисководов: ',(rAX and $CO) shr 6 + 1); {7,8}
  if rAX and $1000 shr 12 = 1 then {Проверка 13-го бита AX}
    Writeln('Есть джойстик');
  Writeln('Количество параллельных портов: ',rAX shr 14); {15,16}
  ReadKey;

  {Запрос объема оперативной памяти}
  asm
    INT 12H
    MOV rAX,AX
  end;
  Writeln('');
  Writeln('Объем базовой оперативной памяти: ',rAX,'K');
  ReadKey;

  {Запрос информации о диске}
  asm
    MOV AH,08H
    MOV DL,80H  {Запрашиваем информацию о жестком диске}
    INT 13H
    MOV rCX,CX
    MOV rDL,DL
  end;
  Writeln('');
  Writeln('Информация о жестком диске:');
  Writeln('Секторов:  ',rCX and $3F); {Первые 6 битов}
  Writeln('Цилиндров:  ',rCX shr 6); {Остальные 10 битов}
  Writeln('Количество жестких дисков на первом контроллере:  ',rDL);
  ReadKey;
end.
```

Прерывания MS-DOS

Некоторые прерывания, обрабатываемые операционной системой MS-DOS, дублируют прерывания BIOS, однако они проще в использовании и меньше зависят от аппаратной реализации, чем BIOS-аналоги. Прерываниям MS-DOS соответствуют номера от $20 до $62, однако основными являются только 8 из них.

- $20 — завершение программы и передача управления системе MS-DOS.
- $21 — запрос функций MS-DOS. Это прерывание MS-DOS используется чаще всего, так как позволяет выполнять множество системных операций в соответствии с номером функции, записанном в регистр AH.
- $22 — адрес подпрограммы обработки завершения задачи.
- $23 — адрес подпрограммы, реагирующей на нажатие комбинации клавиш <Ctrl+Break>.
- $24 — адрес подпрограммы, реагирующей на фатальную ошибку.
- $25 — прямое чтение с диска.
- $26 — прямая запись на диск.
- $27 — завершение программы, которая остается резидентной. Используется для сохранения в памяти программ, выполняемые файлы которых имеют расширение .com.

В этой главе будут рассмотрены только те функции прерывания $21, которые предназначенны для работы с экраном, клавиатурой, принтером и таймером. Другие функции прерывания $21 (например, предназначенные для работы с файловой системой и с дисками) на примерах рассматриваться не будут.

» Справочник по основным прерываниям MS-DOS находится в приложении Г.

Вывод на экран символов и строк

Для вывода на экран символов с текущими параметрами цветности и в текущей позиции используется функция с номером $02 прерывания $21, а для вывода строк — функции с номерами $09 и $40 того же прерывания. Изменим, разработанную в начале этой главы, программу IntIO_2 (листинг 17.4) с использованием функций прерывания $21. Откройте в интегрированной среде Turbo Pascal файл IntIO_2.pas, сохраните его под именем Int21_1.pas. И внесите в него изменения, выделенные полужирным шрифтом в листинге 17.8, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.8. Программа Int21_1.pas

```
program Int21_1;
uses Crt;
var
  i: byte;
  c: integer;
  s: string;

  procedure SetCursor(x,y: byte); assembler;
  asm
    MOV AH,02H
    MOV BH,00H
    MOV DL,x
    MOV DH,y
```

Окончание листинга 17.8

```
    INT ЮН
  end;

begin
  ClrScr;
  {Вывод посимвольно}
  for i := 49 to 57 do
  begin
    SetCursor(0,i-49);
    for c := 1 to i - 48 do
    asm
      MOV AH,02H  {Функция вывода символа}
      MOV DL,i    {Код выводимого символа}
      INT 21H
    end;
  end;

  {Вывод строки на устройство}
  for i := 0 to 24 do
  begin
    s := 'Вывод строки функцией $40';
    SetCursor(30,i);
    c := i+1;
    asm
      MOV AH,40H    {Функция вывода строки}
      MOV BX,01H    {Устройство вывода)
      LEA DX,[s+1]  {Адрес строки s}
      MOV CX,c      {Длина выводимой строки)
      INT 21H
    end;
  end;
  {Стандартный вывод строки}
  SetCursor(0,10);
  s := 'Вывод строки функцией 09$';
  asm
    MOV AH,09H    {Функция вывода строки}
    LEA DX, [s+1] {Адрес строки s}
    INT 21H
  end;
end.
```

В случае использования функции с номером $40 в регистре BX необходимо указать устройство вывода. Значению 01 соответствует экран монитора (стандартное устройство вывода), а значению 04 — принтер. Таким образом, при помощи функции $40 строки можно выводить не только на экран, но и на принтер. Обратите также внимание на то, что в случае вывода строк на экран при помощи функции $09 нельзя указать количество выводимых символов, а конец строки определяется символом "$".

Ввод символов с клавиатуры

Для ввода символов с клавиатуры используются следующие функции прерывания $21.

- $01 — ввод символа с его отображением на экране.

- $06 — ввод символа со стандартного устройства ввода (обычно клавиатуры), а также вывод символа на стандартное устройство вывода (обычно экран монитора).
- $07 — ввод символа с клавиатуры без его вывода на экран и без проверки нажатия комбинации клавиш <Ctrl+Break>.
- $08 — ввод символа с клавиатуры без его вывода на экран и с проверкой нажатия комбинации клавиш <Ctrl+Break>.
- $ OA — ввод символов в буферную область памяти.

Рассмотрим использование этих функций на примере. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int21_2.pas и введите в него текст из листинга 17.9, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.9. Программа Int21_2.pas

```
program Int21_2;
uses Crt;
type
  Buffer = Record
    MaxLen: byte;
    ActLen: byte;
    Chars: array[1..254] of char;
  end;
var s: string;
    i: integer;
    c: char;
    Buf: Buffer;

  procedure EnterString(fNum: byte);
  begin
    s:='';
    Writeln('Введите строку: ');
    for i := 1 to 255 do
    begin
      asm
        MOV AH,fNum  {Функция ввода символа с клавиатуры}
        INT 21H
        MOV c,AL
      .end;
      if c=#13 then break;  {Если нажата клавиша Enter}
      s := s + c;
    end;
    Writeln('Введена строка "',s,'"');
  end;

begin
  ClrScr;
  EnterString($01);  {Ввод символов с отображением}
  EnterString($07);  {Ввод символов без отображения}
  {Ввод строки в буфер}
  Write('Введите длину строки для ввода: ');
  Readln(Buf.MaxLen);
  Inc(Buf.MaxLen);  {Увеличиваем буфер с учетом символа конца строки}
  Writeln('Введите строку и нажмите Enter: ');
```

Окончание листинга 17.9

```
  asm
    MOV AH,0AH
    LEA DX,Buf   {Адрес структуры буфера}
    INT 21H
  end;
  Write('Содержимое буфера: ');
  for i:= 1 to Buf.ActLen do Write(Buf.Chars[i]);
  ReadKey;
end.
```

Ввод строки с отображением символов на экране и без их отображения на экране реализован в процедуре EnterString, в которую в качестве параметра передается номер функции прерывания $21. Для ввода символов с последующей записью в буфер, в программе определена структура типа Buffer, адрес которой записывается в регистр DX перед вызовом прерывания. Эта структура состоит из трех полей: максимальной длины вводимой строки, фактическая длина введенной строки и массив символов. При определении максимальной длины вводимой строки следует учитывать то, что строка в буфере всегда оканчивается символом конца строки (код 13), а значит фактически, максимальная длина уменьшается на единицу. После вызова прерывания фактическая длина введенной строки будет записана в поле ActLen.

Печать

Для вывода на печать используется две функции прерывания $21. Первая из них — с номером $05 — используется для передачи на принтер одного символа, хранимого в регистре DL. Вторая функция — с номером $40 — используется для передачи в принтер данных фиксированной длины. Рассмотрим это на примере. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int21_3.pas и введите в него текст из листинга 17.10, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.10. Программа Int21_3.pas

```
program Int21_3;
uses Crt;
var
  s: string;
  i: integer;
  c: char;
begin
  ClrScr;
  Write('Введите строку: ');
  Readln(s);
  {Печать посимвольно}
  for i := 1 to Length(s) do
  begin
    c := s[i];
    asm
      MOV AH,05
      MOV DL,c
      INT 21H
      MOV DL,0AH  {Символ конца строки}
      INT 21H
```

Окончание листинга 17.10

```
    end;
  end;
  {Печать строки целиком}
  i := Length(s) ;
  asm
    MOV AH,40H
    MOV BX,04H   {Файловый номер принтера}
    MOV CX,i     {Количество выводимых символов}
    LEA DX, s    {Адрес строки}
    INT 21H
  end;
end.
```

В результате вывода символа перевода строки (код 10 — $OA), в первой части программы Int21_3, каждый следующий символ будет напечатан в той же позиции, но на строку ниже. Для того чтобы печать начиналась с начала строки, после символа перевода строки, на принтер должен передаваться символ возврата каретки (код 13 — $0D).

Работа с системным таймером

Для работы с таймером предназначены функции прерывания $21 с номерами от $2A по $2E. Разработаем программу, которая будет выводить в левом верхнем углу экрана информацию о текущей дате, а в правом верхнем — о текущем времени. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Int21_4.pas и введите в него текст из листинга 17.11, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 17.11. Программа Int21_4.pas

```
program Int21_4;
uses Crt;
type
  DateType = record
    Year: word;
    Month: byte;
    Day: byte;
  end;
  TimeType = record
    Hour: byte;
    Minute: byte;
    Second: byte;
  end;
var
  PriorDate,CurDate: DateType;
  PriorTime,CurTime: TimeType;

  procedure SetCursor(x,y: byte); assembler;
  asm
    MOV AH,02H
    MOV BH,00H
    MOV DL,x
    MOV DH,y
    INT 10H
  end;
```

Продолжение листинга 17.11

```
  {Дополнение строки нулями слева}
  function PadL(num,len: byte): string;
  var
    s:string[2];
    i: integer;
  begin
    Str(num,s);  {Преобразуем число в строку}
    if Length(s) < len {Если необходимо дополнить}
      then PadL := '0' + s else PadL := s;
  end;

begin
  ClrScr;
  while not KeyPressed do
  begin
    {Определяем дату}
    asm
      MOV AH,2AH
      INT 21H
      MOV CurDate.Year,CX
      MOV CurDate.Month,DH
      MOV CurDate.Day,DL
    end;
    if (CurDate.Year <> PriorDate.Year) or
       (CurDate.Month <> PriorDate.Month) or
       (CurDate.Day <> PriorDate.Day)
    then begin
      PriorDate.Year := CurDate.Year;
      PriorDate.Month := CurDate.Month;
      PriorDate.Day := CurDate.Day;
      SetCursor(0,0);
      Write(PadL(CurDate.Day,2), '.',
            PadL(CurDate.Month,2), '.',
            CurDate.Year);
    end;
    {Определяем время}
    asm
      MOV AH,2CH
      INT 21H
      MOV CurTime.Hour,CH
      MOV CurTime.Minute,CL
      MOV CurTime.Second,DH
    end;
    if (CurTime.Hour <> PriorTime.Hour) or
       (CurTime.Minute <> PriorTime.Minute) or
       (CurTime.Second <> PriorTime.Second)
    then begin
      PriorTime.Hour := CurTime.Hour;
      PriorTime.Minute := CurTime.Minute;
      PriorTime.Second := CurTime.Second;
      SetCursor(72,0);
      Write(PadL(CurTime.Hour,2), ':',
```

Окончание листинга 17.10

```
            PadL(CurTime.Minute,2) + ':' +
            PadL(CurTime.Second,2));
    end;
  end;
end.
```

В программе Int21_4 функция PadL возвращает число num, преобразованное в строку и дополненное нулями до длины len. KeyPressed — это функция модуля Crt, которая возвращает значение True только в том случае, если была нажата какая-либо клавиша на клавиатуре. Смена даты и времени на экране происходит только в том случае, если предыдущее значение даты и времени не совпадает с текущим значением даты. Таким образом, дата сменяется на экране в полночь, а время — ежесекундно.

Для установки даты и времени используются прерывания $2B и $2D. При этом данные о годе, месяце, дне, часе, минуте и секунде сохраняются в тех же регистрах, из которых они считываются при вызове прерываний $2A и $2C.

Глава 18

Turbo Vision

Turbo Vision — это объектно-ориентированная прикладная система, предназначенная для разработки программ, работающих в мультиоконном диалоговом режиме. Для полного описания этой системы потребуется отдельная книга, поэтому в данной главе рассмотрены только базовые понятия, и разработаны небольшие программы, демонстрирующие эти возможности Turbo Vision.

По сути, Turbo Vision — это иерархия типов (или "классов", если придерживаться терминологии объектно-ориентированного программирования). Программа, написанная с использованием средств Turbo Vision, представляет собой набор объектов (отображаемых и неотображаемых на экране), взаимодействующих друг с другом при помощи механизма обработки *событий*.

Событие — это совершаемое объектом действие, которое обнаруживается программой и может быть соответствующим образом обработано. Процедуры и функции, выполняющие определенные операции в ответ на возникновение события, называются **обработчиками событий.** Например, можно создать обработчик события "Нажатие", связанного с объектом "Кнопка".

В состав Turbo Vision входят средства, позволяющие реализовывать в программах следующие элементы программного интерфейса:

- множественные перекрывающиеся окна переменного размера;
- меню;
- поддержку мыши;
- диалоговые окна;
- настройку цвета;
- кнопки, полосы прокрутки, окна ввода и другие элементы программного интерфейса;
- стандартную обработку нажатий клавиш и манипуляций с мышью.

При написании программ с использованием средств Turbo Vision широко применяются такие методики объектно-ориентированного программирования как *наследование* и *полиморфизм*. Не погружаясь глубоко в теорию, перейдем непосредственно к созданию первой программы средствами Turbo Vision.

« Базовые понятия объектно-ориентированного программирования рассматриваются в главе 12.

Простейшая программа, созданная при помощи средств Turbo Vision

Система Turbo Vision полностью объектно-ориентированная. Даже сама программа, написанная при помощи средств этой системы, является объектом типа TApplication. Для того чтобы использовать возможности Turbo Vision необходимо включить в программу ссылку на модуль App, а затем объявить переменную типа, производного от TApplication — объект приложения. Создайте в интегрированной среде

Turbo Pascal новый файл, сохраните его под именем tv_1.pas и введите в него текст, представленный в листинге 18.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 18.1. Программа TV_1.pas

```
program tv_1;
uses App;
type
  TMyApp = Object(TApplication)
  end;
var
  MyApp: TMyApp;
begin
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
end.
```

Запустите программу tv_1 на выполнение. Результат ее выполнения представлен на рис. 18.1. Как видно на этом рисунке, написав всего лишь несколько строк программного кода можно получить программу, состоящую из пустой строки меню; рабочей области, заполненной фоновым символом, и строки состояния.

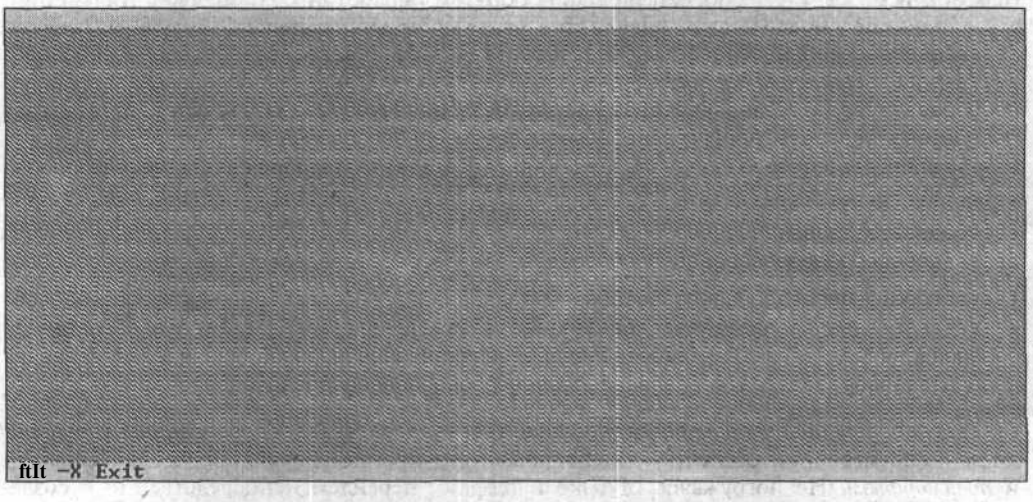


Рис. 18.01. Простейшая программа, созданная при помощи Turbo Vision

Программа tv_1 уже реагирует на нажатие комбинации клавиш <Alt+X>, а также на щелчок мышью на одноименном элементе строки состояния для завершения работы.

Процедуры и функции, реализованные в типах, называются **методами.** Так в программе tv_1 вызываются три базовых метода типа TApplication. Метод Init (конструктор) выполняет начальную инициализацию объекта приложения. Метод Run активизирует объект приложения (то есть приложение "запускается" — начинает реагировать на создание подчиненных объектов и возникновение событий). Метод Done (деструктор), выполняющий удаление объекта приложения вызывается только после того, как пользователь нажмет комбинацию клавиш <Alt+X>, то есть прервет рабочий цикл, активизированный методом Run.

ВНИМАНИЕ

Эти три метода должны вызываться во всех программах, написанных с использованием средств Turbo Vision, хотя, конечно же, возможности типа TApplication далеко не ограничены только методами Init, Run и Done.

Настройка строки состояния

В реализации конструктора TApplication.Init вызываются три виртуальных метода:

TApplication.InitDesktop — инициализация рабочей области;
TApplication.InitMenuBar — инициализация строки меню;
TApplication.InitStatusLine — инициализация строки состояния.

Любой из этих трех методов можно переопределить в программе (то есть создать собственную его реализацию), не изменяя при этом конструктор Init. Для начала изменим метод инициализации строки состояния.

Виртуальный метод InitStatusLine инициализирует объект строки состояния и сопоставляет с ним глобальную переменную StatusLine. Первое, что необходимо сделать — определить границы строки состояния. Поскольку границы строки состояния зависят от границ самой программы, которые в свою очередь изменяются в зависимости от видеорежима, то метод InitStatusLine должен запросить объект программы о его границах, и в соответствии с ними установить собственные границы.

Все отображаемые элементы Turbo Vision прямоугольные, и потому для хранения координат левого верхнего и правого нижнего углов используется специальный тип TRect. Каждый объект типа TRect имеет два поля: A и B. Полю A соответствует левый верхний угол, а полю B — правый нижний угол отображаемого элемента. В свою очередь поля A и B также являются объектами, хранящими координаты столбца и строки.

Отображаемым элементам соответствует метод GetExtent, возвращающий в своем единственном параметре-переменной объект типа TRect, соответствующий границам данного элемента (тип TRect объявлен в модуле Objects). Таким образом, для определения границ строки состояния метод InitStatusLine вызывает метод GetExtent объекта программы, а затем изменяет значение полей возвращенного объекта типа TRect. Внесите изменения в программу tv_1 (см. листинг 18.1), выделенные полужирным шрифтом в листинге 18.2.

Листинг 18.2. Новая версия программы Tv_1.pas

```
program tv_1;
uses App, Objects;
type
  TMyApp = Object(TApplication)
    procedure InitStatusLine; virtual; {Перекрываем метод}
  end;

    procedure TMyApp.InitStatusLine; {Новая реализация метода}
    var R: TRect;
    begin
      GetExtent(R);        {Извлекаем границы обеъкта программы}
      R.A.Y := R.B.Y - 1; {Устанавливаем координату верхней точки на
                    один пиксель выше нижней точки объекта программы}
    end;
```

Окончание листинга 18.2

```
var
  MyApp: TMyApp;
begin
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
end.
```

Если теперь запустить программу tv_1 на выполнение, то строка состояния будет просто заполнена фоновым черным цветом без какого-либо текста, и не будет реагировать на нажатие комбинации клавиш <Alt+x> и щелчки мышью (рис. 18.2).

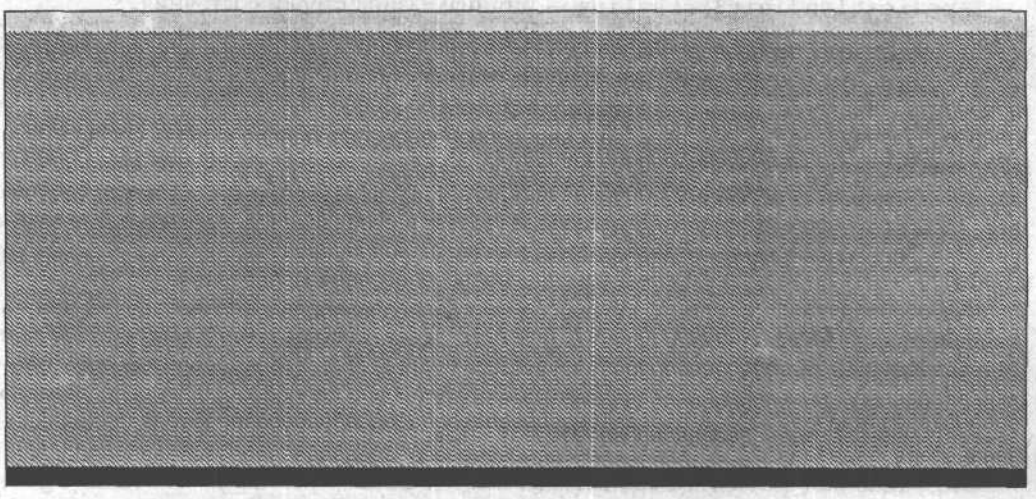


Рис. 18.2. Пустая строка состояния

Это объясняется тем, что был перекрыт базовый метод InitStatusLine и теперь ответственность за отображение строки состояния и организацию взаимодействия с ней возлагается на программиста.

Каждому отображаемому элементу соответствует поле, содержащее контекст его подсказки. Это поле хранит число, используемое системами контекстной справки для определения отображаемого на экране информационного кадра, а также определяет, какая строка состояния отображается в нижней части экрана. Объект строки состояния содержит связанный набор записей, называемых определениями состояния. Определение состояния содержит диапазон контекстов подсказки и список элементов строки состояния или клавиш состояния, которые отображаются при попадании контекста подсказки программы в указанный диапазон. Определение состояния задается путем вызова функции NewStatusDef (определена в модуле Menus).

Создадим в программе tv_1 одно определение состояния с диапазоном, покрывающим все возможные контексты подсказки так, чтобы независимо от текущего контекста подсказки всегда отображалась одна и та же строка состояния. Добавьте в метод InitStatusLine (см. листинг 18.2) строки, выделенные полужирным шрифтом в листинге 18.3.

При помощи процедуры New создается новый объект statusLine, инициализированный следующими параметрами метода Init.

- Границы, указанные в полях объекта R, типаTrect. *

- Определение состояния, созданное функцией NewStatusDef, покрывающее весь диапазон возможных контекстов (номера от 0 до $FFFF) и использующее стандартный набор клавиш состояния, создаваемый функцией StdStatusKeys. Поскольку в программе tv_1 для строки состояния есть только одно определение состояния, то последний параметр функции NewStatusDef — это "пустой" указатель nil. Если бы требовалось создать еще одно определение состояния для какого-нибудь диапазона контекстов, то вместо указателя nil была бы указана еще одна функция NewStatusDef.

Листинг 18.3. Создание определения состояния

```
procedure TMyApp.InitStatusLine;
var R: TRect;
begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  New(StatusLine,
      Init (R, {Используются границы, переданные в R }
            NewStatusDef (0, $FFFF, {Покрывается контекст подсказки в
                                     диапазоне 0..FFFF}
                          StdStatusKeys (nil), {Используются
                                                стандартные клавиши состояния}
                         nil))); {Отсутствие дальнейших определений)
end;
```

Запустите программу tv_1 на выполнение. Теперь можно увидеть, что строка состояния имеет стандартную форму и даже реагирует на нажатие таких стандартных комбинации следующих клавиш: <Alt+x> — выход из программы, <F10> — активизация меню, <Ctrl+F5> — перемещение окна. Хотя программу tv_1 и не отображает никакой информации об этих клавишах. Для отображения элементов строки состояния вместо вызова единственной функции StdStatusKey необходимо создать связанный список вызовов функций NewStatusKey, причем последним элементом в этом списке должен быть вызов функции StdStatusKey, в противном случае программа не будет реагировать на стандартные наборы клавиш. Вставьте в метод InitStatusLine (см. листинг 18.3) строки, выделенные полужирным шрифтом в листинге 18.4.

Листинг 18.4. Завершение определения строки состояния

```
program tv_1;
uses ..., Drivers, Views;
...
procedure TMyApp.InitStatusLine;
var R: TRect;
begin
...
  New(StatusLine,
      -Init (R, NewStatusDef (0, $FFFF,
          NewStatusKey('~F3~Открыть', kbF3, cmOpen,
          NewStatusKey('~Alt+F3~Закрыть', kbAltF3, cmClose,
          NewStatusKey('~Alt+X~Выход', kbAltX, cmQuit,
          StdStatusKeys (nil) )))),
          nil)));
end;
```

Функция NewStatusKey создает объект клавиши состояния. Каждая клавиша состояния состоит из четырех элементов: текстовой строки, отображаемой в строке состояния; кода клавиши активизации; кода команды и указателя на следующую клавишу состояния.

Любой текст, заключенный в тильды ("~"), отображается красным цветом. Если текстовая строка пустая, что текст такого элемента вообще не появляется на экране, хотя и связан с комбинацией клавиш активации и командой. Комбинации клавиш активации могут состоять из любых функциональных клавиш в сочетании с клавишами <Alt> или <Ctrl>. Для определения кодов клавиш активации используются специальные константы, соответствующие всем распространенным комбинациям клавиш. Идентификаторы этих констант, определенных в модуле Drivers, начинаются с префикса kb.

Команды Turbo Vision — это также целочисленные константы, определенные в модуле Views, идентификаторы которых начинаются с префикса cm. Команда связана с комбинацией клавиш активизации и элементом строки состояния. Щелчок мышью на клавише состояния или нажатие комбинации клавиш активизации приводит к выполнению соответствующей команды.

В программе tv_l были определены две дополнительных клавиши состояния, которым соответствуют комбинации клавиш <F3> (команда "Открыть"), и <Alt+F3> (команда "Закрыть"). Кроме того, был определен текст для стандартной клавиши состояния, используемой для выхода из программы (<Alt+X>).

Если теперь запустить программу tv_l на выполнение, то в строке состояния будут отображены три элемента, причем клавиша состояния <Alt+F3> будет недоступна. Это объясняется тем, что для недоступных команд (в данном случае — для команды cmClose) средства Turbo Vision автоматически делают недоступными соответствующие элементы строки состояния (рис. 18.3).

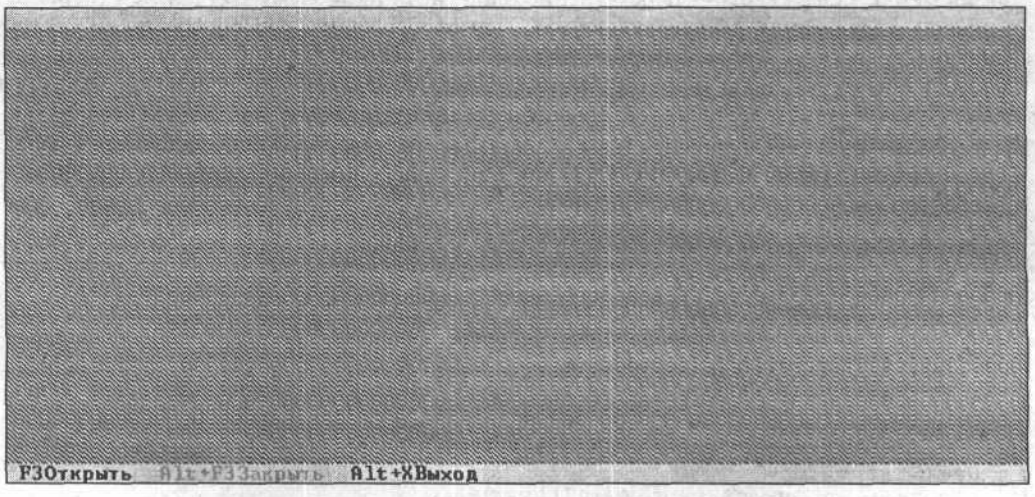


Рис. 18.3. Конечный вид строки состояния в программе tv_l

Настройка строки меню

Точно так же, как метод InitStatusLine инициализирует строку состояния, метод InitMenuBar создает строку меню и сопоставляет ее с глобальной переменной MenuBar. При создании меню в программе необходимо переопределить базовый ме-

тод InitMenuBar. Так же, как и строка состояния, строка меню состоит из связанного перечня пунктов. Пункты меню могут быть либо командами, либо связями с подменю. По умолчанию строка меню не содержит никаких пунктов, поэтому оно создается полностью программистом.

Создадим в программе tv_1 (см. предыдущий разел) собственную реализацию метода InitMenuBar (листинг 18.5).

Листинг 18.5. Перекрытие метода InitMenuBar

```
program tv_1;
uses App, Objects, Menus, Drivers, Views;
type
  TMyApp = Object(TApplication)
    procedure InitStatusLine; virtual;
    procedure InitMenuBar; virtual;
  end;
...
    procedure TMyApp.InitMenuBar;
    var R: TRect;
    begin
      GetExtent(R);
      R.B.Y := R.A.Y + 1;
      MenuBar :=
        New(PMenuBar,
          Init(R, NewMenu(   {Создание строки меню}
            NewSubMenu('Файл', hcNoContext, NewMenu(  {Создание подменю}
               {Создание пунктов подменю}
              NewItem('Открыть', 'F3', kbF3, cmOpen, hcNoContext,
                NewItem('Закрыть', 'Alt-F3', kbAltF3, cmClose,
                        hcNoContext,
                NewLine(   {Создание разделителя}
                NewItem('Вы~х~од', 'Alt-X', kbAltX, cmQuit, hcNoContext,
              nil))))),
            nil))));
    end;
...
```

Для создания строки меню используется функция NewMenu, для создания подменю используется функция NewSubMenu, а для создания пунктов меню и подменю — функция Newltem. В функцию NewMenu передается связанный список пунктов меню или подменю. В функцию NewSubMenu передается заголовок подменю, код справочного контекста и объект подменю, возвращаемый функцией NewMenu. В функцию Newltem передается заголовок пункта меню или подменю, строка с указанием комбинации клавиш, код комбинации клавиш, код команды, код справочного контекста и ссылка на следующий пункт меню или подменю. Для создания разделителя между пунктами меню или подменю используется функция NewLine. Таким образом в программе tv_1 создается меню **Файл** с пунктами **Открыть, Закрыть и Выход,** которым в соответствие поставлены команды cmOpen, cmClose и cmQuit (рис. 18.4).

Реакция на команды

В традиционном программировании, не управляемом событиями, обработка команд выполняется следующим образом: вначале выполняется некоторое действие, по-

сле чего ожидается ввод данных от пользователя и выполняются действия соответствующие полученным данным. Центральной частью в такой модели программирования является **цикл ввода,** за которым обычно следует условный оператор. При программировании, управляемом событиями, вместо многочисленных циклов ввода вся программа имеет один цикл, который называется **циклом событий** и управляет всем интерфейсом с "внешним миром" и переходом в различные части программы.

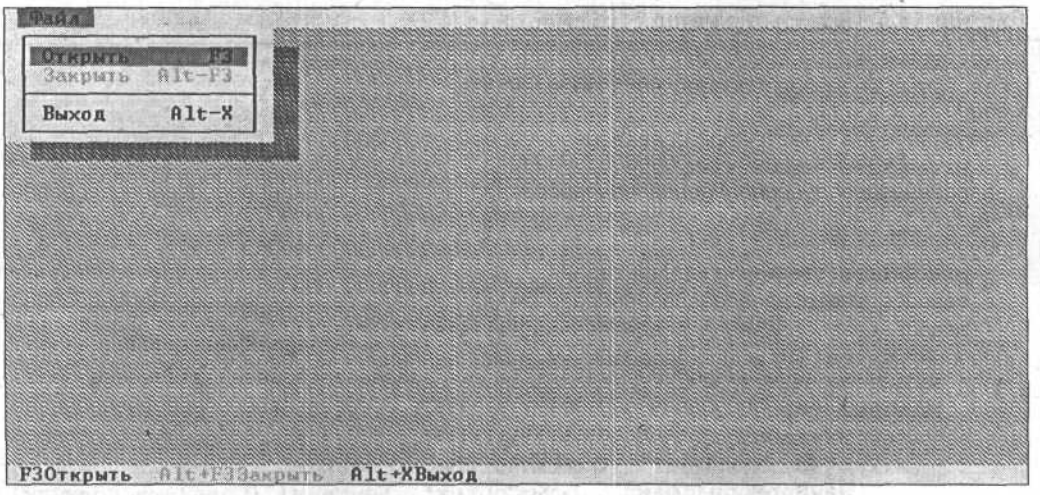


Рис. 18.4. Разработано меню **Файл**

Цикл событий записывает информацию о событии в запись типа TEvent и передает ее объекту, к которому относится событие. Все видимые объекты Turbo Vision имеют виртуальные методы, называемые **обработчиками событий,** которые принимают параметр-переменную типа TEvent. Таким образом, когда цикл событий программы обнаруживает событие, он делает заключение о том, обработчик события какого объекта должен обрабатывать событие, создает запись события и передает эту запись методу обработки. Затем обработчик события объекта проверяет запись события и на основании этой записи принимается решенеие, что с этим событием необходимо сделать.

Каждая запись события содержит поле типа Word с именем What, которое цикл событий заполняет константой, указывающей, какой тип события описывается в записи. Одной из таких констант является evCommand, которая указывает на командное событие. Если событие является командным, то в записи находится также поле с именем Command, содержащее командную константу, связанную с пунктом меню или клавишей состояния, генерирующими командное событие.

Например, если щелкнуть мышью на элементе строки состояния **Alt+F3** (или нажать комбинацию клавиш <Alt+F3>), то цикл событий сгенерирует командное событие, присвоит полю What из записи события значение evCommand, а полю Command — значение cmClose. Затем этот цикл направляет событие в активное окно. Получив команду cmClose, оконные объекты закрываются вызовом метода Close.

Данные в записи события зависят от типа события. Например, если событием является щелчок кнопкой мыши, то запись события содержит экранные координаты указателя мыши и значение, указывающее на наличие двойного щелчка. Нажатие клавиши на клавиатуре посылает событие, включающее в себя код опроса или код символа нажатой клавиши.

Для обработки событий в программе необходимо перекрыть базовый метод TApplication.HandleEvent, в который в качестве параметра передается значе-

ние типа TEvent. Реализуем в программе tv_1 обработку команды cmQuit (листинг 18.6).

Листинг 18.6. Реализация обработки команды cmQuit

```
program tv_1;
uses App, Objects, Menus, Drivers, Views, MsgBox;
type
  TMyApp = Object(TApplication)
    procedure InitStatusLine; virtual;
    procedure InitMenuBar; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
  end;

  procedure TMyApp.HandleEvent(var Event: TEvent);
  begin
    if Event.What = evCommand then {Если команда}
      case Event.Command of
        cmQuit: if MessageBox(#3'Выйти из программы?',nil,
                        mfConfirmation or mfOKCancel) = cmCancel
                then ClearEvent(Event);  {Пометить событие, как
                                          обработанное }
      end;
    inherited HandleEvent(Event);   {Вызывается наследуемый метод}
  end;
...
```

В начале в методе HandleEvent проверяется тип события. Если событие относится к командам (Event.What = evCommand), то сама команда записана в поле Event.Command. Обычно, при выполнении команды cmQuit на экране отображается диалоговое окно запроса, в котором пользователь может подтвердить решение выйти из программы. Для вывода подобного диалогового окна используется функция MessageBox, реализованная в модуле MsgBox. В эту функцию передается три параметра.

1. Строка, отображаемая в диалоговом окне. Если строка начинается с символа #3, то она центрируется, в противном случае строка выравнивает по левому краю окна.
2. Указатель на массив записей элементов данных, выводимых в сообщении. В данном случае такой массив не используется, и потому в качестве второго параметра передается "пустой" указатель nil.
3. Флажок, определяющий заголовок окна сообщения и отображаемые в нем кнопки. Для формирования значения этого флажка константа типа окна (mbInformation, mfWarning, mfConfirmation или mfError) комбинируется с константой набора кнопок (mfOKButton, mfOKCancel или mfYesNoCancel) при помощи оператора or.

Результат, возвращаемый функцией MessageBox, соответствует нажатой в диалоговом окне кнопке, и может принимать значения cmOK, cmCancel, cmYes или cmNo. В данном примере (см. листинг 18.6) в ответ на команду cmQuit метод HandleEvent открывает диалоговое окно подтверждения операции с двумя кнопками: **OK** и **Cancel.** (рис. 18.5).

Если пользователь нажмет кнопку **OK,** то будет вызван унаследованный метод HandleEvent, в котором выполнение команды cmQuit приведет к выходу из программы. Если пользователь выберет в диалоговом окне кнопку **Cancel** или нажмет клавишу <Esc>, то командное событие cmQuit будет очищено при помощи процеду-

ры ClearEvent. Это означает, что запись Event не будет содержать информации ни о каком событии. В результате выхода из программы не произойдет, так как в унаследованный обработчик HandleEvent не будет передано никакого события.

Стандартные диалоговые окна

К стандартным диалоговым окнам относятся следующие окна: открытия файла, сохранения файла и т.п. В среде Turbo Vision такие окна реализованы в модуле StdDlg. В программе tv_1 необходимо использовать диалоговое окно открытия файла, которому соответствует тип TFileDialog. Вставьте в файл tv_1. pas дополнения, выделенные полужирным шрифтом в листинге 18.7.

Листинг 18.7. Реализация процедуры открытия файла

```
program tv_1;
uses ..., StdDlg,Editors;
type
  TMyApp = Object(TApplication)
    procedure InitStatusLine; virtual;
    procedure InitMenuBar; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
    procedure FileOpen;
end;

var
  MyApp: TMyApp;

  procedure TMyApp.HandleEvent(var Event: TEvent);
  begin
    if Event.What = evCommand then {Если команда}
      case Event.Command of
        ...
        cmOpen: FileOpen;
      end;
    inherited HandleEvent(Event); {Вызывается наследуемый метод}
  end;

  function OpenEditor(FileName: FNameStr;
                      Visible: Boolean): PEditWindow;
  var
    P: PWindow;
    R: TRect;
  begin
    DeskTop^.GetExtent(R);
    P := New(PEditWindow, Init(R, FileName, wnNoNumber));
    if not Visible then P^.Hide;
    OpenEditor := PEditWindow(Application^.InsertWindow(P));
  end;

  procedure TMyApp.FileOpen;
  var
    FileName: FNameStr;   {Строка для имени файла }
  begin
    FileName := '*.*';    {Начальная маска для имени файла }
    if ExecuteDialog(New(PFileDialog, Init('*.*', 'Открыть файл',
```

Окончание листинга 18.7

```
      'Имя:', fdOpenButton, 100)), @FileName) <> cmCancel then
       OpenEditor(FileName, True);
  end;
...
```

При выполнении команды cmOpen метод HandleEvent вызывает метод FileOpen. В методе FileOpen вызывается функция ExecuteDialog, в которую передается два параметра: указатель на объект типа TFileDialog и адрес строки, в которую будет возвращено имя открытого файла в том случае, если пользователь нажмет в диалоговом окне кнопку **Открыть.** Конструктор Init типа TFileDialog принимает 5 параметров: начальную маску выбора файлов, заголовок окна, подпись над полем выбора имени файла, набор констант кнопок и номер, который будет использоваться для идентификации списка последних открываемых файлов. Как и функция MessageBox, функция ExecuteDialog возвращает код нажатой в диалоговом окне кнопки. В данном примере, если была нажата кнопка **Открыть** (соответствует константе fdOpenButton), то вызывается функция OpenEditor, создающая в рабочей области программы окно и загружающая в него содержимое выбранного файла.

За управление окнами отвечает глобальный указатель на объект рабочей области DeskTop. Вначале функции OpenEditor в объекте типа TRect сохраняются размеры рабочей области. Затем создается указатель на объект типа TEditWindow (определен в модуле Editors), в конструктор которого передается объект типа TRect с размерами рабочей области, имя файла и номер создаваемого окна (в данном примере номер не определен — константа wnNoNumber). Если необходимо скрыть созданное окно (параметр Visible = False), то вызывается метод TEditWindow.Hide. Последний оператор функции OpenEditor вставляет окно в рабочую область при помощи метода TApplication.InsertWindow и возвращает указатель на это окно.

Если запустить программу в таком виде (см. листинг 18.7) на выполнение и попытаться открыть какой-нибудь файл, то ничего не произойдет. Это объясняется тем, что для открытия окна редактора в рабочей области необходимо глобальной переменной EditorDialog (в конструкторе программы) присвоить значение, возвращаемое функцией создания стандартного окна редактирования StdEditorDialog. Добавьте в программу tv_1 код конструктора TMyApp.Init, выделенный полужирным шрифтом в листинге 18.8.

Листинг 18.8. Конструктор TMyApp.Init

```
...
type
  TMyApp = Object(TApplication)
    ...
    constructor Init;
  end;
  constructor TMyApp.Init;
  begin
    inherited Init;
    EditorDialog := StdEditorDialog;
  end;
...
```

Если теперь запустить программу на выполнение и попытаться открыть какой-нибудь файл, то на экране появится сообщение о том, что для выполнения этой операции недостаточно памяти. Дело в том, что для хранения содержимого файла необходимо зарезерви-

ровать область динамически распределяемой памяти, которая будет использоваться в качестве буфера для загрузки файла. Вставим в программу tv_1 дополнения в метод Init, выделенные полужирным шрифтом в листинге 18.9, позволяющие зарезервировать область динамически распределяемой памяти.

Листинг 18.9. Метод Init

```
program tv_1;
uses ..., Memory;
...
var
  MyApp: TMyApp;
  ClipWindow: PEditWindow;
...
  constructor TMyApp.Init;
  begin
    MaxHeapSize := 8192;
    inherited Init;
    EditorDialog := StdEditorDialog;
    ClipWindow := OpenEditor('', False);
    if ClipWindow <> nil then
    begin
      Clipboard := ClipWindow^.Editor;
      Clipboard^.CanUndo := False;
    end;
  end;
...
```

Переменная MaxHeapSize, определенная в модуле Memory, резервирует часть памяти. Она устанавливает количество 16-байтовых параграфов, которые программа может использовать под динамически распределяемую память, оставляя остальную свободную память для буферов редактирования файла. Присвоение переменной MaxHeapSize значения 8192 означает резервирование 128 Кбайт памяти.

ClipWindow — это указатель на область памяти, связанной с окном редактирования. При инициализации программы функци OpenEditor создает невидимое окно редактирования, на которое ссылается указатель ClipWindow. Затем, если окно создано, для области редактирования в памяти выделяется буфер размером, установленным переменной MaxHeapSize (на этот буфер ссылается глобальный указатель Clipboard). При этом запрещается сохранение истории изменений.

Теперь можно запустить программу tv_1 и открыть какой-нибудь файл при помощи стандартного диалогового окна (рис. 18.6 и рис. 18.7).

Команда cmClose обрабатывается для окна редактирования автоматически и поэтому не требует написания программного кода в методе HandleEvent.

Обзор Turbo Vision

Полностью рассмотреть все средства Turbo Vision в пределах одной главы просто невозможно, поэтому рассмотрим только кратко иерархию типов и ее распределение по модулям. Корневым в иерархии Turbo Vision является тип TObject. От него происходят все остальные типы. Этот объект не имеет полей (или свойств), а только конструктор Init, деструктор Done и процедуру Free, которая уничтожает объект И вызывает деструктор.

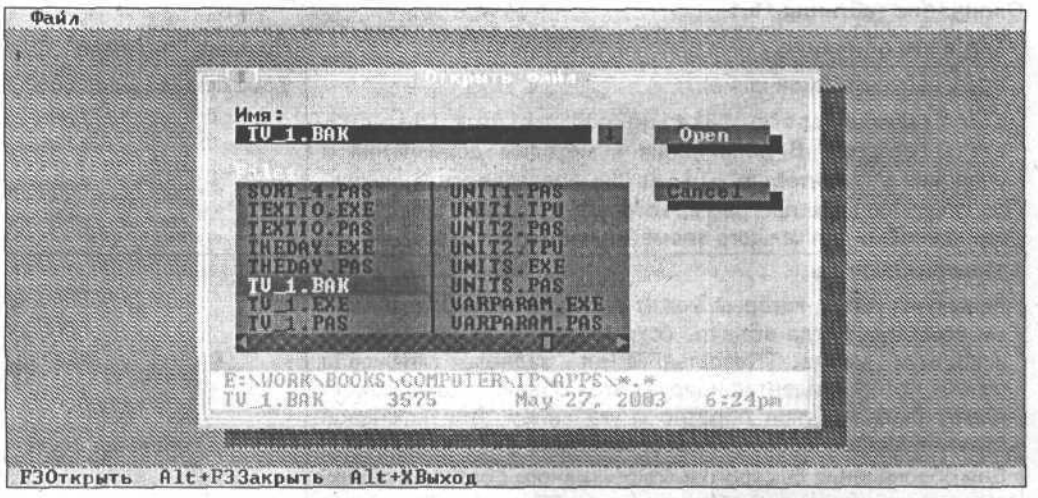


Рис. 18.6. Стандартное диалоговое окно открытия файла

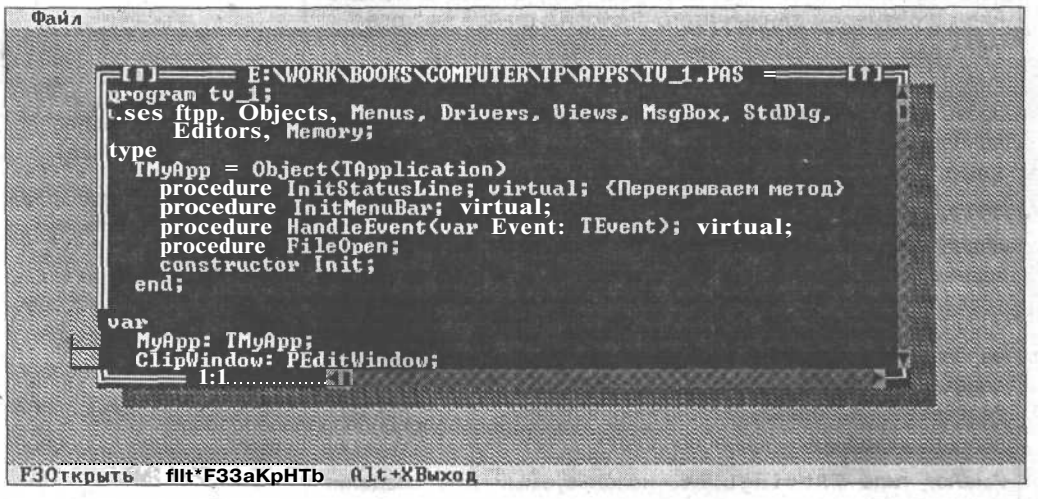


Рис. 18.7. Программа tv_1.pas, открытая в окне одноименного редактора

Прямыми "потомками" TObject **являются типы** TView, TValidator, TCollection, TResourceFile, TStream, TStringList и TStrListMaker. **Назначение каждого из этих типов описано в табл. 18.1.**

Таблица 18.1. Прямые "потомки" типа TObject

Тип и его описание	Прямые "потомки"
TCollection (начало) Основной тип для реализации любого набора элементов, включающего другие объекты. Размер объектов TCollection динамически устанавливается во время выполнения, и обеспечивает базовый тип для многих специализированных типов, таких, как:	TSortedCollection

Окончание таблицы 18.1

Тип и его описание	Прямые "потомки"
TCollection (окончание) TSortedCollection, TStringCollection и TResourceCollection. В дополнение к методам добавления и удаления элементов, TCollection представляет несколько итерационных программ, которые вызывают процедуру или функцию для каждого элемента набора	TSortedCollection
TResourceFile Реализует поток, который может индексироваться ключевыми строками. Когда объекты сохраняются в файле ресурса, используя метод TResourseFile.Put, задается ключевая строка, которая идентифицирует этот объект. Позже объект может быть получен обратно с указанием этой ключевой строки в вызове метода TResourseFile.Get. Для обеспечения быстрого и эффективного доступа к объектам, хранящимся в файле ресурса, TResourseFile хранит ключевые строки в отсортированном наборе строк (с использованием типа TResourseCollection) вместе с положением и размером данных этого ресурса в файле ресурса	
TStream Общий тип потока, обеспечивающий ввод-вывод через область памяти. Можно создавать порожденные объекты потока, переопределяя виртуальные методы GetPos, GetSize, Read, Seek, Truncate и Write	TDosStream TEmsStream TMemoryStream
TSrtingList Механизм для доступа к строкам, хранящимся в потоке. Каждая строка, хранящаяся в списке строк, идентифицируется уникальным номером (ключом) в интервале от 0 до 65535. Списки строк занимают меньше памяти, чем обычные строки, поскольку хранятся в потоке, а не в памяти. Кроме того, списки строк легко решают проблему настройки программ на разные языки, поскольку строки не "встроены" в программу	
TStrListMaker Аналог типа TStringList, используемый для создания списка строк	
Tvalidator Определяет абстрактный объект определителя допустимости данных. Этот тип обеспечивает большую часть абстрактных функций для других определителей допустимости данных	
TView Тип, предназначенный, главным образом для создания порожденных отображаемых объектов, и непосредственно экземпляры объектов TView создаются редко	TbackGround, Tcluster, Teditor, Tframe, Tgroup, Thistory, TinputLine, TlistViewer, TmenuView, ScrollBar, TstaticText, TstatusLine,

» **см. подробнее о потоках в разделе "Создание баз данных средствами Turbo Vision" гл. 21.**

В табл. 18.2 перечислены прямые "потомки" типа TView.

Таблица 18.2. Прямые "потомки" типа TView

Тип и его описание	Прямые "потомки"
TBackGround Обычный отображаемый элемент, содержащий однотонно заполненный прямоугольник. Обычно он принадлежит рабочей области	
TCluster Кластер — это группа элементов управления, которые "откликаются" одинаковым образом. Этот тип обобщает работу с независимыми и зависимыми кнопками	TCheckBoxes TMultiCheckBoxes TRadioButtons
TEditor Реализует простой, быстрый редактор с объемом памяти 64K для использования в программах Turbo Vision. В нем реализована поддержка мыши, отмена изменений, работа с системным буфером, автоматические режимы формирования отступов и изменение режимов вставки/замены, создание оперативных клавиш, а также поиск и замену. Отображаемые элементы редактора можно использовать для редактирования файлов и для многострочных полей комментариев в диалоговых окнах или бланках	TFileEditor TMemo
TFrame Создает различные рамки вокруг окон и диалоговых окон. Непосредственно объекты рамок используются редко, так как они добавляются к окнам автоматически	
TGroup Специальный тип отображаемого элемента. В дополнение ко всем полям и методам, порожденным от TView, группа TGroup имеет добавочные поля и методы (включая многие переопределения), позволяющие управлять динамически связанными списками отображаемых элементов (включая другие группы) как если бы они были одним объектом	TScroller TWindow
TInputLine Обеспечивает базовый редактор строк ввода. Он управляет вводом с клавиатуры, а также нажатием кнопок и перемещениями мыши при отметке блоков и в целом ряде функций редактирования строки	
THistory Реализует список ранее выполненных команд, действий или выборов для их повтора	
TListViewer Базовый тип, производными от которого являются различные объекты просмотра списков	THistoryViewer TListBox
TMenuView Абстрактный тип меню, производными от которого являются строки меню и вертикальные меню	TMenuBar TMenuBox

Окончание таблицы 18.2

Тип и его описание	Прямые "потомки"
TScrollBar Полоса прокрутки	1
TStaticText Простейшие отображаемые элементы: они содержат фиксированный текст и игнорируют все события, переданные им. Обычно эти элементы используются как сообщения или пассивные метки	TLabel TParamText
TstatusLine Строка состояния	

В табл. 18.3 перечислены прямые "потомки" типа TGroup.

Таблица 18.3. Прямые "потомки" типа TGroup

Тип и его описание	Прямые "потомки"
Tscroller Задает виртуальное окно для прокрутки отображаемого элемента, имеющего больший размер. Это означает, что этот отображаемый элемент позволяет пользователю просматривать большой отображаемый элемент в определенных границах	TDeskTop TProgram
TWindow Специализированная группа, в которую обычно включен объект TFrame, внутренний объект TScroller и один или два объекта TScrollBar	TDialog TEditWindow THistoryWindow

В табл. 18.4 перечислены прямые "потомки" типа TScroller.

Таблица 18.4. Прямые "потомки" типа TScroller

Тип и его описание	Прямые "потомки"
Tdesktop Простая группа, в которую обычно включен отображаемый элемент TBackground, на котором появляются окна и другие отображаемые элементы программы. TDeskTop представляет оперативную область экрана, располагающуюся между полосой меню (верхняя граница области) и строкой состояния (нижняя граница области)	
TProgram Обеспечивает базовый шаблон для всех стандартных программ Turbo Vision	TApplication

В табл. 18.5 показано распределение типов по модулям.

Таблица 18.5. Распределение типов по модулям

Модуль	Типы	
App	TApplication	TDeskTop
	TBackGround	**TProgram**
Dialogs	TCheckBoxes	TListBox
	TCluster	**TMultiCheckBoxes**
	THistory	TRadioButtons
	THistoryViewer	TScrollbar
	TInputLine	TStaticText
	TLabel	**TParamTest**
Editors	TFileEditor	TEditWindow
	TEditor	**TMemo**
Menus	TMenuBar	TMenuView
	TMenuBox	**TStatusLine**
	TMenuPopup	
Objects	**TBufStream**	TResourceFile
	TCollection	**TSortedCollection**
	TDosStream	TStrCollection
	TEmsStream	**TStream**
	TMemoryStream	TStringCollection
	TObject	**TStringList**
	TResourceCollection	TStrListMaker
Menus	TMenuBar	TMenuView
	TMenuBox	TStatusLine
	TMenuPopup	
StdDlg	TChDirDialog	TFileDialog
	TDialog	TSortedListBox
Validate	**TLookupValidator**	TStringLookupValidator **TVali-**
	TPXPictureValidator	**dator**
	TRangeValidator	
Views	**TFrame**	**TScroller**
	TGroup	TView
	TListViewer	**TWindow**

За более подробной информацией о средствах Turbo Vision обращайтесь к специализированным изданиям и технической документации.

ЧАСТЬ IV
ПРИМЕРЫ

Эта часть разбита на три главы.

- В главе 19 рассматривается работа с одномерными и двумерными массивами, включая методы их сортировки.
- В главе 20 на игровых примерах показано использование в приложениях звука, мыши и графики.
- В завершающей главе представлены два способа организации систем управления базами данных средствами языка Pascal — при помощи типизированых файлов и объектов Turbo Vision.

Глава 19

Работа с массивами

Основными задачами, связанными с массивами, являются сортировка элементов, поиск в отсортированном массиве, а также операции над матрицами (двухмерными массивами). В этой главе будут рассмотрены некоторые методики решения перечисленных задач.

◀◀ О массиве, как структуре данных, можно узнать в гл. 8.

Сортировка массивов

Сортировка — это упорядочение элементов массива по возрастанию или по убыванию. Существуют различные методы сортировки, однако общепринятыми считаются только четыре из них: сортировка отбором, сортировка методом "пузырька", сортировка вставкой и быстрая сортировка с разделением.

Сортировка отбором

При сортировке отбором последовательно просматривается весь массив, определяется наибольший (в случае сортировки по убыванию) или наименьший (в случае сортировки по возрастанию) элемент, который затем меняется местами с первым элементом массива. Затем просматриваются элементы массива, начиная со второго, и выполняется аналогичная операция по определению наибольшего или наименьшего элемента и его обмена с первым элементом текущей последовательности (например, при третьем просмотре элементов массива первым элементом текущей последовательности является третий элемент массива и т.д.). Этот цикл повторяется N-1 раз, где N — количество элементов массива, а первым элементом текущей последовательности элементов, которую нужно просмотреть, является N-1 элемент массива.

Реализуем этот метод сортировки в виде программы. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Sort_1.pas и введите в него текст, представленный в листинге 19.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.1. Программа Sort_1.pas

```
program Sort_1;
uses Crt;

var
  A: array[1..20] of byte;
  i,j,buf: byte;

  {Вывод элементов массива}
  procedure ShowArray;
  begin
    for i := 1 to 20 do Write(A[i]:4);
  end;
```

Окончание листинга 19.1

```
  {Сортировка массива. Если ByAsc = True, то - по возрастанию.
                       Если ByAsc = False, то - по убыванию. }
  procedure SortArray(ByAsc: boolean);
  begin
    for i := 1 to 19 do
      for j := i+1 to 20 do
        if (ByAsc and (A[i] > A[j])) or
           (not ByAsc and (A[i] < A[j]))
        then begin
          buf := A[i];
          A[i] := A[j];
          A[j] := buf;
        end;
  end;

begin
  ClrScr;
  {Заполняем массив случайными числами}
  Randomize;
  for i := 1 to 20 do A[i] := Random(255);
  Writeln('Массив до сортировки: ');
  ShowArray;
  SortArray(True); {Сортировка по возрастанию}
  Writeln('Массив отсортированный по возрастанию:');
  ShowArray;
  SortArray(False); {Сортировка по убыванию}
  Writeln('Массив отсортированный по убыванию:');
  ShowArray;
  ReadKey;
end.
```

Схематически процесс сортировки по возрастанию элементов массива, состоящего из десяти чисел, методом отбора представлен на рис. 19.1.

Сортировка методом "пузырька"

Метод "пузырька" — это один из самых популярных методов сортировки. Он основан на том, что элементы массива с меньшими (в случае сортировки по убыванию) или с большими (в случае сортировки по возрастанию) постепенно перемещаются к концу массива, подобно пузырьку воздуха перемещающемуся к поверхности воды. При этом каждый элемент сравнивается не со всеми элементами, а только с соседним элементом. Создадим программу сортировки методом "пузырька" на основе программы Sort_1 (см. листинг 19.1). Сохраните файл Sort_1.pas под именем Sort_2.pas и внесите изменения в процедуру SortArray, выделенные в листинге 19.2 полужирным шрифтом, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.2. Программа Sort_2.pas

```
program Sort_2;
...
  {Сортировка массива. Если ByAsc = True, то - по возрастанию.
                       Если ByAsc = False, то - по убыванию.}
  procedure SortArray(ByAsc: boolean);
```

Окончание листинга 19.2

```
  begin
    for i := 1 to 19 do
      for j := 1 to 20-i do
        if (ByAsc and (A[j] > A[j+1])) or
           (not ByAsc and (A[j] < A[j+1]))
        then begin
          buf := A[j];
          A[j] := A[j+1];
          A[j+1] := buf;
        end;
  end;
...
```

	1	2	3	4	5	6	7	8	9	10
Исходный массив	11	200	3	42	0	1	87	218	1	20
i = 1; j = 3	3	200	11	42	0	1	87	218	1	20
1 = 1; j = 5	0	200	11	42	3	1	87	218	1	20
i = 2; j = 3	0	11	200	42	3	1	87	218	1	20
i = 2; j = 5	0	3	200	42	11	1	87	218	1	20
i = 2; j = 6	0	1	200	42	11	3	87	218	1	20
i = 3; j = 4	0	1	42	200	11	3	87	218	1	20
i = 3; j = 5	0	1	11	200	42	3	87	218	1	20
i = 3; J = 6	0	1	3	200	42	11	87	218	1	20
i = 3; j = 9	0	1	1	200	42	11	87	218	3	20
i = 4; j = 5	0	1	1	42	200	11	87	218	3	20
i = 4; j = 6	0	1	1	11	200	42	87	218	3	20
i = 4; j = 9	0	1	1	3	200	42	87	218	11	20
i = 5; j = 6	0	1	1	3	42	200	87	218	11	20
i = 5; j = 9	0	1	1	3	11	200	87	218	42	20
i = 6; j = 7	0	1	1	3	11	87	200	218	42	20
i = 6; j = 9	0	1	1	3	11	42	200	218	87	20
i = 6; j = 10	0	1	1	3	11	20	200	218	87	42
i = 7; j = 9	0	1	1	3	11	20	87	218	200	42
i = 7; j = 10	0	1	1	3	11	20	42	218	200	87
i = 8; j = 9	0	1	1	3	11	20	42	200	218	87
i = 8; j = 10	0	1	1	3	11	20	42	87	218	200
i = 9; j = 10	0	1	1	3	11	20	42	87	200	218

Рис. 19.1. Сортировка элементов массива по возрастанию методом отбора

Схематически процесс сортировки по возрастанию элементов массива, состоящего из десяти чисел, методом "пузырька" представлен на рис. 19.2.

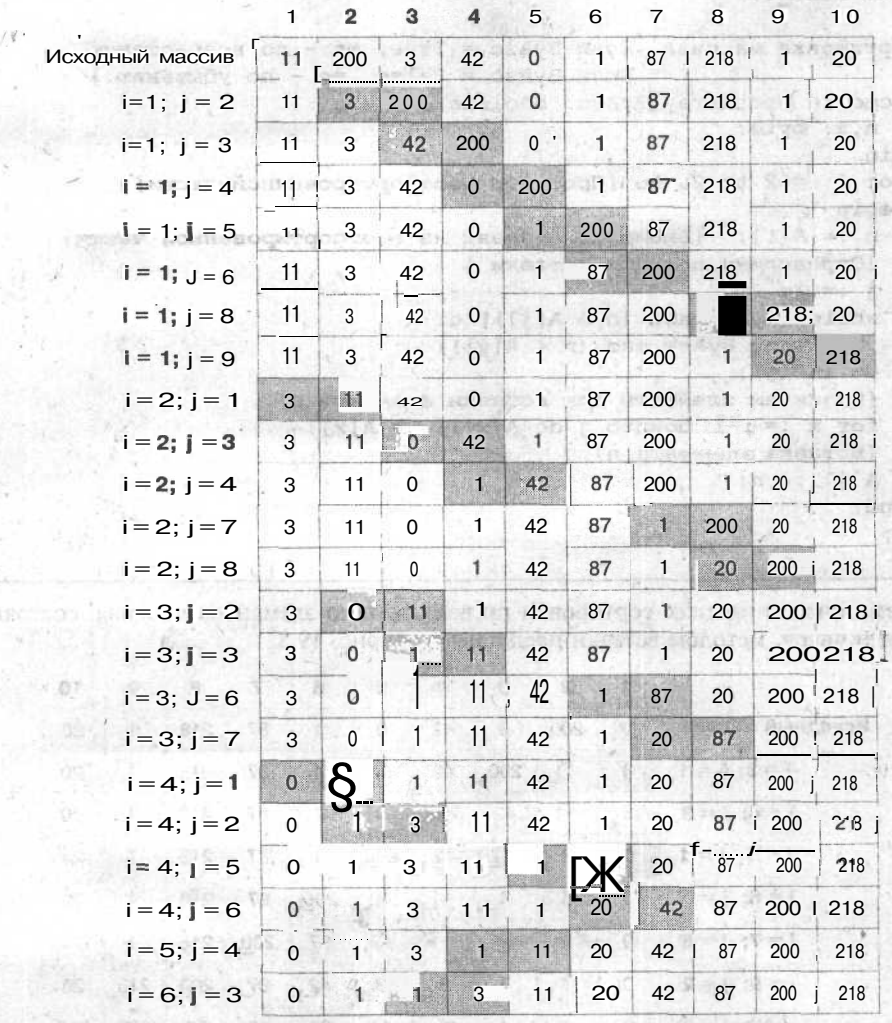


Рис. 19.2. Сортировка элементов массива по возрастанию методом "пузырька"

Сортировка вставкой

При сортировке методом вставки массив разбивается на две части: отсортированную и неотсортированную. Из неотсортированной части поочередно выбираются элементы, которые затем вставляются в отсортированную часть в соответствии со своим значением. В начале в качестве отсортированной части массива выступает только один первый элемент, а в качестве неотсортированной — все остальные элементы массива.

Сохраните файл `Sort_2.pas` (см. предыдущий раздел) под именем `Sort_3.pas` и внесите изменения в процедуру `SortArray`, выделенные в листинге 19.3 полужирным шрифтом, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.3. Программа Sort_3.pas

```
program Sort_3;
...
  {Сортировка массива. Если ByAsc = True, то - по возрастанию.
                       Если ByAsc = False, то - по убыванию.}
  procedure SortArray(ByAsc: boolean);
  var n,x: byte;
  begin
    for i := 2 to 20 do {Просмотр неотсортированной части}
    begin
      n := A[i];   {Выбираем элемент из неотсортированной части}
      {Определяем позицию вставки }
      j :- 1;
      while (ByAsc and (n > A[j])) or
            (not ByAsc and (n < A[j]))
      do Inc(j);
      {Сдвигаем элементы для вставки элемента n}
      for x := i-1 downto j do A[x+1] :=A[x];
      {Вставка элемента n}
      A[j] :=n;
    end;
  end;
...
```

Схематически процесс сортировки по возрастанию элементов массива, состоящего из десяти чисел, методом вставки представлен на рис. 19.3.

	1	2	3	4	5	6	7	8	9	10
Исходный массив	11	200	3	42	0	1	87	218	1	20
i = 3; j = 1	3	11	200	42	0	1	87	218	1	20
i = 4; j = 3	3	11	42	200	0	1	87	218	1	20
i = 5; j = 1	0	3	11	42	200	1	87	218	1	20
i = 6; j = 2	0	1	3	11	42	200	87	218	1	20
i = 7; j = 6	0	1	3	11	42	87	200	218	1	20
i = 9; j = 2	0	1	1	3	11	42	87	200	218	20
i = 10; j = 6	0	1	1	3	11	20	42	87	200	218

Рис. 19.3. Сортировка элементов массива по возрастанию методом вставки

Быстрая сортировка с разделением

При быстрой сортировке вначале определяется элемент, расположенный в центре массива. Затем просматриваются все значения, расположенные в левой части, и те из них, которые больше (в случае сортировки по возрастанию) значения центрального элемента меняются с теми элементами, расположенными в правой части, значения которых меньше значения центрального элемента. Таким образом, по окончанию этого цикла в левой части массива не останется ни одного элемента больше центрального. Затем каждая из частей в свою очередь разбивается на две части и для каждой из них повторяется описанный выше цикл. Такое разбиение выполняется до тех пор, пока одна из частей не будет сведена до одного элемента.

Сохраните файл Sort_3.pas (см. листинг 19.3) под именем Sort_4.pas и внесите изменения в процедуру SortArray, выделенные в листинге 19.4 полужирным шрифтом, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.4. Программа Sort_4 . pas

```
program Sort_4;
...
  {Сортировка массива. Если ByAsc = True, то - по возрастанию.
                       Если ByAsc = False, то - по убыванию.}
  procedure SortArray(ByAsc:boolean; First,Last: byte);
  var
    mid: byte;
  begin
    i := First;  {Левая граница}
    j := Last;   {Правая граница}
    {Определяем центральный элемент}
    mid := A[ (First + Last) div 2];
    repeat
      {Находим в левой части позицию элемента,
       который необходимо перенести в правую часть)
      while (ByAsc and (A[i] < mid)) or (not ByAsc and (A[i] > mid))
      do Inc(i);
       (Находим в правой части
       позицию элемента, который можно перенести в левую часть}
      while (ByAsc and (A[j] > mid» or (not ByAsc and (A[j] < mid))
      do Dec(j);                          j
      if i <= j then
      begin  {Обмениваем местами элементы из левой и правой части)
        buf :=A[i];
        A[i] := A[j];
        A[j] := buf;
        Inc(i); {Смещаем позицию в левой}
        Dec(j); (и в правой части к центру}
      end;
    until i > j; {Повторяем цикл до тех пор, пока не произойдет
                  "встреча" при встречном просмотре}
    (Рекурсивный вызов процедуры для разбиения левой части)
    if First < j then SortArray(ByAsc,First,j);
    (Рекурсивный вызов процедуры для разбиения правой части}
    if i < Last then SortArray(ByAsc,i,Last);
  end;

begin
  ...
  SortArray(True,1,20);  {Сортировка по возрастанию}
  ...
  SortArray(False,1,20); {Сортировка по убыванию}
  ...
end;
```

Схематически процесс быстрой сортировки элементов массива представить сложно по причине множества вложенных рекурсивных вызовов процедуры SortArray.

Бинарный поиск в упорядоченном массиве

Для поиска некоторого элемента в упорядоченном массиве можно воспользоваться простым последовательным перебором его элементов, однако использование такого подхода при работе с массивами большого размера занимает много времени. Одним из наиболее эффективных методов поиска в больших упорядоченных массивах является *бинарный поиск*. При таком подходе массив делится пополам, и определяется его центральный элемент. Если центральный элемент равен искомому значению, то поиск прекращается. Если центральный элемент больше искомого значения, то эта же операция разбиения применяется к левой части массива. Если же центральный элемент меньше искомого значения, то поиск продолжается в правой части.

Откройте в интегрированной среде Turbo Pascal один из файлов с текстом программы сортировки массива (например, листинг 19.4), сохраните его под именем BSearch.pas и вставьте дополнения, выделенные в листинге 19.5 полужирным шрифтом, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.5. Программа BSearch.pas

```
program BSearch;
uses Crt;
var
  A: array[1..20] of byte;
  i,j,buf,n: byte;
  Found: boolean;
...
begin
  ...
  {Поиск в массиве, отсортированном по убыванию}
  Write('Введите число для поиска: ');
  Readln(n);
  buf := 0;
  i := 1;     {Левая граница}
  j := 20;    {Правая граница}
  Found := False;
  repeat
    buf := (i + j) div 2; {Номер центрального элемента}
    if A[buf] = n then    {Если центральный элемент равен искомому)
    begin
      Found := True; {Устанавливаем флажок}
      break;         {Выходим из цикла}
    end;
    {Если искомый элемент больше центрального, то смещаем левую границу}
    if A[buf] < n then j := buf - 1;
    {Если искомый элемент меньше центрального, то смещаем правую границу)
    if A[buf] > n then i := buf + 1;
  until i > j;
  if Found
  then Writeln('Соответствует ',buf,'-му элементу')
  else Writeln('Такое число в массиве отсутствует');
  ReadKey;
end.
```

Операции над матрицами

Рассмотрим ряд примеров, в которых обрабатываются двухмерные массивы или матрицы. В первом примере заполним квадратную матрицу случайными целыми числами, а затем запишем среднее арифметическое элементов строк в те элементы матрицы, которые расположены на центральной диагонали.

Создайте в интегрированной среде программирования Turbo Pascal новый файл, сохраните его под именем `Matrix_1.pas` и введите в него текст, представленный в листинге 19.6, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.6. Программа `Matrix_1.pas`

```
program Matrix_1;
uses Crt;

var
  A: array [1..5,1..5] of real;
  i,j: byte;
  sum: real;

  {Вывод элементов массива}
  procedure ShowArray;
  begin
    for i := 1 to 5 do
    begin
      for j := 1 to 5 do Write(A[i,j]:6:1); '
      Writeln('');
    end;
  end;

begin
  ClrScr;
  {Заполняем матрицу случайными числами}
  Randomize;
  for i := 1 to 5 do
    for j := 1 to 5 do A[i,j] := Random(255);
  Writeln('Исходная матрица:');
  ShowArray;
  {Вычисляем среднее арифметическое элементов строк)
  for i := 1 to 5 do
  begin
    sum := 0;  {Сумма элементов строки}
    for j := 1 to 5 do sum := sum + A[i,j];
    A[i,i] := sum/5;
  end;
  Writeln('Полученная матрица:');
  ShowArray;
  ReadKey;
end.
```

Схематически результат работы программы `Matrix_1` представлен на рис. 19.4.

Во втором примере реализуем сортировку всех элементов матрицы, расположенных слева от главной диагонали, по возрастанию, а расположенных справа — по убыванию. Создайте в интегрированной среде программирования Turbo Pascal новый файл, сохраните его под именем `Matrix_2.pas` и введите в не-

го текст, представленный в листинге 19.7, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

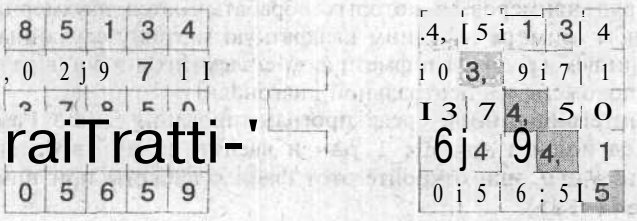


Рис. 19.4. Результат работы программы Matrix_1

Листинг 19.7. Программа Matrix_2.pas

```
program Matrix_2;
uses Crt;
var
  A: array [1..5,1..5] of byte;
  B: array[1..10] of byte;   {Временный массив}
  i,j: byte;

  {Вывод элементов массива}
  procedure ShowArray;
  begin
    for i := 1 to 5 do
    begin
      for j := 1 to 5 do Write(A[i,j]:4);
      Writeln('');
    end;
  end;

  {Сортировка массива. Если ByAsc = True, то - по возрастанию.
                       Если ByAsc = False, то - по убыванию.}
  procedure SortArray(ByAsc: boolean);
  var
    buf: byte;
  begin
    for i := 1 to 9 do
      for j := i+1 to 10 do
        if (ByAsc and (B[i] > B[j])) or
           (not ByAsc and (B[i] < B[j]))
        then begin
          buf := B[i];
          B[i] := B[j];
          B[j] := buf;
        end;
  end;

  {Процедура копирования элементов из одного массива в другой.
   Если FromAtoB = True, то копируем из A в B
   Если FromAtoB = False, то копируем из B в A}
  procedure CopyArrays(Lefthalf, FromAtoB: boolean);
  var
    iBegin, iEnd, jBegin, jEnd, c: byte;
```

Окончание листинга 19.7

```
  begin
    {Определяем начальное и конечное значения
    счетчиков циклов}
    if LeftHalf then
    begin
      {Для левой части матрицы}
      iBegin := 2; iEnd := 5;
      jBegin := 1; jEnd := 1;
    end else
    begin
      {Для правой части матрицы}
      iBegin := 1; iEnd := 4;
      jBegin := 2; jEnd := 5;
    end;
    {Копируем элементы из массива в массив}
    c := 1; {Счетчик временного массива}
    for i := iBegin to iEnd do
    begin
      for j := jBegin to jEnd do
      begin
        if FromAtoB
        then B[c] := A[i,j]
        else Afi,j] := B[c];
        Inc(c);
      end;
      if LeftHalf
        then Inc(jEnd) else Inc(jBegin);
    end;
  end;

  {Если LeftHalf = True, то сортируем
   левую часть, иначе - правую часть}
  procedure SortHalf(LeftHalf: boolean);
  begin
    CopyArrays(LeftHalf,True); {Формируем временный массив}
    SortArray(LeftHalf);       {Сортируем временный массив}
    {Переписываем значения в матрицу}
    CopyArrays(LeftHalf,False);
  end;

begin
  ClrScr;
  {Заполняем матрицу случайными числами}
  Randomize;
  for i := 1 to 5 do
    for j := 1 to 5 do A[i,j] := Random(255);
  Writeln('Исходная матрица:');
  ShowArray;
  SortHalf(True) ;  {Сортируем левую часть}
  SortHalf(False); {Сортируем правую часть}
  Writeln('Полученная матрица:');
  ShowArray;
  ReadKey;
end.
```

Схематически результат работы программы Matrix_2 представлен на рис. 19.5.

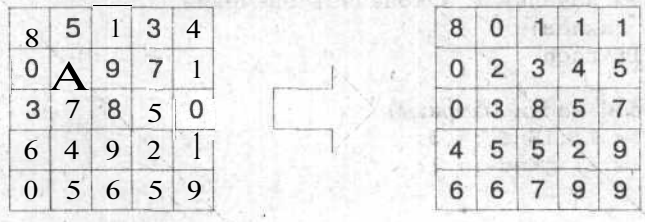


Рис. 19.5. Результат работы программы Matrix_2

В третьем примере выполним зеркальное отображение матрицы относительно главной диагонали. Создайте в интегрированной среде программирования Turbo Pascal новый файл, сохраните его под именем Matrix_3.pas и введите в него текст, представленный в листинге 19.8, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 19.8. Программа Matrix_3.pas

```
program Matrix_3;
uses Crt;
var
  A: array [1..5,1..5] of byte;
  i,j: byte;
  {Вывод элементов массива}
  procedure ShowArray;
  begin
    for i := 1 to 5 do
    begin
      for j := 1 to 5 do Write(A[i,j]:4);
      Writeln('');
    end;
  end;
  procedure TranspArray;
  var
    jEnd,buf: byte;
  begin
    jEnd := 1;
    for i := 2 to 5 do
    begin
      for j := 1 to jEnd do
      begin
        buf := A[i,j];
        A[i,j] := A[j,i];
        A[j,i] := buf;
      end;
      Inc(jEnd);
    end;
  end;
```

Окончание листинга 19.8

```
begin
  ClrScr;
  {Заполняем матрицу случайными числами}
  Randomize;
  for i := 1 to 5 do
    for j := 1 to 5 do A[i,j] :=Random(255);
  Writeln('Исходная матрица:');
  ShowArray;
  TranspArray; {Зеркальное отображение матрицы}
  Writeln('Полученная матрица:');
  ShowArray;
  ReadKey;
end.
```

Схематически результат работы программы Matrix_3 представлен на рис. 19.6.

Рис. 19.6. Результат работы программы Matrix_3

Глава 20

Работа со звуком и мышью

В этой главе рассматривается три примера, в которых используется звуковое сопровождение программ и мышь. Для работы со звуком в языке Pascal используются три процедуры из модуля Crt: Sound, NoSound и Delay. Процедура Sound генерирует звук, частота которого, выраженная в герцах, передается в эту процедуру в качестве единственного параметра типа Word. Отмена звука выполняется вызовом процедуры NoSound. Процедура Delay вызывается между вызовами процедур Sound и NoSound и задерживает звучание на указанное количество миллисекунд. Так, вызов процедуры Delay (1000) приостанавливает работу программы на одну секунду.

Для работы с мышью во втором и третьем примерах, рассмотренных в этой главе, используются различные функции прерывания с номером $33.

◄◄ см. подробно о прерываниях в гл. 17.

Имитация клавиатуры фортепиано

Разработаем программу, отображающую в графическом режиме первую октаву клавиатуры фортепиано и проигрывающую текущий звук, выбранный пользователем. Частоты звуков музыкального лада фортепиано представлены в табл. 20.1.

Таблица 20.1. Частоты звуков музыкального лада фортепиано

Нота	Большая октава	Малая октава	Первая октава	Вторая октава
До (C)	131	262	523	1047
До-диез (C#)	139	278	555	1111
Ре (D)	147	294	587	1174
Ре-диез (D#)	156	312	623	1246
Ми (E)	165	330	659	1318
Фа (F)	175	349	699	1396
Фа-диез (F#)	186	371	742	1482
Соль (G)	196	392	785	1568
Соль-диез (G#)	208	416	832	1664
Ля (A)	220	440	880	1760
Ля-диез (A#)	234	467	934	1868
Си (H)	247	494	988	1975

Для выбора звука в программе используются клавиши, приведенные в табл. 20.2.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем Piano.pas и введите в него текст, представленный в листинге 20.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Таблица 20.2. Клавиши имитирующие клавиатуру фортепиано

Раскладки клавиатуры		Описание
Английская	Русская	
<→>		Переход от белой клавиши к белой клавише вправо
<←>		Переход от белой клавиши к белой клавише влево
<↑>		Переход от белой клавиши к одноименному диезу (если он есть)
<↓>		Пперехо д от белой клавиши к диезу предыдущей белой клавиши (если он есть)
<c> *	<с>	Нота "до"
<f>	<а>	Нота "до-диез"
<v>	<м>	Нота "ре"
<д>	<п>	Нота "ре-диез"
<Ь>	<и>	Нота "ми"
<п>	<т>	Нота "фа"
<j>	<о>	Нота "фа-диез"
<т>	<ь>	Нота "соль"
<k>	<л>	Нота "соль-диез"
<,>	<б>	Нота "ля"
<1>	<д>	Нота "ля-диез"
< . >	<ю>	Нота "си"

Листинг 20.1. Программа `Piano.pas`

```
program Piano;
uses Graph, Crt;

var
  DriverVar, ModeVar, ErrorCode,
  MouseX, MouseY, CurNote, CurDelay: integer;
  PressedKey: Char;
  CurDiez: boolean;
  Notes: string;

  {Процедура прорисовки маркера текущей выбранной ноты note - символ,
   соответствующий ноте. Если diez = True, то клавиша - диезная.
                         Если IsPoint = True, то маркер прорисовывается,
                         если IsPoint = True, то маркер удаляется}
  procedure DrawPosition(note: char; diez,IsPoint: boolean);
  var
    x: integer; (Координата X верхнего левого угла отображаемой клавиши}
  begin
    if diez   {Если рисуем черную (диезную) клавишу}
    then begin
      SetColor(DarkGray); {Цвет прорисовки - темно-серый}
      (Координата X определяется на основании позиции
       текущей ноты в строке нот Notes. 50 - ширина белой клавиши}
      x := 170 + (pos(note,Notes) - 1)*50;
      if IsPoint
      then SetFillStyle (SolidFill, Red)
```

Продолжение листинга 20.1

```
      else SetFillStyle(SolidFill,DarkGray);
      FillEllipse(x,125,5,5); {Рисуем маркер}
    end else begin {Если рисуем белую клавишу}
      SetColor(White); {Цвет прорисовки - белый}
      x := 145 + (pos(note,Notes) - 1>*50;
      if isPoint
      then SetFillStyle(SolidFill,Red)
      else SetFillStyle(SolidFill,White);
      FillEllipse(x,190,5,5);
    end;
  end;

  {Прорисовка клавиш}
  procedure DrawKey(note: char; diez: boolean);
  var
    x: integer; {Координата X верхнего левого угла клавиши}
  begin
    if diez {Если рисуем диезную (черную) клавишу}
    then begin
      x := 155 + (pos(note,Notes) - 1)*50;
      SetFillStyle(SolidFill,DarkGray);
      Bar(x,50,x+30,135); {Заполненный прямоугольник}
    end else begin {Если рисуем белую клавишу}
      x := 120 + (pos(note,Notes) - 1)*50;
      SetFillStyle(SolidFill,White); {Цвет клавиши}
      SetColor(DarkGray); {Цвет края клавиши}
      Rectangle(x, 50, x+50, 200);
      FloodFill(x+1,51,DarkGray);
    end;
  end;

  {Процедура проигрывания ноты}
  procedure PlayNote(note: char; diez: boolean);
  begin
    case note of
  {До}'C': if diez then Sound(555). else Sound(523);
  {Ре}'D': if diez then Sound(623) else Sound(587);
  {Ми}'E': Sound(659);
  {Фа}'F': if diez then Sound(742) else Sound(699);
{Соль}'G': if diez then Sound(832) else Sound(785);
  {Ля}'A': if diez then Sound(934) else Sound(880);
  {Си}'H': Sound(988);
    end;
    Delay(CurDelay*1000); {Длительность звучания
                           CurDelay=1 - четвертная нота
                           CurDelay=2 - половинная нота
                           CurDelay=4 - целая нота}
    NoSound; {Отмена звука}
  end;

  {Процедура прорисовки текущей длительности звучания}
  procedure DrawDelay;
  begin {Закрашиваем область вывода обозначения
         длительности светло-серым цветом фона}
```

Продолжение листинга 20.1

```
    SetFillStyle(SolidFill,LightGray);
    Bar(200,210,500,350);
    {Устанавливаем цвет рисования и заливки}
    SetFillStyle(SolidFill,DarkGray);
    SetColor(DarkGray);
    case CurDelay of
      1: begin {Если длительность - четвертная}
           FillEllipse(300,300,30,15);
           Line(330,300,330,220);
         end;
      2: begin (Если длительность - половинная}
           Ellipse(300,300,0,360,30,15);
           Line(330,300,330,220);
         end;
      4: Ellipse(300,300,0,360,30,15); {Целая нота}
    end;
  end;

begin
  Notes := 'CDEFGAH'; {Последовательность нот}
  CurDelay := 1; {Текущая длительность нот - четвертная}
  {Инициализируем графический режим}
  DriverVar := EGA;
  ModeVar := EGAHI;

InitGraph(DriverVar,ModeVar,'\tp\tp\bgi');
  {Проверяем наличие ошибок инициализации графического режима}
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  SetBkColor(LightGray); {Цвет фона - светло-серый}
  ClearDevice; {Заполняем экран светло-серым цветом}
  DrawKey('C',False);   {Рисуем клавишу До}
  DrawKey('D',False);   {Рисуем клавишу Ре}
  DrawKey('E',False);   {Рисуем клавишу Ми}
  DrawKey('F',False);   {Рисуем клавишу Фа}
  DrawKey('G',False);   {Рисуем клавишу Соль}
  DrawKey('A',False);   {Рисуем клавишу Ля}
  DrawKey('H',False);   {Рисуем клавишу Си}
  DrawKey('C',True);    {Рисуем клавишу До-диез}
  DrawKey('D',True);    {Рисуем клавишу Ре-диез}
  DrawKey('F',True);    {Рисуем клавишу Фа-диез}
  DrawKey('G',True);    {Рисуем клавишу Соль-диез}
  DrawKey('A',True);    {Рисуем клавишу Ля-диез}
  CurNote := 1; {Текущая нота — До}
  CurDiez := False; {Без диеза}
  DrawPosition(Notes[CurNote],CurDiez,True); {Рисуем маркер}
  DrawDelay; {Рисуем обозначение длительности}
  {Обрабатываем в цикле нажатие клавиш клавиатуры}
  repeat
```

Продолжение листинга 20.1

```
{Проигрываем текущую ноту, если PressedKey не равна #0}
if PressedKey o #0 then PlayNote(Notes[CurNote],CurDiez);
PressedKey := ReadKey; {Считываем клавишу}
DrawPosition(Notes[CurNote],CurDiez,False); {Удаляем маркер в
                                             старой позиции}
case PressedKey of
  {Если стрелка вверх и нота до, ре, фа, соль или ля,
   то переходим к диезной черной клавише}
  #72: CurDiez := (CurNote in [1,2,4,5,6]);
  {Если стрелка влево и текущая клавиша - диезная,
   то переходим к одноименной белой клавише}
  #75: if not ((CurNote = 1) and not CurDiez)
       then if CurDiez then CurDiez := False
            else Dec(CurNote);
  {Если стрелка вправо, и нота - меньше "си"}
  #77: if CurNote < 7
       then begin
         Inc(CurNote); {Переходим к следующей белой клавише}
         {Если текущая клавиша диезная, то отменяем диез}
         if CurDiez then CurDiez := False;
       end;
  {Если стрелка вниз, и текущая клавиша - белая, то
   переходим к предыдущей диезной клавише}
  #80: if not ((CurNote = 1) and not CurDiez)
       then if not CurDiez then
            begin
              CurDiez := (CurNote in [2,3,5,6,7]);
              Dec(CurNote);
            end;
  {Прямой доступ к клавишам}
  'C','c','С','с':begin
                    CurNote := 1;
                    CurDiez := False;
                  end;
  'F','f','A','a': begin
                     CurNote := 1;
                     CurDiez := True;
                   end;
  'V','v','М','м': begin
                     CurNote := 2;
                     CurDiez := False;
                   end;
  'G','g','П','п':begin
                     CurNote := 2;
                     CurDiez := True;
                   end;
  'B','b','И','и': begin
                     CurNote := 3;
                     CurDiez := False;
                   end;
  'N','n','Т','т':begin
                     CurNote := 4;
```

Окончание листинга 20.1

```
                        CurDiez := False;
                      end;
    'J','j','O','o': begin
                        CurNote := 4;
                        CurDiez := True;
                      end;
    'M','m','Ь','ь':begin
                        CurNote := 5;
                        CurDiez := False;
                      end;
    'K','k','Л','л':begin
                        CurNote := 5;
                        CurDiez := True;
                      end;
    '<',',','Б','б': begin
                        CurNote := 6;
                        CurDiez := False;
                      end;
    'L','l','Д','д': begin
                        CurNote := 6;
                        CurDiez := True;
                      end;
    '>','.','Ю','ю': begin
                        CurNote := 7;
                        CurDiez := False;
                      end;
    {Увеличение длительности звучания при помощи "+"}
    '+','=': if CurDelay < 4 then
             begin
               {Увеличиваем текущее значение длительности}
               Inc(CurDelay,CurDelay);
               DrawDelay; {Прорисовка обозначения}
             end;
    {Увеличение длительности звучания при помощи "-"}
    '-','_': if CurDelay > 1 then
             begin
               Dec(CurDelay,CurDelay div 2);
               DrawDelay;
             end;
  end;
  {Рисуем маркер на новой выбранной клавише}
  DrawPosition(Notes[CurNote],CurDiez,True);
until PressedKey = #27; {Выход из цикла - по Esc}
CloseGraph; {Выходим из графического режима}
end.
```

Результат работы этой программы представлен на рис. 20.1.

Рисование при помощи мыши

Разработаем программу, позволяющую выбрать при помощи мыши и ячеек палитры фоновый цвет экрана, а также цвет пера. В данной программе будет рассмотрено

только так называемое "свободное" рисование без использования каких-либо геометрических фигур.

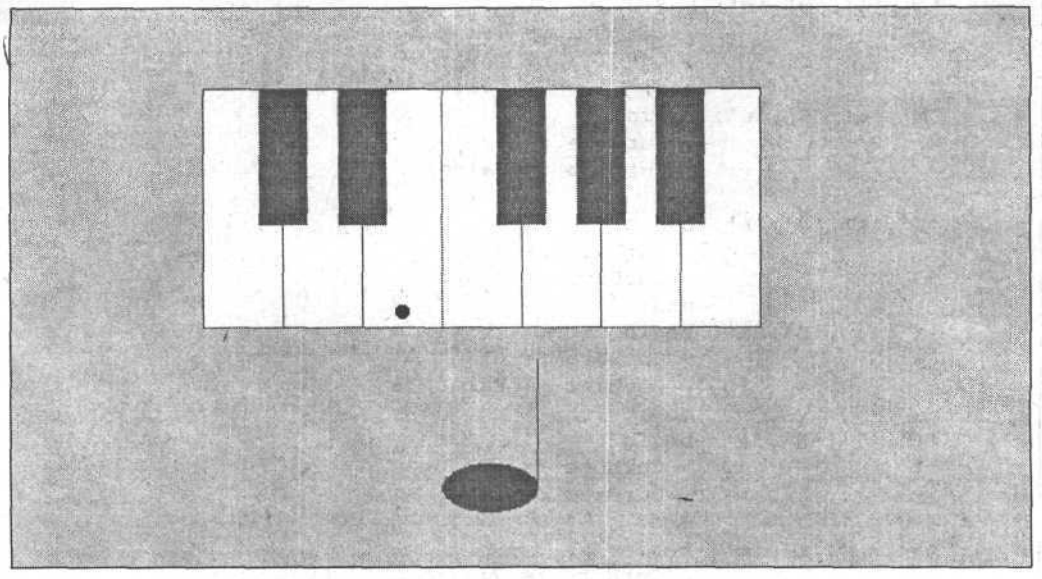


Рис. 20.1. Результат работы программы Piano

Для доступа к мыши в этой программе используются следующие функции прерывания $33.

- $01 — отображение указателя мыши.
- $03 — извлечение данных о нажатой в данный момент кнопке мыши и координатах указателя на экране.
- $04 — позиционирование указателя мыши на экране.
- $07 — установка допустимого диапазона действия мыши по горизонтали.
- $08 — установка допустимого диапазона действия мыши по вертикали.

Скорость перемещения указателя мыши может быть не равномерной, поэтому в программе используется проверка предыдущего положения указателя с текущим. В результате этой проверки, длина отображаемой линии равна не реальному размеру смещения указателя, а только одному пикселю в направлении перемещения указателя. Если не выполнить такой коррекции, то линии при более медленном перемещении указателя мыши будут изображаться как разреженные.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем FreePen.pas и введите в него текст, представленный в листинге 20.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 20.2. Программа FreePen.pas

```
program FreePen;
uses Graph, Crt;
var
  DriverVar, ModeVar, ErrorCode: integer;
  ButtonPressed,MouseX,MouseY,PrevX,PrevY: word;
  i,CurColor: byte;
```

Продолжение листинга 20.2

```
  {Процедура прорисовки маркера выбранного цвета в палитре}
  procedure DrawMarker;
  begin
    SetColor(15 - CurColor); {Устанавливаем цвет, инверсный текущему}
    OutTextXY(26+CurColor*39, 10,'X'); {Выводим "X"}
  end;

begin
  {Инициализация графического режима}
  DriverVar := EGA;
  ModeVar := EGAHI;
  InitGraph(DriverVar,ModeVar,'\tp\tp\bgi');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt (1);
  end;
  {Текущий цвет рисования — белый}
  SetColor(White);
  {Рисуем палитру}
  for i := 0 to 15 do
  begin
    SetFillStyle(SolidFill,i); {Заполнение текущим цветом палитры}
    Rectangle(10+i*39, 5, 10+(i+1)*39, 20); {Рисуем ячейку палитры }
    FloodFill(11+i*39, 6, White);
  end;
  CurColor := White; {Текущий цвет — белый}
  DrawMarker; {Рисуем маркер текущего цвета}
  asm
    MOV AX,01   {Функция отображения указателя)
    INT* 33H
    MOV AX,07   {Функция установки допустимого диапазона
                 перемещения указателя по горизонтали}
    MOV CX,01   {Координата X1 диапазона}
    MOV DX,640  {Координата X2 диапазона}
    INT 33H
    MOV AX,08   (Функция установки допустимого диапазона
                 перемещения указателя по вертикали}
    MOV CX,01   {Координата Y1 диапазона}
    MOV DX,350  {Координата Y2 диапазона}
    INT 33H
    MOV AX,04   {Позиционирование указателя на экране}
    MOV CX,320  {Координата X}
    MOVDX,175  {Координата Y}
    INT 33H
  end;
  {Цикл, выполняемый до тех пор, пока не будет нажата какая-либо клавиша}
  while not KeyPressed do
  begin
    asm
      MOV AX,03   {Функция извлечения данных о мыши}
      INT 33H
```

Окончание листинга 20.2

```
      MOV ButtonPressed,BX {В регистре BX - номер нажатой кнопки мыши}
      MOV MouseX,CX        {В регистре CX - координата X указателя мыши}
      MOV MouseY,DX        {В регистре DX - координата Y указателя мыши}
    end;
    if ButtonPressed = 0 {Если не нажата никакая кнопка}
    then PrevX := 0  {Обнуляем предыдущие координаты}
    else {Если нажата какая-либо кнопка мыши}
         if MouseY <= 20 then {Если указатель находится над палитрой }
         begin
           case ButtonPressed of
             1: begin {Если нажата левая кнопка}
                  {Перенос маркера в палитре:}
                  {Закрашиваем маркер в текущей позиции}
                  SetFillStyle(SolidFill,CurColor);
                  Bar(22+CurColor*39,6,
                      33+CurColor*39,19);
                  {Изменяем текущий выбранный цвет}
                  CurColor := (MouseX-10) div 39;
                  {Рисуем маркер на новой позиции}
                  DrawMarker;
                end;
             2: begin {Если нажата правая кнопка мыши}
                  {Устанавливаем выбранный цвет для заливки}
                  SetFillStyle(SolidFill,(MouseX-10) div 39);
                  Bar (0,21,640,350); {Заполняем экран}
                end;
           end;
         end
         else begin {Если кнопа мыши нажата над
                     областью рисования}
           SetColor(CurColor); {Выбираем для рисования
                                текущий цвет в палитре}
           if PrevX > 0 then begin {Если новая линия}
             {Коррекция конечной точки линии - смещение уменьшается до
              одного пикселя в направлении перемещения указателя мыши}
             if MouseX > PrevX
             then MouseX := PrevX+1
             else MouseX := PrevX-1;
             if MouseY > PrevY
             then MouseY := PrevY+1
             else MouseY := PrevY-1;
             {Рисуем линию от предыдущего положения
              указателя до текущего с учетом коррекции}
             Line(PrevX,PrevY,MouseX,MouseY);
           end;
           {Запоминаем последнее положение указателя}
           PrevX := MouseX;
           PrevY := MouseY;
         end;
  end;
  CloseGraph; {Выход из графического режима}
end.
```

Результаты работы этой программы представлены на рис. 20.2.

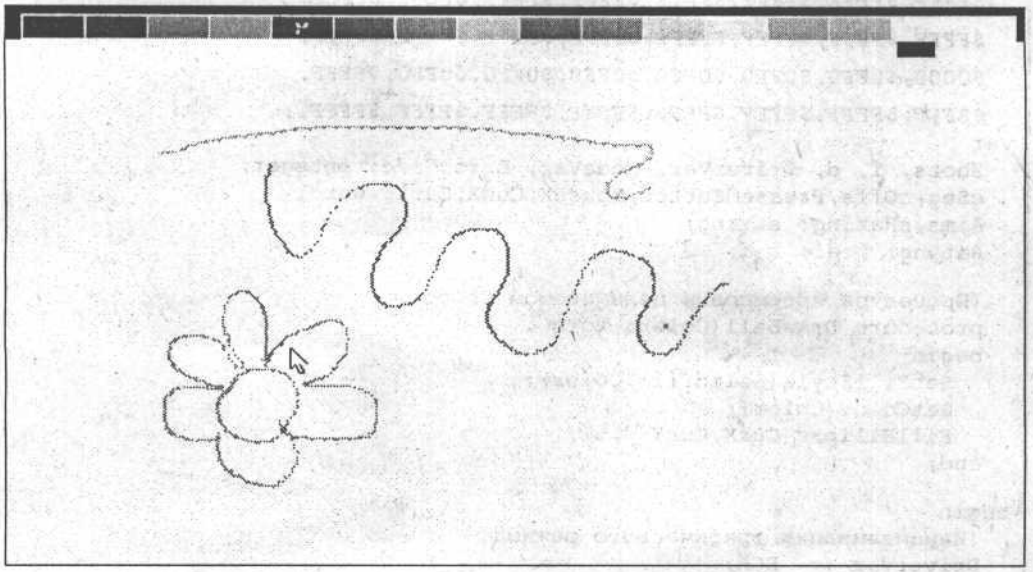


Рис. 20.2. Результат работы программы FreePen

Игра "Охотник"

Продолжим рассмотрение возможностей использования мыши и звука на примере простой игры "Охотник". Суть игры заключается в том, что указатель мыши выполняет роль "охотника", который может перемещаться строго вдоль одной горизонтальной линии по экрану и "стрелять" вверх при помощи нажатия левой кнопки мыши. Над охотником на разных высотах поочередно пролетают "утки" (в игре им соответствуют желтые эллипсы). Если "охотник" попадает в "утку", то она исчезает, а на экране отображается количество сбитых "уток" и общий рейтинг "охотника". Рейтинг определяется как соотношение количества сбитых "уток" к количеству сделанных "выстрелов". Выстрелы и попадания в "утку" имеют соответствующее звуковое сопровождение и отображаются на экране при помощи специальных графических эффектов.

В этой программе используется нестандартный указатель мыши, который устанавливается при помощи функции $09 прерывания $33. При этом в регистр ES загружается сегмент (при помощи функции Seg), а в регистр DX — смещение адреса массива Cursor (при помощи функции Ofs), в котором хранится двоичное описание формы указателя. Координаты центра нового указателя относительно его левого верхнего угла задаются в регистрах BX и CX.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем `Hunter.pas` и введите в него текст, представленный в листинге 20.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 20.3. Программа `Hunter.pas`

```
program Hunter;
uses Graph, Crt;
const Cursor:array [1..32] of word=
```

Продолжение листинга 20.3

```
  ($FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,
   $FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,
   $0000,$0FF0,$0FF0,$0FF0,$0FF0,$0FF0,$0FF0,$FFFF,
   $FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF);
var
  Shots, i, d, DriverVar, ModeVar, ErrorCode: integer;
  cSeg,cOffs,PressedButton,MouseX,CurX,CurY: word;
  Aims,sRating: string;
  Rating: real;

  {Процедура прорисовки цели цветом Color}
  procedure DrawBall(Color: Word);
  begin
    SetFillStyle(SolidFill,Color);
    SetColor(Color);
    FillEllipse(CurX,CurY,7,5);
  end;

begin
  {Инициализация графического режима}
  DriverVar := EGA;
  ModeVar := EGAHI;
  InitGraph(DriverVar,ModeVar,'\tp\tp\bgi');
  ErrorCode := GraphResult;
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);
  end;
  cSeg := Seg(Cursor);  {Определяем сегмент массива Cursor}
  cOffs := Ofs(Cursor); {Определяем смещение адреса массива Cursor}
  asm
    MOV AX,01 {Функция отображения указателя }
    INT 33H
    MOV AX,09 {Функция определения формы указателя мыши}
    MOV BX,7  {Относительная координата X центра указателя}
    MOV CX,0  {Относительная координата Y центра указателя}
    MOV ES,cSeg  {Указываем сегмент для формы указателя}
    MOV DX,cOffs {Указываем смещение для формы указателя}
    INT 33H
    MOV AX,07  {Функция установки допустимого диапазона
                перемещения указателя по горизонтали}
    MOV CX,01  {Координата X1 диапазона}
    MOV DX,620 {Координата X2 диапазона}
    INT 33H
    MOV AX,08  {Функция установки допустимого диапазона
                перемещения указателя по вертикали}
    MOV CX,300 {Указатель может перемещаться только}
    MOV DX,300 {по одной линии — Y1 = Y2 = 300}
    INT 33H
    MOV AX,04  {Позиционирование указателя на экране}
    MOV CX,320 {Координата X}
```

Продолжение листинга 20.3

```
    MOV DX,300 {Координата Y}
    INT 33H
  end;
  Randomize;    {Активизируем генератор случайных чисел}
  CurY := 0;    {Текущая координата Y для цели}
  CurX := 700; {Текущая координата X для цели (за пределами экрана)}
  Aims := '';  {Строка, фиксирующая количество сбитых целей}
  Shots := 0;   {Общее количество выстрелов}
  {Выполняется цикл до тех пор, пока не будет
   нажата какая-либо клавиша на клавиатуре}
  while not KeyPressed do
  begin
    asm
      MOV AX,03  {Функция извлечения состояния мыши}
      INT 33H
      MOV PressedButton,BX {Нажатая кнопка}
      MOV MouseX,CX  {Координата X указателя мыши}
    end;
    DrawBall(Black);  {Удаляем цель в текущем положении}
    if CurX > 640 then {Если цель вышла за пределы экрана}
    begin
      CurX := 0;  {Устанавливаем отсчет с начала экрана}
      CurY := 5 + Random(200);  {Определяем случайным образом высоту}
    end;
    Inc(CurX);  {Увеличиваем координату X цели}
    DrawBall(Yellow);  {Рисуем цель в новом положении}
    Delay(300);  {Небольшая задержка, определяющая скорость движения
                  цели. Чем больше задержка, тем медленнее движется цель}
    if PressedButton = 1 then {Если нажата левая кнопка мыши}
    begin                     {то совершаем "выстрел"}
      Inc(Shots);     {Увеличиваем количество выстрелов}
      Sound(1900);    {Генерируем звук "выстрела"}
      {Отображаем выстрел в виде красной вертикальной
       линии, выходящей из середины указателя мыши}
      SetColor(Red);
      Line(MouseX,1,MouseX,300);
      Delay(100);  {Для того чтобы "выстрел"
                    был заметен, делаем небольшую задержку}
      {Затираем красный "след выстрела" фоновым цветом}
      SetColor(Black);
      Line(MouseX,1,MouseX,300);
      NoSound;  {Отключаем звук "выстрела"}
      {Если ось "выстрела" отстоит от центра цели не
       далее, чем на 5 пикселей/ то...}
      if abs(MouseX-CurX) = 5 then
      begin
        {Отображаем""поражение цели"}
        DrawBall(Red);  {Закрашиваем цель красным цветом}
        {Создаем звуковой эффект начала ``взрыва цели"}
        -for i := 200 to 800 do
        begin
          if (i mod 20) = 0 then Sound(i);
          Delay(20);
```

Окончание листинга 20.3

```
        end;
        DrawBall(Black); {Удаляем цель, закрасив ее черным цветом фона}
        {Создаем эффект "затухания взрыва" цели}
        for i := 1000 downto 200 do
        begin
          {Понижающийся звук}
          if (i mod 10) = 0 then Sound(i);
          {Красным цветом рисуем уменьшающиеся "следы
           от взрыва" - линии, выходящие из центра цели}
          SetColor(Red);
          d := i div 100;
          Line(CurX,CurY,CurX-d,CurY-d);
          Line(CurX,CurY,CurX,CurY-d);
          Line(CurX,CurY,CurX+d,CurY-d);
          Line(CurX,CurY,CurX+d,CurY);
          Line(CurX,CurY,CurX+d,CurY+d);
          Line(CurX,CurY,CurX,CurY+d);
          Line(CurX,CurY,CurX-d,CurY+d);
          Line(CurX,CurY,CurX-d,CurY);
          Delay(30);
          {Закрашиваем место "взрыва" цветом фона}
          SetColor(Black);
          SetFillStyle(SolidFill,Black);
          Bar(CurX-d,CurY-d,CurX+d,CurY+d);
        end;
        Aims := Aims + '*'; {Прибавляем к строке "сбитых"
                             целей еще одну "звездочку"}
        Rating := Length(Aims)/Shots*100; {Рейтинг}
        Str(Rating:6:2,sRating); {Преобразуем рейтинг в строку}
        {Очищаем область в правом нижнем углу экрана,
         в которой отображается рейтинг "охотника"}
        SetColor(Black);
        SetFillStyle(SolidFill,Black);
        Bar(570,330,640,350);
        {Отображаем количество сбитых целей и рейтинг}
        SetColor(White);
        OutTextXY(5,330,Aims);
        OutTextXY(570,330,sRating);
        NoSound; {Отключаем звук}
        CurX := 700; {Цель поражена}  ,
      end;
    end;
  end;
  {Отображаем красным цветом сообщение об окончании игры}
  SetColor(Red);
  SetFillStyle(SolidFill,Black);
  Bar(5,330,565,350);
  OutTextXY(5,330,'Игра оаончена! Нажмите любую клавишу');
  ReadKey; {Две процедуры ReadKey объясняются тем, что}
  ReadKey; {первая считывает клавишу, нажатую во время
            работы цикла repeat функцией KeyPressed}
  CloseGraph;
end.
```

Результат работы этой программы представлен на рис. 20.3.

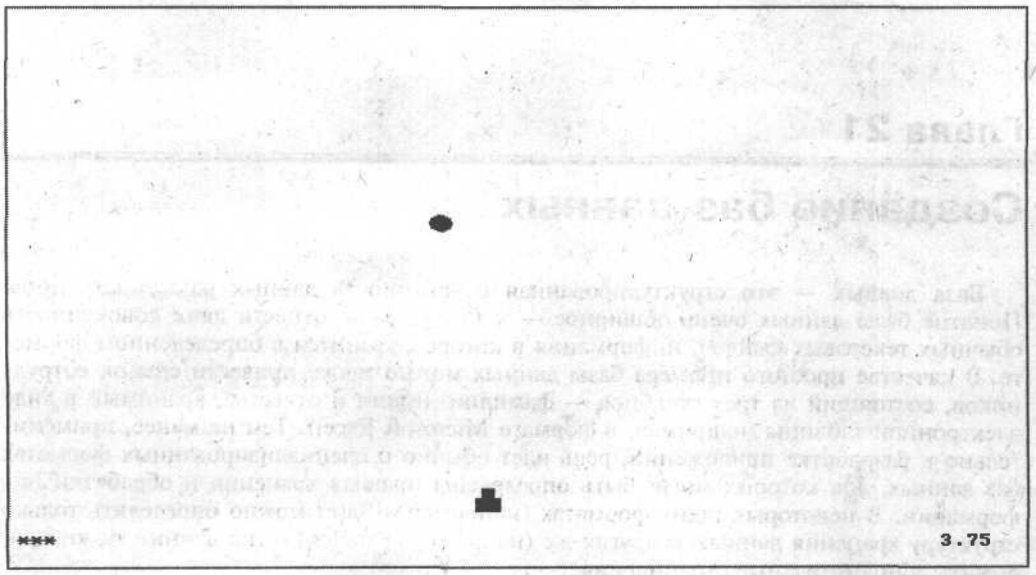


Рис. 20.3. Результат работы программы Hunter

Глава 21

Создание баз данных

База данных — это структурированная совокупность данных различных типов. Понятие базы данных очень обширное — к нему можно отнести даже совокупность обычных текстовых файлов, информация в которых хранится в определенном формате. В качестве простого примера базы данных можно также привести список сотрудников, состоящий из трех столбцов — фамилии, имени и отчества, хранимый в виде электронной таблицы (например, в формате Microsoft Excel). Тем не менее, применительно к разработке приложений, речь идет обычно о специализированных форматах баз данных, для которых могут быть определены правила хранения и обработки информации. В некоторых таких форматах (например, dBase) можно определить только структуру хранения данных, в других же (например, Paradox) — на данные можно наложить дополнительные ограничения.

Для работы с различными форматами баз данных используются специализированные программные средства, и рассмотрение внутренней структуры таких стандартных форматов выходит за тематические рамки этой книги. Данная глава посвящена созданию простейших баз данных при помощи типизированных файлов, хранящих наборы записей, а также объектов Turbo Vision типа TCollection и файловых потоков ввода-вывода.

База данных, созданная при помощи типизированных файлов

В главе 9 была разработана программа Audio, предназначенная для хранения информации о компакт-дисках в типизированном файле на диске. Подобное использование типизированных файлов вполне приемлемо для небольших объемов данных, когда количество хранимых записей не превышает нескольких десятков. Когда же речь идет о тысячах и даже о десятках тысяч записей в файле, тогда возникает необходимость оптимизации структуры типизированных файлов.

Под оптимизацией структуры подразумевается такое распределение данных по взаимосвязанным файлам, чтобы объем хранимой в каждом файле информации был максимально уменьшен. Так, например, в программе Audio хранение данных о звуковых дорожках в одном файле с данными о самих дисках избыточно, так как в каждой записи для каждой звуковой дорожки постоянно повторяется информация и о компакт-диске.

Правильным считается подход, когда данные о дисках и о дорожках хранятся в отдельных файлах, взаимосвязанных друг с другом по *ключевому полю*.

Ключевое поле — это поле, в котором хранится уникальный номер записи или какие-либо другие данные, используемые при поиске требуемых записей в файле. Так, например, в файле с данными о звуковых дорожках каждой записи отводится отдельное поле для хранения уникального номера компакт-диска. Информация о дисках хранится в одном файле, а информация о звуковых дорожках — в другом. В результа-

те, выбрав данные о диске, по его уникальному номеру можно без труда выбрать данные о связанных с ним звуковых дорожках из другого файла.

Разработаем программу AudioDB, позволяющую вести учет компакт-дисков. Данные о самих дисках будут храниться в файлах с расширением .cdd. При этом для каждого файла .cdd в том же каталоге должен создаваться одноименный подкаталог, в котором будут храниться типизированные файлы с данными о звуковых дорожках (файлах с расширением .trd). Каждой записи в файле .cdd соответствует отдельный файл .trd.

ПРИМЕЧАНИЕ

С точки зрения классической теории баз данных такой подход нетипичен, так как в подобной ситуации все данные о звуковых дорожках хранились бы в одном файле и для поиска необходимой информации использовались бы *индексы* и *фильтры*. Тем не менее, программная реализация индексации — это достаточно сложный вопрос, который *в* этой книге не рассматривается. По этой причине программа AudioDB упрощена с тем, чтобы максимально ускорить процесс выборки информации из файлов, благодаря разбиению данных о звуковых дорожках на отдельные фрагменты.

Данные в файле .cdd сортируются по имени исполнителя и названию диска, а в файлах .trd — по номеру дорожки. Программа позволяет добавлять и удалять данные в двух отдельных частях экрана. В верхней части организована работа с записями, хранящими данные о дисках, а в нижней части — с записями, хранящими данные о звуковых дорожках, выбранного в данный момент диска.

Вся информация представлена в табличном виде с указанием текущей позиции в наборе данных о дисках и общего количества дорожек, связанных с выбранным в данный момент диском. Для переключения между верхней и нижней частью экрана используется клавиша <TaЬ>, для добавления данных в текущей части экрана — клавиша <Insert>, а для удаления текущей записи — клавиша <Delete>. Для перемещения по списку дисков и звуковых дорожек используются клавиши <t> и <↓>. Программа достаточно сложная, поэтому для упрощения исходного текста все элементы интерфейса были реализованы в черно-белом варианте, без использования таких процедур из модуля Crt как TextColor и TextBackGround.

Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем AudioDB.pas и введите в него текст, представленный в листинге 21.1, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 21.1. Программа AudioDB.pas

```
program AudioDB;
uses Crt;
type
  {Структура для хранения данных об одной дорожке}
  Track = record
            DiscNum: word;     {Номер диска}
            TrackNum: word;    {Номер дорожки}
            Name: string[40]; {Название дорожки}
            Hours: 0..3;  / " {Часы}
            Minutes: 0..59;   {Минуты}
            Seconds: 0..59;   {Секунды}
          end;
  {Структура для хранения данных о дисках}
  CD = record
```

Продолжение листинга 21.1

```
        DiscNum: word;          {Номер диска}
        Author: string[20];     {Автор}
        Title: string[40];      {Название}
        Year: integer;          {Год}
        Kind: string[10]        {Жанр}
      end;
var
  c: char;
  DBName,DBDir: string;
  FileCDs: File of CD;
  FileTracks: File of Track;
  CDRec: CD;
  TrackRec: Track;
  i,j,CurY: byte;
  CurRec,CurTrack,TopCDRec,TopTrack: Word;
  PressedKey: char;
  InCDArea: boolean;

  {Процедура вывода на экран линии из знаков "-"}
  procedure DrawLine;
  begin
    for i := 1 to 79 do Write('-');
    Writeln('');
  end;

  {Процедура отображения текущей позиции в
   наборе данных о дисках в виде "текущая/всего"}
  procedure ShowPosition;
  begin
    CurY := WhereY; {Запоминаем текущее положение курсора по вертикали}
    GotoXY(1,1);    (Выводим информацию о позиции}
    Write(' ':10);  {в наборе данных в левом}
    GotoXY(1,1);    {верхнем углу экрана}
    Write(CurRec,'/',FileSize(FileCDs));
    GotoXY(1,CurY); {Возвращаем курсор в прежнее положение на экране}
  end;

  {Процедура вывода строки с данными о выбранном
   в данный момент компакт-диске}
  procedure CDDataLine;
  begin
    Read(FileCDs,CDRec);
    Writeln(CDRec.Author:20,'|',CDRec.Title:40, ' | ',
            CDRec.Year:4,'.   |',CDRec.Kind);
  end;

  {Процедура вывода строки с данными о выбранной
   в данный момент звуковой дорожке)
  procedure TrackDataLine;
  begin
    Writeln(TrackRec.TrackNum,'. ',
    , .     TrackRec.Name,'  (',
            TrackRec.Hours,'ч',
```

Продолжение листинга 21.1

```
            TrackRec.Minutes,'м',
            TrackRec.Seconds,'с',')')
  end;

  {Процедура открытия файла .trd, соответствующего
   выбранному в данный момент диску. Если Creating = True,
   то файл создается, в противном случае - открывается}
  procedure OpenTrackFile(Creating: boolean);
  var
    NumStr: string[8];
  begin
    if FileSize(FileCDs) > 0 then {Если файл с }
    begin                         {данными о дисках не пустой}
      {Перемещаемся в файле к записи,
       соответствующей выбранному в данный момент диску}
      Seek(FileCDs,CurRec-1);
      Read(FileCDs,CDRec); {Считываем запись}
      {Преобразовываем номер диска в строку}
      Str(CDRec.DiscNum,NumStr);
      {Св'язываем файловую переменную
       FileTracks с файлом "NumStr".trd в подкаталоге DBDir}
      Assign(FileTracks,DBDir+'\'+NumStr+'.trd');
      if Creating then
      begin
        Rewrite(FileTracks);
        Close(FileTracks);
      end else Reset(FileTracks);
    end;
  end;

  {Процедура, отображающая количество
   дорожек для выбранного в данный момент диска}
  procedure ShowTracksCount;
  begin
    GotoXY(69,21);
    Write('Всего: ',FileSize(FileTracks));
  end;

  {Процедура вывода списка звуковых
   дорожек для выбранного в данный момент диска}
  procedure ShowTracks;
  begin
    CurY := WhereY;
    {Заполняем область, отображающую
     данные о дорожках, пробелами (7 строк)}
    GotoXY(1,14);
    for i := 1 to 7 do Writeln(' ':79);
    if FileSize(FileCDs)> 0 then {Если файл с }
    begin                        {данными о дисках непустой}
      {Открываем файл .trd,
       соответствующий выбранному в данный момент диску}
      OpenTrackFile(False);
      ShowTracksCount; {Выводим количество дорожек}
```

Продолжение листинга 21.1

```
      i := 1;
      {Выводим перечень дорожек}
      GotoXY(1,14);
      while not Eof(FileTracks) do
      begin
        Read(FileTracks,TrackRec);
        TrackDataLine;
        inc(i);
        if i = 8 then break; {Только 7 первых дорожек}
      end;
      Close(FileTracks);
    end;
    GotoXY(1,CurY);
    CurTrack := 1; {Текущая дорожка в списке - первая}
  end;

  {Функция вставки записи в файл .cdd. Возвращает номер
   записи после вставки в отсортированную последовательность}
  function InsertCD: word;
  var
    CurCDRec: CD;
    InsertPos: word; {Позиция для вставки записи}
  begin
    InsertPos := 0;
    {Определяем порядковый номер нового диска}
    if FileSize(FileCDs) = 0 {Если файл .cdd пустой}
    then CDRec.DiscNum := 1
    else
    begin {Если в файле .cdd есть записи}
      {Считываем последнюю запись в файле}
      Seek(FileCDs,FileSize(FileCDs)-1);
      Read(FileCDs,CurCDRec);
      {Номер добавляемой записи на 1 номера последней записи}
      CDRec.DiscNum := CurCDRec.DiscNum + 1;
      {Определяем место вставки}
      Seek(FileCDs,0);
      while not Eof(FileCDs) do {Просматриваем файл с данными о дисках}
      begin
        Read(FileCDs,CurCDRec);
        {Если строка "Автор"+"Название"
         у добавляемой записи меньше, чем у текущей в файле}
        if (CurCDRec.Author + CurCDRec.Title) >
           (CDRec.Author + CDRec.Title)
        then Break; {Найдена позиция вставки,
                     поэтому прерываем цикл просмотра файла}
        Inc(InsertPos);
      end;
      {Сдвигаем записи от места вставки вниз}
      if InsertPos < FileSize(FileCDs) then
      begin
        Seek(FileCDs,FileSize(FileCDs)); {Добавляем}
        Write(FileCDs,CDRec);            {запись в конец файла}
```

Продолжение листинга 21.1

```
        {Просматриваем файл с конца в направлении позиции вставки записи}
        for i := (FileSize(FileCDs)-1)
            downto InsertPos + 1 do
        begin
          {"Сдвигаем" записи к концу файла}
       ' 'Seek(FileCDs,i-1);
          Read(FileCDs,CurCDRec);
          Write(FileCDs,CurCDRec);
        end;
      end;
    end;
    {Вставляем новую запись в требуемую позицию}
    Seek(FileCDs,InsertPos);
   Write(FileCDs,CDRec);
    InsertCD := FilePos(FileCDs);
end;

{Процедура вставки записи в файл .trd}
procedure InsertTrack;
var
  CurTrackRec: Track;
  InsertPos: word; {Позиция вставки}
begin
  OpenTrackFile(False); {Открываем файл .trd}
  InsertPos := 0;
  if FileSize(FileTracks) > 0 then {Если файл .trd не пустой }
  begin
    {Просматриваем файл с данными о дорожках}
    Seek(FileTracks,0);
    while not Eof(FileTracks) do
    begin
      Read(FileTracks,CurTrackRec);
      {Если номер добавляемой дорожки                    }
       меньше номера текущей дорожки в файле}
      if CurTrackRec.TrackNum >
         TrackRec.TrackNum then Break; {Найдена позиция вставки}
      Inc(InsertPos); {Коррекция позиции вставки}
    end;
    {Сдвигаем записи от места вставки вниз}
    Seek(FileTracks,FileSize(FileTracks));
    {Добавляем запись в конец файла}
    Write(FileTracks,TrackRec);
    {Просматриваем файл .trd с конца в направлении позиции вставки}
    for i := (FileSize(FileTracks)-1) downto InsertPos + 1 do
    begin
      Seek(FileTracks,i-1);
      {Сдвигаем записи по направлению к концу файла}
      Read(FileTracks,CurTrackRec);
      Write(FileTracks,CurTrackRec);
    end;
    {Перемещаемся к позиции вставки записи}
    Seek(FileTracks,InsertPos);
  end;
```

Продолжение листинга 21.1

```
    Write (FileTracks,TrackRec); {Вставляем запись}
    ShowTracksCount;             {Выводим количество дорожек}
    Close(FileTracks);           {Закрываем файл .trd}
  end;

  {Процедура удаления выбранного в данный момент диска}
  procedure DelCD;
  var
    CurCDRec: CD;
    CurPos: word;
  begin
    CurPos := CurRec; {Начиная с текущей позиции}
    while CurPos < FileSize(FileCDs) do {Просматриваем файл .cdd}
    begin
      {Сдвигаем записи, затирая каждую текущую последующей}
      Seek(FileCDs,CurPos);
      Read(FileCDs,CurCDRec);
      Seek(FileCDs,CurPos-1);
      Write(FileCDs,CurCDRec);
      Inc(CurPos);
    end;
    {Удаляем последнюю запись файла}
    Seek(FileCDs,FileSize(FileCDs)-1);
    Truncate(FileCDs);
  end;

  {Процедура удаления текущей дорожки}
  procedure DelTrack;
  var
    CurTrackRec: Track;
    CurPos: word;
  begin
    OpenTrackFile(False); {Открываем файл .trd}
    {Просматриваем файл, начиная с текущей позиции}
    CurPos := CurTrack;
    while CurPos < FileSize(FileTracks) do
    begin
      {Сдвигаем записи по направлению к началу}
      Seek(FileTracks,CurPos);
      Read(FileTracks,CurTrackRec);
      Seek(FileTracks,CurPos-1);
      Write(FileTracks,CurTrackRec);
      Inc(CurPos);
    end;

    {Удаляем последнюю запись в файле}
    Seek(FileTracks,FileSize(FileTracks)-1);
    Truncate(FileTracks);
    ShowTracksCount; {Выводим количество дорожек}
  end;

{Начало программы}
begin
  {Цикл, пока пользовтель не выберет вариант "0"}
```

Продолжение листинга 21.1

```
while c <> '0' do begin
  ClrScr; {Очищаем экран}
  repeat {Цикл выбора варианта продолжения}
    Writeln('Укажите дальнейшее действие:');
    Writeln('0 - Выход из программы');
    Writeln('1 - Создание новой базы данных');
    Writeln('2 - Открытие существующей баз данных');
    Readln(c);
  until c in ['0'..'2'];
  if c <> '0' then
  begin
    Write('Введите имя базы данных: ');
    Readln(DBName);
    {Если пользователь не ввел
     расширение, то добавляем его к имени файла}
    if pos('.cdd',DBName) = 0 then
      DBName := DBName + '.cdd';
    Assign(FileCDs,DBName);
    {Определяем каталог размещения файла}
    DBDir:='' ;
    for i := Length(DBName) downto 1 do
      if DBName[i] = '\' then    {Находим позицию}
      begin                      {последнего символа "\"}
        DBDir := copy(DBName,1,i);
        Break;
      end;
    {Определяем подкаталог для данных о дорожках}
    DBDir := DBDir + copy(DBName,Length(DBDir)+1,
                          pos('.',DBName)-Length(DBDir));
    if c = '1' then
    begin {Создание базы данных}
      Rewrite(FileCDs);
      Writeln('Файл ',DBName,'.cdd создан...');
      MkDir(DBDir); {Создаем подкаталог}
      Writeln('Каталог ',DBDir,' создан...');
    end;
    {Открытие базы данных}
    Reset(FileCDs);
    Writeln('Файл ',DBName,'.cdd открыт...');
    ClrScr;
    GotoXY(1,24);
    {Выводим внизу экрана строку подсказки}
    DrawLine;
    Write('Esc - Выход, Insert - Добавить ',
          'Delete - Удалить, Tab - Переход');
    GotoXY(1,1);
    {Отображаем данные о дисках}
    Writeln('          | Автор    |',
            '                Название                !
            ' Год|    Жанр   ');
    DrawLine;
    {Выводим данные о первых 7 дисках}
```

Продолжение листинга 21.1

```
Seek(FileCDs,0);
for i := 1 to 7 do
  if Eof(FileCDs) then
  begin
    for j := i to 7 do
      Writeln(' ':20,'|',' ':40,'|',' ':4,'|',' ');
    break;
  end else CDDataLine;
DrawLine;
CurRec := 1; {Выбран первый диск}
TopCDRec := 1; {Первым в списке виден первый диск}
{Отображаем область ввода нового диска}
Writeln('Новый диск:');
WritelnC ':20,'|',' ':40,'|',' ':4,'|',' ');
DrawLine;
ShowTracks; {Выводим список дорожек для первого диска}
{Отображаем область ввода новой дорожки }
GotoXY(1,21);
DrawLine;
Writeln('Новая дорожка:');
WritelnC ':3,'.',' ':40,' ( ч м с)');
GotoXY(1,3);
TopTrack := 1; {Первой в списке видна первая дорожка}
InCDArea := True; {Курсор - в области просмотра данных о дисках}
repeat {Цикл до тех пор, пока
        пользователь не нажмет клавишу Esc}
  (Перемещаемся в файле .cdd к текущему диску}
  Seek(FileCDs,CurRec-1);
  ShowPosition; (Отображаем текущую позицию}
  PressedKey := ReadKey; (Считываем клавишу}
  case PressedKey of
    #09: begin {Табуляция}
           {Переключение между областями
           просмотра данных о дисках и дорожках}
           InCDArea := not InCDArea;
           {Устанавливаем курсор в позицию,
            соответствующую активной области}
           if InCDArea
           then GotoXY(1,3 + CurRec - TopCDRec)
           else GotoXY(1,14);
         end;
    #72: begin  (Стрелка вверх}
           if InCDArea then (Если в списке дисков}
             if CurRec > 1 then (Если не первый диск}
             begin
               Dec(CurRec); {Переходим к предыдущему диску}
               Seek(FileCDs,CurRec-1);
               if WhereY > 3 {Если не в первом ряду}
               then GotoXY(1,WhereY-1)
               else (Если в первом ряду и не на первом диске}
               begin
                 {Удаляем седьмую строку в списке}
                 GotoXY(1,9);
```

Продолжение листинга 21.1

```
                DelLine;
                {Вставляем строку в
                 позиции первой строки в списке}
                GotoXY(1,3);
                InsLine;
                {Выводим данные о предыдущем диске}
                CDDataLine;
                {Уменьшаем номер записи,
                 которая отображается в списке первой}
                Dec(TopCDRec);
                {Смещаем курсор вверх}
                GotoXY(1,WhereY-1);
              end;
              {Показываем данные о дорожках}
              ShowTracks;
            end;
          {Если активен список дорожек}
          if not InCDArea then
            if CurTrack > 1 then
            begin
              Dec(CurTrack);
              if WhereY > 14 {Если не в первом ряду}
              then GotoXY(1,WhereY-1)
              else {Если в первом ряду и не на первой дорожке}
              begin
                GotoXY(1,20);
                DelLine;
                GotoXY(1,14);
                InsLine;
                OpenTrackFile(False);
                Seek(FileTracks,CurTrack-1);
                Read(FileTracks,TrackRec);
                Close(FileTracks);
                TrackDataLine;
                Dec(TopTrack);
                GotoXY(1,WhereY-1);
              end;
            end;
        end;
#80:    begin {Стрелка вниз}
          if InCDArea then
            if CurRec < FileSize(FileCDs) then
            begin
              Inc(CurRec);
              Seek(FileCDs,CurRec-1);
              {Если выбрана не последняя видимая строка, то...}
              if WhereY < 9 then
                {Просто смещаем курсор вниз}
                GotoXY(1,WhereY+1)
              else
              begin
                {Если была выбрана последняя видимая в списке
                 строка, то удаляем первую строку в списке}
```

Продолжение листинга 21.1

```
                    GotoXY(1,3);
                    DelLine;
                    {Вставляем пустую строку в
                     позиции последней строки в списке}
                    GotoXY(1,9);
                    InsLine;
                    {Выводим данные о диске}
                    CDDataLine;
                    {Увеличиваем номер первой
                     записи,видимой в списке дисков}
                    Inc(TopCDRec);
                    {Смещаем курсор вверх}
                    GotoXY(1,WhereY-1);
                  end;
                  ShowTracks; {Выводим данные о дорожках}
                end;
              if not InCDArea then {Для списка дорожек}
              begin
                OpenTrackFile(False);
                if CurTrack < FileSize(FileTracks) then
                begin
                  Inc(CurTrack);
                  if WhereY < 20 then
                    GotoXY(1,WhereY+1)
                  else
                  begin
                    GotoXY(1,14);
                    DelLine;
                    GotoXY(1,20);
                    InsLine;
                    Seek(FileTracks,CurTrack-1);
                    Read(FileTracks,TrackRec);
                    TrackDataLine;
                    Inc(TopTrack);
                    GotoXY(1,WhereY-1);
                  end;
                end;
                Close(FileTracks);
              end;
            end;
#82:        if InCDArea then  {Вставка данных}
            begin
              {Вставка данных о диске}
              CurY := WhereY;
              GotoXY(1,12); {Вводим данные в 12-й строке экрана}
              Readln(CDRec.Author);
              GotoXY(22,12);
              Readln(CDRec.Title);
              GotoXY(63,12);
              Readln(CDRec.Year);
              GotoXY(68,12);
              Readln(CDRec.Kind);
              {Вставляем запись в файл .cdd}
```

Продолжение листинга **21.1**

```
      CurRec := InsertCD;
      {Создаем файл для хранения данных о дорожках}
      OpenTrackFile(True);
      GotoXY(1,12);
      Writeln(' ':20,'|',' ':40,'|',' ':4,'|',' ':10);
      {Если вставляемая запись
       попадает в диапазон отображаемых записей}
      if CurRec in [TopCDRec..TopCDRec+6] then
      begin
        {Вставляем запись в таблицу}
        GotoXY(1,9);
        DelLine;
        GotoXY(1,3+CurRec-TopCDRec);
        InsLine;
        Seek(FileCDs,CurRec-1);
        CDDataLine;
        GotoXY(1,WhereY-1);
      end else begin
       {Иначе перемещаемся к строке,
        которая была выбрана до вставки новой записи}
        GotoXY(1,CurY);
        CurRec := TopCDRec + CurY - 3;
      end;
      ShowTracks;
    end else
    begin
      {Вставка данных о звуковой дорожке}
      CurY := WhereY;
      {Сохраняем номер текущего диска}
      TrackRec.DiscNum := CDRec.DiscNum;
      {Вводим данные о дорожке в 23-й строке экрана}
      GotoXY(1,23);
      Readln(TrackRec.TrackNum);
      GotoXY(5,23);
      Readln(TrackRec.Name);
      GotoXY(47,23);
      Readln(TrackRec.Hours);
      GotoXY(50,23);
      Readln(TrackRec.Minutes);
      GotoXY(53,23);
      Readln(TrackRec.Seconds);
      {Вставляем дорожку в файл .trd}
      InsertTrack;
      {Очищаем поля ввода новой дорожки}
      GotoXY(1,23);
      WritelnC ':3,'.','':40,' ( ч м с)');
      {Если вставляемая запись
       попадает в диапазон отображаемых записей}
      if TrackRec.TrackNum in [TopTrack..TopTrack+6]
      then begin
        {Вставляем запись в таблицу}
        GotoXY(1,20);
        DelLine;
```

Продолжение листинга 21.1

```
                GotoXY(1,14+TrackRec.TrackNum-TopTrack);
                InsLine;
                TrackDataLine;
                GotoXY(1,WhereY-1);
                CurTrack := TrackRec.TrackNum;
              end else GotoXY(1,CurY);
            end;
#83:        {Удаление записи}
            if InCDArea then  {Удаление данных о диске}
            begin
              CurY := WhereY;
              DelCD; {Удаляем запись из файла}
              DelLine; {Удаляем текущую строку экрана}
              {Вставляем в позиций последней
               строки списка дисков пустую строку}
              GotoXY(1,9);
              InsLine;
              {Если в файле .cdd меньше 7 записей
               или была удалена последняя запись} '
              if (FileSize(FileCDs) < 7) or
                 (CurRec > FileSize(FileCDs))
              then begin
                {Заполняем последнюю
                 строку списка "пустыми" значениями}
                Writeln(''':20,' | ',' ' :40,'|',
                        ' ':4,'|',' ':10);
                {Если была удалена последняя
                 запись, то текущей становится новая
                 последняя запись}
                if CurRec > FileSize(FileCDs)
                then CurRec := FileSize(FileCDs);
              end else
              begin
                Seek(FileCDs,CurRec-1);
                CDDataLine;
              end;
              {Если позиция курсора до удаления теперь находится
               ниже последней строки в списке дисков}
              if (3 + CurRec - TopCDRec) < CurY
              then Dec(CurY); {Перемещаем курсор}
              GotoXY(1,CurY);
              ShowTracks;
            end else {Удаление данных о дорожке}
            begin
              CurY := WhereY;
              DelTrack;
              GotoXY(1,CurY);
              DelLine;
              GotoXY(1,20);
              InsLine;
              if (FileSize(FileTracks) < 7) or
                 (CurTrack > FileSize(FileTracks))
              then begin
```

Окончание листинга 21.1

```
                    Writeln(' ':79);
                    if CurTrack > FileSize(FileTracks)
                     then CurTrack := FileSize(FileTracks);
                  end else
                  begin
                    Seek(FileTracks,CurTrack-1);
                    Read(FileTracks,TrackRec)';
                    TrackDataLine;
                  end;
                  Close(FileTracks);
                  if (14 + CurTrack - TopTrack) < CurY
                  then Dec(CurY);
                  GotoXY(1,CurY);
                end;
            end;
          until PressedKey = #27; {Нажата клавиша Esc}
          Close(FileCDs); {Закрываем файл .cdd}
        end;
      end;
end.
```

Результаты работы этой программы представлены на рис. 21.1.

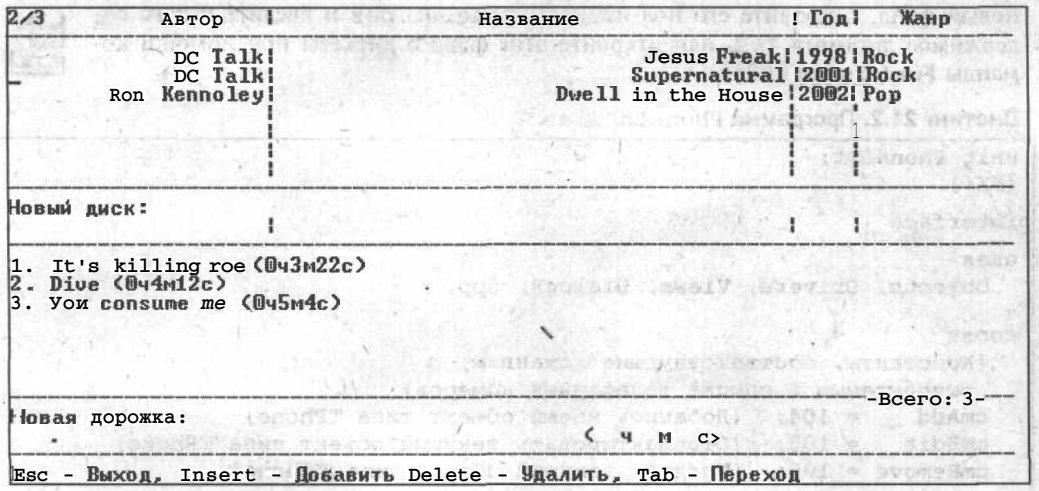


Рис. 21.1. Результат работы программы AudioDB

Создание баз данных средствами Turbo Vision

В этом разделе рассмотрено создание базы данных телефонных номеров при помощи объектно-ориентированных средств Turbo Vision. При этом список номеров хранится в программе в объекте типа, производного от TCollection. Тип TCollection используется для хранения набора объектов определенного типа. В данном случае в программе для хранения данных об одном номере будет использоваться объект пользовательского типа TPhone, состоящий из двух полей Name и

Number, хранящих имя и номер телефона, и двух методов Load и Save, используемых для считывания данных из файлового потока и записи данных в файловый поток.

Поток — это объект типа, производного от TStream, обеспечивающий абстрактный ввод-вывод через буфер памяти. В потоках можно сохранять объекты любого типа с последующей их записью в файл, а также выполнять обратную операцию считывания данных из файла в объект некоторого типа.

Для того чтобы некоторый тип можно было использовать при работе с потоками ввода-вывода, в модуле, в котором этот тип объявлен, должен быть реализован вызов процедуры RegisterType. В создаваемой программе, кроме типа TPhone, для хранения списка имен и телефонных номеров используется объект типа TphoneColl. Тип TphoneColl производный от типа TCollection. Следовательно, при работе с потоками, необходимо для использования этих типов в том модуле, где они объявлены, дважды вызвать процедуру RegisterType:

```
RegisterType(TPhone);
RegisterType(TPhoneColl);
```

Перед вызовом процедуры RegisterType для каждого из типов должна быть определена запись типа TStreamRec, в которой указываются следующие данные: идентификатор типа, смещение таблицы виртуальных методов для объектов данного типа, адрес метода считывания данных из потока и адрес метода записи данных в поток.

Создадим программу, в которой типы TPhone и TPhoneColl будут реализованы в отдельном модуле PhoneLst.pas. Создайте в интегрированной среде Turbo Pascal новый файл, сохраните его под именем PhoneLst.pas и введите в него содержимое листинга 21.2, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 21.2. Программа PhoneLst.pas

```
unit PhoneLst;
{$X+}

interface

uses
  Objects, Drivers, Views, Dialogs, App;

const
  {Константы, соответствующие командам,
   выполняемым в списке телефонных номеров}
  cmAdd    = 104;  {Добавить новый объект типа TPhone)
  cmEdit   = 105;  {Отредактировать текущий объект типа TPhone}
  cmRemove = 106;  {Удалить текущий объект типа TPhone}

type
  NameStr = String[40];   {Строка для хранения имени}
  NumberStr = String[12]; {Строка для хранения номера}

  {Объявление типа TPhone.
   Объекты этого типа хранят имя и номер
   телефона, а также содержат методы Load и Store, используемые
   для считывания данных из потока и для записи в поток}
  PPhone = ^TPhone;
  TPhone = object (TObject)
    Name: NameStr;
    Number: NumberStr;
```

Продолжение листинга 21.2

```
    constructor Init (AName: NameStr; ANumber: NumberStr);
    constructor Load (var S: TStream);
    procedure Store (var S: TStream);
  end;

  {Объявление типа TPhoneListBox.
   Этот тип - производный от типа Turbo Vision TListBox,
   с добавлением специального метода Ge,tText, предназначенного
   для отображения списка объектов типа TPhone}
  PPhoneListBox = ^TPhoneListBox;
  TPhoneListBox = object(TListBox)
    function GetText (Item: Integer; MaxLen: Integer): String;
    virtual;
  end;

  {Объявление типа TPhoneColl.
   Это типа - производный от типа Turbo Vision TCollection,
   с добавлением метода Show, открывающего диалоговое окно,
   в котором выполнятся просмотр и редактирование списка
   телефонных номеров}
  PPhoneColl = ^TPhoneColl;
  TPhoneColl = object(TCollection)
    function Show: Word;
  end;

  {Объявление типа TViewDialog.
   Этот тип - производный от типа Turbo Vision TDialog.
   В нем дополнительно реализован просмотр информации,
   содержащейся в объекта типа TPhone, а также ее
   редактирование.
   Поле PhoneColl - это указатель на объект типа TPhoneColl.
   Поле L - это указатель на список, отображающий набор
   объектов типа TPhone}
  PViewDialog = ^TViewDialog;
  TViewDialog = object (TDialog)
    PhoneColl: PPhoneColl;
    L: PPhoneListBox;
    constructor Init (APhoneColl: PPhoneColl);
    procedure HandleEvent (var Event: TEvent); virtual;
    end;

{Объявление процедуры RegisterPhone.
 Эта процедура отвечает за регистрацию новых объектных типов.
 Данные зарегистрированных типов могут быть записаны
 в поток или считаны из потока}
procedure RegisterPhone;

implementation

{Реализация методов типа TPhone}

{Конструктор Init - инициализация объекта типа TPhone.
 AName - имя, ANumber - номер телефона}
constructor TPhone.Init (AName: NameStr; ANumber: NumberStr);
```

Продолжение листинга 21.2

```
begin
  TObject.Init; {Вызов унаследованного конструктора}
  {Инициализация полей объекта типа TPhone}
  Name := AName;
  Number := ANumber;
end;

{Конструктор Load - считывание данных объекта типа TPhone
 из. потока (объекта типа Turbo Vision TSrteam) }
constructor TPhone.Load (var S: TStream);
begin
  S.Read(Name, SizeOf(NameStr));      {Считываем имя}
  S.Read(Number, SizeOf(NumberStr)); {Считываем номер}
end;

{Метод Store - запись данных объекта типа TPhone в
 поток (объект типа Turbo Vision TSream)}
procedure TPhone.Store (var S: TStream);
begin
  S.Write(Name, SizeOf(NameStr));      {Запись имени}
  S.Write(Number, SizeOf(NumberStr)); {Запись номера}
end;

{Реализация методов типа TPhoneListBox}

{Метод GetText возвращает строку, состоящую из имени и
 номера телефона, хранимых в указанном объекте типа TPhone.
 Параметр Item - номер элемента в списке.
 Параметр MaxLen оставлен для соответствия с наследуемым методом}
function TPhoneListBox.GetText (Item: Integer; MaxLen: Integer):
String;
var
  S: String; {Строка для хранения имени и телефонного номера}
begin
  S := '                                                    ';
  {List^.At(Item) - указатель на элемент списка с номером Item.
   PPhone(List^.At(Item)) - приведение указателя на
   элемент списка к типу PPhone, чтобы можно было
   обращаться к полям TPhone.Name и TPhone.Number}
  {Записываем в строку S, начиная с первого символа,
   подстроку из поля Name, начиная с первого символа,
   длиной Length(Name)}
  Move(PPhone(List^.At(Item))^.Name[1], S[1],
       Length(PPhone(List^.At(Item))^.Name));
  {Записываем в строку S, начиная с 42-го символа,
   подстроку из поля Number, начиная с первого символа,
   длиной Length(Number)}
  Move(PPhone(List^.At(Item))^.Number[1], S[42],
       Length(PPhone(List^.At(Item))^.Number));
  GetText := S; {Возвращаем результат}
end;

{Реализация методов типа TPhoneColl}
```

Продолжение листинга 21.2

```
{Метод TPhoneColl.Show вызывает метод ExecView типа
 Turbo Vision TViewDialog, и возвращает результат,
 возвращенный этим методом}
function TPhoneColl.Show: Word;
begin
  Show := DeskTop^.ExecView(New(PViewDialog,Init(@Self)));
end;

{Метод ModifyRecord создает диалоговое окно, используемое для
 добавления новой или редактирования существующей записи типа TPhone.
 Результат, возвращаемый этим методом, равнозначен результату,
 возвращаемому при вызове метода диалогового окна ExecView.
 Если пользователь выбрал в диалоговом окне
 кнопку "Отмена", то объект типа TPhone остается неизменным}
function ModifyRecord (Phone: PPhone; Title: TTitleStr): Word;
var
  R: TRect;    {Объект для хранения прямоугольных границ}
  D: PDialog; {Указатель на объект диалогового окна}
  N,P: PInputLine; {Указатели на строки ввода
                    имени и телефонного номера}
  Result: Word;     {Возвращаемый результат}
begin
  R.Assign(27,11,73,21); {Инициализируем границы окна}
  {Создаем объект диалогового окна с границами,
   определенным в R}
  D := New(PDialog, Init(R, Title + ' запись'));
  R.Assign(2,2,44,3);  {Инициализируем границы поля ввода имени}
  {Создаем объект поля ввода имени с границами, определенным в R}
  N := New(PInputLine, Init(R,40));
  {Записываем в поле ввода имени содержимое поля TPhone.Name}
  N^.SetData(Phone^.Name);
  D^.Insert(N);  {Размещаем поле ввода имени в диалоговом окне}
  R.Assign(2,1,44,2); {Инициализируем границы надписи "Имя"}
  {Размещаем в диалогом окне объект-надпись "Имя"}
  D^.Insert(New(PLabel, Init(R,'Имя',N)));
  R.Assign(2,5,16,6); {Инициализируем границы поля ввода номера}
  {Создаем объект поля ввода номера с границами,определенным в R}
  P := New(PInputLine, Init(R,12));
  {Записываем в поле ввода номера содержимое поля TPhone.Number}
  P^.SetData(Phone^.Number);
  D^.Insert (P); {Размещаем поле ввода в диалоговом окне}
  R.Assign(2,4,44,5) ; {Инициализируем границы надписи "Телефон"}
  {Размещаем в диалогом окне объект-надпись "Телефон"}
  D^. Insert(New(PLabel, Init(R,'Телефон',N)));
  R.Assign(5,7,15,9); {Инициализируем границы кнопки "OK"}
  {Размещаем в диалогом окне объект-кнопку "OK".
   Параметр bfDefault означает, что эта кнопка выбрана по
   умолчанию, то есть нажимается при нажатии клавиши <Enter>}
  D^.Insert(New(PButton, Init(R,'OK',cmOK,bfDefault)));
  R.Assign (20,7,30,9); {Инициализируем границы кнопки "Отмена"}
  D^.Insert(New(PButton, Init(R,'Отмена',cmCancel,bfNormal)));
  D^.SelectNext(False); {Выбирается кнопка "OK"}
```

Продолжение листинга 21.2

```
  {Раскрываем диалоговое окно с размещенными на нем элементами}
  Result := DeskTop^.ExecView(D);
  if Result o cmCancel then {Если выбрана кнопка "OK"}
  begin
    N^.GetData (Phone^.Name);    {В поле ввода имени записываем
                                  содержимое поля TPhone.Name}
    P^.GetData(Phone^.Number); {В поле ввода номера записываем
                                  содержимое поля TPhone.Number}
  end;
  Dispose(D,Done); {Уничтожаем объект диалогового окна}
  ModifyRecord := Result;
end;

{Реализация методов типа TViewDialog}

{Конструктор TViewDialog.Init - инициализация
диалогового окна}
constructor TViewDialog.Init(APhoneColl: PPhoneColl);
var
  R: TRect;  {Объект для хранения прямоугольных границ}
  SB: PScrollBar; {Указатель на объект "полоса прокрутки"}
begin
  R.Assign(10,5,70,16);  {Инициализируем границы окна}
  {Инициализируем диалоговое окно с указанным заголовком
   и границами, хранимыми в объекте R}
  TDialog.Init(R, 'Список телефонов');
  PhoneColl := APhoneColl; {Указатель на коллекцию номеров}
  R.Assign(57, 2, 58,7); {Инициализируем границы полосы прокрутки}
  SB := New(PScrollBar, Init(R)); {Создаем полосу прокрутки}
  Insert(SB); {Размещаем полосу прокрутки в диалоговом окне}
  R.Assign(2,2,57,7); {Инициализируем границы списка номеров}
  {Создаем объект-список типа TPhoneListBox с полосой прокрутки}
  L := New(PPhoneListBox, Init(R,1,SB));
  L^.NewList(PhoneColl); {Сопоставляем с ним список телефонов}
  Insert(L); {Размещаем список в диалоговом окне}
  R.Assign(2,1,57,2); {Инициализируем границы строки заголовка}
  {Размещаем в диалоговом окне объект статического текста}
  Insert(New(PStaticText, Init(R,
  ' Имя                                    Телефон')));
  R.Assign(1,8,13,10); {Инициализируем границы кнопки "Добавить"}
  {Размещаем в диалоговом окне кнопку "Добавить"}
  Insert(New(PButton, Init(R,'Добавить',cmAdd,bfNormal)));
  R.Assign(13, 8,25,10); {Инициализируем границы кнопки "Изменить"}
  {Размещаем в диалоговом окне кнопку "Изменить"}
  Insert(New(PButton, Init(R,'Изменить',cmEdit,bfNormal)));
  R.Assign(25,8,36,10); {Инициализируем границы кнопки "Удалить"}
  {Размещаем в диалоговом окне кнопку "Удалить"}
  Insert(New(PButton, Init(R,'Удалить',cmRemove,bfNormal)));
  R.Assign(36,8,49,10); {Инициализируем границы кнопки "Сохранить"}
  {Размещаем в диалоговом окне кнопку "Сохранить"}
  Insert(New(PButton, Init(R,'Сохранить',cmOK,bfDefault)));
  R.Assign(49,8,59,10); {Инициализируем границы кнопки "Отмена"}
  {Размещаем в диалоговом окне кнопку "Отмена"}
```

Продолжение листинга 21.2

```
  Insert(New(PButton, Init(R,'Отмена',cmCancel,bfNormal)));
  SelectNext(False); {Выбираем кнопку "Сохранить"}
end;

{Метод TViewDialog.HandleEvent обрабатывает команды cmAdd, cmEdit и
 cmRemove, используемые типом TViewDialog. Кроме того, он в случае
 необходимости обновляет содержимое списка телефонов, а также делает
 недоступными команды cmEdit и cmRemove, если набор номеров типа
 TPhoneColl пуст}
procedure TViewDialog.HandleEvent (var Event: TEvent);
var
  P: PPhone;
begin
  TDialog.HandleEvent(Event); {Вызов унаследованного метода}
  if Event.What = evCommand then {Если событие - команда}
  begin
    case Event.Command of
      cmAdd: begin {Команда "Добавить"}
        {Создается новый объект типа TPhone с
         пустыми значениями имени и телефонного номера}
              P := New (PPhone, Init ('',''));
              {Вызывается диалоговое окно "Добавить запись"
               с двумя полями ввода "Имя" и "Номер"}
              if ModifyRecord(P,'Добавить') o cmCancel then
                {Если была нажата кнопка "OK", то добавляем в набор
                 номеров новый объект типа TPhone, содержащий указанные
                 в диалоговом окне значения полей Name и Number}
                PhoneColl^.Insert(P)
              else Dispose(P,Done); {Если была выбрана кнопка "Отмена",
                                     то уничтожаем объект типа TPhone}
            end;
      cmEdit: {Команда "Изменить"}
              {Вызывается диалоговое окно "Изменить запись" с
               двумя полями ввода "Имя" и "Номер", в которых
               отображаются текущие значения полей объекта типа
               TPhone,соответствующего текущему элементу списка.
               Текущий выбранный элемент списка - L^.Focused.
               Указатель, на объект списка - PhoneColl^.At(L^.Focused)}
            ModifyRecord(PPhone(PhoneColl^.At(L^.Focused)), 'Изменить');
      cmRemove: {Команда "Удалить"}
              {Удаляется элемент списка PhoneColl в текущей
               позиции, возвращаемой функцией L^.Focused}
             PhoneColl^.AtDelete(L^.Focused);
    end;
    {Обновление диапазона отображаемых
     элементов списка и состояния полосы прокрутки}
    L^.SetRange(L^.List^.Count);
    L^.DrawView; {Перерисовка списка номеров}
  end;
  if PhoneColl^.Count >= 1 {Если количество
                            элементов в списке PhoneColl >= 1}
  then EnableCommands([cmRemove,cmEdit]) {Делаем кнопки
                      "Удалить" и "Изменить" доступными}
```

Окончание листинга 21.2

```
  else DisableCommands([cmRemove,cmEdit]); {Делаем кнопки "Удалить"
                                            и "Изменить" недоступными}
end;

{Записи, используемые для регистрации потока}
const
  srPhone     = 10001;   {Поток для номеров}
  srPhoneColl = 10002;   {Поток для списка номеров}
  RPhone: TStreamRec = (
    ObjType: srPhone;   {Тип объекта}
    VMTLink: Ofs(TypeOf(TPhone)^); {Смещение таблицы виртуальных
                                    методов для объектов типа TPhone}
    Load: @TPhone.Load;   {Адрес метода считывания из
                           потока данных типа TPhone}
    Store: @TPhone.Store  {Адрес метода записи в поток
                           данных типа TPhone}
  );

  RPhoneColl: TStreamRec = (
    ObjType: srPhoneColl;    {Тип объекта}
    VMTLink: Ofs(TypeOf(TPhoneColl)^); {Смещение таблицы виртуальных
                                        методов для объектов типа TPhoneColl}
    Load: @TPhoneColl.Load;   {Адрес метода считывания из
                               потока данных типа TPhoneColl}
    Store: @TPhoneColl.Store  {Адрес метода записи в
                               поток данных типа TPhoneColl}
  );

  {Процедура регистрации новых объектных типов, которые
   будут записываться в поток или считываться из потока}
procedure RegisterPhone;
begin
  RegisterType(RPhone);      {Регистрация типа TPhone}
  RegisterType(RPhoneColl);  {Регистрация типа TPhoneColl}
end;

end.
```

Теперь создайте в интегрированной среде программирования Turbo Pascal основной файл программы с именем Phone.pas и введите в него текст из листинга 21.3, или откройте этот файл с дискеты при помощи команды **File | Open** (<F3>).

Листинг 21.3. Программа Phone.pas

```
program Phone;
{$X+}
{$S-}
uses
  Dos, Drivers, Objects, Views, Menus, Dialogs,
  StdDlg, MsgBox, App, PhoneLst;
const
  {Константы, соответствующие командам меню "Файл"}
```

Продолжение листинга 21.3

```
  cmNew    = 101;   {Команда "Создать"}
  cmOpen   = 102;   {Команда "Открыть"}

{Тип TPhoneApp - потомок типа TApplication.
 Поле PhoneList типа TPhoneColl - набор
 телефонных номеров (этот тип объявлен в модуле PhoneLst).
 Поле CurrentFile содержит путь к выбранному в
 данный момент файлу базы данных телефонных номеров (.phn)}
type
  PPhoneApp = ^TPhoneApp;
  TPhoneApp = object (TApplication)
    PhoneList: PPhoneColl;    {Набор номеров}
    CurrentFile: PathStr;     {Путь к файлу базы данных}
    constructor Init;         {Инициализация}
    procedure NewPhoneList;   {Создание нового списка}
    procedure OpenPhoneList;  {Открытие существующего списка}
    procedure SavePhoneList;  {Сохранение списка в файле .phn}
    procedure HandleEvent (var Event: TEvent); virtual;
    procedure InitMenuBar; virtual;
    procedure InitStatusLine; virtual;
  end;

{Функция FileExists возвращает значение True, если файл,
 указанный в качестве параметра, существует. В противном ,
 случае возвращается значение False}
function FileExists(FileName: PathStr): Boolean;
var
  F: File;
begin
  Assign(F,FileName);
  {$I-}
  Reset(F);
  {$I+}
  if IOResult <> 0 then FileExists := False
  else begin
    FileExists := True;
    Close(F);
  end;
end;

{Реализация методов типа TPhoneApp}

{Конструктор Init - инициализация}
constructor TPhoneApp.Init;
begin
  TApplication.Init; {Вызов унаследованного конструктора}
  {Вызов процедур регистрации всех типов, используемых в программе.
   Это необходимо для того, чтобы эти типы можно было использовать
   при считывании данных из потока ввода-вывода и записи данных в
   поток ввода-вывода. Каждая из этих процедур вызывает процедуру
   RegisterType для соответствующих типов объектов. Например, процедура
   RegisterDialogs регистрирует такие типы, определенные в модуле
   Dialogs, как TDialog, TInputLine, TButton, TCluster, TRadioButtons,
   TCheckBoxes, TListBox, TStaticText, TParamText, TLabel и THistory}
```

Продолжение листинга 21.3

```
  RegisterObjects; {Регистрация типов, определенных в модуле Objects}
  RegisterViews;   {Регистрация типов, определенных в модуле Views}
  RegisterMenus;   {Регистрация типов, определенных в модуле Menus}
  RegisterDialogs; {Регистрация типов, определенных в модуле Dialogs}
  RegisterApp;     {Регистрация типов, определенных в модуле App}
  RegisterPhone;   {Регистрация типов, определенных в модуле PhoneLst}
  CurrentFile := ''; {В данный момент Не
                      выбран ни один файл базы данных .phn}
end;

{Метод NewPhoneList - создание нового пустого объекта типа TPhoneColl
 и вызов диалогового окна при помощи метода PhoneList^.Show. В этом
 диалоговом окне пользователь может записывать данные в объект типа
 TPhoneColl. Если результат, возвращаемый методом Show при закрытии
 диалогового окна, не равен cmCancel (выбор кнопки "Отмена"), то объект
 типа TPhoneColl (набор номеров) сохраняется на диске в файле .phn}
procedure TPhoneApp.NewPhoneList;
begin
  {Создаем объект типа TPhoneColl.
  Начальный размер набора - 10 элементов.
  При добавлении новых элементов набор расширяется по 10 элементов}
  PhoneList := New(PPhoneColl,Init(10,10));
  CurrentFile := '';
  {Отображаем диалоговое окно с пустым списком.
   Метод PhoneList^.Show реализован в модуле PhoneLst. Если в диалоговом
   окне была нажата кнопка "OK", то вызываем процедуру SavePhoneList}
  if PhoneList^.Show <> cmCancel then SavePhoneList;
end;

{Метод OpenPhoneList отличается от метода NewPhoneList тем, что объект
 типа TPhoneColl (набор номеров) загружается из внешнего файла .phn}
procedure TPhoneApp.OpenPhoneList;
var
  D: PFileDialog; {Указатель на объект
                   стандартного диалогового окна выбора файла}
  S: TBufStream;  {Тип TBufStream предназначен для
                   работы с дисковыми файлами через поток}
begin
  {Создаем объект стандартного диалогового окна выбора файла}
  D := New(PFileDialog,
           Init('*.PHN','Открыть список телефонов',
                'Имя',fdOKButton + fdHelpButton,100));
  {Открываем диалоговое окно}
  if Desktop^.ExecView(D) <> cmCancel then
  begin {Если нажата кнопка "OK"}
    D^.GetFileName(CurrentFile); {Извлекаем имя выбранного файла}
    if FileExists (CurrentFile) then
    begin {Если такой файл существует}
      {Инициализируем поток для чтения данных из файла
       CurrentFile. При этом в памяти создается буфер в 512 байт}
      S.Init(CurrentFile,,stOpenRead,512);
      {Извлекаем данные из выбранного файла и
       сохраняем их в объекте типа TPhoneColl}
```

Продолжение листинга 21.3

```
      PhoneList := PPhoneColl(S.Get);
      S.Done; {Уничтожаем объект S}
      {Открываем диалоговое окно просмотра списка номеров
       (реализовано в модуле PhoneLst). Если в нем не будет
       нажата кнопка "Отмена", то сохраняем список в файле}
      if PhoneList^.Show <> cmCancel then SavePhoneList;
    end
    else begin {Если файл не найден}
      {Открывается диалоговое окно сообщения
       (реализовано в модуле MsgBox) с заголовком
       "Error" и кнопкой "OK"}
      MessageBox('Файл ' + CurrentFile + ' не найден.',
                 nil,mfError + mfOkButton);
      CurrentFile := '';
    end;
  end;
  Dispose (D,Done); {Уничтожение объекта диалогового окна}
end;

{Процедура SavePhoneList - сохранение активного в данный момент
 объекта типа TPhoneColl (набор номеров). Если поле CurrentFile
 пустое, то процедура SavePhoneList открывает стандартное диалоговое
 окно сохранения файла, в противном случае набор PhoneList
 сохраняется в файле с именем, указанном в CurrentFile}
procedure TPhoneApp.SavePhoneList;
var
  D: PFileDialog; {Указатель на объект
                   стандартного диалогового окна выбора файла}
  S: TBufStream; {Тип TBufStream предназначен
                  для работы с дисковыми файлами через поток}
begin
  if CurrentFile = '' then {Если имя файла не указано}
  begin
    {Создаем объект стандартного диалогового окна сохранения файла}
    D := New(PFileDialog,
             Init('*.PHN','Сохранить список номеров',
                  'Имя',fdOKButton + fdHelpButton,100));
    {Если нажата кнопка "OK"}
    if Desktop^.ExecView(D) <> cmCancel then
      {Имя, указанное в диалоговом окне,
       записывается в поле CurrentFile}
      D ^.GetFileName(CurrentFile);
    Dispose(D,Done); {Уничтожаем объект диалогового окна}
  end;
  if FileExists(CurrentFile)
  then {Если файл существует}
    {Инициализируем поток для записи данных в файл
     CurrentFile. При этом в памяти создается буфер в 512 байт}
    S.Init(CurrentFile,stOpenWrite,512)
  else
    {Инициализируем поток для создания файла CurrentFile и записи в
    него данных. При этом в памяти создается буфер размером 512 байт}
    S.Init(CurrentFile,stCreate,512);
```

Продолжение листинга 21.3

```
  S.Put(PhoneList); {Записываем в поток вывода список
       номеров, что означает запись в файл CurrentFile}
  S.Done; {Уничтожаем объект S}
  Dispose(PhoneList,Done); {Уничтожаем объект списка номеров}
end;

{Процедура обработки событий}
procedure TPhoneApp.HandleEvent (var Event: TEvent);
begin
  TApplication.HandleEvent (Event) ; {Вызов унаследованного метода}
  if Event.What = evCommand then   {Если событие - команда}
  begin
    case Event.Command of
      cmNew: NewPhoneList;    {Команда "Создать"}
      cmOpen: OpenPhoneList; {Команда "Открыть"}
    else Exit; {В противном случае - выходим из процедуры}
    end;
    ClearEvent(Event); {Очистка события}
  end;
end;

{Инициализация строки меню}
procedure TPhoneApp.InitMenuBar;
var
  R: TRect;
begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R,NewMenu(
    NewSubMenu ('Список',hcNoContext,NewMenu(
      NewItem ('Создать','F3',kbF3,cmNew,hcNoContext,
      NewItem ('Открыть','F5',kbF5,cmOpen,hcNoContext,
      NewLine (NewItem('Выход','Alt-X',kbAltX,cmQuit,
                  hcNoContext,nil))))),nil))));
end;

{Инициализация строки состояния}
procedure TPhoneApp.InitStatusLine;
var
  R: TRect;
begin
  GetExtent (R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New (PStatusLine,Init (R,
    NewStatusDef (0,$FFFF,
    NewStatusKey ('~F3~ Создать',kbF3,cmNew,
    NewStatusKey ('~F5~ Открыть',kbF5,cmOpen,
    NewStatusKey ('~Alt-X~ Выход',kbAltX,cmQuit,
    NewStatusKey ('',kbF10,cmMenu,nil)))),nil)));
end;

var
  PhoneApp: TPhoneApp;
```

Окончание листинга 21.3

```
begin
  PhoneApp.Init;   {Инициализация приложения}
  PhoneApp.Run;    {Запуск приложения}
  PhoneApp.Done;   {Выгрузка приложения}
end.
```

Для компиляции этой программы, а также модуля PhoneLst в системе Turbo Pascal должны быть установлены средства Turbo Vision.

» Требования по установке Turbo Pascal описаны в приложении А.

Откомпилируйте сначала модуль PhoneLst, а затем файл `Phone.pas` и запустите программу на выполнение. Окно программы при открытом меню будет выглядеть, как показано на рис. 21.2.

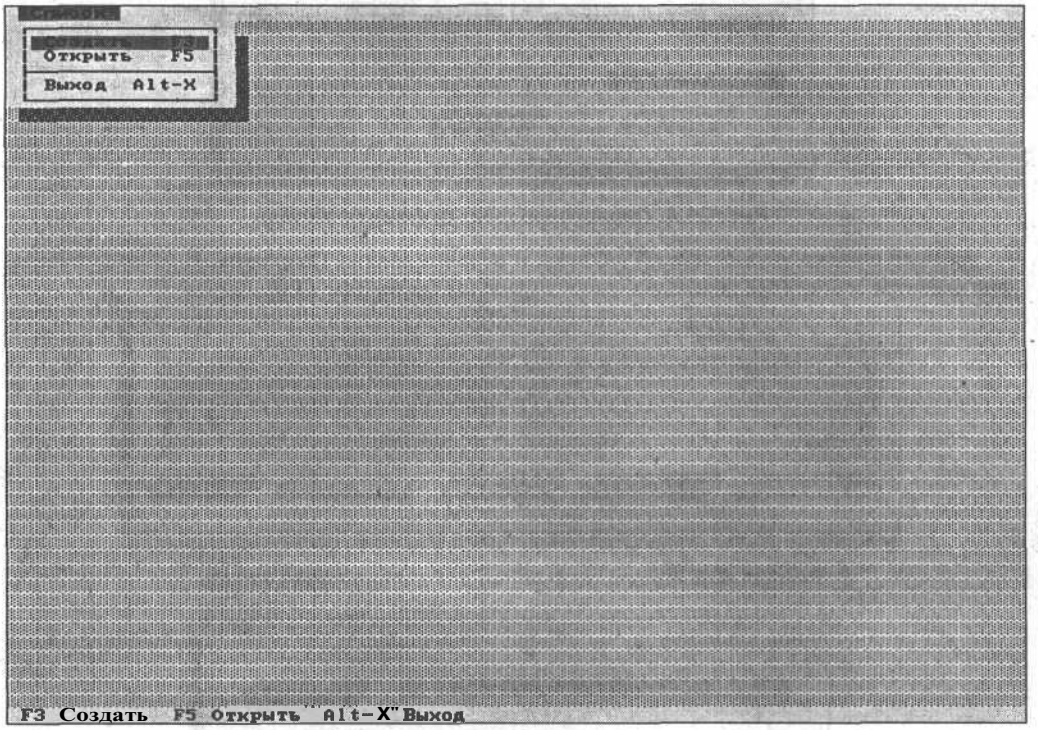


Рис. 21.2. Программа Phone

Выполните команду **Создать.** В результате на экране появится диалоговое окно **Список телефонов,** представленное на рис. 21.3.

Нажмите в окне **Список телефонов** кнопку **Добавить.** В результате на экране появится диалоговое окно **Добавить запись** (рис. 21.4).

Введите данные в полях **Имя** и **Телефон** и нажмите кнопку OK. В результате указанное имя и номер телефона будут добавлены в список, как показано на рис. 21.5.

Как видно на рис. 2.15 кнопки **Изменить и Удалить** стали доступными. Если нажать кнопку **Изменить,** то будет открыто диалоговое окно **Изменить запись,** аналогичное диалоговому окну **Добавить запись** (см. рис. 2.14) в котором можно внести

изменения в имя или номер текущей записи. Если нажать кнопку **Удалить,** то текущая запись будет удалена.

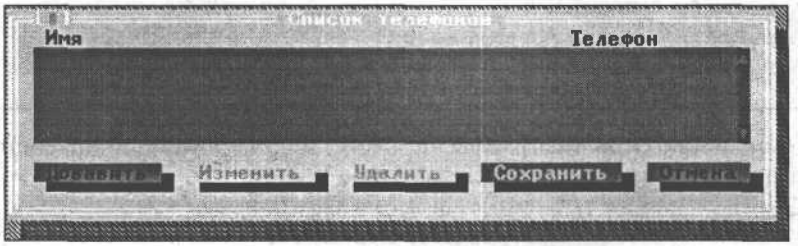


Рис. 21.3. Диалоговое окно **Список телефонов**

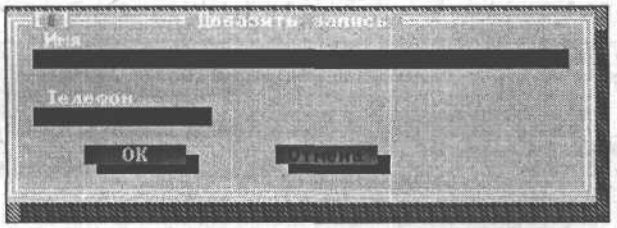


Рис. 21.4. Диалоговое окно **Добавить запись**

Добавьте сколько угодно записей в список.и нажмите кнопку **Сохранить.** В результате на экране появится стандартное диалоговое окно сохранения файла (рис. 21.6).

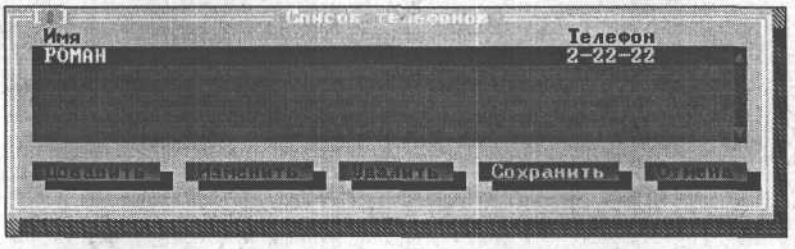


Рис. 21.5. В список телефонов добавлена новая запись

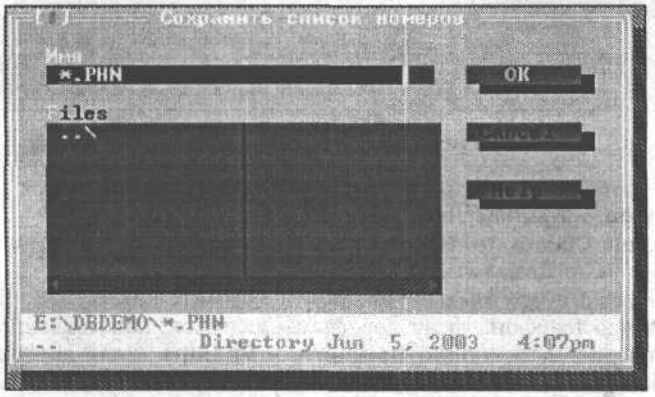


Рис. 21.6. Диалоговое окно **Сохранить список номеров**

Укажите в этом окне каталог расположения и имя файла .phn, в котором будет сохранен текущий список номеров, и нажмите кнопку OK.

Для того чтобы открыть файл базы данных телефонных номеров в программе Phone необходимо выполнить команду **Список | Открыть** (клавиша <F5>) и выбрать в стандартном диалоговом окне требуемый файл . phn.

Приложение А

Установка Turbo Pascal

Установка системы Turbo Pascal может выполняться двумя способами.

- Из дистрибутива, когда весь процесс установки возлагается на специальную программу.
- Вручную, обычным копированием с одного компьютера на другой.

Установка из дистрибутива

Специализированная программа (она может называться setup.exe или install.exe) значительно упрощает процесс установки, так как в этом случае отпадает необходимость выяснять, какие файлы требуется переписывать на компьютер, а какие — нет. Программа установки позволяет указать каталог, в который должны быть скопирована система. Затем сама создает структуру каталогов Turbo Pascal и переписывает файлы, необходимые для работы указанного пользователем набора средств. Например, в программе установки можно отказаться от установки тех или иных средств и библиотек (например, Turbo Vision или средств поддержки Windows).

Обычно установка из дистрибутива интуитивно понятна и никаких сложностей не вызывает. Дело обстоит несколько иначе с установкой простым копированием. Сложность здесь заключается в том, что неопытный пользователь не знает, какие файлы обязательно необходимы для работы в среде Turbo Pascal. Именно разрешению этого вопроса и посвящено данное приложение.

Минимальная установка

Для работы в среде Turbo Pascal обязательно должны быть установлены файлы turbo.exe (интегрированная среды программирования) и turbo.tpl (библиотека стандартных модулей Turbo Pascal). Можно работать, также, и без использования интегрированной среды программирования. В этом случае вместо файла turbo.exe обязательно должен быть установлен компилятор командной строки tpc.exe. Стандартные модули System, Crt, Printer и Dos входят в состав библиотеки Turbo.tpl, поэтому дополнительно копировать соответствующие файлы .tpu не обязательно. Таким образом, для начала разработки программ можно скопировать на компьютер всего два файла.

Полная установка

В случае минимальной установки системы Turbo Pascal, будут недоступны следующие средства: графический режим, поддержка строк PChar, средства Turbo Vision и интерактивная справочная система. Рассмотрим, какие файлы необходимы для работы каждого из перечисленных средств.

- Для работы в графическом режиме должен быть скопирован файл graph. tpu (модуль с графическими процедурами и функциями) и хотя бы один файл графического драйвера .bgi. В общем случае вполне достаточно одного файла .bgi, соответствующего установленному в системе видеоадаптеру (например, egavga.bgi для видеоадаптеров EGA и VGA). Кроме того, если в графическом режиме должны использоваться какие-либо внешние шрифты, то потребуется скопировать и соответствующие файлы . ch (при автоматической установке они размещаются вместе с драйверами .bgi в каталоге \TP\BGI).

ПРЕДУПРЕЖДЕНИЕ

Если файлы .tpu копируются не в тот же каталог, что и файл turbo.exe, то путь к ним должен быть указан при помощи команды **Options | Directories** (см. рис. 1.4) в интегрированной среде Turbo Pascal. В противном случае система не сможет их обнаружить.

- Для поддержки строк типа PChar необходимо скопировать модуль Strings . tpu.
- Для поддержки средств Turbo Vision необходимо скопировать модули App. tpu, Objects.tpu, Drivers.tpu, Memory.tpu, HistList.tpu, Views.Tpu, Menus.tpu, Dialogs.tpu, Validate.tpu и MsgBox.tpu. В рассматриваемых в этой книге примерах (см. гл. 18) использовались модули Editors.tpu и StdDlg.tpu.
- Для работы с интерактивной справочной системой необходимо скопировать файлы с расширением . tph. Для подключения дополнительных файлов справки используется команда меню **Help | Files** интегрированной среды Turbo Pascal. Файл turbo. tph используется как файл справки по умолчанию, и поэтому специального подключения не требует — достаточно просто скопировать его в тот же каталог, что и файл turbo.exe.

Достаточная установка для разработки и компиляции программ из этой книги

Для разработки и компиляции программ, представленых в частях I и II этой книги, достаточно минимальной установки, а для некоторых программ, рассматриваемых в остальных главах, следует дополнительно скопировать следующие модули:

- Модули App, Dialogs, Drivers, Editors, Memory, Menus, MsgBox, Objects, StdDlg, Views — программа Tv_1 (глава 18) и программа Phone (глава 22).
- Модуль Graph.
  - Глава 15, программы: Graph_01, Graph_02, Graph_03, Graph_04.
  - Глава 20, программы: FreePen, Hunter, Piano.
  - **Приложение Ж, программы:** FuncAbs, FuncAtan, FuncCos, FuncExp, FuncGBkC, FuncGDrN, FuncGetC, FuncGetX, FuncGetY, FuncGGrM, FuncGMdN, FuncGMxC, FuncGMxM, FuncGMxX, FuncGMxY, FuncGPix, FuncGPSz, FuncGrEM, FuncGrRs, FuncImgS, FuncInUF, FuncLn, FuncSin, FuncTxtH, FuncTxtW, ProcArc, ProcBar, ProcBar3, ProcClsG, ProcClVP., ProcCrcl, ProcDetG, ProcDPol, ProcElps, ProcFEll, ProcFFil, ProcFPol, ProcGDef, FuncGImg, ProcGtAC, ProcGtAR, ProcGtDP, ProcGtFP, ProcGtFS, ProcGtLS, ProcGtMr, ProcGtPa, ProcGtVS, ProcIniG, ProcLine, ProcLnRl, ProcLnTo, ProcMvRl, ProcMvTo, ProcOTXY, ProcOutT,

ProcPieS, FuncPImg, ProcPPxl, ProcRCRM, ProcRect, ProcSAlP, ProcSect, ProcSetC, ProcSetP, ProcSRGB, ProcStAP, ProcStAR, ProcStBC, ProcStFP, ProcStFS, FuncStGM, ProcStGS, ProcStLS, ProcStTJ, ProcStTS, ProcStVP, ProcSDCS, ProcSVPg.

- Модуль Strings — приложение Ж, программы: FuncSCat, FuncSCmp, FuncSCpy, FuncSDsp, FuncSECp, FuncSEnd, FuncSICm, FuncSLCm, FuncSLCp, FuncSLCt, FuncSLen, FuncSLIC, FuncSLow, FuncSMov, FuncSNew, FuncSPas, FuncSPCp, FuncSPos, FuncSRSc, FuncSScn, FuncSUpp.

Работа в Turbo Pascal с дискеты 3.5"

Если использовать следующий минимальный набор файлов:

- Turbo.exe;
- Turbo.tpl;
- Graph.tpu;
- Egavga.bgi,

то общий занимаемый ими объем будет составлять около 1 МБайта. Другими словами, минимальный набор файлов, требуемый для работы в интегрированной среде Turbo Pascal, вполне помещается на одной дискете.

Приложение Б

Команды меню интегрированной среды Turbo Pascal

В этом приложении представлены команды меню интегрированной среды Turbo Pascal, а также соответствующие им комбинаций клавиш для быстрого доступа к этим командам.

Кроме комбинаций клавиш быстрого доступа существует общее правило выполнения любой команды меню. Для того чтобы открыть меню, необходимо нажать клавишу <Alt>, и удерживая ее, нажать клавишу с первой буквой названия меню. Далее, для доступа к какой-нибудь команде меню необходимо, не отпуская клавишу <Alt>, нажать клавишу с буквой, выделенной в названии требуемой команды красным цветом.

Рис. Б.1. Меню **File**

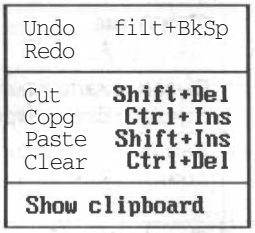


Рис. Б.2. Меню Edit

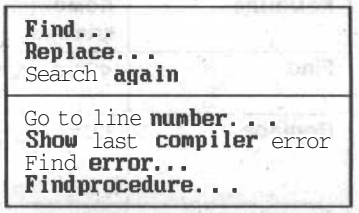


Рис. Б.3. Меню **Search**

Таблица Б.1. Меню **File** (рис. Б.1) — работа с файлами и принтером

Команда	Комбинация клавиш	Описание
New		Создать файл
Open	<F3>	Открыть файл
Save	<F2>	Сохранить файл
Save as		Сохранять файл как
Save all		Сохранить все открытые файлы
Change dir		Изменить рабочий каталог. Рабочим является каталог, в котором система Turbo Pascal по умолчанию сохраняет и ищет файлы
Print		Печать. Используется для печати содержимого активного окна редактора
Printer setup		Вызов диалогового окна настроек принтера
DOS shell		Временный выход в MS-DOS. Для возврата в Turbo Pascal необходимо выполнить в командной строке MS-DOS команду `exit`
Exit	<Alt+X>	Выход из Turbo Pascal

Таблица Б.2. Меню **Edit** (рис. Б.2) — команды редактора

Команда	Комбинация клавиш	Описание
Undo	<Alt+Bksp>	Отменить предыдущее действие, выполненное в редакторе
Redo		Повторить предыдущее действие, отмененное при помощи команды Undo
Cut	<Shift+Del>	Вырезать выделенный фрагмент текста в буфер обмена. Для выделения фрагмента текста необходимо воспользоваться клавишами управления курсором при нажатой клавише <Shift>
Copy	<Ctrl+ Insert>	Скопировать выделенный фрагмент текста в буфер обмена
Paste	<Shift+ Insert>	Вставить содержимое буфера обмена в текст
Clear	<Ctrl+Del>	Удалить выделенный фрагмент текста
Show clipboard		Показать содержимое буфера обмена

Таблица Б.3. Меню **Search** (рис. Б.3) — команды поиска и замены по тексту программы

Команда	Комбинация клавиш	Описание
Find	<Ctrl+Q+F>	Вызов диалогового окна, позволяющего ввести строку и параметры поиска
Replace	<Ctrl+Q+A>	Вызов диалогового окна, позволяющего ввести образец текста для поиска, и текст для замены этого образца
Search again	<Ctrl+L>	Повторяет последнюю выполненную команду Find или Replace
Go to line number		Переход к строке по ее номеру
Show last compiler error		Показать последнюю ошибку компиляции
Find error	<Alt+F8>	Поиск ошибки времени выполнения программы. После выполнения этой команды определяется адрес возникновения ошибки в виде "сегмент:смещение"
Find procedure		Поиск процедуры

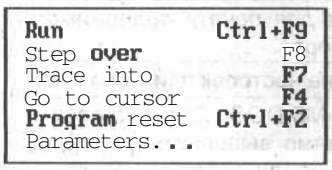


Рис. Б.4. Меню **Run**

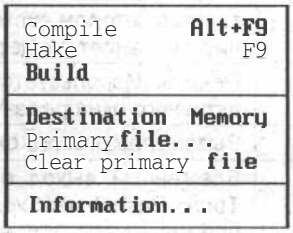


Рис. Б.5. Меню **Compile**

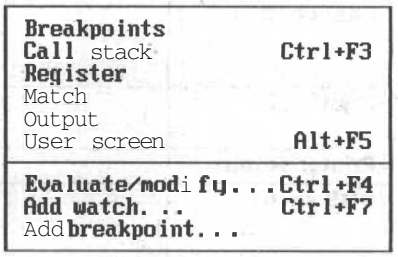


Рис. Б.6. Меню **Debug**

Таблица Б.4. Меню **Run** (рис. Б.4) — компиляция и выполнение программы

Команда	Комбинация клавиш	Описание
Run	<Ctrl+F9>	Компиляция и выполнение программы
Step over	<F8>	Пошаговый режим выполнения без захода в подпрограммы
Trace into	<F7>	Пошаговый режим выполнения с заходом в подпрограммы
Go to cursor	<F4>	Выполнить программу до курсора
Program reset	<Ctrl+F2>	Принудительное прекращение работы программы
Parameters		Позволяет указать набор параметров командной строки, передаваемые программе при запуске из MS-DOS

Таблица Б.5. Меню **Compile** (рис. Б.5) — компиляция и сборка программы

Команда	Комбинация клавиш	Описание
Compile	<Alt+F9>	Компиляция активного файла редактора
Make	<F9>	Сборка программного проекта
Build		Полная сборка программного проекта
Destination		Выбор направления для записи выполняемого файла (на диске или в памяти)
Primary file		Используется для определения основного файла программного проекта
Clear primary file		Отмена выбора основного файла программного проекта
Information		Информация о последней откомпилированной программе и текущем состоянии памяти

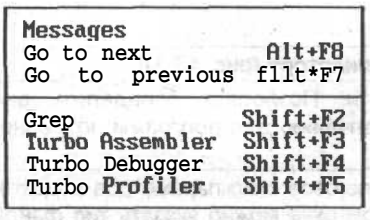


Рис. Б.7. Меню Tools

Рис. Б.8. Меню **Options**

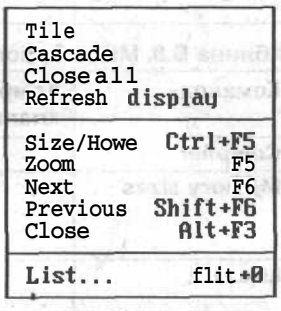


Рис. Б.9. Меню **Window**

Таблица Б.6. Меню **Debug** (рис. Б.6) — средства отладки

Команда	Комбинация клавиш	Описание
Breakpoints		Работа с точками прерывания (см. рис. 13.4)
Call stack	<Ctrl+F3>	Просмотр стека вызовов подпрограмм

Окончание таблицы Б.6

Команда	Комбинация клавиш	Описание
Register		Просмотр содержимого регистров процессора
Watch		Вызов окна, в котором можно просмотреть и изменить значения переменных и полей
Output		Просмотр результата работы программы
User screen	<Alt+F5>	Переключение между окном просмотра результатов выполнения программы и интегрированной средой Turbo Pascal
Evaluate/modify	<Ctrl+F4>	Оценка и изменение значений переменных и полей
Add watch	<Ctrl+F7>	Добавить расположенное под курсором значение в список окна Watch
Add breakpoint	<Ctrl+F8>	Установить точку прерывания

Таблица Б.7. Меню **Tools** (рис. Б.7) — сервисные средства

Команда	Комбинация клавиш	Описание
Messages		Раскрывает окно с информацией, выдаваемой программой при помощи фильтра DOS (типа GREP). Для отслеживания строк программы необходимо выбрать требуемое сообщение и нажать клавишу <SpaceBar> (Пробел)
Go to next	<Alt+F8>	Переход к следующему элементу списка имен программ, которые можно запускать, не выходя из среды Turbo Pascal
Go to previous	<Alt+F7>	Переход к предыдущему элементу списка имен программ, которые можно запускать, не выходя из среды Turbo Pascal
Grep	<Shift+F2>	Запуск средств поиска, позволяющих просматривать текст одновременно в нескольких файлах

Таблица Б.8. Меню **Options** (рис. Б.8) — параметры среды Turbo Pascal

Команда	Комбинация клавиш	Описание
Compiler		Параметры компилятора (рис. 13.1)
Memory sizes		Размеры памяти. Позволяет определить выбранные по умолчанию для программ потребности в памяти
Linker		Раскрывает диалоговое окно параметров редактора связей. В этом окне можно указать тип файла карты (MAP-файл) и место размещения буфера компоновки. Если буфер размещен в памяти (Memory), то это увеличивает скорость компиляции, однако для больших программ может не хватить памяти
Debugger		Параметры отладчика
Directories		Пути поиска файлов (см. рис. 1.4)

Окончание таблицы Б.8

Команда	Комбинация клавиш	Описание
Tools		Раскрывает диалоговое окно, в котором можно определить набор сервисных средств, отображаемых в меню Tools
Environment		Параметры интегрированной среды Turbo Pascal: общие параметры, параметры редактора, параметры мыши, параметры при запуске, цветовые настройки (см. рис. 1.5)
Open		Раскрывает диалоговое окно, позволяющее найти параметры Turbo Pascal, сохраненные ранее при помощи команды Options | Save
Save		Сохранение текущих параметров интегрированной среды Turbo Pascal с возможностью дальнейшего их восстановления. Набор параметров сохраняется в файле с расширением . tp
Save as		См. предыдущую команду

Таблица Б.9. Меню **Window** (рис. Б.9) — работа с окнами

Команда	Комбинация клавиш	Описание
Tile		Расположить окна таким образом, чтобы они не перекрывали друг друга
Cascade		Расположить окна каскадом друг над другом
Close all		Закрыть все окна
Refresh display		Обновить экран, если в результате выполнения программы изображение было искажено
Size/Move	<Ctrl+F5>	Перемещение и изменение размеров окна
Zoom	<F5>	Раскрывает активное окно во весь экран или восстанавливает его предыдущий размер
Next	<F6>	Активизирует следующее окно
Previous	<Shift+F6>	Активизирует предыдущее окно
Close	<Alt+F3>	Закрывает активное окно
List	<Alt+0>	Выводит список всех открытых окон
	<Alt+N>	Доступ к открытому окну по его номеру (N)

Таблица Б.10. Меню **Help** (рис. Б.10) — справочная система

Команда	Комбинация клавиш	Описание
Contents		Открывает окно справки с основной таблицей содержания
Index	<Shift+F1>	Вызов алфавитного оглавления справочной системы
Topic search	<Ctrl+F1>	Тематический поиск. При выборе этой команды отображается справочная информация о том элементе языка Pascal, на котором в данный момент расположен курсор

Окончание таблицы Б.10

Команда	Комбинация клавиш	Описание
Previous topic	<Alt+F1>	Вывод предыдущего окна справки
Using help		Подсказка по использованию справочной системы
Files		Файлы, входящие в состав справочной системы
Compiler directives		Справка о директивах компилятора
Reserved words		Справка о зарезервированных словах
Standard units		Справка о стандартных модулях
Turbo Pascal Language		Справка по синтаксису языка Turbo Pascal
Error messages		Справка о сообщениях об ошибках
About		Открытие окна с информацией о системе Turbo Pascal

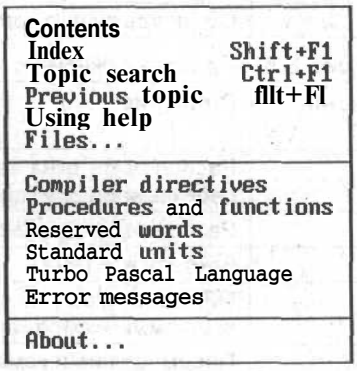


Рис. Б.10. Меню **Help**

Приложение B

Коды клавиатуры

В этом приложении рассматриваются коды клавиатуры, возвращаемые функцией `ReadKey`. При нажатии символьных клавиш возвращается их значение, соответствующее таблице ASCII (см. приложение Е). При нажатии управляющих клавиш возвращается два значения: символ #0 и некоторая величина, соответствующая нажатой клавише. Подобные коды клавиатуры называют **расширенными.** Расширенные коды клавиатуры представлены в табл. B.1.

Таблица B.1. Расширенные коды клавиатуры

Код	Клавиша	Код	Клавиша
16	<Alt+Q>	50	**<Alt+M>**
17	<Alt+W>	59	<F1>
18	<Alt+E>	60	<F2>
19	**<Alt+R>**	61	<F3>
20	<Alt+T>	62	<F4>
21	<Alt+Y>	63	<F5>
22	<Alt+U>	64	<F6>
23	<Alt+I>	65	<F7>
24	<Alt+0>	66	<F8>
25	<Alt+P>	67	<F9>
30	<Alt+A>	68	<F10>
31	<Alt+S>	71	<Home>
32	<Alt+D>	72	<t>
33	<Alt+F>	73	**<PageUp>**
34	<Alt+G>	75	**<←>**
35	<Alt+H>	77	**<→>**
36	<Alt+I>	79	<End>
37	<Alt+J>	80	**<↓>**
38	<Alt+K>	81	<PageDown>
39	**<Alt+L>**	82	<Insert>
44	<Alt+Z>	83	<Delete>
45	<Alt+X>	84	<Shift+Fl>
46	<Alt+C>	85	<Shift+F2>
47	<Alt+V>	86	<Shift+F3>
48	<Alt+B>	87	<Shift+F4>
49	<Alt+N>	88	**<Shift+F5>**

Окончание таблицы В.1

Код	Клавиша	Код	Клавиша
89	<Shift+F6>	115	<Cltr+←>
90	<Shift+F7>	116	<Ctrl+→>
91	<Shift+F8>	117	<Ctrl+End>
92	<Shift+F9>	118	<Ctrl+PageUp>
93	<Shift+F10>	119	<Ctrl+Home>
94	<Ctrl+Fl>	120	<Alt+l>
95	<Ctrl+F2>	121	<Alt+2>
96	<Ctrl+F3>	122	<Alt+3>
97	<Ctrl+F4>	123	<Alt+4>
98	<Ctrl+F5>	124	<Alt+5>
99	<Ctrl+F6>	125	<Alt+6>
100	<Ctrl+F7>	126	<Alt+7>
101	<Ctrl+F8>	127	<Alt+8>
102	<Ctrl+F9>	128	<Alt+9>
103	<Ctrl+F10>	129	<Alt+0>
104	<Alt+Fl>	130	<Alt+—>
105	<Alt+F2>	131	<Alt+=>
106	<Alt+F3>	132	<Ctrl+PageDown>
107	<Alt+F4>	133	<F11>
108	<Alt+F5>	134	<F12>
109	<Alt+F6>	135	<Shift+F11>
110	<Alt+F7>	136	<Shift+Fl2>
111	<Alt+F8>	137	<Ctrl+Fll>
112	<Alt+F9>	138	<Ctrl+F12>
113	<Alt+F10>	139	<Alt+Fll>
114	<Ctrl+PrtScr>	140	<Alt+F12>

Кроме того, в программах могут использоваться **коды опроса клавиатуры** — стандартные коды персонального компьютера, генерируемые при нажатии какой-либо клавиши. Коды опроса клавиатуры представлены в табл. В.2.

Таблица В.2. Коды опроса клавиатуры

Клавиша	Код опроса		Клавиша	Код опроса	
	В 16-ом виде	В 10-ом виде		В 16-ом виде	В 10-ом виде
<Escape>	01	1	<^>, <б>	07	7
<!>, <1>	02	2	<&>, <7>	08	8
<@>, <2>	03	3	<*>, <8>	09	9
<#>,<3>	04	4	<(>, <9>	0A	10
<$>, <4>	05	5	<)>, <0>	0B	11
<%>,<5>	06	6	<_>, <->	0C	12

Окончание таблицы В.2

Клавиша	Код опроса		Клавиша	Код опроса	
	В 16-ом виде	В **10-ом** виде		**В 16-ом** виде	В **10-ом** виде
<+>, <=>	OD	13	<M>	32	50
<BackSpace>	OE	14	<<>, <,>	33	51
<Tab>	OF	15	<>>, <.>	34	52
<Q>	10	16	<?>, </>	35	53
<W>	11	17	<Shift> (справа)	36	54
<E>	12	18	<PrintScreen>	37	55
<R>	13	19	<Alt>	**38**	56
<T>	14	20	<Space>	39	57
<Y>	15	21	<Caps Lock>	3A	58
<U>	16	22	<F1>	3B	59
<I>	17	23	<F2>	3C	60
<O>	18	24	<F3>	3D	61
<p>	19	25	<F4>	3E	62
<{>, <[>	1A	26	<F5>	3F	63
<}>, <]>	1B	27	<F6>	40	64
<Enter>	1C	**28**	<F7>	41	65
<Ctrl>	1D	29	<F8>	42	66
<A>	**1E**	30	<F9>	43	67
<S>	**1F**	31	<F10>	44	68
<D>	20	32	<Num Lock>	45	69
<F>	21	33	<Scroll Lock>	46	70
<G>	22	34	<Home>	47	71
<H>	23	35	<t>	48	72
<J>	24	36	<PageUp>	49	73
<K>	25	37	<—>	4A	74
<L>	26	38	<←>	4B	75
<:>, <;>	27	39	<5>	4C	76
<">, <'>	28	40	<→>	4D	77
<~>, <`>	29	41	<+>	4E	78
<Shift> (слева)	2A	**42**	<End>	4F	79
<|>, <\>	2B	43	<↓>	50	80
<Z>	2C	**44**	<PageUp>	51	81
<X>	2D	45	<Insert>	52	82
<C>	2E	46	<Delete>	53	83
<V>	2F	47	<F11>	D9	217
<B>	30	48	<F12>	DA	218
<N>	31	49			

Приложение Г

Основные прерывания BIOS и MS-DOS

В этом приложении описаны основные прерывания BIOS и MS-DOS, которые могут использоваться при разработке программ в среде Turbo Pascal. Те прерывания, которые здесь не предстапвлены, относятся либо к системным прерываниям (то есть к таким, которые вызываются не из пользовательских программ, а самой системой), либо к специфическим и редко используемым.

Прерывания BIOS

Прерывание $05 — печать экрана

Входные данные: нет.
Выходные данные: нет.

Это прерывание вызывается напрямую прерыванием $09 в ответ на нажатие на клавиатуре клавиши <PrintScreen>. Кроме того, его можно вызывать программно.

Прерывание $10 — работа с дисплеем

Это прерывание имеет множество функций, предназначенных для управления выводом на экран и считыванием данных из видеопамяти. Номер функции сохраняется в регистре AH. Остальные входные данные зависят от выбранной функции.

Функция $00 — установка видеорежима

Входные данные
AH = 0
AL — номер видеорежима (табл. Г.1).

Выходные данные: нет.

Номер видеорежима выбирается в соответствии с табл. ГЛ.

Таблица Г.1. Номера видеорежимов, используемые в функции $00 прерывания $10

Регистр AL	Тип режима	Размер экрана, в символах	Размер символа, в пикселях	Цвета	Адаптер
0	Текстовый	40x25	8x8, 8x14	16/8	CGA, EGA
1	Текстовый	40x25	8x8, 8x14	16	CGA, EGA
2	Текстовый	80x25	8x8, 8x14	16/8	CGA, EGA
3	Текстовый	80x25	8x8, 8x14	16	CGA, EGA
4	Графический	320x200	8x8	4	CGA, EGA

Окончание таблицы В.2

Регистр AL	Тип режима	Размер экрана, в символах	Размер символа, в пикселях	Цвета	Адаптер
5	Графический	320x200	8x8	4*	CGA, EGA
6	Графический	640x200	8x8	2	CGA, EGA
7	Текстовый	80x25	9x14	3	MDA, EGA
$0D	Графический	320x200	8x8	16	EGA, VGA
$OE	Графический	640x200	8x8	16	EGA, VGA
$OF	Графический	640x350	8x14	3	EGA, VGA
$10	Графический	640x350	8x14	4 или 16	EGA, VGA
$11	Графический	640x480	8x16	2	VGA
$12	Графический	640x480	8x16	16	VGA
$13	Графический	640x480	8x16	256	VGA

* — оттенки серого цвета.

Функция $01 — установка формы и размера курсора

Входные данные

AH = 1

CH — высота верхней линии матрицы курсора.

CL — высота нижней линии матрицы курсора.

Выходные данные: нет.

Высота верхней линии матрицы курсора выбирается в диапазоне от 0 (наибольшая высота) до $1F (наименьшая высота). Если CH = $20, то курсор невидим. Высота нижней линии также выбирается в диапазоне от 0 до $1F.

Высота матрицы курсора соответствует высоте символов (в пикселях), и поэтому "отличается для различных видеорежимов:

- в текстовом режиме VGA (25 строк) высота матрицы курсора составляет 16 пикселей;
- в режиме EGA (25 строк) — 14 пикселей;
- в режимах CGA (25 строк);
- в режиме EGA (43 строки);
- в режиме VGA (50 строк) — 8 пикселей.

Функция $02 — установка позиции курсора

Входные данные

AH = 2

BH — номер видеостраницы (начиная с 0).

DH — номер строки (начиная с 0).

DL — номер столбца (начиная с 0).

Выходные данные: нет.

Функция $03 — запрос о позиции и размере курсора

Входные данные

AH = 3

BH — номер видеостраницы (начиная с 0).

Выходные данные
CH — высота верхней линии матрицы курсора.
CL — высота нижней линии матрицы курсора.
DH — текущая строка (начиная с 0).
DL — текущий столбец (начиная с 0).

Функция $05 — выбор видеостраницы

Входные данные
AH = 5
AL — номер видеостраницы (начиная с 0).

Выходные данные: нет.

В режимах CGA, EGA и VGA поддерживается до 8 видеостраниц, но в большинстве режимов MDA — только одна.

Функция $06 — очистка области экрана сдвигом строк вниз

Функция $07 — очистка области экрана сдвигом строк вверх

Входные данные
AH = 6 (7)
AL — количество сдвигаемых строк. Нулю соответствует очистка всей области с заданными координатами.
BH — байт атрибутов цветности (◄◄ рассматривается в главе 15) для очищаемой области.
CH, CL — строка и столбец расположения правого нижнего угла очищаемой области.
DH, DL — строка и столбец расположения левого верхнего угла очищаемой области.

Выходные данные: нет.

Функция $08 — считывание символа и байта атрибутов цветности в текущей позиции курсора

Входные данные
AH = 8
BH — номер видеостраницы (начиная с 0).

Выходные данные
AL — код ASCII считанного символа (►► коды ASCII см. в приложении Е).
BH — байт атрибутов цветности (◄◄ рассматривается в главе 15).

Функция $09 — вывод символа с установленными атрибутами цветности в текущей позиции курсора

Входные данные
AH = 9
AL — код ASCII выводимого символа (►► коды ASCII см. в приложении Е). Символы с кодами, превышающими 127, в графическом режиме определяются таблицей, устанавливаемой прерыванием $1F.
BH — номер видеостраницы (начиная с 0).
BL — в текстовом режиме — байт атрибутов цветности; в графическом режиме — номер цвета (◄◄ рассматриваются в главе 15).

CX — количество повторений символа (начиная с 1).

Выходные данные: нет.

Функция $0A — вывод символа с текущими атрибутами цветности в текущей позиции курсора

Входные данные
AH = $0A
AL — код ASCII выводимого символа (» коды ASCII см. в приложении Е). Символы с кодами, превышающими 127, в графическом режиме определяются таблицей, устанавливаемой прерыванием $1F.
BH — номер видеостраницы (начиная с 0).
CX — количество повторений символа (начиная с 1).

Выходные данные: нет.

Функция $0B — выбор графической палитры или цвета границы экрана в текстовом режиме

Входные данные
AH = $0B
BL — в текстовом режиме — значение в диапазонах от 0 до $0F (цвета с низкой интенсивностью) и от $10 до $1F (цвета с высокой интенсивностью). В графическом режиме — значения 0 (палитра green/red/brown) или 1 (палитра cyan/magenta/white).

Выходные данные: нет.

Функция $0C — вывод пикселя в графическом режиме

Входные данные
AH = $0C
AL — номер цвета (« рассматривается в главе 15). Значению $80 соответствует операция XOR с текущим значением цвета.
BH — номер видеостраницы (начиная с 0).
CX — координата по горизонтали (начиная с 1).
DX — координата по вертикали (начиная с 1).

Выходные данные: нет.

Функция $0D — считывание цвета пикселя в графическом режиме

Входные данные
AH = $0D
BH — номер видеостраницы (начиная с 0).
CX — координата по горизонтали (начиная с 1).
DX — координата по вертикали (начиная с 1).

Выходные данные: AL — номер цвета (« рассматривается в главе 15).

Функция $0E — вывод символа с переносом курсора

Входные данные
AH = $0E
AL — код ASCII выводимого символа (коды ASCII см. в приложении Е).
BL — в графическом режиме — номер цвета фона (« рассматривается в главе 15).

Выходные данные: AL — номер цвета (« рассматривается в главе 15).

Эта функция выводит символ, код которого указан в регистре AL, в текущей позиции экрана и изменяет положение курсора. Если курсор выходит за пределы конца последней строки экрана, то текст сдвигается на одну строку вверх.

Функция $OF — запрос информации о текущем видеорежиме

Входные данные: AH = $OF.

Выходные данные

AL — номер текущего видеорежима (см. табл. Г.1).
AH — ширина экрана в символах.
BH — номер активной в данный момент видеостраницы (начиная с 0).

Функция $11 — функции генератора символов (EGA/VGA)

Эта функция имеет 15 подфункций, номер которых записывается в регистр AL.

Подфункция $00 — загрузка шрифта, определенного пользователем

Входные данные

AH = $11
AL = 0
BH — высота каждого символа.
BL — блок шрифта для загрузки (для EGA — от 0 до 3, для VGA — от 0 до 7).
CX — количество переопределяемых символов.
DX — код ASCII первого символа из области, адрес которой находится в ES:BP (▸▸ коды ASCII см. в приложении Е).
ES:BP — адрес области памяти, в которой хранится информация о шрифте.

Выходные данные: нет.

Подфункция $01 — загрузка из ПЗУ набора символов размерами 8x14

Подфункция $02 — загрузка из ПЗУ набора символов размерами 8x8

Входные данные

AH = $11
Al = 1 (2)
BL — блок шрифта для загрузки (для EGA — от 0 до 3, для VGA — от 0 до 7).

Выходные данные: нет.

Подфункция $03 — активизация блока шрифта

Входные данные

AH = $11
Al = 3
BL — код селектора блока шрифта (для EGA — от 0 до 3, для VGA — от 0 до 7).

Выходные данные: нет.

Подфункция $04 — загрузка из ПЗУ набора символов размерами 8x16 (VGA)

Входные данные

AH = $11
AL = 4
BL — блок шрифта для загрузки (от 0 до 7).

Выходные данные: нет.

Подфункция $10 — загрузка и активизация шрифта, определенного пользователем

Входные данные

AH = $11
AL = $10
BH — высота каждого символа.
BL — блок шрифта для загрузки (для EGA — от 0 до 3, для VGA — от 0 до 7).
CX — количество переопределяемых символов.
DX — код ASCII первого символа из области, адрес которой находится в ES:BP (коды ASCII см. в приложении Е).
ES:BP — адрес области памяти, в которой хранится информация о шрифте.

Выходные данные: нет.

Подфункция $11 — загрузка из ПЗУ и активизация набора символов размерами 8x14

Подфункция $12 — загрузка из ПЗУ и активизация набора символов размерами 8x8

Входные данные

AH = $11
AL = $11 ($12)
BL — блок шрифта для загрузки (для EGA — от 0 до 3, для VGA — от 0 до 7).

Выходные данные: нет.

Подфункция $14 — загрузка из ПЗУ и активизация набора символов размерами 8x16 (VGA)

Входные данные

AH = $11
AL = $14
BL — блок шрифта для загрузки (от 0 до 7).

Выходные данные: нет.

Подфункция $20 — установка вектора для прерывания $1F

Входные данные

AH = $11
AL = $20
ES:BP — адрес области памяти, в которой находится информация, описывающая шрифт размерами 8x8 для символов с кодами ASCII, превышающими 127.

Выходные данные: нет.

Подфункция $21 — установка шрифта, определенного пользователем, для графического режима

Входные данные

AH = $11
AL = $21
BL — код размерности экрана по вертикали:
0 — определен пользователем (в регистре DL);
1 — 14 строк;
2 — 25 строк;
3 — 43 строки.
CX — количество байтов, отводимых на описание символа.

DL — размерность экрана по вертикали (количество строк) при BL = 0.
ES:BP — адрес области памяти, в которой находится информация, описывающая шрифт.

Выходные данные: нет.

Подфункция $22 — установка шрифта ПЗУ размерами 8x14 для графического режима

Подфункция $23 — установка шрифта ПЗУ размерами 8x8 для графического режима

Подфункция $24 — установка шрифта ПЗУ размерами 8x16 для графического режима

Входные данные
AH = $11
AL = $22 ($23, $24)
BL — код размерности экрана по вертикали:
0 — определен пользователем (в регистре DL);
1 — 14 строк;
2 — 25 строк;
3 — 43 строки.
DL — размерность экрана по вертикали (количество строк) при BL = 0.

Выходные данные: нет.

Подфункция $30 — извлечение информации о текущем генераторе символов

Входные данные
AH = $11
AL = $30
BH — код запроса на извлечение адреса:
0 — в вектор прерывания $1F;
1 — в вектор прерывания $43;
2 — шрифта ПЗУ размерами 8x14;
3 — шрифта ПЗУ размерами 8x8;
4 — шрифта ПЗУ размерами 8x8 (вторая часть);
5 — дополнительного шрифта ПЗУ размерами 8x14;
6 — шрифта ПЗУ размерами 8x16 (для VGA);
7 — дополнительного шрифта ПЗУ размерами 8x16 (для VGA).

Выходные данные
CX — высота символа (число байтов, отводимых для отображения 1 символа).
DL — высота экрана в строках.
ES:BP — адрес запрашиваемой таблицы описания шрифта.

Функция $12 — специальные функции EGA/VGA

Эта функция применима только для видеоадаптеров EGA и VGA, и недоступна для систем CGA или MDA. Она имеет ряд подфункций, номера которых записываются в регистр BL. Рассмотрим некоторые из них.

Подфункция $10 — извлечь информацию о режиме EGA

Входные данные
AH = $12
BL = $10

Выходные данные

BH — установка BIOS по умолчанию (0 — цветной режим; 1 — монохромный режим).

BL — код размера памяти (0 — 64 Кбайта; 1 — 128 Кбайт; 2 — 192 Кбайта; 3 — 256 КБайт). Если BL > 3, то используется BIOS CGA или MDA.

CH — биты характеристик (соответствуют контактам разъемов RCA).

CL — установки переключателей.

Подфункция $30 — установить количество точек экрана по вертикали для текстового режима

Входные данные

AH = $12

AL — код размерности по вертикали:

0 — 200 точек (EGA/VGA);

1 — 350 точек (EGA/VGA);

2 — 400 точек (только для VGA).

BL = $30

Выходные данные

AH — если в данный момент выбран режим VGA, то в регистр AH возвращается значение `$12`.

Подфункция $36 — отключение/включение обновления экрана

Входные данные

AH = $12

AL — установка: 0 — включить обновление экрана; 1 — отключить обновление экрана.

BL = $36

Выходные данные

AH — если в AL было передано корректное значение, то в регистр AH возвращается значение `$12`.

Сложная графическая информация будет отображаться быстрее, если отключить обновление экрана. После завершения загрузки информации в видеопамять обновление экрана опять должно быть включено.

Функция $13 — вывод строки (AT/EGA/VGA)

Эта функция имеет четыре подфункции в соответствии с характером вывода строки.

Подфункция $00 — вывод строки без перемещения курсора

Подфункция $01 — вывод строки с перемещением курсора

Входные данные

AH = $13

AL = O (1)

BH — номер видеостраницы (начиная с 0).

BL — байт атрибутов цветности (◀◀ рассматривается в главе 15).

CX — длина выводимой строки.

DH, DL — строка и столбец позиции вывода.

ES:BP — адрес области памяти, в которой хранится выводимая строка.

Выходные данные: нет.

Подфункция $02 — вывод строки с форматированием атрибутов цветности без переноса курсора

Подфункция $03 — вывод строки с форматированием атрибутов цветности с переносом курсора

Входные данные

AH = $13

AL = 2(3)

BH — номер видеостраницы (начиная с 0).

CX — количество выводимых символов.

DH, DL — строка и столбец позиции вывода.

ES:BP — адрес области памяти, в которой хранится выводимая строка с размещенными в ней байтами атрибутов цветности.

Выходные данные: нет.

В случае использования этой функции после каждого символа в строке должен располагаться его байт атрибутов цветности. Таким образом, фактическая длина строки должна в два раза превышать количество выводимых символов.

Предположим, необходимо вывести на экран строку "Привет!". При этом восклицательный знак должен отображаться другим цветом. Для этого строковой переменной (например, s) должно быть присвоено значение, подобное следующему:

```
s := 'П'+#7+'р'+#7+'и'+#7+'в'+#7+'е'+#7+'т'+#7+'!'+#15;
```

Затем при помощи команды встроенного ассемблера

```
LEA BP, S+1
```

адрес строки s сохраняется в регистре BP. После каждого символа в строке хранится байт атрибутов цветности. В данном случае символы до восклицательного знака будут выведены светло серым цветом на черном фоне (атрибут цветности $07), а символ "!" — белым цветом на черном фоне (атрибут цветности $OF = 15).

Прерывание $11 — запрос списка подключенного оборудования

Входные данные: нет.

Выходные данные

AX — информация о подключенных устройствах.

Распределение битов значения, возвращаемого в регистр AX представлено в табл. Г.2.

Таблица Г.2. Значения битов в регистре AX после вызова прерывания $11

Биты	Значение
0	Если этот бит установлен в "1", то в системе используется как минимум одно дисковое устройство
1	Если этот бит установлен в "Г, то в системе используется сопроцессор
2, 3	Код размера памяти, установленной на материнской плате: 01 — 16 Кбайт; 10 — 32 КБайта; 11 — 64 Кбайта и больше
4, 5	Активный в данный момент видеоадаптер: 00 — не используется; 01 — цветной режим 40x25; 10 — цветной режим 80x25; 11 — черно-белый режим 80x25
6, 7	Количество обнаруженных дисководов: 00 — 1; 01 — 2; 10 — 3; 11 — 4
8	Если этот бит установлен в "1", то в системе используется оборудование прямого доступа к памяти

Окончание таблицы Г.2

Биты	Значение
9–11	Количество последовательных портов RS-232: 000 — 0; 001 — 1; ... 100 — 4; ... 111— 7
12	Если этот бит установлен в "Г, то в системе используется игровой адаптер.
13	Если этот бит установлен в "1", то к системе подключен принтер через последовательный порт
14, 15	Количество параллельных портов, как правило, используемых для подключения принтеров: 00 — 0; 01 — 1 ; 10 — 2; 11 — 3

Прерывание $12 — запрос размера физической памяти

Входные данные: нет.
Выходные данные: AX — размер памяти в килобайтах.

Прерывание $13 — дисковые операции ввода/вывода

Это прерывание имеет ряд функций, используемых для прямого доступа к адаптерам дисководов и жестких дисков. Номер функции записывается в регистр AH.

Функция $00 — сброс контроллера диска

Входные данные: AH = 0.
Выходные данные: AH — код ошибки BIOS.

Вызов этой функции устанавливает для всех дисков нулевую головку на нулевую дорожку. Если при вызове этой функции возникла ошибка, то код этой ошибки возвращается в регистр AH. Перечень кодов ошибок ввода/вывода, возникающих при работе с дисководов и с жесткими дисками представлены в табл. Г.3 и табл. Г.4.

Таблица Г.3. Коды ошибок ввода/вывода, возникающих при работе с дисководами

Код ошибки	Описание
$00	Нет ошибок
$01	Неверная команда — ошибка запроса к контроллеру
$01	Неверная команда — ошибка запроса к контроллеру
$02	Неверная метка адреса
$03	Попытка записи на дискету, защищенную от записи
$04	Идентификатор сектора неверен или не найден
$06	В дисководе отсутствует дискета
$08	Неудачная попытка прямого доступа к памяти
$09	Выход за границы памяти в режиме прямого доступа — попытка записи данных за пределами области в 64 КБайта
$OC	Некорректный тип носителя
$10	Обнаружена ошибка в данных в результате контроля при помощью циклического избыточного кода (CRC)
$20	Отказ контроллера дисковода
$40	Ошибка поиска — запрашиваемая дорожка не найдена
$80	Превышение допустимого времени простоя

Таблица Г.4. Коды ошибок ввода/вывода, возникающих при работе с жесткими дисками

Код ошибки	Описание
$00	Нет ошибок
$01	Неверная команда — ошибка запроса к контроллеру
$02	Неверная метка адреса
$03	Попытка записи на диск, защищенный от записи
$04	Идентификатор дорожки/сектора неверен или не найден
$05	Неудачная попытка сброса контроллера
$06	Дискета извлечена из дисковода после последнего обращения
$07	Ошибка в рабочих параметрах устройства
$08	Неудачная попытка прямого доступа к памяти (DMA-доступ — direct memory access), данные переданы слишком быстро
$09	Выход за границы памяти в режиме прямого доступа (DMA-доступ) — попытка чтения/записи данных за пределами области в 64 КБайта
$0A	Обнаружен флажок запорченного сектора
$0B	Обнаружен флажок запорченного цилиндра
$0D	В формате неверно указано количество секторов
$0E	Обнаружена метка адреса управляющих данных
$0F	Выход за диапазон уровня разрешения конфликтов при прямом доступе к памяти
$10	Ошибка контроля ECC или CRC, не поддающаяся корректировке
$11	Ошибка в данных, откорректированных с использованием кода ECC
$20	Отказ контроллера жесткого диска
$40	Ошибка поиска — запрашиваемая дорожка не найдена
$80	Превышение допустимого времени ожидания для жесткого диска или отсутствие дискеты в дисководе
$AA	Устройство не готово
$BB	Неопределенная ошибка
$CC	Ошибка записи на выбранное устройство
$EO	Полная недоступность устройства
$FF	Ошибка операции считывания

ПРИМЕЧАНИЕ

ECC (Error Correction Code) — кодирование с исправлением ошибки — метод кодирования информации, который позволяет выявлять и исправлять ошибки, возникающие при передаче или хранении данных. ECC характеризуется двумя основными показателями: максимальным числом ошибочных бит, которые можно обнаружить, и максимальным числом бит, которые можно исправить. Если число ошибочных бит превышает эти пределы, ошибки могут остаться невыявленными или неисправленными.

CRC (Cyclic Redundancy Check) — контроль передачи данных избыточным циклическим кодом. В этом способе контроля применяется сложная система вычислений, в результате которых на основе переданной информации генерируется некоторое число. Устройство-отправитель выполняет вычисления перед передачей основных данных и посылает результат вычисления в устройство-получатель. Последнее устройство, получив данные, повторяет те же вычисления. Если в обоих устройствах результаты вычислений совпадают, считается, что передача данных прошла без ошибок.

ОКОНЧАНИЕ ПРИМЕЧАНИЯ

Такой контроль называется избыточным потому, что в каждый передаваемый блок данных включаются дополнительные (избыточные) данные.

Функция $01 — запрос об ошибке диска

Входные данные

AH = 1

DL — от $00 до $03 — запрос информации о дисководах; $80 или $81 — запрос информации о состоянии контроллера жесткого диска.

Выходные данные: AH — код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Функция $02 — чтение секторов

Входные данные

AH = 2

AL — количество секторов.

CH — номер дорожки (цилиндра), начиная с 0.

CL — номер сектора, начиная с 1.

DL — устройство: 0—3 — дисководы; $80 или $81 — жесткий диск.

DH — номер головки.

ES:BX — адрес принимающего буфера данных.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4). Если AH = 0, то буфер, адрес которого указан в ES:BX, будет содержать считанные данные.

ПРИМЕЧАНИЕ

Фактически, для этой функции регистр CX разбивается не на равные 8-битные части, а на два поля — 6 бит (младшие биты регистра CL) и 10 бит (регистр CH + два бита регистра CL). Это означает, что номер дорожки не может превышать 1024, а номер сектора — 64.

Функция $03 — запись секторов

Входные данные

AH = 3

AL — количество секторов.

CH — номер дорожки (цилиндра), начиная с 0.

CL — номер сектора, начиная с 1.

DL — устройство: 0—3 — дисководы; $80 или $81 — жесткий диск.

DH — номер головки.

ES:BX — адрес буфера с записываемыми данными.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Функция $04 — проверка секторов

Входные данные

AH = 4

AL — количество секторов.

CH — номер дорожки (цилиндра), начиная с 0.

CL — номер сектора, начиная с 1.
DL — устройство: 0–3 — дисководы; $80 или $81 — жесткий диск.
DH — номер головки.

Выходные данные
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Функция $05 — форматирование дорожки

Входные данные
AH = 5
AL — количество секторов.
CX — номер дорожки (цилиндра) в старших 10 битах (◄◄ см. примечание для функции $02).
DL — устройство: 0–3 — дисководы; $80 или $81 — жесткий диск.
DH — номер головки.
ES:BX — адрес буфера, содержащего информацию о формате.

Выходные данные
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Адрес информации о формате записывается в регистр ES:BX. Эта информация отличается в зависимости от типа системы и характера выполняемой операции. Для систем PC и любых операций с дискетами буфер формата содержит последовательность 4-байтовых значений — по одному для каждого сектора в дорожке (табл. Г.5).

Таблица Г.5. Информация о формате дорожки для систем PC

Байт	Описание
0	Номер дорожки (начиная с 0)
1	Номер головки (начиная с 0)
2	Номер сектора (начиная с 1)
3	Код длины одной записи: 0 — 128 байт; 1 — 125 байт; 2 — 512 байт; 3 — 1 024 байт

Для систем AT, всех современных жестких дисков и BIOS в буфер формата записываются 2-байтные значения — по одному для каждого сектора дорожки (табл. Г.6).

Таблица Г.6. Информация о формате дорожки для систем AT

Байт	**Описание**
0	Флажок: $00 — сектор нормальный; $80 — сектор запорчен или не используется
1	Номер головки (начиная с 0)

Для очень старых систем XT вместо области памяти, адрес которой хранится в ES:BX, в качестве буфера используется регистр AL, содержащий значение в диапазоне от 1 до 16.

Функция $08 — извлечение параметров устройства

Входные данные
AH = 8
DL — устройство: 0-3 — дисководы; $80 или $81 — жесткий диск.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

CX — максимальное значение для цилиндра и сектора (для сектора — младшие 6 битов CL, для цилиндра — CH + два бита CL).

DL — количество жестких дисков на первом контроллере.

DH — максимальное значение для головки.

ES:DI — адрес таблицы параметров жесткого диска.

Таблица параметров жесткого диска — это 16-байтовая структура, представленная в табл. Г.7.

Таблица Г.7. Информация о формате дорожки для систем PC

Смещение	Длина	Описание
0	2	Максимальное количество цилиндров
2	1	Максимальное количество дорожек
3	2	Начальный цилиндр при записи сокращенным потоком
5	2	Начальный цилиндр при записи со смещением битовых элементов (предкомпенсацией)
7	1	Максимальная длина пакета данных ECC
8	1	Параметры устройства: биты 0–2 — вариант выбора устройства; бит 6 — отключение повторов; бит 7 — отключение контроля ECC
9	1	Стандартное значение времени простоя
10	1	Значение времени простоя для форматирования
11	1	Значение времени простоя для проверки
12	4	Зарезервированы

Функция $0A — чтение сектора и данных ECC

Входные данные

AH = $0A

AL — количество секторов (обычно 1).

CH — номер дорожки (цилиндра), начиная с 0.

CL — номер сектора, начиная с 1.

DL — устройство: $80 или $81 — жесткий диск.

DH — номер головки.

ES:BX — адрес принимающего буфера данных.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4). Если AH = 0, то буфер, адрес которого указан в ES:BX, будет содержать считанные данные.

Эта функция работает аналогично стандартной функции $02 за исключением того, что она считывает от четырех до семи байтов данных ECC, расположенных сразу за содержимым сектора.

◄◄ см. также примечание к функции $00 прерывания $13.

Функция $0B — запись сектора и данных ECC

Входные данные
AH = $0B
AL — количество секторов (обычно 1).
CH — номер дорожки (цилиндра), начиная с 0.
CL — номер сектора, начиная с 1.
DL — устройство: $80 или $81 — жесткий диск.
DH — номер головки.
ES:BX — адрес буфера с записываемыми данными.

Выходные данные
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и **табл.** Г.4).

Эта функция работает аналогично стандартной функции $03 за исключением того, что данные ECC не вычисляются контроллером, а считываются из последних байтов буфера (от четырех до семи); адрес которого указан в регистре **ES:BX**.

Функция $0C — поиск цилиндра

Входные данные
AH = $0C
CX — номер дорожки (цилиндра) — старшие 10 бит (◀◀ см. примечание к функции $02 прерывания $13).
DL — устройство: $80 или **$81** — жесткий диск.
DH — номер головки.

Выходные данные
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3, и **табл.** Г.4).

Функция $0D — сброс контроллера жесткого диска

Входные данные
AH = $0D
DL — устройство: $80 или **$81** — жесткий диск.

Выходные данные
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и **табл.** Г.4).

Функция $10 — проверка готовности устройства

Входные данные
AH = $10
DL — устройство: $80 или **$81** — жесткий диск.

Выходные **данные**
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Функция $11 — повторная проверка устройства

Входные данные
AH = $11
DL — устройство: $80 или **$81** — жесткий диск.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Функция $14 — самопроверка контроллера

Входные данные: AH = $14

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Функция $15 — определение типа дискеты или проверка наличия жесткого диска

Входные данные

AH = $15

DL — устройство: от 0 до 3 — дисководы; $80 или $81 — жесткий диск.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4). Если флажок CF не установлен, то содержимое регистра AH имеет следующее значение: 0 — устройство не установлено; 1 — дисковод (невозможно определить наличие дискеты в дисководе); 2 — дисковод (можно определить наличие дискеты в дисководе); 3 — жесткий диск установлен.

Функция $16 — определение смены носителя

Входные данные

AH = $16

DL — номер дисковода от 0 до 3.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4). Если флажок CF не установлен, то содержимое регистра AH имеет следующее значение: 0 — дискета извлечена; 1 — неверный номер устройства; 2 — определение смены дискеты не поддерживается или была вставлена другая дискета; $80 — дисковод не готов или не установлен.

Функция $17 — выбор скорости передачи данных

Входные данные

AH = $17

AL — скорость передачи в зависимости от типа дискеты и дисковода:

1 — дискета объемом 360 Кбайт в устройстве для дискет объемом 360 Кбайт;
2 — дискета объемом 360 Кбайт в устройстве для дискет объемом 1.2 Мбайт;
3 — дискета объемом 1.2 Мбайт в устройстве для дискет объемом 1.2 Мбайт;
4 — дискета объемом 720 Кбайт в устройстве для дискет объемом 720 Кбайт.

DL — номер дисковода от 0 до 3.

Выходные данные

AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

ПРИМЕЧАНИЕ

Этой функцией не поддерживаются дисководы, работающие с дискетами объемом 1.44 Мбайт и 2.88 Мбайт. Для подобных дисководов используется функция $18 (рассматривается в следующем подразделе).

Функция $18 — выбор типа носителя для форматирования

Входные данные
AH = $18
CH — максимальное количество дорожек.
CL — максимальное количество секторов на дорожку.
DL — номер дисковода от 0 до 3.

Выходные данные
AH — если флажок CF установлен, то возвращен код ошибки BIOS (см. табл. Г.3 и табл. Г.4).

Прерывание $14 — ввод/вывод через последовательные порты

Это прерывание состоит из четырех функций, обеспечивающих доступ к четырем последовательным портам RS-232. Номер функции записывается в регистр AH.

Функция $00 — инициализация последовательного порта

Входные данные
AH = 0
AL — флажки инициализации.
DX — номер COM-порта (от 0 до 3).

Выходные данные
AH — состояние линии.
AL — состояние модема.

Перед вызовом этой функции в регистр AL записывается байт с флажками инициализации. Структура этого байта представлена в табл. Г.8.

Таблица Г.8. Структура байта инициализации COM-порта

Биты	Описание
0, 1	Длина слова: 10 — 7 битов; 11 — 8 битов
2	Количество стоповых битов: 0 — 1, 1 — 2
3, 4	Код четности: 00,10 — отсутствует; 01 — нечетный; 11 — четный
5–7	Скорость передачи: 000 — 110; 001 — 150; 010 — 300; 011 — 600; 100 — 1200; 101 — 2400; 110 — 4800; 111 — 9600 BIOS не поддерживает скорости передачи, превышающие 9600 Кбит/с

После вызова этой функции в регистре AH хранится байт состояния линии, а в регистре AL — байт состояния модема. Структура этих байтов представлена в табл. Г.9 и Г.10.

Таблица Г.9. Значение флажков состояния линии

Биты	Описание
0	Данные готовы
1	Ошибка переполнения
2	Ошибка четности
3	Ошибка синхронизации кадров данных
4	Обнаружен разрыв
5	Регистр продолжения передачи пуст
6	Регистр сдвига передачи пуст
7	Превышение допустимого времени ожидания (или указывает на наличие любой ошибки)

Таблица Г.10. Значение флажков состояния модема

Биты	Описание
0	Фрагмент данных может быть передан
1	Фрагмент набора данных готов
2	Обнаружен звонок на нисходящем сигнале
3	В линии обнаружен сигнал приема фрагмента данных
4	Свободен для передачи
5	Набор данных готов
6	Обнаружен звонок
7	Принят сигнал обнаружения линии

Функция $01 — передача символа

Входные данные

AH = 1

AL — передаваемый байт.

DX — номер COM-порта (от 0 до 3).

Выходные данные

AH — код ошибки (см. табл. Г.9). Если восьмой бит находится в состоянии "О", то символ передан без ошибок, в противном случае в первых семи битах хранится код состояния линии.

Функция $02 — прием символа

Входные данные

AH = 2

DX — номер COM-порта (от 0 до 3).

Выходные данные

AH — индикатор успешного приема. Если регистр AH содержит 0, то символ принят без ошибок, в противном случае регистр AH содержит байт состояния линии (см. табл. Г.9).

AL — принятый символ.

Функция $03 — запрос о состоянии последовательного порта

Входные данные

AH = 3

DX — номер COM-порта (от 0 до 3).

Выходные данные

AH — состояние линии (см. табл. Г.9).

AL — состояние модема (см. табл. Г.10).

Прерывание $16 — работа с клавиатурой

Это прерывание содержит ряд функций, предназначенных для работы с клавиатурой. Номер функции записывается в регистр AH.

Функция $00 — ожидание следующего нажатия клавиши

Входные данные: AH = 0

Выходные данные

AH — код опроса клавиатуры (см. приложение В).

AL — код ASCII символа (см. приложение Е) или расширенный код клавиши (см. приложение В).

Функция $01 — запрос о состоянии клавиатуры

Входные данные: AH = I

Выходные данные

Флажок ZF — если этот флажок сброшен в "О", то код клавиши считан из буфера клавиатуры.

AH — код опроса клавиатуры (см. приложение В), если ZF = 0.

AL — код ASCII (см. приложение Е) или расширенный код клавиши (см. приложение В), если ZF = 0.

Эта функция только проверяет наличие кода клавиши в буфере клавиатуры, не очищая его.

Функция $02 — запрос о состоянии вспомогательных клавиш

Входные данные: AH = 2

Выходные данные

AL — байт флажков состояния вспомогательных клавиш (табл. Г.11).

Таблица Г.11. Структура байта флажков состояния вспомогательных клавиш

Биты	Описание
0	Если этот флажок установлен в "Г, значит нажата комбинация *правой* клавиши <Shift> и какой-либо символьной клавиши
1	Если этот флажок установлен в "1", значит нажата комбинация *левой* клавиши <Shi£t> и какой-либо символьной клавиши
2	Если этот флажок установлен в "Т, значит нажата комбинация клавиш <Ctrl+Shift>
3	Если этот флажок установлен в "1", значит нажата комбинация клавиш <Alt+Shift>
4	Состояние клавиши <Scroll Lock>
5	Состояние клавиши <Num Lock>
6	Состояние клавиши <Caps Lock>
7	Состояние клавиши <Insert>

Функция $03 — установка периода повторения символа, когда клавиша находится в нажатом состоянии

Входные данные

AH = 3

AL = 5

BH — код времени задержки (0 — 250 мс; 1 — 500 мс; 2 — 750 мс; 3 — 1 с).

BL — код скорости повторения клавиши (0 — 30 повторений в секунду; 1 — 26,7 повторений в секунду; 2 — 24 повторения в секунду; $1F — 2 повторения в секунду).

Выходные данные: нет.

Функция $05 — передача данных о нажатии клавиши в буфер клавиатуры

Входные данные

AH = 5

CH — код опроса клавиатуры, который необходимо передать (см. приложение В).

CL — код ASCII (см. приложение Е) или расширенный код клавиши (см. приложение В).

Выходные данные

AL — 0 — данные успешно переданы; 1 — данные не переданы, так как буфер клавиатуры занят.

Прерывание $17 — работа с принтером

Это прерывание обеспечивает вывод данных на принтер и содержит три функции (номер функции записывается в регистр AH).

Функция $00 — печать одного символа

Входные данные

AH = 0

AL — код ASCII (см. приложение Е).

Выходные данные

AH — байт флажков состояния принтера (табл. Г.12).

Таблица Г.12. Структура байта флажков состояния принтера

Биты	Описание
0	Если этот флажок установлен в "1", то это означает, что превышено допустимого времени ожидания
3	Если этот флажок установлен в "1", то это означает, что произошла ошибка ввода/вывода
4	Если этот флажок установлен в "1", то это означает, что принтер выбран
5	Если этот флажок установлен в "Г, то это означает, что закончилась бумага в лотке принтера
6	Если этот флажок установлен в "1", то это означает, что принтер подключен
7	Если этот флажок установлен в "1", то это означает, что принтер не занят

Функция $01 — инициализация порта принтера

Входные данные

AH = 1

DX - номер порта (0 - LPT1; 1 - LPT2; 2 - LPT3).

Выходные данные

AH — байт флажков состояния принтера (см. табл. Г.12).

Функция $02 — запрос о состоянии принтера

Входные данные

AH = 1

DX - номер порта (0 - LPT1; 1 - LPT2; 2 - LPT3).

Выходные данные

AH — байт флажков состояния принтера (см. табл. Г.12).

Прерывание $1A — запрос и установка текущего времени и даты

Прерывание $1A предоставляет доступ к системному таймеру. При этом, для измерения времени, система BIOS использует единицы, называемые "тиками". Так, одна секунда примерно равна 18 тикам, одна минута — 1092 тикам, один час — 65543 тикам, и один день — 1573040 тикам. Таймер обнуляется автоматически при включении и перезагрузке компьютера. Например, его можно использовать для определения продолжительности выполнения какой-нибудь задачи. Для этого необходимо перед выполнением задачи сбросить таймер в ноль, а по окончании — считать количество тиков.

Функция $00 — запрос о системном времени

Входные данные: AH = 0

Выходные данные

AL — 0, если таймер не переполнялся в течение последних 24 часов после перезагрузки компьютера.

CX, DX — количество тиков (CX — старшая часть, DX — младшая часть значения).

Функция $01 — установка системного времени

Входные данные

AH = 1

CX, DX — требуемое количество тиков.

Выходные данные

Флажок CF — 0, если установка времени была выполнена без ошибок; 1 — в случае неудачной установки.

Прерывания MS-DOS

Прерывание $21 — функции MS-DOS

Функция $00 — завершение программы

Входные данные

AH = 0

CS — сегмент префикса программного сегмента (PSP — Program Segment Prefix) прерывающего процесса.

Выходные данные: нет.

Эта функция выходит в "родительский" процесс и передает управление по вектору прерывания в PSP.

Функция $01 — ввод символа с клавиатуры

Входные данные: AH = 1

Выходные данные

AL — код ASCII (см. приложение Е) символа, извлеченного со стандартного устройства ввода (клавиатуры). Для извлечения расширенных кодов управляющих клавиш (см. приложение В) эту функцию необходимо вызвать дважды.

Функция $02 — вывод символа на экран

Входные данные
AH = 2
DL — код ASCII выводимого символа (см. приложение Е).

Выходные данные: нет.

Функция $03 — прием символа из стандартного асинхронного порта (COM1 или AUX)

Входные данные: AH = 3

Выходные данные
AL — код ASCII полученного символа (см. приложение Е).

Функция $04 — вывод символа в стандартный асинхронный порт (COM1 или AUX)

Входные данные
AH = 4
DL — код ASCII выводимого символа (см. приложение Е).

Выходные данные: нет.

Функция $05 — вывод символа на печать (в порт LPT1)

Входные данные
AH = 5
DL — код ASCII выводимого символа (см. приложение Е).

Выходные данные: нет.

Функция $06 — консольный ввод/вывод

Входные данные
AH = 6
DL — от 0 до $0FE — код ASCII (см. приложение Е) символа, выводимого на стандартное устройство вывода (монитор); $0FF — запрос на прием символа от стандартного устройства ввода.

Выходные данные (если при вызове прерывания регистр DL = $0FF).
Флажок ZF — 0, если символ получен.
AL — код ASCII (см. приложение Е) принятого символа, если ZF = 0.

Функция $07 — ввод с клавиатуры без эха на экран и без проверки нажатия комбинации клавиш <Ctrl+Break>

Входные данные: AH = 7

Выходные данные
AL — код ASCII символа, принятого от стандартного устройства ввода (см. приложение Е).

Функция $08 — ввод с клавиатуры без эха на кран и с проверкой нажатия комбинации клавиш <Ctrl+Break>

Входные данные: AH = 8

Выходные данные
AL — код ASCII символа, принятого от стандартного устройства ввода (см. приложение Е).

Функция $09 — отображение строки

Входные данные

AH = 9

DS:DX — адрес строки, оканчивающейся символом "$".

Выходные данные: нет.

Функция $0A — ввод с клавиатуры с буферизацией

Входные данные

AH = $0A

DS:DX — адрес буфера, в котором первый байт отведен для хранения максимальной длины вводимой строки. Введенная строка заканчивается символом конца строки (#13).

Выходные данные .

DS:DX — адрес буфера, содержащего введенную строку, завершающуюся символом конца строки (#13). Во второй байт буфера будет возвращена фактическая длина введенной строки. Поскольку два байта буфера отведены для хранения вспомогательной информации, длина вводимой строки ограничена 254 символами.

Функция $0B — проверка состояния стандартного устройства ввода

Входные данные: AH = $0B

Выходные данные

AL — $0FF, если со стандартного устройства ввода принят символ; 0, если символ не принят.

Функция $0C — очистка буфера ввода с клавиатуры и запрос на ввод

Входные данные

AH = $0C

AL — номер какой-либо функции прерывания $21, используемой для ввода с клавиатуры ($01, $06, $07, $08 или $0A).

Выходные данные: нет.

Функция $0D — сброс диска

Входные данные: AH = $0D

Выходные данные: нет.

Эта функция очищает все файловые буферы, однако никак не изменяет длину файла в каталоге.

Функция $0E — выбор дисковода по умолчанию

Входные данные

AH = $0E

DL — номер устройства, (0 — A; 1 — B и т.д.).

Выходные данные: AL — общее количество системных устройств.

ПРИМЕЧАНИЕ

Если будет выбран дисковод, не содержащий дискету, возникнет фатальная ошибка (вызов прерывания $24).

Функция $OF — открытие файла через FCB

FCB — это аббревиатура понятия File Control Block (управляющий блок файла). Этот блок представляет собой структуру, занимающую 37 байт памяти, в которой хранится вся информация, необходимая для работы с дисковыми файлами. Содержимое FCB представлено в табл. Г. 13.

Таблица Г.13. Структура блока FCB

Смещение	Длина	Описание
0	1	Идентификатор устройства. Перед открытием файла: 0 — устройство по умолчанию; 1 — A; 2 — B и т.д. После открытия файла: 0 — A; 1 — B и т.д.
1 '	8	Имя файла, дополненное справа пробелами до полной длины поля
9	3	Расширение файла, дополненное справа пробелами до полной длины поля
12($OC)	2	Номер текущего блока
14 ($0E)	2	Размер логической записи
16 ($10)	4	Длина файла
20 ($14)	2	Дата создания/модификации файла: биты 0–4 — день; биты 5–8 — месяц; биты 9–15 — количество лет, прошедших после 1980 года
22 ($16)	8	Зарезервированы
32 ($20)	1	Текущая позиция в текущем блоке (от 0 до 127)
33 ($21)	4	Текущий номер записи относительно начала файла в целом (относительный адрес)

Входные данные
AH = $OF
DS:DX — адрес неоткрытого блока FCB.

Выходные данные
AL — 0, если файл открыт без ошибок (и блок FCB обновлен); $0FF — если была обнаружена ошибка и файл не может быть открыт.

Файл, определенный в неоткрытом блоке FCB, должен уже существовать в текущем каталоге на диске, указанном в первом байте FCB. После открытия файла номер текущего блока в FCB устанавливается равным 0, а размер записи — равный 128.

Функция $10 — закрытие файла через FCB

Входные данные
AH = $10
DS:DX — адрес открытого блока FCB.

Выходные данные
AL — 0, если файл закрыт без ошибок; $OFF — если файл не обнаружен.

Функция $11 — поиск первого файла по шаблону через FCB

Входные данные

AH = $11

DS:DX — адрес закрытого блока FCB (в имени файла допускаются символы шаблона "?").

Выходные данные

AL — 0, если файл с указанным именем найден; $OFF — если файл не обнаружен.

Функция $12 — поиск следующего файла по шаблону через FCB

Входные данные

AH = $12

- DS:DX — адрес закрытого блока FCB (в имени файла допускаются символы шаблона "?").

Выходные данные

AL — 0, если файл с указанным именем найден; $OFF — если файл не обнаружен.

Функция $13 — удаление файлов по шаблону через FCB

Входные данные

AH = $13

DS:DX — адрес закрытого блока FCB (в имени файла допускаются символы шаблона "?").

Выходные данные

AL — 0, если файл с указанным именем был успешно удален; $OFF — если файл не обнаружен или к нему нет доступа, напрмиер, он занят другим процессом.

Функция $14 — последовательное чтение файла через FCB

Входные данные

AH = $14

DS:DX — адрес открытого блока FCB.

Выходные данные

AL — 0, если чтение было выполнено без ошибок; 1, если достигнут конец файла и данные не прочитаны; 2, если будет добавлен размер записи FCB к текущей позиции и после этого в области передачи данных (DTA — Data Transfer Array) произойдет выход за пределы сегмента (чтение не выполняется); 3, если достигнут конец файла и запись была прочитана только частично (остальная часть заполнена нулями).

Эта функция считывает количество байтов, соответствующее длине записи FCB, начиная с адреса, определяемого номером текущего блока и номером текущей записи в области передачи данных DTA. Затем файловый адрес увеличивается.

Функция $15 — последовательная запись в файл через FCB

Входные данные

AH = $15

DS:DX — адрес открытого блока FCB.

Выходные данные

AL — 0, если запись была выполнена без ошибок; 1, если диск переполнен и данные не были записаны; 2, если будет добавлен размер записи FCB к текущей позиции и после этого в области передачи данных (DTA) произойдет выход за пределы сегмента (запись не выполняется).

Эта функция записывает количество байтов, соответствующее длине записи FCB, начиная с адреса, определяемого номером текущего блока и номером текущей записи в области передачи данных DTA. Затем файловый адрес увеличивается.

Функция $16 — создание файла через FCB

Входные данные
AH = $16
DS:DX — адрес закрытого блока FCB.

Выходные данные
AL — 0, если файл создан без ошибок (и блок FCB заполнен); $OFF — если возникла ошибка.

Функция $17 — переименование файла через FCB

Входные данные
AH = $17
DS:DX — адрес блока FCB, имеющего специальный формат.

Выходные данные
AL — 0, если файл успешно переименован; $OFF — если возникла ошибка (файл не найден, файл с новым именем уже существует и т.п.).

В случае использования этой функции новое имя и расширение файла указывается в блоке FCB, начиная с байта со смещением $11 (18-й байт).

Функция $19 — запрос о диске, выбранном MS-DOS по умолчанию

Входные данные: AH = $19

Выходные данные
AL — номер диска, выбранного MS-DOS по умолчанию (0 - A, 1 — B и т.д.).

В случае использования этой функции новое имя и расширение файла указывается в блоке FCB, начиная с байта со смещением $11 (18-й байт).

Функция $1A — установка области передачи данных (DTA)

Входные данные
AH = $1A
DS:DX — адрес области передачи данных (DTA), которая используется при последовательной записи в файл и последовательном чтении из файла, а также при поиске файлов при помощи функций $11, $12, $4E и $4F.

Выходные данные: нет.

Функция $1B — получение информации о текущем диске

Входные данные: AH = $1B

Выходные данные
DS:BX — адрес байта идентификации таблицы размещения файлов (FAT — File Allocation Table).
DX — общее количество кластеров на диске.
AL — количество секторов в одном кластере.
CX — количество байтов в одном секторе.

Эту функцию можно использовать для вычисления общего объема диска по формуле DX * AL * CX.

Функция $1C — получение информации о любом диске

Входные данные
AH = $1C
DL — номер диска (0 — выбранный по умолчанию; 1 — A, и т.д.).

Выходные данные
DS:BX — адрес байта идентификации таблицы размещения файлов (FAT).
DX — общее количество кластеров на диске.
AL — количество секторов в одном кластере.
CX — количество байтов в одном секторе.

Функция $1F — получение блока параметров текущего диска

Входные данные: AH = $1F

Выходные данные
AL — 0, если функция выполнена без ошибок, в противном случае — $OFF.
DS:BX — адрес блока параметров диска. Структура этого блока представлена в табл. Г.14.

Таблица Г.14. Структура блока параметров диска

Смещение	Длина	Описание
0	1	Номер диска: 0 — A; 1 — B и т.д.
1	1	Номер типа устройства
2	2	Количество байт в секторе
4	1	Количество секторов в кластере минус один (максимальный номер сектора)
5	1	Смещение "кластер-сектор" (степень, в которую необходимо возвести число 2, чтобы получить количество секторов в кластере)
6	2	Номер сектора первой копии таблицы FAT
8	1	Количество таблиц FAT
9	2	Количество элементов, допустимых для размещения в корневом каталоге
11 ($0B)	2	Номер сектора второго кластера (первого кластера данных)
13 ($0D)	2	Общее количество кластеров плюс один (наибольший номер кластера)
15 ($0F)	1	Количество секторов, требуемых для таблицы FAT
16 ($10)	2	Номер сектора начала корневого каталога
18 ($12)	4	Адрес заголовка устройства для данного диска
22 ($16)	1	Байт описания типа носителя. Некоторые значения: $FO — дискета 3,5" (1,44 M), 2 стороны, 18 секторов; дискета 3,5" (2,88 M), 2 стороны, 36 секторов; $F8 — жесткий диск; $F9 — дискета 3,5" (720 K); 2 стороны, 9 секторов; $FB — дискета 3,5" (640 K); 2 стороны; 8 секторов

Окончание таблицы Г.14

Смещение	Длина	Описание
23 ($17)	1	Значение $00 в этом байте означает, что был получен доступ к данному блоку параметров диска. В противном случае в этом поле будет храниться значение $FF
24 ($18)	4	Адрес следующего блока параметров диска
28 ($1C)	2	Последний расположенный кластер
30 ($1E)	2	Количество свободных кластеров

Функция $21 — чтение прямым доступом из файла через FCB

Входные данные

AH = $21

DS:DX — адрес открытого блока FCB.

Выходные данные

AL — 0, если чтение было выполнено без ошибок (и область DTA заполнена данными); 1, если достигнут конец файла и данные не прочитаны; 2, если после добавления размера записи FCB к текущей позиции в области передачи данных (DTA) произойдет выход за пределы сегмента (чтение не выполняется); 3, если достигнут конец файла и запись была прочитана только частично (остальная часть заполнена нулями).

Эта функция считывает количество байтов, соответствующее длине записи FCB, начиная с относительного адреса, указанного в последнем поле блока FCB.

Функция $22 — запись прямым доступом в файл через FCB

Входные данные

AH = $22

DS:DX — адрес открытого блока FCB.

Выходные данные

AL — 0, если запись была выполнена без ошибок; 1, если диск переполнен и данные не были записаны; 2, если после добавления размера записи FCB к текущей позиции в области передачи данных (DTA) произойдет выход за пределы сегмента (запись не выполняется).

Эта функция записывает количество байтов, соответствующее длине записи FCB, начиная с относительного адреса, указанного в последнем поле блока FCB.

Функция $23 — определение размера файла через FCB

Входные данные

AH = $23

DS:DX — адрес открытого блока FCB.

Выходные данные

AL — 0, если файл найден; $OFF — если указанного файла в текущем каталоге нет.

После вызова этой функции в последнем поле блока FCB будет храниться размер файла, измеряемый в записях. Размер этих записей также указывается в FCB.

Функция $24 — установка в FCB номера записи для прямого доступа к файлу

Входные данные

AH = $24

DS:DX — адрес открытого блока FCB.

Выходные данные: нет.

Эта функция записывает в последнее поле блока FCB файловый адрес, соответствующий номеру текущего блока и номеру текущей записи.

Функция $27 — чтение из файла блока записей прямым доступом

Входные данные
AH = $27
CX — количество считываемых записей.
DS:DX — адрес открытого блока FCB.

Выходные данные
AL — 0, если чтение было выполнено без ошибок (и область DTA заполнена данными); 1, если достигнут конец файла и данные не прочитаны; 2, если после добавления к текущей позиции в области передачи данных (DTA) размера записи FCB, умноженного на содержимое регистра CX, произойдет выход за пределы сегмента (чтение не выполняется); 3, если достигнут конец файла и запись была прочитана только частично (остальная часть заполнена нулями).
CX — фактическое количество прочитанных записей.

Эта функция считывает количество байтов, соответствующее длине записи FCB, умноженной на содержимое регистра CX, начиная с относительного адреса, указанного в последнем поле блока FCB.

Функция $28 — запись в файл блока записей прямым доступом

Входные данные
AH = $28
CX — количество считываемых записей.
DS:DX — адрес открытого блока FCB.

Выходные данные
AL — 0, если запись была выполнена без ошибок; 1, если диск переполнен и данные не были записаны; 2, если после добавления к текущей позиции в области передачи данных (DTA) размера записи FCB, умноженной на содержимое регистра CX, произойдет выход за пределы сегмента (запись не выполняется).

Эта функция записывает количество байтов, соответствующее длине записи FCB, умноженной на содержимое регистра CX, начиная с относительного адреса, указанного в последнем поле блока FCB.

Функция $29 — создание закрытого блока FCB на основании строки с именем файла

Входные данные
AH = $29
AL — байт флажков, определяющих выбор операции анализа имени файла. Используются только первые четыре бита. Значение каждого из флажков, установленных в "1": бит 0 — не учитывать лидирующие разделители; бит 1 — использовать по умолчанию диск, указанный в блоке FCB; бит 2 — использовать по умолчанию имя файла, указанное в блоке FCB; бит 3 — использовать по умолчанию расширение, указанное в блоке FCB.
DS:SI — адрес исходной анализируемой текстовой строки.
ES:DI — адрес буфера, в который будет записан результирующий закрытый блок FCB.

Выходные данные

AL — 0, если в полученном блоке FCB не используется символов шаблона; 1, если полученный блок FCB содержит символы шаблона; $OFF — если в строке, определяющей имя файла, указан неверный идентификатор диска.

DS:DI — указывает на символ, расположенный сразу же после имени файла.

ES:DI — указывает на закрытый блок FCB.

Эта функция не анализирует имена файлов, для которых указан полный путь. Строка для анализа может иметь только следующий вид: "D:FileName.ext".

Функция $2A — получение даты

Входные данные: AH = $2A

Выходные данные

AL — день недели (0 — воскресенье, 1 — понедельник и т.д.).
CX - год (от 1980 до 2099).
DH — месяц.
DL — день месяца.

Функция $2B — установка даты

Входные данные

AH = $2B
CX - год (от 1980 до 2099).
DH — месяц.
DL — день месяца.

Выходные данные

AL — 0, если дата корректна; в противном случае — $0FF.

Функция $2C — получение времени

Входные данные: AH = $2C

Выходные данные

CH — часы (от 0 до 23).
CL — минуты (от 0 до 59).
DH — секунды (от 0 до 59).
DL — сотые доли секунды (от 0 до 99).

Функция $2D — установка времени

Входные данные

AH = $2D
CH — часы (от 0 до 23).
CL — минуты (от 0 до 59).
DH — секунды (от 0 до 59).
DL — сотые доли секунды (от 0 до 99).

Выходные данные

AL — 0, если время корректно; в противном случае — $0FF.

Функция $2E — установка/отмена проверки при записи на диск

Входные данные: AH = $2E

Выходные данные: нет.

Эта функция устанавливает или отменяет проверку при записи данных в каждый сектор, которую выполняет MS-DOS при помощи кодов CRC *(44* см. примечание в разделе, посвященном функции $00 прерывания $13). Эта проверка замедляет операции записи на диск, однако обеспечивает максимальную степень целостности данных.

Функция $2F — получение адреса области передачи данных DTA

Входные данные: AH = $2F

Выходные данные: ES:BX — адрес DTA.

Функция $30 — получение версии MS-DOS

Входные данные: AH = $30

Выходные данные
AL — основная часть версии.
AH — второстепенная часть версии.

Функция $31 — завершение программы, после которого она остается резидентной в памяти

Входные данные
AH = $31
AL — код выхода, доступный для функции $4F.
DX — объем памяти, отводимый для резидентной программы (выраженный в 16-байтовых параграфах).

Выходные данные: нет.

Функция $32 — получение блока параметров диска

Входные данные
AH = $32
DL — номер диска (0 — выбранный по умолчанию; 1 — A; 2 — B и т.д.).

Выходные данные
AL — 0, если номер диска в DL указан верно; в противном случае — $OFF.
DS:BX — адрес блока параметров указанного диска (см. табл. Г.14).

Функция $36 — получение размера свободно места на диске

Входные данные
AH = $36
DL — номер диска (0 — выбранный по умолчанию; 1 — A; 2 — B и т.д.).

Выходные данные
AX — количество секторов в кластере, если в DL был указан корректный номер диска, в противном случае — $FFFF.
BX — количество доступных кластеров.
CX — количество байт в секторе (обычно 512).
DX — общее количество кластеров на диске.

Таким образом, в случае успешного выполнения этой функции, для определения свободного места на диске в байтах используется формула AX * CX * BX, а для определения общего объема диска — формула AX * CX * DX.

Функция $39 — создание каталога

Входные данные
AH = $39
DS:DX — адрес строки с именем каталога, которая завершается символом #0.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $3A — удаление пустого каталога

Входные данные
AH = $3A
DS:DX — адрес строки с именем каталога, которая завершается символом #0.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $3B — установка каталога MS-DOS по умолчанию

Входные данные
AH = $3B
DS:DX — адрес строки с именем каталога, которая завершается символом #0.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $3C — создание файла через дескриптор

Входные данные
AH = $3C
DS:DX — адрес строки с именем файла, которая завершается символом #0.
CX — флажки атрибутов файла. Установка одного из флажков в "1" соответствует следующим атрибутам: бит 0 — только для чтения; бит 1 — скрытый; бит 2 — системный; бит 3 — метка тома; бит 4 — подкаталог; бит 5 — копия файла не создавалась.

Выходные данные
AX — код ошибки, если флажок CF установлен в "1", в противном случае — дескриптор файла.

ПРИМЕЧАНИЕ

Дескриптор файла — 16-битовое значение, которые используется для уникальной идентификации файлов при операциях чтения/записи, поиска данных в файле или закрытия файла.

Функция $3D — открытие файла через дескриптор

Входные данные
AH = $3D
AL — режим доступа (0 — чтение; 1 — запись; 2 — чтение/запись).
DS:DX — адрес строки с именем файла, которая завершается символом #0.

Выходные данные
AX — код ошибки, если флажок CF установлен в "1", в противном случае — дескриптор файла.

Функция $3E — закрытие файла через дескриптор

Входные данные
AH = $3E
BX — дескриптор файла.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $3F — чтение из файла через дескриптор

Входные данные
AH = $3F
BX — дескриптор файла.
CX — количество считываемых байтов.
DS:DX — адрес буфера для приема данных.

Выходные данные
AX — код ошибки, если флажок CF установлен в "1", в противном случае — количество фактически считанных байтов.

Функция $40 — запись в файл через дескриптор

Входные данные
AH = $40
BX — дескриптор файла.
CX — количество записанных байтов (0 равнозначен обрезке файла, начиная с текущей файловой позиции).
DS:DX — адрес буфера, содержащего данные для записи.

Выходные данные
AX — код ошибки, если флажок CF установлен в "1", в противном случае — количество фактически записанных байтов.

Функция $41 — удаление файла

Входные данные
AH = $41
DS:DX — адрес строки, завершающейся символом #0, которая содержит имя удаляемого файла.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $42 — установка файлового указателя для последовательного доступа

Входные данные
AH = $42
AL — номер подфункции: 0 — перемещение указателя в начало файла с последующим смещением **CX:DX;** 1 — перемещение указателя файла от текущей позиции на смещение **CX:DX;** 2 — перемещение указателя в конец файла с последующим смещением CX:DX.
BX — дескриптор файла.
CX:DX — смещение файлового указателя в байтах (CX*65536 + DX).

Выходные данные
AX — код ошибки, если флажок CF установлен в "1".
DX:AX — новая позиция файлового указателя.

Функция $43 — установка и запрос атрибутов файла

Эта функция имеет две подфункции: первая выполняет запрос, а вторая — установку атрибутов файла.

Подфункция $00 — запрос атрибутов файла

Входные данные
AH = $43

AL = 0

DS:DX — адрес строки, заканчивающейся символом #0, в которой хранится имя файла.

Выходные данные

AX — код ошибки, если флажок CF установлен в "1".

CX — атрибуты указанного файла (если нет ошибки).

◄◄ Атрибуты рассматриваются в разделе, посвященном функции $3C прерывания $21.

Подфункция $01 — установка атрибутов файла

Входные данные

AH = $43

AL = 1

CX — атрибуты, назначаемые файлу (◄◄ биты атрибутов рассматриваются в разделе, посвященном функции $3C прерывания $21).

DS:DX — адрес строки, заканчивающейся символом #0, в которой хранится имя файла.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $47 — получение каталога, выбранного по умолчанию

Входные данные

AH = $47

DL — номер диска (0 — выбранный по умолчанию; 1 — A, и т.д.).

DS:SI — адрес локального буфера для хранения пути (64 байта).

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $4B — загрузка/выполнение программы

Эта функция имеет четыре подфункции, номер которых записывается в регистр AL.

Подфункция $00 — загрузка и выполнение программы

Входные данные

AH = $4B

AL — 0

DS:DX — адрес строки, завершающейся символом #0, в которой хранится имя выполняемой программы.

ES:BX — адрес структуры, хранящей параметры выполнения. Описание полей этой структуры представлено в табл. Г. 15.

Таблица Г.15. Описание структуры, хранящей параметры выполнения программ

Смещение	Длина	Описание
0	2	Сегмент для среды окружения выполняемой программы
2	4	Адрес текста командной строки, который будет размещен в префиксе программного сегмента (PSP) по адресу $0080
6	4	Адрес блока FCB, который будет размещен в префиксе программного сегмента по адресу $005C (первый командный параметр)
10 ($0A)	4	Адрес блока FCB, который будет размещен в префиксе программного сегмента по адресу $006C (второй командный параметр)

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Подфункция $01 — загрузка программы без выполнения

Входные данные

AH = $4B

AL — 1

DS:DX — адрес строки, завершающейся символом #0, в которой хранится имя выполняемой программы.

ES:BX — адрес структуры, хранящей параметры загрузки. Описание полей этой структуры представлено в табл. Г. 16.

Таблица Г.16. Описание структуры, хранящей параметры загрузки программ

Смещение	Длина	Описание
0	2	Сегмент для среды окружения выполняемой программы.
2	4	Адрес текста командной строки, который будет размещен в префиксе программного сегмента (PSP) по адресу $0080
6	4	Адрес блока FCB, который будет размещен в префиксе программного сегмента по адресу $005C (первый командный параметр)
10($OA)	4	Адрес блока FCB, который будет размещен в префиксе программного сегмента по адресу $006C (второй командный параметр)
14($OE)	4	Двойное слово, принимающее адрес входа в программу
18 ($12)	4	Двойное слово, принимающее адрес указателя стека программы

Выходные данные

AX — код ошибки, если флажок CF установлен в "1".

ES:BX — некоторая информация, которая возвращается в структуру и была использована для хранения параметров загрузки.

Подфункция $05 — загрузка и выполнение программы

Входные данные

AH = $4B

AL — 5

DS:DX — адрес структуры, хранящей параметры для выполнения загруженной программы системой MS-DOS. Описание полей этой структуры представлено в табл. Г.17.

Таблица Г.17. Описание структуры, хранящей параметры для выполнения загруженной программы системой MS-DOS

Смещение	Длина	Описание
0	2	Зарезервированы
2	2	0 — COM; 1 — EXE
4	2	Адрес строки, завершающейся символом #0, которая хранит имя программы
6	2	Сегмент префикса программного сегмента программы (PSP)
8	2	32-битный адрес точки входа в программу
12 ($0A)	2	32-битный размер программы, включая префикс программного сегмента

Выходные данные: нет.

Функция $4C — завершение программы

Входные данные
AH = $4C
AL — код завершения для возврата в вызывающий процесс.

Выходные данные: нет.

Функция $4D — получение кода завершения программы

Входные данные: AH = $4D

Выходные данные
AL — код завершения последнего прерываемого процесса.
AH — метод выхода:
0 — нормальное завершение;
1 — завершение нажатием комбинации клавиш <Ctrl+Break>;
2 — завершение через возникновение фатальной ошибки;
3 — завершение через функцию $31 прерывания $21.

Функция $4E — поиск первого файла по шаблону

Входные данные
AH = $4E
CX — атрибуты, которые должны совпадать с атрибутами искомого файла (◄ биты атрибутов рассматриваются в разделе, посвященном функции $3C прерывания $21).
DS:DX — адрес строки, завершающейся символом #0, которая хранит имя искомого файла (допускаются символы шаблона).

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Если ошибки не обнаружено, то область передачи данных DTA заполняется данными о файле: байты 0—21 — зарезервированы; байт 22 — атрибуты найденного файла; байты 22, 23 — время создания/модификации файла; байты 24,25 — дата создания/модификации файла; байты 26—29 — размер файла в байтах; байты 30-42 — имя и расширение найденного файла.

Функция $4F — поиск следующего файла по шаблону

Входные данные
AH = $4F
DS:DX — адрес данных, возвращенных функцией $4E.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Если ошибки не обнаружено, то область передачи данных DTA заполняется данными о файле (см. предыдущий подраздел, посвященный функции $4E).

Функция $54 — получение состояния проверки записи на диск

Входные данные: AH = $54

Выходные данные
AL — текущее состояние проверки записи на диск при помощи кодов CRC (◄ см. примечание в разделе, посвященном функции $00 прерывания $13): 0 — проверка отключена; 1 — проверка активна.

Функция $56 — переименование/перемещение файла

Входные данные

AH = $56

DS:DX — адрес строки, завершающейся символом #0, с именем существующего файла.

ES:DI — адрес строки, завершающейся символом #0, с именем нового файла.

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $57 — получение/установка даты и времени изменения файла

Эта функция имеет две подфункции: первая — для получения, а вторая — для установки даты и времени изменения файла.

Подфункция $00 — получение даты и времени изменения файла

Входные данные

AH = $57

AL = 0

BX — дескриптор открытого файла (◄◄ см. примечание в разделе, посвященном функции $3C прерывания $21).

Выходные данные

AX — код ошибки, если флажок CF установлен в "1".

CX — время изменения файла (биты 0-4 — секунды, выраженные в единицах по 2 секунды каждая; биты 5–10 — минуты; биты 11–15 — часы).

DX — дата изменения файла (биты 0–4 — день; биты 5-8 — месяц; биты 9-15 — количество лет, прошедших после 1980 года).

Подфункция $01 — установка даты и времени изменения файла

Входные данные

AH = $57

AL = 1

BX — дескриптор открытого файла.

CX — время изменения файла (биты 0-4 — двухсекундные единицы; биты 5–10 — минуты; биты 11–15 — часы).

DX — дата изменения файла (биты 0–4 — день; биты 5–8 — месяц; биты 9–15 — количество лет, прошедших после 1980 года).

Выходные данные: AX — код ошибки, если флажок CF установлен в "1".

Функция $62 — получение префикса программного сегмента

Входные данные: AH = $62

Выходные данные

BX — адрес префикса программного сегмента (PSP — Program Segment Prefix) выполняемого в данный момент процесса.

Прерывание $33 — работа с мышью

Это прерывание содержит ряд функций, номера которых при вызове записывается в регистр AL. Рассмотрим некоторые из этих функций.

Функция $00 — запрос о состоянии установленной мыши

Входные данные: AL = $00

Выходные данные
AX — $0000 — мышь не установлена; $FFFF — мышь установлена.
BX — количество кнопок мыши.

Функция $01 — отобразить указатель мыши

Входные данные: AL = $01

Выходные данные: нет.

Функция $02 — скрыть указатель мыши

Входные данные: AL = $02

Выходные данные: нет.

Функция $03 — получить информацию о положении указателя и состоянии кнопок мыши

Входные данные: AL = $03

Выходные данные
BX — состояние кнопок. Бит 0 — значению "1" соответствует нажатая левая кнопка мыши; бит 1 — значению "1" соответствует нажатая правая кнопка мыши; бит 2 — значению "1" соответствует нажатая средняя кнопка мыши.
CX — координата указателя мыши по *горизонтали.*
DX — координата указателя мыши по *вертикали,*

Функция $04 — установка координат указателя мыши

Входные данные
AL = $04
CX — координата указателя мыши по *горизонтали.*
DX — координата указателя мыши по *вертикали.*

Выходные данные: нет.

Функция $05 — получение количества нажатий кнопок мыши

Входные данные
AL = $05
BX — номер кнопки (0 — левая; 1 — правая; 2 — средняя).

Выходные данные
AX — состояние кнопок. Бит 0 — значению "1" соответствует нажатая левая кнопка мыши; бит 1 — значению "1" соответствует нажатая правая кнопка мыши; бит 2 — значению "1" соответствует нажатая средняя кнопка мыши.
BX — количество нажатий кнопки мыши со времени последнего вызова этой функции.
CX — координата указателя по горизонтали во время последнего нажатия любой кнопки мыши.
DX — координата указателя по вертикали во время последнего нажатия любой кнопки мыши.

Функция $06 — получение количества отпусканий кнопки мыши

Входные данные
AL = $06
BX — номер кнопки (0 — левая; 1 — правая; 2 — средняя).

Выходные данные

AX — состояние кнопок. Бит 0 — значению "1" соответствует отпущенная левая кнопка мыши; бит 1 — значению "1" соответствует нажатая правая кнопка мыши; бит 2 — значению "1" соответствует нажатая средняя кнопка мыши.

BX — количество отпусканий кнопки мыши со времени последнего вызова этой функции.

CX — координата указателя по *горизонтали* в тот момент, когда любая кнопка была отпущена в последний раз.

DX — координата указателя по *вертикали* в тот момент, когда любая кнопка была отпущена в последний раз.

Функция $07 — установка допустимого диапазона перемещения указателя мыши по горизонтали

Входные данные

AL = $07

CX — *минимальная* допустимая координата по горизонтали (в пикселях).

DX — *максимальная* допустимая координата по горизонтали (в пикселях).

Выходные данные: нет.

Функция $08 — установка допустимого диапазона перемещения указателя мыши по вертикали

Входные данные

AL = $08

CX — *минимальная* допустимая координата по вертикали (в пикселях).

DX — *максимальная* допустимая координата по вертикали (в пикселях).

Выходные данные: нет.

Функция $09 — установка формы графического указателя мыши

Входные данные

AL = $09

BX — координата "острия" точки указателя по *горизонтали* относительно левого верхнего угла указателя.

CX — координата "острия" точки указателя по *вертикали* относительно левого верхнего угла указателя.

ES:DX — адрес области памяти размером 64 байта, хранящей данные о форме указателя.

Выходные данные: нет.

Функция $10 — получение расстояния, пройденного указателем мыши

Входные данные: AL = $10

Выходные данные

CX — перемещение по горизонтали со времени последнего вызова этой функции (между двумя вызовами этой функции может быть выполнено более одного перемещения мыши).

DX — перемещение по вертикали со времени последнего вызова этой функции.

Перемещения измеряются в единицах под названием "мики", равных 1/200 дюйма.

Функция $OC — назначение обработчика событий мыши

Входные данные

AL = $OC

CX — маска событий, передаваемых обработчику: бит 0 — перемещение указателя; бит 1 — нажата левая кнопка; бит 2 — отпущена левая кнопка; бит 3 — нажата правая кнопка; бит 4 — отпущена правая кнопка; бит 5 — нажата средняя кнопка; бит 6 — отпущена средняя кнопка. Таким образом, для обработки всех событий в регистр CX должно быть записано значение $007F.

ES:DX — адрес обработчика событий.

Выходные данные: нет.

Функция $OF — установить скорость указателя мыши

Входные данные

AL = $OF

CX — требуемая скорость по горизонтали, выражаемая в "миках" на 8 пикселей (1 мик = 1/200 дюйма).

DX — требуемая скорость по вертикали, выражаемая в "миках" на 8 пикселей (1 мик = 1/200 дюйма).

Выходные данные: нет.

Функция $1F — отключение драйвера мыши

Входные данные: AL = $1F

Выходные данные

AX — состояние: $001F — отключение драйвера выполнено успешно; $FFFF — драйвер не отключен.

Функция $20 — активизация драйвера мыши

Входные данные: AL = $20

Выходные данные: нет.

Эта функция устанавливает соединение между оборудованием мыши и драйвером.

Функция $21 — переустановка драйвера мыши

Входные данные: AL = $21

Выходные данные

AX — состояние: $0021 — переустановка драйвера выполнена успешно; $FFFF — драйвер не переустановлен.

BX — количество кнопок.

Функция $24 — получение информации о типе мыши и версии драйвера

Входные данные: AL = $24

Выходные данные

BH — основная часть номера версии драйвера.

BL — второстепенная часть номера версии драйвера.

CH — тип мыши по способу подключения: 1 — к шине; 2 — к последовательному порту; 3 — к внутреннему порту; 4 — к порту PS/2; 5 — мышь HP.

CL — номер запроса на прерывание.

Приложение Д

Команды языка Ассемблер для процессоров 8x86

В специальной литературе, посвященной языку Ассемблер, команды называют еще инструкциями.

AAA — ASCII-коррекция после сложения

Преобразует сумму в регистре AL (после выполнения команды ADD) в два ASCII-байта. Если правые четыре бита (младшие) регистра AL содержат значение больше 9 или флаг AF установлен в 1, то команда AAA выполняет следующие действия: прибавляет 1 к регистру AH, прибавляет *6* к регистру AL и *устанавливает* флаги AF и CF, то есть делает их равными 1. Команда всегда *очищает* четыре левых бита в регистре AL, то есть делает их равными 0.

Синтаксис: AAA операндов нет

AAD — ASCII-коррекция перед делением

Корректирует ASCII-величины для деления. Команда AAD используется перед делением неупакованных десятичных чисел в регистре AX (удаляет тройки ASCII-кода). Эта команда корректирует делимое в двоичное значение в регистре AX для последующего двоичного деления. Затем умножает содержимое регистра AH на 10, прибавляет результат к содержимому регистра AL и очищает AH (AH=0).

Синтаксис: AAD операндов нет

AAM — ASCII-коррекция после умножения

Команда AAM используется для коррекции результата умножения двух неупакованных десятичных чисел в регистре AL. Команда делит содержимое регистра AL на 10, записывает частное в регистр AH, а остаток в регистр AL.

Синтаксис: AAM операндов нет

AAS — ASCII-коррекция после вычитания

Корректирует разность двух ASCII-байтов в регистре AL. Если правые четыре бита (младшие) имеют значение больше 9 или флаг CF установлен в 1, то команда AAS вычитает 6 из регистра AL и 1 из регистра AH и устанавливает флаги AF и CF в 1. Команда всегда очищает левые четыре бита (старшие) в регистре AL (устанавливает их в 0).

Синтаксис: AAS операндов нет

ADC — сложение с переносом

Обычно используется при сложении многословных величин для учета бита переполнения в последующих фазах операции. Если флаг CF установлен в 1, то команда ADC сначала прибавляет 1 к первому операнду. Команда всегда прибавляет

ADC сначала прибавляет 1 к первому операнду. Команда всегда прибавляет операнд 2 к операнду 1, аналогично команде ADD.

Синтаксис: ADC регистр/память, регистр/память/непосредственное значение

ADD — сложение двоичных чисел

Прибавляет один байт или одно слово в памяти, регистре или непосредственно из операнда к содержимому регистра, или прибавляет один байт или слово из регистра или непосредственно из операнда к значениям из памяти.

Синтаксис: ADC регистр/память, регистр/память/непосредственное значение

AND — логическое "И"

Команда выполняет поразрядную конъюнкцию (логическое "И") битов двух операндов. Операнды могут быть величины длинной в байт, слово или двойное слово, находящимися в регистре или памяти. Второй операнд может содержать непосредственные данные. Команда AND проверяет два операнда поразрядно. Если два проверяемых бита равны 1, то в первом операнде устанавливается единичное значение бита, в других случаях — нулевое.

Синтаксис: ADC регистр/память, регистр/память/непосредственное значение

CALL — вызов процедуры

Выполняет короткий (типа near) или длинный (типа far) вызов процедуры для связи подпрограмм. Для возврата из процедуры используется команда RETN — для процедуры, объявленной как NEAR и RETF — для FAR.

В случае вызова процедуры NEAR команда CALL уменьшает содержимое регистра SP на 2 и сохраняет в стеке адрес следующей команды (из IP), а затем устанавливает в регистре IP относительный адрес процедуры. Впоследствии команда RETN использует значение в стеке для возврата.

В случае вызова процедуры FAR команда CALL выполняет межсегментный вызов процедуры, сначала уменьшая SP, затем сохраняет в стеке адрес из регистра CS, помещает в регистр CS адрес сегмента, в котором находится вызываемая процедура, затем сохраняет в стеке значение из регистра IP, а в регистре IP сохраняет смещение первой инструкции вызывающей процедуры в ее сегменте.

Синтаксис: CALL регистр/память

CBW — преобразование байта в слово

Расширяет однобайтовое арифметическое значение в регистре AL до размеров слова. Команда CBW размножает знаковый бит (7) в регистре AL по всем битам регистра AH.

Синтаксис: CBW операндов нет

CLC — сброс флага переноса CF

Сбрасывает значение флага переноса (CF=0). В этом случае, например, команда ADC не прибавляет единичный бит.

Синтаксис: CLC операндов нет

CLD — сброс флага направления DF

Сбрасывает значение флага направления (DF=0). В результате такие строковые операции, как CMPS или MOVS, обрабатывают данные слева направо.

Синтаксис: CLD операндов нет

CMC — переключение флага переноса CF

Инвертирует флаг CF, то есть преобразует нулевое значение флага CF в единичное и наоборот.

Синтаксис: CMC операндов нет

CMP — сравнение

Сравнивает содержимое двух полей данных. Фактически команда CMP вычитает второй операнд из первого, но содержимое полей не изменяет, то есть не сохраняет результат. Однако результат влияет на состояние флагов AF, CF, OF, PF, SF и ZF. Операнды должны иметь одинаковую длину: байт или слово. Команда CMP может сравнивать содержимое регистра, памяти или непосредственное значение с содержимым регистра; или содержимое регистра или непосредственное значение с содержимым памяти.

Синтаксис: CMP регистр/память, регистр/память/непосредственное значение

CMPS (CMPSB/CMPSW/CMPSD) - сравнение строк (по байтам/словам/двойным словам)

Сравнивают строки любой длины в памяти. Команда CMPSB сравнивает память по байтам, команда CMPSW — по словам, а команда CMPSD— по двойным словам (для процессоров 80386 и более поздних). Первый операнд этих команд адресуется регистровой парой DS:SI, а второй — регистровой парой ES:DI.

Этим командам обычно предшествует префикс REPn, а количество сравнений указывается в регистре CX. Префикс REPNE вызывет завершение выполнения команды при обнаружении первого совпадения, префикс REPE — первого несовпадения. Если регист CX содержит ноль, выполнение команды завершается при любом префиксе.

Если флаг DF установлен в 0, то сравнение происходит слева направо, а регистры SI и DI при этом увеличиваются после каждого сравнения: на 1 — для байта, на 2 — для слова и на 4 — для двойного слова. Если флаг DF установлен в 1, то сравнение происходит справа налево, а регистры SI и DI при этом уменьшаются после каждого сравнения, аналогично увеличению.

Синтаксис: [REPn] CMPS/CMPSB/CMPSW операндов нет

CWD — преобразование слова в двойное слово

Расширяет арифметическое значение в регистре AX до размеров двойного слова в регистровой паре DX:AX, копируя при этом знаковый бит (15 бит) регистра AX в биты регистра DX. Обычно используется для получения 32-битового делимого.

Синтаксис: CWD операндов нет

DEC — декремент

Вычитает 1 из байта, слова или двойного слова в регистре или в памяти. Для выполнения обратной операции применяется команда INC.

Синтаксис: DEC регистр/память

DIV — деление

Выполняет деление беззнакового делимого на беззнаковый делитель. Левый (старший) единичный бит рассматривается, как бит данных, а не как минус для отрицательных чисел.

Для 8-битового деления делимое должно находиться в регистре AX, а 8-битовый делитель может находиться в регистре или в памяти, например DIV BH. Частное от деления сохраняется в регистре AL, а остаток — в регистре AH.

Для 16-битового деления делимое должно находиться в регистровой паре DX:AX, а 16-битовый делитель может находиться в регистре или в памяти, например DIV CX. Частное от деления сохраняется в регистре AX, а остаток — в регистре DX.

Для 32-битового деления делимое должно находиться в регистровой паре EDX:EAX, а 32-битовый делитель может находиться в регистре или в памяти, например DIV ECX. Частное от деления сохраняется в регистре EAX, а остаток — в регистре EDX.

Синтаксис: DIV регистр/память

ESC — переключение на сопроцессор

Обеспечивает использование сопроцессора для выполнения специальных операций. Для выполнения арифметических операций над числами с плавающей запятой используются сопроцессоры 8087 или 80287. Команда ESC передает в сопроцессор инструкцию и операнд для выполнения необходимой операции.

Синтаксис: ESC непосредственное значение, регистр/память

HLT — останов микропроцессора

Приводит процессор в состояние останова, в котором происходит ожидание прерывания. При завершении команды HLT регистры CS:IP указывают на следующую команду. При возникновении прерывания процессор записывает в стек регистры CS и IP и выполняет подпрограмму обработки прерывания. При возврате из подпрограммы команда IRET восстанавливает регистры CS и IP из стека и управление передается на команду, следующую за командой HLT. Для разрешения аппаратных прерываний, перед остановом микропроцессора, необходимо выполнить команду STI для установки флага IF в 1.

Синтаксис: HLT операндов нет

IDIV — знаковое (целочисленное) деление

Выполняет деление знакового (целочисленного) делимого на знаковый (целочисленный) делитель. Левый (старший) единичный бит рассматривается как знак минус для отрицательных чисел.

Для 8-битового деления делимое должно находиться в регистре AX, а 8-битовый делитель может находиться в регистре или в памяти, например DIV BH. Частное от деления сохраняется в регистре AL, а остаток — в регистре AH.

Для 16-битового деления делимое должно находиться в регистровой паре DX:AX, а 16-битовый делитель может находиться в регистре или в памяти, например DIV CX. Частное от деления сохраняется в регистре AX, а остаток — в регистре DX.

Для 32-битового деления делимое должно находиться в регистровой паре EDX:EAX, а 32-битовый делитель может находиться в регистре или в памяти, например DIV ECX. Частное от деления сохраняется в регистре EAX, а остаток — в регистре EDX.

Синтаксис: IDIV регистр/память

IMUL — знаковое (целочисленное) умножение

Выполняет умножение на знаковый (целочисленный) множитель. Левый (старший) единичный бит рассматривается как знак минус для отрицательных чисел.

Для 8-битового умножения множимое должно находиться в регистре AL, а множитель может находиться в регистре или в памяти, например IMUL BL. Произведение сохраняется в регистре AX.

Для 16-битового умножения множимое должно находиться в регистре AX, а множитель может находиться в регистре или в памяти, например IMUL BX. Произведение сохраняется в регистровой паре DX:AX.

Для 32-битового умножения множимое должно находиться в регистровой паре EDX:EAX, а множитель может находиться в регистре или в памяти, например, DIV ECX. Произведение сохраняется в регистровой паре EDX:EAX.

Синтаксис: IMUL регистр/память

Для процессоров 80286 (и более поздних) множимое может находиться в регистре (операнд 1), а множителем может быть непосредственное значение (операнд 2). Произведение сохраняется в регистре, в котором находилось множимое. Например, IMUL BX, 16.

Еще один метод умножения для таких процессоров: множимое может находиться в регистре или в памяти (операнд 2 — имя регистра или адрес памяти), множителем может быть непосредственное значение (операнд 3), а произведение сохраняется в регистре, указанном в качестве операнда 1. Например, IMUL BX, CX, 50.

Синтаксис: IMUL регистр, непосредственное значение
IMUL регистр, регистр/память, непосредственное значение

Для процессоров 80386 (и более поздних) множимое может находиться в регистре (операнд 1), а множитель может находиться в регистре или в памяти (операнд 2). Например, IMUL EBX, EDX.

Синтаксис: IMUL регистр, регистр/память

IN — ввод байта или слова из порта

Передает из вводного порта один байт в регистр AL, или слово (два байта) в регистр AX, или двойное слово в регистр EAX. Порт кодируется как фиксированный числовой операнд — номер порта, или как переменная в регистре DX. Если номер порта больше 256, то используется регистр DX. Кроме того, процессоры 80186, 80286 и 80386 имеют команду INS (Input String — ввод строки).

Синтаксис: IN AL/AX/EAX, номер порта/DX

INC — инкремент

Прибавляет 1 к байту или слову в регистре или в памяти, например INC CX.

Синтаксис: INC регистр/память

INT — прерывание

Прерывает выполнение программы и передает управление по одному из 256 векторов прерывания.

Синтаксис: INC номер прерывания

INTO — прерывание по переполнению

Приводит к прерыванию при возникновении переполнения (флаг OF установлен в 1) и выполняет команду IRET 04H. Адрес подпрограммы обработки прерывания (вектор прерывания) находится по адресу ЮН.

Синтаксис: INTO операндов нет

IRET — возврат из обработчика прерывания

Обеспечивает возврат типа FAR из подпрограммы обработки прерывания (обработчика прерывания): извлекает слово из вершины стека в регистр IP, увеличивает значение SP на 2, сохраняет слово из вершины стека в CS, увеличивает значение IP на 2 и сохраняет слово из вершины стека в регистр флагов.

Синтаксис: IRET операндов нет

J-команды условного перехода

Команды предназначенные для передачи управления (перехода) программой по адресу, указанному в паре регистров CS:IP. Переход по указанному адресу происходит, если выполняется условие, заданное флагами. Для выполнения перехода в паре регистров CS:IP формируется новый адрес, путем прибавления к регистру IP значения операнда J-команды, являющегося относительным смещением и называемым меткой для перехода. В случае, если условие, заданное флагами, не выполняется, происходит переход к команде, расположенной после J-команды.

Синтаксис: J-команда метка

JA/JNBE — переход по "больше"/"не меньше или равно"

Используется после проверки *беззнаковых* данных. Переход по метке выполняется, если флаг CF=0 (нет переноса) *и* флаг ZF=0.

JAE/JNB/JNC — переход по "больше или равно"/"не меньше"/если нет переноса

Используется после проверки *беззнаковых* данных. Переход по метке выполняется, если флаг CF=0 (нет переноса).

JB/JNAE/JC — переход по "меньше"/"не больше или равно"/переход по переносу

Используется после проверки *беззнаковых* данных. Переход по метке выполняется, если флаг CF=1 (есть перенос).

JBE/JNA — переход по "меньше или равно"/"не больше"

Используется после проверки *беззнаковых* данных. Переход по метке выполняется, если флаг CF=1 (есть перенос) *или* флаг AF=1.

JE/JZ — переход по "равно"/"нулю"

Используется после проверки *знаковых* или *беззнаковых* данных. Команда JZ применяется для перехода по метке, когда эти данные равны нулю. Переход по метке выполняется, если флаг ZF=1.

JG/JNLE — переход по “больше”/“не меньше или равно"

Используется после проверки *знаковых* данных. Переход по метке выполняется, если флаг ZF=0 *и* флаги SF и OF равны (SF=OF).

JGE/JNL — переход по "больше или равно”/“не меньше"

Используется после проверки *знаковых* данных. Переход по метке выполняется, если флаги SF и OF равны (SF=OF).

JL/JNGE — переход по “меньше”/“не больше или равно"

Используется после проверки *знаковых* данных. Переход по метке выполняется, если флаги SF и OF неравны (SF≠OF).

JLE/JNG — переход по "меньше или равно”/“не больше"

Используется после проверки *знаковых* данных. Переход по метке выполняется, если флаг ZF=1 *или* флаги SF и OF неравны (SF≠OF).

JNE/JNZ — переход по "не равно" или по "не нуль"

Используется после проверки *знаковых* или *беззнаковых* данных. Команда JNZ применяется для перехода по метке, когда эти данные неравны нулю. Переход по метке выполняется, если флаг ZF=0.

JNO — переход, если нет переполнения

Используется после обнаружения, что отсутствует переполнение. Переход по метке выполняется, если флаг OF=0 (нет переполнения).

JNP/JPO — переход, если нет паритета/паритет нечетный

Используется после обнаружения, что отсутствует паритет или нечетный паритет. В данном случае нечетный паритет означает, что число битов, установленных в 1, *нечетное* в младших восьми битах. Переход по метке выполняется, если флаг PF=0 (нечетный паритет).

JNS — переход, если нет знака

Используется, если в результате операции получен *положительный* знак. Переход по метке выполняется, если флаг SF=0.

JO — переход по переполнению

Используется, если в результате выполнения операции получено переполнение. Переход по метке выполняется, если флаг OF=1 (переполнение).

JP/JPE — переход, если есть паритет/паритет четный

Используется после обнаружения паритета или четного паритета. В данном случае четный паритет означает, что число битов, установленных в 1, *четное* в младших восьми битах. Переход по метке выполняется, если флаг PF=1 (четный паритет).

JS — переход по знаку

Используется, если в результате операции получен *отрицательный* знак. Переход по метке выполняется, если флаг SF=1.

JCXZ — переход по CX=0

Переход по метке выполняется, если значение в регистре CX равно нулю. Команда JCXZ может быть полезна в начале циклов LOOP.

Синтаксис: JCXZ метка

JMP — безусловный переход

Переход по метке выполняется при любых условиях. При межсегментном переходе адрес перехода сохраняется в регистре IP, а адрес сегмента, в который осуществляется переход, в регистр CS (◄◄ см. также раздел "J-команды условного перехода").

Синтаксис: JMP метка

LAHF — загрузка флагов в регистр AH

Загружает значение флагового регистра в регистр AH. Данная команда обеспечивает совместимость с процессором 8080. Команда LAHF сохраняет правый байт флагового регистра (8 младших битов) в регистр AH в следующем виде: SZ*A*P*C, где символом "*" обозначены неиспользуемые биты.

Синтаксис: LAHF операндов нет

LDS — загрузка в сегментный регистр

Инициализирует начальный адрес сегмента данных и адрес смещения к переменной для обеспечения доступа к данной переменной. Команда LDS загружает четыре байта (два слова) из области памяти (операнд 2), содержащей *относительный адрес* (первое слово) и *сегментный адрес* (второе слово), в любой из общих или индексных регистров или в регистровый указатель (операнд 1) — относительный адрес и в регистр DS — сегментный адрес.

Синтаксис: LDS регистр, память

Например, следующая команда загружает сегментный адрес в регистр DS, а относительный адрес в регистр DI: LDS DI, адрес памяти.

LEA — загрузка эффективного (относительного) адреса

Загружает в регистр (операнд 1) двухбайтное смещение (адрес) из памяти (операнд 2).

Синтаксис: LEA регистр, память

LES — загрузка в дополнительные сегментные регистры

Инициализирует начальный адрес дополнительного сегмента и адрес смещения к переменной для обеспечения доступа к данной переменной.

◄◄ см. описание для команды LDS.

LOCK — блокировка шины доступа к данным

Запрещает математическим сопроцессорам и другим процессорам изменять элементы данных одновременно с главным процессором. Команда LOCK представляет собой однобайтовый префикс, который можно кодировать непосредственно перед любой командой. Данная операция посылает сигнал в другой процессор, запрещая использование данных до тех пор, пока не будет завершена следующая команда.

Синтаксис: LOCK команда

LODS (LODSB/LODSW/LODSD) - загрузка строки (по байтам/словам/двойным словам)

Загружает из памяти один байт в регистр AL или одно слово в регистр AX. Несмотря на то, что команда LODS выполняет строковую операцию, нет смысла использовать ее с префиксом REP. Регистровая пара DS:SI адресует в памяти байт (для LODSB), или слово (для LODSW), или двойное слово (для LODSD — для процессоров 80386 и более поздних), которые загружаются в регистр AL, AX и EAX, соответственно. Если флаг DF=0, то операция прибавляет 1 (для байта), 2 (для слова) или 4 (для двойного слова) к содержимому регистра SI. Если флаг DF=1, то выполняется операция вычитания из регистра SI, величин аналогичных операции сложения, когда DF=0.

Синтаксис: LODS память
LODS сегмент:память
LODSB/LODSW/LODSD операндов нет

LOOP — цикл

Управляет выполнением группы команд определенное число раз. До начала цикла в регистр CX должно быть загружено число выполняемых циклов. Команда LOOP находится в конце цикла, где она уменьшает значение в регистре CX на единицу. Если значение в регистре CX не равно нулю, то команда передает управление по адресу, указанному в операнде (прибавляет к регистру IP значение операнда); в противном случае управление передается на следующую после LOOP команду (происходит выход из цикла).

Синтаксис: LOOP метка

LOOPE/LOOPZ - цикл повторять, если “равно”/“ноль”

Выполненяет группу команд число раз, определенное в регистре CX *или* пока флаг ZF установлен в единичное состояние. Команды LOOPE/LOOPZ аналогичны команде LOOP, за исключением того, что по этим командам цикл прекращается либо по нулевому значению в регистре CX, либо когда значение флага ZF=0.

Синтаксис: LOOPE/LOOPZ метка

LOOPNE/LOOPNZ - цикл повторять, если "не равно”/“не ноль"

Выполненяет группу команд число раз, определенное в регистре CX ***или*** пока флаг ZF сброшен в нулевое состояние. Команды LOOPNE/LOOPNZ аналогичны команде LOOP за исключением того, что по этим командам цикл прекращается либо по нулевому значению в регистре CX, либо когда значение флага ZF=1.

Синтаксис: LOOPNE/LOOPNZ метка

MOV — пересылка данных

Пересылает один байт или одно слово между регистрами или между регистром и памятью, а также передает непосредственное значение в регистр или в память. Источником перемещаемых данных является операнд 1. Размер данных может быть 1, 2 или 4 байта, а размер операндов (источника и приемника) должны совпадать. Команда MOV не может передавать данные между двумя адресами памяти (для этой цели служит команда MOVS).

Синтаксис: MOV регистр/память, регистр/память/непосредственное значение

MOVS (MOVSB/MOVSW/MOVSD) - пересылка строки (по байтам/словам/двойным словам)

Пересылает данные между областями памяти. Команда MOVSB пересылает любое число байтов, команда MOVSW — любое число слов, а команда MOVSD— любое число двойных слов. Перед выполнением команды регистровая пара DS:SI должна адресовать источник пересылки (операнд 2), а регистровая пара ES:DI — получателя пересылки (операнд 1).

Если флаг DF=0, то операция пересылает данные *слева направо* и увеличивает регистры SI и DI на: 1 — для байта, 2 — для слова, 4 — для двойного слова.

Если флаг DF=1, то операция пересылает данные *справа налево* и уменьшает регистры SI и DI на величину, аналогичную операции сложения, когда DF=0.

Команды MOVS(B/W/D) обычно используются с префиксом REP, который уменьшает значение регистра CX на единицу после каждой операции. Если регистр CX=0, выполнение операции завершается.

Синтаксис: [REP] MOVS/MOVSB/MOVSW/MOVSD операндов нет

MUL — беззнаковое умножение

Умножает беззнаковое множимое на беззнаковый множитель. Левый единичный бит (старший) воспринимается как бит данных, а не как знак минус для отрицательных чисел.

Для 8-битового умножения множимое должно находиться в регистре AL, а множитель может находиться в регистре или в памяти, например MUL CL. Произведение записывается в регистр AX.

Для 16-битового умножения множимое должно находиться в регистре AX, а множитель может находиться в регистре или в памяти, например MUL BX. Произведение записывается в регистровой паре DX:AX.

Для 32-битового умножения множимое должно находиться в регистре EAX, а множитель может находиться в регистре или в памяти, например MUL ECX. Произведение записывается в регистровой паре EAX:EDX.

Синтаксис: MUL регистр/память

NEG — изменение знака числа

Команда меняет знак двоичного числа, преобразуя положительное число в такое же отрицательное число и наоборот, вычисляя двоичное дополнение операнда, вычитая операнд из 0 и прибавляя к результату 1. Операндом может быть байт, слово или двойное слово в регистрах или в памяти.

Синтаксис: NEG регистр/память

NOP — нет операции

Применяется для удаления или вставки машинных кодов или для задержки выполнения программы. Команда NOP выполняет операцию XCHG AX,AX, которая ничего не меняет.

Синтаксис: NOP операндов нет

NOT — логическое "НЕТ"

Меняет нулевые биты на единичные и наоборот. Операндом может быть байт, слово или двойное слово в регистрах или в памяти.

Синтаксис: NOT регистр/память

OR — логическое "ИЛИ"

Выполняет поразрядную дизъюнкцию (логическое "ИЛИ") над битами двух операндов. Операндами являются байты, слова или двойные слова в регистрах или в памяти, второй операнд может быть непосредственным значением. Если любой из пары проверяемых бит равен единице, то бит в первом операнде устанавливается равным единице, в противном случае бит в первом операнде не изменяется.

Синтаксис: OR регистр/память, регистр/память/непосредственное значение

OUT — вывод байта, слова или двойного слова в порт

Передает в порт вывода байт из регистра AL, слово из регистра AX или двойное слово из регистра EAX. Порт кодируется как фиксированный числовой операнд — номер порта, или как переменная в регистре DX. Если номер порта больше 256, то используется регистр DX. Кроме того, процессоры 80186, 80286 и 80386 имеют команду OUTS (Output String — вывод строки).

Синтаксис: OUT номер порта/DX, AL/AX/EAX

POP — извлечение слова или двойного слова из стека

Передает слово или двойное слово, помещенное ранее в стек, в указанное в операнде место: общий регистр, сегментный регистр, память. Регистр SP указывает на текущее слово (двойное слово) в вершине стека. Команда POP извлекает слово (двойное слово) из стека и увеличивает значение в регистре SP на 2 (4 — двойного слова).

Синтаксис: POP регистр/память

POPA/POPAD — извлечение из стека данных во все общие регистры

Для процессора 80286 (и более ранних) используется команда POPA, извлекающая из стека восемь значений в регистры DI, SI, BP, SP, BX, DX, CX, AX в указанной последовательности и увеличивающая значение регистра SP на 16, чтобы отбросить его значение. Записать в стек содержимое перечисленных выше регистров можно при помощи команды PUSHA.

Для процессора 80386 (и более поздних) используется команда POPAD, извлекающая из стека восемь двойных слов значений в регистры EDI, ESI, EBP, ESP, EBX,

EDX, ECX, EAX в указанной последовательности и увеличивающая значение регистра ESP на 32, чтобы отбросить его значение. Записать в стек содержимое перечисленных выше регистров можно при помощи команды PUSHAD.

Синтаксис: POPA/POPAD операндов нет

POPF/POPFD — извлечение флагов из стека

Передает биты (помещенные ранее в стек) во флаговый регистр. Регистр SP указывает на текущее слово (двойное слово) в вершине стека. Команда POPF (для процессора 80286 и более ранних) передает биты из этого слова (двойного слова — команда POPFD для процессора 80386 и более поздних) во флаговый регистр и увеличивает значение в регистре SP на 2 (4 — для двойного слова). Обычно команда PUSHF/PUSHFD записывает значения флагов в стек, а команда POPF/POPFD восстанавливает эти флаги.

Синтаксис: POPF/POPFD операндов нет

PUSH — сохранение слова или двойного слова в стеке

Сохраняет значение слова или двойного слова (адрес или элемент данных) в стеке для его последующего использования. Регистр SP указывает на текущее слово (двойное слово) в вершине стека. Команда PUSH уменьшает значение в регистре SP на 2 или в регистре ESP на 4 (для двойного слова) и передает слово (двойное слово) из указанного операнда в новую вершину стека. Операндом команды PUSH может быть общий регистр, сегментный регистр, слово (двойное слово) в памяти или непосредственное значение для процессоров 80286 и более поздних.

Синтаксис: POP регистр/память/непосредственное значение

PUSHA/PUSHAD - сохранение в стеке всех общих регистров

Для процессора 80286 (и более ранних) используется команда PUSHA, записывающая в стек восемь значений регистров AX, CX, DX, BX, SP, BP, SI, DI в указанной последовательности и уменьшающая регистр SP на 16. Извлечь из стека данные в перечисленные выше регистры можно при помощи команды POPA.

Для процессора 80386 (и более поздних) используется команда PUSHAD, записывающая в стек восемь двойных слов значений регистров EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX в указанной последовательности и уменьшающая значение регистра ESP на 32. Извлечь из стека данные в перечисленные выше регистры можно при помощи команды PUSHAD.

Синтаксис: PUSHA/PUSHAD операндов нет

PUSHF/PUSHFD — сохранение флагов в стеке

Сохраняет значений флагов из флагового регистра в стеке для их последующего использования. Регистр SP указывает на текущее слово (PUSHF) или двойное слово (PUSHFD — для процессора 80386 и более поздних) в вершине стека. Команда PUSHF уменьшает значение в регистре SP на 2, а команда PUSHFD — на 4. Обе команды передают флаги в новую вершину стека.

Синтаксис: PUSHF/PUSHFD операндов нет

RCL/RCR — циклический сдвиг влево/вправо через перенос

Выполняет циклический побитовый сдвиг (ротацию) влево или вправо байта, слова или двойного слова (для процессора 80386 и более поздних) в регистре или в памяти через флаг CF. Для процессора 80286 (и более ранних) ротация на один бит указывается в команде непосредственным значением 1, в качестве операнда; ротация более, чем на один бит, требует в качестве операнда указания регистра CL, который содержит величину сдвига (счетчик бит). Для процессора 80386 (и более поздних) в качестве операнда может указываться непосредственное значение до 31 включительно.

Выходящий слева за пределы операнда бит при выполнении команды RCL, записывается во флаг CF, а значение флага CF предварительно записывается в бит О (крайний слева байт) операнда; все *последующие* биты сдвигаются *влево*. Для команды RCR значение флага CF записывается в самый левый бит (старший) операнда, а бит О записывается во флаг CF; все *предыдущие* биты сдвигаются *вправо*.

Синтаксис: RCL/RCR регистр/память, CL/непосредственное значение

REP/REPE/REPZ/REPNE/REPNZ - повтор строковой команды при условии

Повторяет строковую операцию определенное число раз или до наступления некоторого события. Используется в качестве префикса повторения перед строковыми командами CMPS, MOVS, SCAS, STOS, изменяющих флаг ZF. Счетчик повторений должен быть загружен в регистр CX до выполнения строковой команды. Операция уменьшает регистр CX на 1 при каждом выполнении строковой команды. Для префикса REP операция повторяется, пока содержимое регистра CX не достигнет нуля. Для префикса REPE или REPZ (условие: равно или ноль) операция повторяется, пока флаг ZF равен 1 (равно или ноль) и регистр CX содержит ненулевое значение. Для префикса REPNE или REPNZ (условие: неравно или не ноль) операция повторяется, пока флаг ZF равен 0 (неравно или не ноль) и регистр CX содержит ненулевое значение.

Синтаксис: REP/REPE/REPZ/REPNE/REPNZ строковая команда

RET/RETF/RETN — возврат из процедуры

Возвращает управление из процедуры, вызванной ранее командой CALL. Для возврата из процедуры NEAR, используется команда RETN, а для процедуры, объявленной как FAR — команда RETF.

В случае возврата из процедуры NEAR команда RET сохраняет слово из вершины стека в регистре IP и увеличивает значение SP на 2. В случае возврата из процедуры FAR команда RET сохраняет двойное слово из вершины стека в регистрах IP и CS и увеличивает значение SP на 4.

Любой числовое значение, указанное в качестве операнда команды, (например, RET 4) прибавляется к указателю стека SP.

Синтаксис: RET/RETF/RETN [непосредственное значение]

ROL/ROR — циклический сдвиг влево/вправо

Выполняет циклический побитовый сдвиг (ротацию) влево или вправо байта, слова или двойного слова (для процессора 80386 и более поздних) в регистре или в памяти.

Для процессора 80286 (и более ранних) ротация на один бит указывается в команде непосредственным значением 1, в качестве операнда; ротация более, чем на один бит, требует в качестве операнда указания регистра CL, который содержит величину сдвига (счетчик бит).

Для процессора 80386 (и более поздних) в качестве операнда может указываться непосредственное значение до 31 включительно.

Выходящий слева за пределы операнда бит, при выполнении команды ROL, записывается в бит 0 (крайний слева байт) операнда, а все *последующие* биты сдвигаются *влево*. Для команды ROR бит 0 записывается в самый левый бит (старший) операнда, а все *предыдущие* биты сдвигаются *вправо*.

Синтаксис: ROL/ROR регистр/память, CL/непосредственное значение

SAHF — установка флагов из регистра AH

Данная команда обеспечивает совместимость с процессором 8080 для пересылки значений флагов из регистра AH в младшие 8 разрядов регистра флагов. Определенные биты из регистра AH пересылаются в регистор флагов в следующем виде: SZ*A*P*C, где символ "*" обозначает неиспользуемые биты.

SAL/SAR — алгебраический сдвиг влево/вправо

Выполняет побитовый сдвиг влево или вправо байта, слова или двойного слова (для процессора 80386 и более поздних) в регистре или в памяти.

Для процессора 80286 (и более ранних) сдвиг на один бит указывается в команде непосредственным значением 1, в качестве операнда; сдвиг более, чем на один бит, требует в качестве операнда указания регистра CL, который содержит величину сдвига (счетчик бит). Для процессора 80386 (и более поздних) в качестве операнда может указываться непосредственное значение до 31 включительно.

Команда SAL сдвигает биты влево определенное число раз и правый освобождающийся бит заполняется нулевым значением. Команда SAR выполняет арифметический сдвиг вправо, который учитывает знак сдвигаемого значения: сдвигает биты вправо определенное число раз и левый, освобождающийся бит, заполняется значением знакового бита (0 или 1). Значения битов, выдвигаемых за пределы операнда, теряются.

Синтаксис: SAL/SAR регистр/память, CL/непосредственное значение

SBB — вычитание с переносом (заемом)

Обычно используется при вычитании многобайтных двоичных величин для учета единичного бита переполнения в последующей фазе операции. Если флаг CF=1, то команда SBB сначала вычитает 1 из операнда 1. Команда SBB всегда вычитает операнд 2 из операнда 1, аналогично команде SUB.

Синтаксис: SBB регистр/память, регистр/память/непосредственное значение

SCAS (SCASB/SCASW/SCASD) - поиск (байта, слова или двойного слова) в строке

Выполняет поиск определенного значения в строке, находящейся в памяти по адресу, указанному в паре регистров ES:DI. Команда SCASB ищет в строке байт, который предварительно загружается в регистр AL. Команда SCASW ищет в строке слово, которое предварительно загружается в регистр AX. Команда SCASD ищет в строке двойное

слово, которое предварительно загружается в регистр EAX (для процессора 80386 и более поздних).

Если флаг DF=0, то операция сканирует память *слева направо* и увеличивает регистр DI на 1 с каждым байтом. Если флаг DF=1, то операция сканирует память *справа налево* и уменьшает регистр DI на 1 с каждым байтом.

Данные команды обычно используются с префиксом REPE или REPNE. В регистр CX предварительно записывается величина, позволяющая ограничить число выполнений команд поиска. С каждым выполнением команды поиска префикс REPE/REPNE уменьшает значение регистра CX на 1. Поиск прекращается, если CX=0. Пара регистров DI и SI содержит адрес элемента строки, следующим за совпадающим/несовпадающим элементом. Обычно, префикс REPE применяется, если ищется первое несовпадение, для поиска первого совпадения применяется префикс REPNE.

Синтаксис: [REPn] SCASB/SCASW/SCASD операндов нет

SHL/SHR — логический сдвиг влево/вправо

Команды SHL и SHR выполняют логический сдвиг и рассматривают знаковый бит как обычный бит данных.

Команда SHL выполняется аналогично команде SAL (◀◀ см. описание команд SAL/SAR).

Команда SHR аналогична команде SAR, но в случае логического сдвига вправо левые, освобождающиеся биты, заполняются нулевым значением, а не знаковым битом (0 или 1), как при арифметическом сдвиге вправо.

Синтаксис: SHL/SHR регистр/память, CL/непосредственное значение

STC — установка флага переноса CF

Устанавливает значение флага CF в 1.

Синтаксис: STC операндов нет

STD — установка флага направления DF

Устанавливает значение флага DF в 1. В результате строковые операции, такие, как MOVS или CMPS, обрабатывают данные *справа налево*.

STOS и STOSB/STOSW/STOSD — запись строки длинной в байт/слово/двойное слово

Сохраняет байт, слово или двойное слово (для процессора 80386 и более поздних) в памяти. Команда STOSB повторяет байт из регистра AL, команды команда STOSW – слово из регистра AX, а команда STOSD — двойное слово из регистра AEX. Регистровая пара ES:DI указывает область памяти, куда должен быть записан байт/слово/двойное слово. Если флаг DF=0, то операция записывает данные в память *слева направо* (в порядке возрастания адреса) и увеличивает значение в регистре DI на 1. Если флаг DF=1, то операция записывает в память *справа налево* (в порядке уменьшения адреса) и уменьшает значение в регистре DI на 1.

При использовании префикса REP операция дублирует значение байт/слово/двойное слово число раз, определенное в регистре CX, что делает ее удобной для очистки областей памяти. С каждым выполнением команды записи префикс REP уменьшает значение регистра CX на 1. Выполнение команд записи прекращается, если CX=0.

Синтаксис: [REP] STOSB/STOSW/STOSD операндов нет

SUB — вычитание двоичных чисел

Вычитает байт, слово или двойное слово в регистре, памяти или непосредственное значение из регистра; или вычитает байт, слово или двойное слово в регистре или непосредственное значение из памяти.

Синтаксис: SUB регистр/память, регистр/память/непосредственное значение

TEST — проверка битов

Команда выполняет проверку байта, слова или двойного слова на определенную битовую комбинацию в регистре или памяти. Операнды могут быть байтом, словом или двойным словом. Второй операнд может быть непосредственным значением. Команда TEST действует аналогично команде AND, устанавливая флаги в соответствии с логической функцией "И", но не изменяет результирующий операнд: флаг ZF сбрасывается (ZF=0), если любая пара сравниваемых битов содержит 1, в противном случае флаг ZF устанавливается (ZF=1).

Синтаксис: TEST регистр/память, регистр/память/непосредственное значение

WAIT — перевод процессора в состояние ожидания

Позволяет процессору оставаться в состоянии ожидания, пока не произойдет внешнее прерывание. Данная операция необходима для обеспечения синхронизации процессора с внешним устройством или с сопроцессором. Процессор остается в состоянии ожидания, пока внешнее устройство или сопроцессор не закончат работу или вычисления, и продолжает свою работу, получив сигнал с входа TEST.

Синтаксис: WAIT операндов нет

XCHG — перестановка

Переставляет два байта или два слова между двумя регистрами, например, XCHG AH, BL, или между регистром и памятью, например, XCHG CX, word.

Синтаксис: XCHG регистр/память, регистр/память

XLAT — перекодировка

Транслирует байты в другой формат, например, при переводе нижнего регистра в верхний или при перекодировке ASCII-кода в EBCDIC-код. Для выполнения данной команды необходимо определить таблицу преобразования байт и загрузить ее адрес в регистр BX или EBX для 32-разрядного размера операнда. Регистр AL должен содержать байт, который будет преобразован с помощью команды XLAT. Операция использует значение в регистре AL как смещение в таблице, выбирает байт по этому смещению и помещает его в регистр AL.

Синтаксис: XLAT [AL]

XOR — исключающее "ИЛИ"

Выполняет логическую операцию исключающего "ИЛИ" над битами двух операндов. Операндами являются байты, слова или двойные слова в регистрах или в памяти, второй операнд может быть непосредственным значением. Команда XOR обрабатывает операнды побитово. Если проверяемые биты одинаковы, то команда XOR устанавливает бит в операнде 1 равным нулю, если биты различны, то бит в операнде 1 устанавливается равным единице.

Синтаксис: XOR регистр/память, регистр/память/непосредственное значение

Приложение E

Кодировка символов ASCII

Таблица E.1. Неотображаемые (управляющие) символы ASCII

Код		Название символа	Символ	Код		Название символа	Символ
0	$00	Ноль		16	$10	Выход из линии данных	fe
1	$01	Начало заголовка	©	17,	$11	Управляющий 1	◄
2	$02	Начало текста		18	$12	Управляющий 2	▌
3	$03	Конец текста	♥	19	$13	Управляющий 3	‼
4	$04	Конец передачи		20	$14	Управляющий 4	¶
5	$05	Запрос	♣	21	$15	Отрицательное подтверждение	§
6	$06	Подтверждение	♠	22	$16	Синхронизация	
7	$07	Звуковой сигнал	◦	23	$17	Конец блока	↨
8	$08	Возврат на шаг	a	24	$18	Отмена	↑
9	$09	Горизонтальная табуляция	o	25	$19	Конец линии передачи	▪
10	$OA	Перевод строки	т	26	$1A	Замена	→
11	$OB	Вертикальная табуляция	♂	27	$1B	<Escape>	←
12	$OC	Прогон страницы	9	28	$1C	Разделитель файла	∟
13	$0D	Возврат каретки	f	29	$1D	Разделитель группы	↔
14	$OE	Сдвиг наружу	♫	30	$1E	Разделитель записи	A
15	$OF	Сдвиг внутрь	☼	31	$1F	Разделитель блока	T

Таблица E.2. Отображаемые символы ASCII

Код		Символ	Код		Символ	Код		Символ	Код		Символ
32	$20		41	$29	)	50	$32	2	59	$3B	;
33	$21	!	42	$2A	*	51	$33	3	60	$3C	<
34	$22	"	43	$2B	+	52	$34	4	61	$3D	9в
35	$23	#	44	$2C	,	53	$35	5	62	$3E	>
36	$24	$	45	$2D	-	54	$36	6	63	$3F	?
37	$25	%	46	$2E	.	55	$37	7	64	$40	©
38	$26	&	47	$2F	/	56	$38	8	65	$41	A
39	$27	'	48	$30	0	57	$39	9	66	$42	B
40	$28	(	49	$31	1	58	$3A	:	67	$43	C

Продолжение таблицы Е.2

Код		Символ	Код		Символ	Код		Символ	Код		Символ
68	$44	D	107	$6B	k	146	$92	Т	185	$B9	╣
69	$45	E	108	$6C	l	147	$93	У	186	$BA	║
70	$46	F	109	$6D	m	148	$94	ф	187	$BB	╗
71	$47	G	110	$6E	n	149	$95	Х	188	$BC	╝
72	$48	H	111	$6F	o	150	$96	Ц	189	$BD	Л
73	$49	I	112	$70	P	151	$97	Ч	190	$BE	J
74	$4A	J	113	$71	q	152	$98	Ш	191	$BF	┐
75	$4B	K	114	$72	r	153	$99	Щ	192	$CO	└
76	$4C	L	115	$73	s	154	$9A	Ъ	193	$C1	┴
77	$4D	M	116	$74	t	155	$9B	Ы	194	$C2	┬
78	$4E	N	117	$75	u	156	$9C	Ь	195	$C3	├
79	$4F	O	118	$76	v	157	$9D	Э	196	$C4	─
80	$50	P	119	$77	w	158	$9E	Ю	197	$C5	┼
81	$51	Q	120	$78	x	159	$9F	Я	198	$C6	╞
82	$52	R	121	$79	y	160	$AO	а	199	$C7	╟
83	$53	S	122	$7A	z	161	$A1	б	200	$C8	╚
84	$54	T	123	$7B	{	162	$A2	в	201	$C9	╔
85	$55	U	124	$7C	|	163	$A3	г	202	$CA	╩
86	$56	V	125	$7D	}	164	$A4	д	203	$CB	╦
87	$57	W	126	$7E	~	165	$A5	е	204	$CC	╠
88	$58	X	127	$7F	Q	166	$A6	ж	205	$CD	═
89	$59	Y	128	$80	А	167	$A7	з	206	$CE	╬
90	$5A	Z	129	$81	Б	168	$A8	и	207	$CF	╧
91	$5B	[	130	$82	В	169	$A9	й	208	$D0	╨
92	$5C	\	131	$83	Г	170	$AA	к	209	$D1	╤
93	$5D	]	132	$84	Д	171	$AB	л	210	$D2	╥
94	$5E	л	133	$85	Е	172	$AC	м	211	$D3	╙
95	$5F	_	134	$86	Ж	173	$AD	н	212	$D4	╘
96	$60	`	135	$87	З	174	$AE	о	213	$D5	╒
97	$61	a	136	$88	И	175	$AF	п	214	$D6	╓
98	$62	b	137	$89	Й	176	$BO	▒	215	$D7	╫
99	$63	c	138	$8A	К	177	$B1	┐	216	$D8	╪
100	$64	d	139	$8B	Л	178	$B2	▓	217	$09	┘
101	$65	e	140	$8C	М	179	$B3	|	218	$DA	┌
102	$66	f	141	$8D	Н	180	$B4	┤	219	$DB	▌
103	$67	g	142	$8E	О	181	$B5	╡	220	$DC	▄
104	$68	h	143	$8F	П	182	$B6	╢	221	$DD	▐
105	$69	i	144	$90	Р	183	$B7	т	222	$DE	▀
106	$6A	j	145	$91	С	184	$B8	┐	223	$DF	■

Окончание таблицы Е.2

Код		Символ	Код		Символ	Код		Символ	Код		Символ
224	$EO	р	232	$E8	ш	240	$FO	Ё	248	$F8	°
225	$E1	с	233	$E9	щ	241	$F1	ё	249	$F9	·
226	$E2	т	234	$EA	ъ	242	$F2	е	250	$FA	·
227	$E3	у	235	$EB	ы	243	$F3	е	251	$FB	√
228	$E4	ф	236	$EC	ь	244	$F4	Т	252	$FC	№
229	$E5	х	237	$ED	э	245	$F5	Т	253	$FD	а
230	$E6	ц	238	$EE	ю	246	$F6	Ў	254	$FE	■
231	$E7	ч	239	$EF	я	247	$F7	ў	255	$FF	

Приложение Ж

Модули, процедуры и функции

В этом приложении рассмотрены процедуры и функции из стандартных библиотечных модулей System, Crt, Dos, Graph и Strings. Для доступа ко всем модулям необходимо указать ссылку на них в разделе uses. Исключение составляет только модуль System, который подключается к программе автоматически.

Модуль System

Арифметические вычисления

В некоторых примерах, рассматриваемых в данном разделе, будут отображаться графики функций в графическом режиме, поэтому имеет смысл вынести процедуру инициализации графического режима и рисования осей координат в отдельный модуль, а затем включать этот модуль в программы примеров. Текст модуля CoordSys.pas представлен в листинге Ж.1.

ПРИМЕЧАНИЕ

Напоминаем, что к книге прилагается дискета со всеми, рассматриваемыми в ней, примерами программ. Все программы, рассмотренные в этом приложении, находятся на дискете в папке app. Открыть файл с дискеты в среде Turbo Pascal можно при помощи команды **File | Open** (<F3>).

Листинг Ж.1. Модуль CoordSys.pas

```
Unit CoordSys;
interface
  uses Graph;
  const
    ZeroX = 320; {Координата X начала координат}
    ZeroY = 175; {Координата Y начала координат}
 ' procedure DrawCoordSys;
implementation
  procedure DrawCoordSys;
  var
    DriverVar, ModeVar, ErrorCode: integer;
  begin
    {Инициализация графического режима}
    DriverVar :=  EGA;
    ModeVar := EGAHI;
    InitGraph(DriverVar,ModeVar,'\tp\bgi');
    ErrorCode := GraphResult;
    {Проверка корректности инициализации графического режима}
    if ErrorCode <> grOK then
```

Окончание листинга Ж.1

```
    begin
      Writeln(GraphErrorMsg(ErrorCode));
      Halt(1);
    end;
    ClearDevice; {Очищаем экран}
    SetColor(White); {Устанавливаем белый цвет для рисования}
    Line(ZeroX,10,ZeroX,340); {Рисуем ось Y}
    Line(10,ZeroY,630,ZeroY); {Рисуем ось X}
  end;
end.
```

Для использования функции DrawCoordSys в программе необходимо указать в разделе uses ссылку на модуль CoordSys, при этом путь к откомпилированному модулю CoordSys.tpu должен быть указан в поле **Unit directories** диалогового окна **Directories** (см. рис. 1.4).

ПРИМЕЧАНИЕ

Путь к каталогу bgi в процедуре InitGraph указывайте в соответствии с его размещением на конкретном компьютере. Например, если компилятор tpc.exe или файл turbo.exe, предназначенный для запуска интегрированной среды Turbo Pascal, находятся в папке C:\TP\BIN, а bgi-файлы расположены в папке C:\TP\BGI то вместо, указанного в листинге Ж.2 пути '\tp\bgi' можно указать путь '..\bgi'. Это, так называемое, относительное указание пути доступа к папке, в котором одно двоеточие указывает на возврат в папку верхнего уровня, в данном случае папку C:\TP.

◀◀ Методы компиляции и выполнения программ подробно расмотрены в главе 13.

Функции

Функция Abs

Возвращает абсолютное значение принятого параметра.

Синтаксис: Abs (X)

Параметр X — выражение вещественного или целочисленного типа.

Тип возвращаемого результата: совпадает с типом принятого параметра.

Пример использования

Программа вывода на экран графика функции y = |x| представлена в листинге Ж.2, а результат работы программы на рис. Ж.1.

Листинг Ж.2. Программа FuncAbs.pas

```
program FuncAbs;
uses Crt, Graph, CoordSys;
var
  x, y: integer;
begin
  DrawCoordSys; {Процедура из модуля CoordSys (листинг Ж.1)}
  for x := -150 to 150 do {Вычисляем значения функции}
  begin                   {в диапазоне x от -150 до 150}
    y := abs(x); {y = |x|}
    PutPixel(ZeroX+x, ZeroY-y, Yellow); {Рисуем на экране желтую точку
                  с координатами, соответствующими значениям x и y}
```

Окончание листинга Ж.2

```
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция ArcTan

Возвращает арктангенс принятого параметра.

Синтаксис: ArcTan (X)

Параметр X — выражение вещественного типа.

Тип возвращаемого результата: Real.

Пример использования

Программа вывода на экран графика функции y = ArcTan(x) представлена в листинге Ж.3, а результат работы программы на рис. Ж.2.

Листинг Ж.3. Программа FuncATan.pas

```
program FuncATan;
uses Crt, Graph, CoordSys;
const
  UnitX = 3; {Количество пикселей на одно деление по оси X}
  UnitY = 10; {Количество пикселей на одно деление по оси Y}
var
  x: integer;
  y: real;
begin
  DrawCoordSys; {Процедура из модуля CoordSys (листинг Ж.1)}
  SetColor(Yellow); {Устанавливаем желтый цвет для рисования}
  for x := -150 to 150 do
  begin
    y := ArcTan(x);
    LineTo(ZeroX+x*UnitX, ZeroY-round(y*UnitY));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

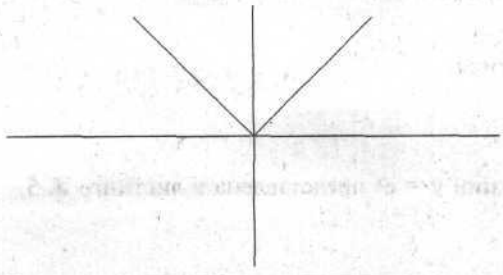


Рис. Ж.1. Построение графика функции y = |x| при помощи программы FuncAbs

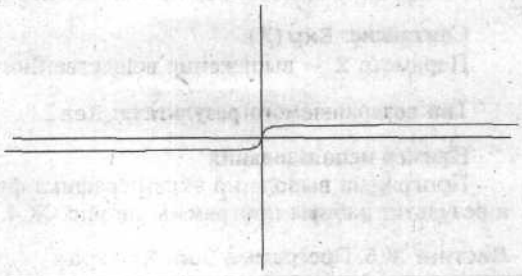


Рис. Ж.2. Построение графика функции y = ArcTan(x) при помощи программы FuncATan

Константы UnitX и UnitY используются для масштабирования графика вдоль осей X и Y. Обратите также внимание на то, что значение y не прибавляется к координате ZeroY, а вычитается из нее. Это объясняется тем, что на экране положительные значения координаты Y меньше отрицательных значений, то есть расположены выше.

Функция Cos

Возвращает косинус принятого параметра (угол в радианах).

Синтаксис: Cos (X)
Параметр X — выражение вещественного типа.

Тип возвращаемого результата: Real.

Пример использования
Программа вывода на экран графика функции y = Cos(x) представлена в листинге Ж.4, а результат работы программы на рис. Ж.3.

Листинг Ж.4. Программа FuncCos.pas

```
program FuncCos;
uses Crt, Graph, CoordSys;
const
  UnitX = 12;  {Количество пикселей на одно деление по оси X}
  UnitY = 8;   {Количество пикселей на одно деление по оси Y}
var
  x: integer;
  y: real;
begin
  DrawCoordSys;  {Процедура из модуля CoordSys  (листинг Ж.1)}
  SetColor(Yellow);  {Устанавливаем желтый цвет для рисования}
  for x := -150 to 150 do
  begin
    y := Cos(x);
    LineTo(ZeroX+x*UnitX, ZeroY-round(y*UnitY));
  end;
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph;  {Выходим из графического режима}
end.
```

Функция Exp

Возвращает экспоненту принятого параметра (e^x).

Синтаксис: Exp (X)
Параметр X — выражение вещественного типа.

Тип возвращаемого результата: Real.

Пример использования
Программа вывода на экран графика функции y = e^x представлена в листинге Ж.5, а результат работы программы на рис. Ж.4.

Листинг Ж.5. Программа FuncExp.pas

```
program FuncExp;
uses Crt, Graph, CoordSys;
const
  UnitX = 40;  {Количество пикселей на одно деление по оси X)
```

Окончание листинга Ж.5

```
  UnitY = 10; {Количество пикселей на одно деление по оси Y}
var
  x, y: real;
begin
  DrawCoordSys; {Процедура из модуля CoordSys (листинг Ж.1)}
  SetColor(Yellow); {Устанавливаем желтый цвет для рисования}
  x := -20;
  while x <= 5 do
  begin
    y := Exp(x);
    LineTo(ZeroX+x*UnitX, ZeroY-round(y*UnitY));
    x := x + 0.1;
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

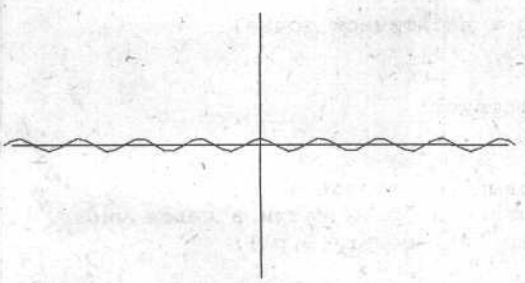


Рис. Ж.3. Построение графика функции y = Cos(x) при помощи программы FuncCos

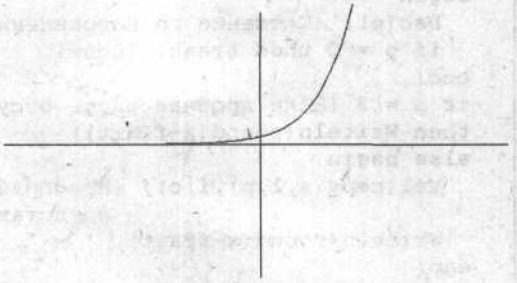


Рис. Ж.4. Построение графика функции y = e^x при помощи программы FuncExp

Функция Frac

Возвращает дробную часть принятого параметра.

Синтаксис: `Frac(X)`

Параметр X — выражение вещественного типа.

Тип возвращаемого результата: Real.

Пример использования функции Frac представлен в листинге Ж.6.

Листинг Ж.6. Программа `FuncFrac.pas`

```
program FuncFrac;
uses Crt;
var
  x,fPart: real;
  s: string;
  p: byte;
  {Функция возведения в степень}
  function Power(Num, Pow: byte): longint;
  var
    i: integer;
    Res: longint;
```

Окончание листинга Ж.6

```
  begin
    Res := 1;
    for i := 1 to Pow do Res := Res * Num;
    Power := Res;
  end;

begin
  ClrScr; {Очищаем экран}
  Write('Введите вещественное число:');
  Readln(x);
  fPart := Frac(x); {Определяем дробную часть числа}
  Str(fPart:10:8,s); (Преобразуем ее в строку}
  Delete(s,1,2);     {Удаляем из строки первый нуль и десятичную точку}
  (Определяем позицию последней значащей цифры в
   дробной части, просматривая строку с конца}
  p := Length(s);
  while s[p] = '0' do (Пока текущая цифра - 0}
  begin
    Dec(p); {Смещение по направлению к десятичной точке}
    if p = 0 then Break; {Если}
  end;
  if p = 0 {Если дробная часть отсутствует}
  then Writeln(round(x-fPart))
  else begin
    Val(copy(s,1,p),i,c); {Преобразовываемстроковое
                           представление дробной части в целое число}
    Writeln(round(x-fPart), ' и ', i, '/', Power(10,p));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

В этой программе выполняется представление десятичной дроби в виде простой дроби. Для этого выделяется дробная часть введенного вещественного числа, а затем в ней определяется количество значащих цифр. Это количество используется в качестве степени числа 10 — знаменателя простой дроби.

Функция Int

Возвращает целую часть принятого параметра.

Синтаксис: `Int(X)`

Параметр X — выражение вещественного типа.

Тип возвращаемого результата: `Real`.

Пример использования функции Int представлен в листинге Ж.7.

Листинг Ж.7. Программа `FuncInt.pas`

```
program FuncInt;
uses Crt;
var
  x: real;
begin
  ClrScr; {Очищаем экран}
  Write('Введите вещественное число:');
```

Окончание листинга Ж.7

```
  Readln(x);
  GotoXY(Round(Int(x)),3); {Перемещаемся в 3-й строке в позицию,
                            соответствующую целой части числа}
  Write(Chr(Round(Int(x))+32)); {Выводим один из символов, расположенных
                                 в таблице ASCII после пробела}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Pi

Возвращает число Пи (3.1415926535897932385).

Синтаксис: `Pi`

Тип возвращаемого результата: `Real`.

Пример использования

Программа, вычисляющая площадь круга, представлена в листинге Ж.8.

Листинг Ж.8. Программа `FuncPi.pas`

```
program FuncPi;
uses Crt;
var
  r: word;
begin
  ClrScr; {Очищаем экран}
  Write('Введите радиус круга: ');
  Readln(r);
  Writeln('Площадь круга = ', Pi*Sqr(r):5:2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Sin

Возвращает синус принятого параметра (угол в радианах).

Синтаксис: `Sin(X)`

Параметр X — выражение вещественного типа.

Тип возвращаемого результата: `Real`.

Пример использования

Программа вывода на экран графика функции y = Sin(x) представлена в листинге Ж.9, а результат работы программы на рис. Ж.5.

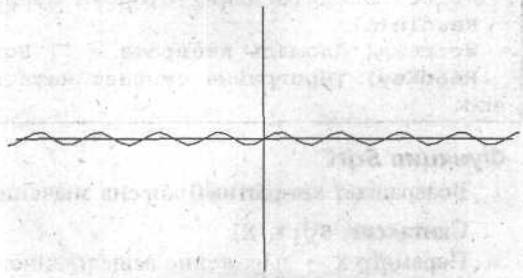


Рис. Ж.5. Построение графика функции y= Sin(x) при помощи программы `FuncSin`

Листинг Ж.9. Программа `FuncSin.pas`

```
program FuncSin;
uses Crt, Graph, CoordSys;
const
  UnitX = 12; {Количество пикселей на одно деление по оси X}
  UnitY = 8;  {Количество пикселей на одно деление по оси Y}
var
```

Окончание листинга Ж.9

```
  x: integer;
  y: real;
begin
  DrawCoordSys; {Процедура из модуля CoordSys (листинг Ж.1)}
  SetColor(Yellow); {Устанавливаем желтый цвет для рисования}
  for x := -150 to 150 do
  begin
    y := Sin(x);
    LineTo(ZeroX+x*UnitX, ZeroY-round(y*UnitY));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция Sqr

Возвращает значение принятого параметра, возведенное в квадрат.

Синтаксис: Sqr (X)

Параметр X — выражение вещественного или целого типа.

Тип возвращаемого результата: совпадает с типом принятого параметра.

Пример использования

Программа, вычисляющая площадь квадрата, представлена в листинге Ж. 10.

Листинг Ж.10. Программа FuncSqr.pas

```
program FuncSqr;
uses Crt;
var
  a: word;
begin
  ClrScr; {Очищаем экран}
  Write('Введите длину стороны квадрата: ');
  Readln(a);
  Writeln('Площадь квадрата = ', Sqr(a));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Sqrt

Возвращает квадратный корень значения принятого параметра.

Синтаксис: Sqrt(X)

Параметр X — выражение вещественного типа.

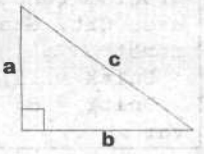
Тип возвращаемого результата: Real.

Пример использования

Программа, вычисляющая длину гипотенузы прямоугольного треугольника, представлена в листинге Ж. 11.

ПРИМЕЧАНИЕ

Напомним, что согласно теоремы Пифагора для прямоугольного треугольного справедливо равенство $c^2 = a^2 + b^2$ — отсюда следует, что $c = \sqrt{a^2 + b^2}$, где c — гипотенуза, a a и b — катеты прямоугольного треугольника.



Листинг Ж.11. Программа FuncSqrt.pas

```
program FuncSqrt;
uses Crt;
var
  a,b: word;
begin
  ClrScr; {Очищаем экран}
  Write('Введите длину 1-го катета, a: ');
  Readln(a);
  Write('Введите длину 2-го катета, b);: ');
  Readln(b);
  Writeln('Длина гипотенузы, c = ', Sqrt(Sqr(a)+Sqr(b)):10:2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Управление программой

Процедуры

Процедура Break

Прерывает ход выполнения циклических операторов.

Синтаксис: Break

Пример использования процедуры Break в различных операторах циклов представлен в листинге Ж. 12.

Листинг Ж.12. Программа ProcBrk.pas

```
program ProcBrk;
uses Crt;
var
  i: integer;
begin
  ClrScr; {Очищаем экран}
  {Вывод чисел от 1 до 4}
  for i := 1 to 10 do
    if i = 5 then Break else Writeln(i);
  {Вывод чисел от 5 до 7}
  i := 5; {Установить значение счетчика}
  while i <= 10 do
  begin
    Writeln(i);
    Inc(i); {Увеличить значение счетчика на 1}
    if i = 8 then Break;
  end;
  repeat {Вывод чисел от 8 до 10}
    Writeln(i);
    Inc(i);
    if i = 11 then Break;
  until i = 20;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Continue

Продолжает выполнение циклического оператора.

Синтаксис: Continue

Пример использования процедуры Continue в различных операторах циклов представлен в листинге Ж.13.

Листинг Ж.13. Программа ProcCntn.pas

```
program ProcCntn;
uses Crt;
var
  i: integer;
begin
  ClrScr; {Очищаем экран}
  {Вывод нечетных чисел от 1 до 20}
  for i := 1 to 20 do
    if (i mod 2) = 0 then Continue else Writeln(i);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Exit

Выход из выполняемого блока во внешний блок. Если эта процедура вызывается в функции или процедуре, то выполнение передается в вызывающую функцию или процедуру. Если эта процедура вызывается непосредственно из основного блока, то работа программы прерывается.

Синтаксис: Exit

Пример использования процедуры Exit представлен в листинге Ж.14.

Листинг Ж.14. Программа ProcExit.pas

```
program ProcExit;
uses Crt;
var
  i: integer;
  c: char;

  procedure TwentyStars;
  {Вывод на экран 20 символов '*' в одну строку}
  begin
    for i := 1 to 20 do
      if i = 20 then Exit else Write('*');
  end;

begin
  ClrScr; {Очищаем экран}
  repeat
    TwentyStars;
    Writeln;
    Write('Для вывода еще 20 звезд нажмите клавишу У или Д?');
    Write('Для выхода нажмите любую клавишу... Ввод: ');
    {Напоминаем, что любой ввод должен
     быть подтвержден нажатием клавиши <Enter>}
    Readln(c);
```

Окончание листинга Ж.14

```
    if not (c in ['Y','y','д','Д']) then Exit;
  until False;
end.
```

Процедура Halt

Прекращение выполнения программы и передача управления среде программирования или системе DOS.

Синтаксис: Halt(ExitCode)

Необязательный параметр ExitCode типа Word определяет код завершения программы.

Пример использования процедуры Halt представлен в листинге Ж.15.

Листинг Ж.15. Программа ProcHalt.pas

```
program ProcHalt;
uses Crt;
  procedure PressEsc;
  begin
    repeat
      Writeln('Для выхода из программы нажмите Esc');
      if ReadKey = #27 then Halt;
    until False;
  end;
begin
  ClrScr; {Очищаем экран}
  PressEsc;
end.
```

Процедура RunError

Прекращение выполнения программы и генерация ошибки времени выполнения.

Синтаксис: RunError(ErrorCode)

Параметр ErrorCode типа Byte определяет код ошибки времени выполнения (» коды ошибок перечислены в приложении 3).

Пример использования процедуры RunError представлен в листинге Ж.16.

Листинг Ж.16. Программа ProcRnEr.pas

```
program ProcRnEr;
uses Crt;
var
  a,b: integer;
begin
  ClrScr; {Очищаем экран}
  repeat
    Write('Введите делимое a: ');
    Readln(a);
    Write('Введите делитель b: ');
    Readln(b);
    if b = 0
    then RunError(200)   {Деление на ноль}
```

Окончание листинга Ж.18

```
    else begin
           Writeln('a/b = ', a div b);
           Writeln('Для выхода нажмите Esc.
                    Для продолжения любую клавишу...');
           if ReadKey = #27 then Exit;
         end;
    ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  until False;
end.
```

Ввод/вывод

Функции

Функция Eof

Возвращает значение True, если достигнут конец файла.

Синтаксис: Eof (F)

Необязательный параметр-переменная F — файловая переменная. Если этот параметр отсутствует, то подразумевается использование вместо него стандартной файловой переменной Input.

Тип возвращаемого результата: Boolean.

Пример использования

Программа, выводящая на экран содержимое текстового файла, представлена в листинге Ж. 17, а результат работы программы на рис. Ж.6.

Листинг Ж.17. Программа FuncEof .pas

```
program FuncEof;
uses Crt;
var
  F: Text;
  s: string;
begin
  ClrScr; {Очищаем экран}
  Write('Укажите путь и имя файла: ');
  Readln(s);
  Assign(F,s); {Связываем имя файла с файловой переменной}
  Reset(F);    {Открываем файл}
  while not Eof(F) do
  begin
    Readln(F,s); {Читаем строку файла в переменную}
    Writeln(s);  {Выводим содержимое переменной на экран}
  end;
  Close(F); {Закрываем файл}
  Write('Нажмите любую клавишу... ');
  ReadKey;
end.
```

Функция Eoln

Возвращает значение True по достижению конца строки при чтении данных из текстового файла.

Синтаксис: Eoln(F)

Необязательный параметр-переменная F — файловая переменная. Если этот параметр отсутствует, то подразумевается использование вместо него стандартной файловой переменной Input.

Тип возвращаемого результата: Boolean.

```
Укажите путь и имя файла: c:\autoexec.bat
SET COMSPEC=C:\WINDOWS\COMMAND.COM
SET windir=C:\WINDOWS
SET winbootdir=C:\WINDOWS
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
SET PROMPT=$p$g
SET TEMP=C:\WINDOWS\TEMP
SET TMP=C:\WINDOWS\TEMP
SET BLASTER=A220 I5 D1 P338
Нажмите любую клавишу...
```

Рис. Ж.6. Вывод на экран содержимого файла autoexec.bat при помощи программы FuncEof

Пример использования

Программа, выводящая на экран содержимое первой строки текстового файла, представлена в листинге Ж. 18.

Листинг Ж.18. Программа FuncEoln.pas

```
program FuncEoln;
uses Crt;
var
  F: Text;
  s: string;
  c: char;
begin
  ClrScr; {Очищаем экран}
  Writeln('Укажите путь и имя файла: ');
  Readln(s);
  Assign(F,s); {Связываем имя файла с файловой переменной}
  Reset(F);    {Открываем файл}
  while not Eoln(F) do
  begin
    Read(F,c); {Читаем строку файла в переменную}
    Write(c);  {Выводим содержимое переменной на экран}
  end;
  Close(F); {Закрываем файл}
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция FilePos

Возвращает текущую файловую позицию в указанном файле.

Синтаксис: FilePos(F)

Параметр-переменная F — файловая переменная любого файлового типа, кроме текстового файла.

Тип возвращаемого результата: Longint.

Пример использования

Программа, подсчитывающая размер файла на основании файловой позиции последнего байта, представлена в листинге Ж. 19.

Листинг Ж.19. Программа FuncFPos.pas

```
program FuncFPos;
uses Crt;
var
  F : file of Byte;
  s: string;
begin
  ClrScr; {Очищаем экран}
  Write('Введите путь и имя файла: ');
  Readln(s);
  Assign(F,s); {Связываем имя файла с файловой переменной}
  Reset(F); {Открываем файл}
  Seek(F, FileSize(F)); {Перемещаемся в конец файла}
  WriteLn('Размер файла: ', FilePos(F), ' байт');
  Close(F); {Закрываем файл}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция FileSize

Возвращает размер указанного файла.

Синтаксис: FileSize(F)

Параметр-переменная F — файловая переменная любого файлового типа, кроме текстового файла.

Тип возвращаемого результата: Longint.

Пример использования

Программа, определяющая размер файла, представлена в листинге Ж.20.

Листинг Ж.20. Программа FuncFSz.pas

```
program FuncFSz;
uses Crt;
var
  F : file of Byte;
  s: string;
begin
  ClrScr; {Очищаем экран}
  Write('Введите путь и имя файла: ');
  Readln(s);
  Assign(F,s); {Связываем имя файла с файловой переменной}
  Reset(F); {Открываем файл}
  WriteLn ('Размер файла: ', FileSize(F), ' байт');
  Close(F); {Закрываем файл}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция IOResult

Возвращает состояние последней выполненной операции ввода/вывода. Используется при отключенной директиве компилятора {$I-}.

Синтаксис: IOResult(F)

Параметр-переменная F — файловая переменная любого файлового типа, кроме текстового файла.

Тип возвращаемого результата: Integer.

Пример использования

Программа, определяющая фактическое наличие указанного файла, представлена в листинге Ж.21.

Листинг Ж.21. Программа FuncIORs.pas

```
program FuncIORs;
uses Crt;
var
  s: string;

  function FileExists (FName: string): boolean;
  var
    F: file;
    isError: boolean;
  begin
    Assign(F,Fname); {Связываем имя файла с файловой переменной}
    {$I-}
    Reset(F); {Открываем файл}
    {$I+}
    isError := IOResult > 0;
    if isError
    then Writeln('Файла с таким именем не существует!')
    else Close(F); {Закрываем файл}
    FileExists := not isError;
  end;

begin
  ClrScr; {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  if FileExists(s) then
    Writeln('Такой файл существует');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция SeekEof

Определяет, достигнут ли конец текстового файла.

Синтаксис: SeekEof(F)

Параметр-переменная F — файловая переменная типа Text.

Тип возвращаемого результата: Boolean.

Пример использования функции SeekEof представлен в листинге Ж.22.

Листинг Ж.22. Программа FuncSEof.pas

```
program FuncSEof;
uses Crt;
var
  F: Text;
  i: Integer;
begin
  ClrScr; {Очищаем экран}
```

Окончание листинга Ж.22

```
  Assign(F,'test.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F); {Создаем файл}
   {Файл будет создан в каталоге, указанном в процедуре Assign.
    В данном случае, путь не указан, поэтому txt-файл будет создан в
    каталоге, в котором находится откомпилированный exe-файл программы
    ProcApnd.pas или файл turbo.exe, запускающий интегрированную среду
    Turbo Pascal}
   {Записываем в файл 8 цифр и пробелы в конце строк}
  Writeln(F, '1 2 3 4 ');
  Writeln(F, '5 6 7 8 ');
  Reset(F); {Открываем файл}
   {Считываем данные. SeekEoln возвращает True, если в данной строке
    больше нет цифр;SeekEof возвращает True, если в файле больше нет
    текста (кроме пробелов) }
  while not SeekEof(F) do
  begin
    if SeekEoln(F) then Readln; {Переходим к следующей строке}
    Read(F,i);   {Читаем строку файла в переменную}
    Writeln(i); {Выводим содержимое переменной
                 на экран с переходом на новую строку}
  end;
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
  Close(F); {Закрываем файл}
end.
```

Функция SeekEoln

Определяет, достигнут ли конец строки в текстовом файле.

Синтаксис: SeekEoln(F)
Параметр-переменная F — файловая переменная типа Text.

Тип возвращаемого результата: Boolean.

Пример использования функции SeekEoln представлен в листинге Ж.22 (◀◀ см. предыдущий раздел).

Процедуры

Процедура Append

Открывает существующий текстовый файл для дополнения данных.

Синтаксис: Append(F)
Параметр-переменная F — файловая переменная типа Text.

Пример использования процедуры Append представлен в листинге Ж.23.

Листинг Ж.23. Программа ProcApnd.pas

```
program ProcApnd;
uses Crt;
var
  F: Text;
  s: string;
begin
```

Окончание листинга Ж.23

```
  ClrScr;          {Очищаем экран}
  Assign(F, 'test.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F);      {Создаем новый файл}
  Writeln(F, 'Первая строка'); {Записываем в файл с
                                переходом на новую строку}
  Close(F);        {Закрываем файл}
  Append(F);       {Добавляем данные к концу файла}
  Writeln(F, 'Вторая строка');
  Close(F);
  Reset(F);        {Открываем файл}
  Writeln('Содержимое файла test.txt:');
  while not Eof(F) do
  begin
    Readln(F,s);  {Читаем строку файла в переменную}
    Writeln(s);   {Выводим содержимое переменной
                   на экран с переходом на новую строку}
  end;
  Write('Нажмите любую клавишу... ');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Assign

Связывает имя внешнего файла с файловой переменной.

Синтаксис: `Assign(F, Name)`

Параметр-переменная F — файловая переменная любого файлового типа.

Параметр Name — выражение строкового типа (или типа PChar в случае использование расширенного синтаксиса).

Пример использования процедуры Assign при создании пустых файлов представлен в листинге Ж.24. Параметр Name может быть именем файла, например, `'test1.txt'`, и в этом случае файл будет создан в каталоге, в котором находится откомпилированный exe-файл программы или файл turbo.exe, запускающий интегрированную среду Turbo Pascal. Чтобы файл был создан в другом каталоге (уже существующем на диске, например, `c:\ex`), путь к нему нужно указать перед названием файла, например, `'c:\ex\test1.txt'`.

Листинг Ж.24. Программа `ProcAsgn.pas`

```
program ProcAsgn;
var
  F1: Text;
  F2: File of Integer;
  F3: File;
begin
  {Связываем имена файлов с файловыми переменными}
  Assign(F1, 'test1.txt');
  Assign(F2, 'test2.int');
  Assign(F3, 'test3.dat');
  {Создаем новые файлы}
  Rewrite(F1);
  Rewrite(F2);
```

Окончание листинга Ж.24

```
  Rewrite(F3);
  {Закрываем файлы}
  Close(F1);
  Close(F2);
  Close(F3);
end.
```

Процедура BlockRead

Считывает из нетипизированного файла в переменную одну или более записей.

Синтаксис: BlockRead(F, Buf, Count, Result)

Параметр-переменная F — нетипизированная файловая переменная.

Параметр-переменная Buf — любая переменная, в которой будут сохраняться считанные записи.

Параметр Count — значение типа Word, определяющее количество считываемых записей.

Параметр-переменная Result — необязательный параметр типа Word, в котором возвращается фактическое количество считанных записей.

Пример использования процедуры BlockRead при копировании файлов представлен в листинге Ж.25.

Листинг Ж.25. Программа ProcBlRd.pas

```
program ProcBlRd;
var
  F: Text;
  FFrom, FTo: File;
  c: char;
begin
  {Создаем файл-источник}
  Assign(F,'test1.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F); {Создаем и открываем новый файл}
  Writeln(F,'Файл-источник'); {Записываем в файл текстовую
                               строку с переходом на новую строку}
  Close(F); {Закрываем файл}
  {Копируем содержимое файла в другой файл}
  Assign(FFrom,'test1.txt');
  Assign(FTo,'test2.txt');
  Reset(FFrom,1);  {Открываем файл}
  Rewrite(FTo,1);
  while not Eof(FFrom) do
  begin
    BlockRead(FFrom, c, 1); {Считываем побайтно}
    BlockWrite(FTo, c, 1);  {Записываем побайтно}
  end;
  {Закрываем файлы}
  Close(FTo);
  Close(FFrom);
end.
```

Процедура BlockWrite

Записывает в нетипизированный файл рдну или более записей на основании значения переменной.

Синтаксис: BlockWrite(F, Buf, Count, Result)

Параметр-переменная F — нетипизированная файловая переменная. .

Параметр-переменная Buf — любая переменная, значение которой записывается в файл.

Параметр Count — значение типа Word, определяющее количество записываемых записей.

Параметр-переменная Result — необязательный параметр типа Word, в котором возвращается фактическое количество записанных записей.

Пример использования процедуры BlockWrite представлен в листинге Ж.26.

Листинг Ж.26. Программа ProcBlWr.pas

```
program ProcBlWr;
var
  F: File;
  c: char;
  i: byte;
begin
  {Создаем файл-источник}
  Assign(F,'test1.dat'); {Связываем имя файла с файловой переменной}
  Rewrite(F,1); {Создаем файл и открываем его для записи по 1 байту}
  {Записываем в файл символы с кодами от 0 до 255}
  for i := 0 to 255 do
  begin
    c := Chr(i); {Сохранить символ с кодом i в переменной c}
    BlockWrite(F,c,1); {Записываем посимвольно в файл}
  end;
  Close(F); {Закрываем файл}
end.
```

Процедура Close

Закрывает открытый файл.

Синтаксис: Close(F)

Параметр-переменная F — переменная любого файлового типа.

Пример использования процедуры Close представлен в листинге Ж.27.

Листинг Ж.27. Программа ProcClos.pas

```
program ProcClos;
var
  F1: Text;
  F2: File of Integer;
  F3: File;
begin
  {Связываем имена файлов с файловыми переменными}
  Assign(F1, 'test1.txt');
  Assign(F2, 'test2.int');
  Assign(F3, 'test3.dat');
```

Окончание листинга Ж.27

```
  {Создаем новые файлы}
  Rewrite(F1);
  Rewrite(F2);
  Rewrite(F3);
  {Закрываем файлы}
  Close(F1);
  Close(F2);
  Close(F3);
end.
```

Процедура Flush

Очищает буфер текстового файла, открытого для вывода.

Синтаксис: Flush(F)

Параметр-переменная F — переменная типа Text.

Пример использования процедуры Flush представлен в листинге Ж.28.

Листинг Ж.28. Программа ProcFlsh.pas

```
program ProcFlsh;
uses Crt;
var
  F: Text;
  i: word;
begin
  ClrScr;              {Очищаем экран}
  Assign(F, 'test.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F);          {Создаем новый файл}
  i := 0; {Устанавливаем счетчик}
  Writeln;
  Writeln('Для завершения нажмите любую клавишу...');
  while not KeyPressed do
  begin
                       {Устанавливаем курсор на экране в позицию
    GotoXY(1,1) ;       с координатами (1,1) - левый верхний угол экрана}
    Write(i);          {Выводим значение счетчика на экран}
    Writeln(F,i);      {Записываем значение счетчика в файл}
    inc(i);            {Увеличиваем значение счетчика на 1}
  end;
  Flush(F);          {Очищаем буфер файла - принудительная запись на
                      диск всех записываемых при помощи Writeln данных}
end.
```

Если в программе ProcFlsh не вызвать процедуру Flush, то несколько последних значений, записанных в файл при помощи процедуры Writeln, не успеют сохраниться на диске до выхода из программы.

Процедура Read

Для типизированного файла — считывает компонент файла в переменную.

Для текстового файла — считывает одно или более значений в одну или более переменных.

Для строковых значений процедура Read считывает все символы до следующего маркера конца строки или пока функция Eof не вернет значение True.

Для целочисленных и вещественных значений процедура Read пропускает любые пробелы, символы табуляции и маркеры конца строки.

Синтаксис: `Read(F, V1, V2 ...)`

Для текстовых файлов F — параметр-переменная типа Text, для типизированных файлов F — параметр типизированного файлового типа.

V1, V2 и т.д — параметры-переменные, в которых сохраняются считанные значения.

Пример использования процедуры Read представлен в листинге Ж.29.

Листинг Ж.29. Программа `ProcRead.pas`

```
program ProcRead;
uses Crt;
var
  F: Text;
  c1, c2, c3: char;
begin
  ClrScr;      {Очищаем экран}
  Assign(F, 'test.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F); {Создаем новый файл}
  Writeln(F,'123'); {Записываем в файл}
  Reset(F);    {Открываем файл}
  Read(F,c1); {Считываем из файла в переменные}
  Read(F,c2,c3);
  Close(F);    {Закрываем файл}
  Writeln(c1,c2,c3); {Выводим на экран содержимое переменных}
  ReadKey;     {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Readln

Считывает значение из текстового файла, а затем переходит к следующей строке.

Синтаксис: `Readln(F, V1, V2 ...)`

Параметр-переменная F — переменная типа Text.

V1, V2 и т.д — параметры-переменные, в которых сохраняются считанные значения.

Пример использования процедуры Readln представлен в листинге Ж.ЗО.

Листинг Ж.ЗО. Программа `ProcRdln.pas`

```
program ProcRdln;
uses Crt;
var
  F: Text;
  c1, c2: char;
begin
  ClrScr;        {Очищаем экран}
  Assign(F, 'test.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F);    {Создаем файл}
  Writeln(F,'123'); {Записываем в файл}
  Writeln(F,'456');
  Reset(F);      {Открываем файл}
```

Окончание листинга Ж.30

```
  Readln(F,c1); {Считываем 1-й символ 1-й строки}
  Readln(F,c2); {Считываем 1-й символ 2-й строки}
  Close(F);     {Закрываем файл}
  Writeln(c1);  {Выводим на экран содержимое переменных}
  Writeln(c2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Reset

Открывает существующий файл и устанавливает файловую позицию на начало файла. Перед вызовом этой процедуры файловой переменная должна быть связана с внешним файлом при помощи процедуры Assign.

Синтаксис: Reset(F, RecSize)

Параметр-переменная F — переменная любого файлового типа.

RecSize — необязательный параметр, определяющий размер блока в байтах при работе с нетипизированными файлами, по умолчанию RecSize = 128 байт.

Пример использования процедуры Reset при выводе на экран содержимого файла представлен в листинге Ж.31.

Листинг Ж.31. Программа ProcRset.pas

```
program ProcRset;
uses
  Crt;
var
  F: File;
  c: char;
  s: string;
begin
  ClrScr;      {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  Assign(F,s); {Связываем имя файла с файловой переменной}
  Reset(F,1); {Открываем файл для считывания блоками размером 1 байт}
  while not Eof(F) do
  begin
    BlockRead(F, c, 1); {Считываем блоки в переменную c}
    Write (c); {Выводим на экран содержимое переменной c}
  end;
  Close(F);    {Закрываем файл}
  ReadKey;     {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Rewrite

Создает и открывает новый файл.

Синтаксис: Rewrite(F, RecSize)

Параметр-переменная F — переменная любого файлового типа.

RecSize — необязательный параметр, определяющий размер блока в байтах при работе с нетипизированными файлами, по умолчанию RecSize = 128 байт.

Пример использования процедуры Rewrite представлен в листинге Ж.32.

Листинг Ж.32. Программа ProcRwrt.pas

```
program ProcRwrt;
var
  F1: Text;
  F2: File of Word;
  F3: File;
  i: word;
begin
  {Связываем имена файлов с файловыми переменными}
  Assign(F1, 'test1.txt');
  Assign(F2, 'test2.int');
  Assign(F3, 'test3.dat');
  Rewrite(F1);  {Создаем и открываем файл для записи}
  Writeln(F1,'текст'); {Записываем строку текста в файл}
  Rewrite(F2);
  i := 1;
  Write(F2,i);
  Rewrite(F3,2) ; {Создаем и открываем файл для записи по 2 байта}
  for i := 1 to 10 do BlockWrite(F3,i,1);
  {Закрываем файлы}
  Close (F1);
  Close(F2) ;
  Close(F3);
end.
```

Процедура Seek

Перемещает текущую файловую позицию к указанному компоненту.

Синтаксис: Seek(F, N)

Параметр-переменная F — переменная любого файлового типа, кроме Text.

Параметр N — выражение типа Longlnt, определяющее номер компонента, к которому необходимо переместиться (начиная с 0).

Пример использования процедуры Seek представлен в листинге Ж.33.

Листинг Ж.33. Программа ProcSeek.pas

```
program ProcSeek;
uses Crt;
var
  F: File of Byte;
  s: string;
  c: byte;
  procedure ShowChar(N: Longint);
  begin
    Seek(F, N);
    Read(F, c);
    Writeln (Chr(c));
  end;

begin
  ClrScr;  {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
```

Окончание листинга Ж.33

```
  Assign(F, s); {Связываем имя файла с файловой переменной}
  Reset(F); {Открываем файл}
  Writeln('Первый символ: ');
  ShowChar(0);
  Writeln('Средний символ: ');
  ShowChar(FileSize(F) div 2);
  Writeln('Последний символ: ');
  ShowChar(FileSize(F)-1);
  Close(F); {Закрываем файл}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура SetTextBuf

Назначает текстовому файлу буфер ввода/вывода.

Синтаксис: SetTextBuf(F, Buf, Size)

Параметр-переменная F — переменная типа Text.

Параметр-переменная Buf — буфер для хранения данных, считываемых из файла или записываемых в файл.

Size — необязательный параметр типа Word, определяющий объем считываемых/записываемых данных за один раз.

Пример использования процедуры SetTextBuf представлен в листинге Ж.34.

Листинг Ж.34. Программа ProcStTB.pas

```
program ProcStTB;
uses Crt;
var
  F: Text;
  c: Char;
  Buf: array [1..4095] of Char; {4Кб буфер}
  s: string;
begin
  ClrScr; {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  Assign(F,s); {Связываем имя файла с файловой переменной}
  {Определяем буфер для ускорения чтения}
  SetTextBuf(F,Buf);
  Reset(F); {Открываем файл}
  {Выводим содержимое файла на экран}
  while not Eof(F) do
  begin
    Read(F,c); {Читаем строку из файла в переменную}
    Writeln(c); {Выводим на экран содержимое переменной}
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Truncate

Удаляет часть файла, начиная с текущей файловой позиции.

Синтаксис: Truncate(F)

Параметр-переменная F — переменная любого файлового типа, кроме Text.

Пример использования процедуры Truncate представлен в листинге Ж.35.

Листинг Ж.35. Программа ProcTrun.pas

```
program ProcTrun;
uses Crt;
var
  F: File of word;
  i: word;
begin
  ClrScr; {Очищаем экран}
  Assign(F,'test.dat'); {Связываем имя файла с файловой переменной}
  Rewrite(F); {Создаем файл и открываем для записи}
  {Заполняем файл числами от 1 до 10}
  for i := 1 to 10 do Write(F,i);
  Seek(F,5); {Перемещаемся к середине файла}
  Truncate(F); {Удаляем вторую половину}
  Reset(F); {Открываем файл и перемещаемся в его начало}
  {Выводим содержимое урезанного файла на экран}
  while not Eof(F) do
  begin
    Read(F,i); {Читаем строку из файла в переменную}
    Writeln(i); {Выводим на экран содержимое переменной}
  end;
  Close(F); {Закрываем файл}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Write

Для типизированных файлов — записывает значение переменной в компонент файла.

Для текстовых файлов — записывает в файл одно или более значений.

Синтаксис: Write (F, V1, V2, ...)

Для текстовых файлов F — параметр-переменная типа Text, для типизированных файлов — параметр типизированного файлового типа.

V1, V2 и т.д. — переменные, значения которых записываются в файл.

Пример использования процедуры Write представлен в листинге Ж.36.

Листинг Ж.36. Программа ProcWrit.pas

```
program ProcWrit;
var
  F1: File of word;
  F2: Text;
  i: word;
begin
  {Связываем имена файлов с файловыми переменными}
  Assign(F1,'test.dat');
  Assign(F2,'test.txt');
  {Создаем файлы и открываем их для записи}
  Rewrite(F1);
```

Окончание листинга Ж.36

```
  Rewrite(F2);
  i := 1000;
  {Записываем данные в файлы}
  Write(F1,i);
  Write(F2,i,i+1);
  {Закрываем файлы}
  Close(F1);
  Close(F2);
end.
```

Процедура WriteIn

Записывает одно или более значений в текстовый файл, а затем выполняет переход на новую строку.

Синтаксис: Write(F, V1, V2, ...)

Параметр-переменная F — файловая переменная типа Text.

V1, V2 и т.д. — переменные, значения которых записываются в файл.

Пример использования процедуры WriteIn представлен в листинге Ж.37.

Листинг Ж.37. Программа ProcWrtl.pas

```
program ProcWrtl;
var
  F: Text;
  i: word;
begin
  Assign(F,'test.txt'); {Связываем имя файла с файловой переменной}
  Rewrite(F); {Создаем файл}
  for i := 1 to 100 do Writeln(F,i,i+1); {Записываем в файл}
  Close (F);   {Закрываем файл}
end.
```

Работа с файловой системой

Процедуры

Процедура ChDir

Изменяет текущий каталог.

Синтаксис: ChDir (S)

Параметр s — строковое значение типа String, определяющее имя каталога, который должен стать текущим.

Пример использования процедуры ChDir представлен в листинге Ж.38.

Листинг Ж.38. Программа ProcCDir.pas

```
program ProcCDir;
uses Crt;
var
  s: string;
```

Окончание листинга Ж.38

```
  procedure ShowCurDir;
  begin
    GetDir(0,s); {Определяем текущий каталог}
    Writeln('Текущий каталог: ', s) ;
  end;

begin
  ClrScr;   {Очищаем экран}
  ShowCurDir;
  Write('Укажите каталог, который необходимо сделать текущим: ');
  Readln(s);
  ChDir(s); {Изменяем текущий каталог}
  ShowCurDir;
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Erase

Удаляет внешний файл на диске.

Синтаксис: Erase(F)

Параметр F — файловая переменная любого файлового типа. При вызове процедуры Erase файл, связанный с переменной F, обязательно должен быть закрыт.

Пример использования процедуры Erase представлен в листинге Ж.39.

Листинг Ж.39. Программа ProcEras.pas

```
program ProcEras;
var
  F: File;
begin
  {Связываем имя файла с файловой переменной}
  Assign(F,'test.dat');
  Rewrite(F);  {Создаем файл}
  Close(F);    {Закрываем файл}
  Erase(F);    {Удаляем файл}
end.
```

Процедура GetDir

Возвращает текущий каталог указанного диска.

Синтаксис: GetDir(D,S)

Параметр D — значение типа Byte, задающее номер диска, для которого определяется текущий каталог: 0 — текущий диск, 1 — диск A, 2 — диск B, 3 — диск C и т.д.

Параметр-переменная S — строковая переменная типа String, в которой сохраняется имя текущего каталога

Пример использования процедуры GetDir представлен в листинге Ж.40.

Листинг Ж.40. Программа ProcGDir.pas

```
program ProcGDir;
uses Crt;
var
  s: string;
```

Окончание листинга Ж.40

```
  procedure  ShowCurDir;
  begin
    GetDir(0,s); {Определяем текущий каталог}
    Writeln('Текущий каталог: ', s);
  end;
begin
  ClrScr;    {Очищаем экран}
  ShowCurDir;
  Write('Укажите каталог, который необходимо сделать текущим: ');
  Readln(s);
  ChDir(s); {Изменяем текущий каталог}
  ShowCurDir;
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура MkDir

Создает подкаталог.

Синтаксис: MkDir(S)

Параметр S — строковая переменная типа String, определяющая имя создаваемого подкаталога.

Пример использования процедуры MkDir представлен в листинге Ж.41.

Листинг Ж.41. Программа ProcMDir.pas

```
program ProcMDir;
uses Crt;
yar
  s: string;
begin
  ClrScr;       {Очищаем экран)
  GetDir(0,s); {Определяем текущий каталог)
  Writeln('Текущий каталог: ', s);
  Write('Укажите подкаталог, который необходимо создать: ');
  Readln(s);
  MkDir(s);     {Создаем подкаталог}
end.
```

Процедура Rename

Переименовывает внешний файл на диске.

Синтаксис: Rename (F, NewName)

Параметр F — файловая переменная любого файлового типа.

Параметр NewName — строковое значение типа String или PChar, если включен расширенный синтаксис.

После вызовы процедуры Rename все операции, выполняемые с файловой переменной F, применяются к файлу с новым именем NewName.

Пример использования процедуры Rename представлен в листинге Ж.42.

Листинг Ж.42. Программа ProcRenm.pas

```
program ProcRenm;
uses Crt;
```

Окончание листинга Ж.42

```
var
  F: Text;
  s: string;
begin
  ClrScr;        {Очищаем экран}
  Assign(F, 'test.dat');
  Rewrite(F);    {Создаем файл}
  Writeln(F,1); {Записываем данные в новый файл}
  Write('Укажите новое имя файла test.dat: ');
  Readln(s);
  Rename(F,s);   {Переименовываем файл}
  Writeln(F,2); {Записываем данные в файл с новым именем}
  Close(F);      {Закрываем файл}
end.
```

Процедура RmDir

Удаляет пустой каталог.

Синтаксис: RmDir(S)

Параметр S — строковая переменная типа String, определяющая имя удаляемого каталога.

Пример использования процедуры RmDir представлен в листинге Ж.43.

Листинг Ж.43. Программа ProcRDir.pas

```
program ProcRDir;
uses Crt;
var
  s: string;
begin
  ClrScr;    {Очищаем экран}
  Write('Введите имя создаваемого подкаталога: ');
  Readln(s);
  MkDir(s); {Создаем подкаталог}
  RmDir(s); {Удаляем подкаталог}
end.
```

Работа с памятью и указателями

Функции

Функция Addr

Возвращает адрес объекта, переданного в качестве параметра.

Синтаксис: Address(X)

Параметр X — идентификатор переменной, процедуры или функции.

Тип возвращаемого результата: указатель на объект X.

Пример использования функции Addr представлен в листинге Ж.44.

Листинг Ж.44. Программа FuncAddr.pas

```
program FuncAddr;
uses Crt;
```

Окончание листинга Ж.44

```
var
  i: integer;
  P: ^Integer;
begin
  ClrScr;          {Очищаем экран}
  i := 1;
  P := Addr(i); {Определяем указатель на i}
  pл := P^ + 1; {Равнозначно i:=i+1}
  Writeln(i);    {Будет выведено 2}
  ReadKey;       {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Assigned

Проверяет, является ли передаваемый в качестве параметра указатель или процедурная переменная значением nil.

Синтаксис: Assigned(P)

Параметр-переменная P — ссылка на переменную указательного или процедурного типа.

Тип возвращаемого результата: Boolean.

Пример использования функции Assigned представлен в листинге Ж.45.

Листинг Ж.45. Программа FuncAsnd.pas

```
program FuncAsnd;
uses Crt;
var
  i: Integer;
  P: ^Integer;

  procedure IsAssigned;
  begin
    if Assigned(P)
    then Writeln('Указатель P назначен')
    else Writeln('Указатель P не назначен');
  end;

begin
  ClrScr;          {Очищаем экран}
  IsAssigned;
  P := Addr(i); {Определяем указатель на i}
  IsAssigned;
  ReadKey;       {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция CSeg

Возвращает текущее содержимое регистра CS.

Синтаксис: CSeg

Тип возвращаемого результата: Word.

Пример использования функции CSeg представлен в листинге Ж.46.

Листинг Ж.46. Программа FuncCSeg.pas

```
program FuncCSeg;
uses Crt;
  {Вывод числа в шестнадцатеричном формате}
  procedure WriteHexWord(W : Word);
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    Write(HexChars[Hi(W) shr 4],   {4-й разряд}
          HexChars[Hi(W) and $F] , {3-й разряд}
          HexChars[Lo(W) shr 4],   {2-й разряд}
          HexChars[Lo(W) and $F]); {1-й разряд}
  end;
begin
  ClrScr;  {Очищаем экран}
  Write('Текущий сегмент кода: $');
  WriteHexWord(CSeg);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция DSeg

Возвращает текущее содержимое регистра DS.

Синтаксис: DSeg

Тип возвращаемого результата: Word.

Пример использования функции DSeg представлен в листинге Ж.47.

Листинг Ж.47. Программа FuncDSeg.pas

```
program FuncDSeg;
uses Crt;
  {Вывод числа в шестнадцатеричном формате}
  procedure WriteHexWord(W : Word);
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    Write(HexChars[Hi(W) shr 4],   {4-й разряд}
          HexChars[Hi(W) and $F],  {3-й разряд}
          HexChars[Lo(W) shr 4],   {2-й разряд}
          HexChars[Lo(W) and $F]); {1-й разряд}
  end;
begin
  ClrScr;  {Очищаем экран}
  Write('Сегмент данных: $');
  WriteHexWord(DSeg);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Hi

Возвращает старший байт принятого параметра.

Синтаксис: Hi(X)

Тип возвращаемого результата: Byte.

Параметр X — выражение типа Integer или Word.

Пример использования функции Hi для представления числа в шестнадцатеричной форме представлен в листинге Ж.48.

Листинг Ж.48. Программа FuncHi.pas

```
program FuncHi;
uses Crt;
var
  i: Word;

  {Представление числа в шестнадцатеричном формате}
  function DecToHex(n: Word): string;
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    DecToHex := HexChars[Hi(n) shr 4] +
                HexChars[Hi(n) and $F] +
                HexChars[Lo(n) shr 4] +
                HexChars[Lo(n) and $F];
  end;

begin
  ClrScr;  {Очищаем экран}
  Write('Введите целое число: ');
  Readln(i);
  Writeln(i,' = $',DecToHex(i));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Lo

Возвращает младший байт принятого параметра.

Синтаксис: Lo(X)

Тип возвращаемого результата: Byte.

Параметр X — выражение типа Integer или Word.

Пример использования функции Lo для представления числа в шестнадцатеричной форме представлен в листинге Ж.49.

Листинг Ж.49. Программа FuncLo.pas

```
program FuncLo;
uses Crt;
var
  i: Word;

  (Представление числа в шестнадцатеричном формате}
  function DecToHex(n: Word): string;
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    DecToHex := HexChars[Hi(n) shr 4] +
                HexChars[Hi(n) and $F] +
                HexChars[Lo(n) shr 4] +
```

Окончание листинга Ж.49

```
                HexChars[Lo(n) and $F];
  end;
begin
  ClrScr;   {Очищаем экран}
  Write('Введите целое число: ');
  Readln(i);
  Writeln(i,' = $',DecToHex(i));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция MaxAvail

Возвращает размер самого большого непрерывного блока памяти в *куче*.

ПРИМЕЧАНИЕ

***Куча* — область памяти, отводимая программе для размещения данных, объем которых неизвестен до начала ее выполнения. Программа может запрашивать из *кучи* свободную память для хранения таких данных, использовать эту память по своему усмотрению, а затем освобождать ее. В языках программирования C и Pascal предусмотрены функции и процедуры запроса и освобождения памяти из *кучи*. В отличие от другой резервируемой области памяти, называемой стеком, куча распределяется блоками разных размеров, в соответствии с нуждами программы, предоставляемыми из разных мест *кучи* — отовсюду, где найдется блок подходящего размера. По мере выполнения программы усиливается фрагментация *кучи* и возникает необходимость в проведении так называемого уплотнения *кучи*, при котором маленькие блоки сливаются в более крупные области, что позволяет эффективней использовать память.**

Синтаксис: MaxAvail

Тип возвращаемого результата: Longint.

Пример использования функции MaxAvail представлен в листинге Ж.50.

Листинг Ж.50. Программа FuncMaxA.pas

```
program FuncMaxA;
uses Crt;
type
  MyRec = record
    Num: Longint;
    Name: array[1..3] of string;
  end;
begin
  ClrScr;   {Очищаем экран}
  Writeln('Максимально большой свободный блок в куче: ',
          MaxAvail, ' байт');
  if MaxAvail < SizeOF(MyRec)
  then Write('Недостаточно ')
  else Write('Достаточно ');
  Writeln('для размещения переменной типа MyRec');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция MemAvail

Возвращает объем всей свободной памяти в куче.

◀◀ см. о куче в примечании из предыдущего раздела.

Синтаксис: MemAvail

Тип возвращаемого результата: Longint.

Пример использования функции MemAvail представлен в листинге Ж.51.

Листинг Ж.51. Программа FuncMemA.pas

```
program FuncMemA;
uses Crt;
begin
  ClrScr;   {Очищаем экран}
  Writeln('В куче свободно ', MemAvail, ' байт');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Ofs

Возвращает смещение указанного объекта.

Синтаксис: Ofs(X)

Параметр X — переменная или идентификатор процедуры или функции.

Тип возвращаемого результата: Word.

Пример использования функции Ofs представлен в листинге Ж.52.

Листинг Ж.52. Программа FuncOfs.pas

```
program FuncOfs;
uses Crt;
var
  i: Integer;
  {Вывод числа в шестнадцатеричном формате}
  procedure WriteHexWord(W : Word);
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    Write(HexChars[Hi(W) shr 4],   {4-й разряд}
          HexChars[Hi(W) and $F],  {3-й разряд}
          HexChars[Lo(W) shr 4],   {2-й разряд}
          HexChars[Lo(W) and $F]); {1-й разряд}
  end;

begin
  ClrScr;   {Очищаем экран}
  Write('Переменная i по смещению - $');
  WriteHexWord(Ofs(i));
  Write (' в сегменте $');
  WriteHexWord(Seg(i));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Ptr

Преобразует адрес, состоящий из сегмента и смещения, в указатель.

Синтаксис: `Ptr(Seg,Ofs)`

Параметр Seg — значение типа Word, определяющее сегментную часть адреса. Параметр Ofs — значение типа `Word`, определяющее смещение.

Тип возвращаемого результата: указатель.

Пример использования функции Ptr представлен в листинге Ж.53.

Листинг Ж.53. Программа `FuncPtr.pas`

```
program FuncPtr;
uses Crt;
var
  P: ^Byte;
  Mode: string;
begin
  ClrScr;  {Очищаем экран}
  P := Ptr($40, $49);
  case P^ of
    0: Mode := 'Черно-белый 40x25';
    1: Mode := 'Цветной 40x25';
    2: Mode := 'Черно-белый 80x25';
    3: Mode := 'Цветной 80x25';
  end;
  WriteLn('Текущий видеорежим - ', Mode);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Seg

Возвращает сегментную часть адреса указанного объекта.

Синтаксис: Seg (X)

Параметр X — переменная или идентификатор процедуры или функции.

Тип возвращаемого результата: `Word`.

Пример использования функции Seg представлен в листинге Ж.52 (◀◀ см. раздел, посвященный функции Ofs).

Функция SizeOf

Возвращает количество байтов, занимаемых в памяти принятым параметром.

Синтаксис: SizeOf (X)

Параметр X — идентификатор переменной или имя типа.

Тип возвращаемого результата: Integer.

Пример использования функции SizeOf представлен в листинге Ж.54.

Листинг Ж.54. Программа FuncSzOf.pas

```
program FuncSzOf;
uses Crt;
type
  MyRec = record
```

Окончание листинга Ж.54

```
    F1: integer;
    F2: string;
  end;
var
  c: char;
  i: integer;
  r: real;
  s: string;
  Rec: MyRec;

begin
  ClrScr;  {Очищаем экран}
  Writeln('Переменная c занимает в памяти ', SizeOf(c),' байт');
  Writeln('Переменная i занимает в памяти ', SizeOf(i),' байта');
  Writeln('Переменная r занимает в памяти ', SizeOf(r),' байт');
  Writeln('Переменная s занимает в памяти ', SizeOf(s),' байтов');
  Writeln('Переменная Rec занимает в памяти ', SizeOf(Rec),' байтов');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция SPtr

Возвращает текущее содержимое регистра SP.

Синтаксис: `SPtr`

Тип возвращаемого результата: Word.

Пример использования функции SPtr представлен в листинге Ж.55.

Листинг Ж.55. Программа `FuncSPtr.pas`

```
program FuncSPtr;
uses Crt;
var
  i: Integer;

  {Вывод числа в шестнадцатеричном формате}
  procedure WriteHexWord(W : Word) ;
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    Write(HexChars[Hi(W) shr 4],   {4-й разряд}
          HexChars[Hi(W) and $F],  {3-й разряд}
          HexChars[Lo(W) shr 4],   {2-й разряд}
          HexChars[Lo(W) and $F]); {1-й разряд}
  end;

begin
  ClrScr;  {Очищаем экран}
  Write('Указатель на стек: $');
  WriteHexWord(SPtr);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция SSeg

Возвращает текущее содержимое регистра SS.

Синтаксис: SSeg

Тип возвращаемого результата: Word.

Пример использования функции SSeg представлен в листинге Ж.56.

Листинг **Ж.56.** Программа FuncSSeg.pas

```
program FuncSSeg;
uses Crt;
var
  i: Integer;

  {Вывод числа в шестнадцатеричном формате}
  procedure WriteHexWord(W : Word);
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    Write(HexChars[Hi(W) shr 4],   {4-й разряд}
          HexChars[Hi(W) and $F],  {3-й разряд}
          HexChars[Lo(W) shr 4],   {2-й разряд}
          HexChars[Lo(W) and $F]); {1-й разряд}
  end;

begin
  ClrScr;  {Очищаем экран}
  Write('Сегмент стека $');
  WriteHexWord(SSeg);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Swap

Меняет местами старший и младший байты параметра.

Синтаксис: Swap(X)
Параметр X — выражение типа Integer или Word.

Тип возвращаемого результата: совпадает с типом параметра.

Пример использования функции Swap представлен в листинге Ж.57.

Листинг Ж.57. Программа FuncSwap.pas

```
program FuncSwap;
uses Crt;
var
  i: word;
  {Представление в шестнадцатеричном формате}
  function DecToHex(n: Word): string;
  const
    HexChars: array [0..15] of Char = '0123456789ABCDEF';
  begin
    DecToHex := HexChars[Hi(n) shr 4] +
                HexChars[Hi(n) and $F] +
                HexChars[Lo(n) shr 4] +
```

Окончание листинга Ж.57

```
                HexChars[Lo(n) and $F];
  end;

begin
  ClrScr;  {Очищаем экран}
  Write('Введите целое число: ');
  Readln(i);
  Writeln('До вызова Swap i = $ ',DecToHex(i) ) ;
  Writeln('После вызова Swap i = $',DecToHex(Swap(i)));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура Dispose

Уничтожает динамическую переменную.

Синтаксис: `Dispose(P, Destructor)`
Параметр-переменная P — указатель.
Необязательный параметр Destructor — идентификатор деструктора для уничтожения динамических объектов.

Пример использования функции Dispose представлен в листинге Ж.58.

Листинг Ж.58. Программа `ProcDisp.pas`

```
program ProcDisp;
uses Crt;
var
  P: ^integer;
begin
  ClrScr;  {Очищаем экран}
  New(P);  {Создаем динамическую переменную}
  P^ := 100;
  Writeln(P^);
  Dispose(P);  {Уничтожаем динамическую переменную}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура FillChar

Заполняет побайтно область памяти заданным значением.

Синтаксис: `FillChar(X, Count, Value)`
Параметр-переменная X — буфер, который необходимо заполнить.
Параметр Count — значение типа Word, определяющее количество символов.
Параметр Value — заполнитель типа Byte или Char.

Пример использования функции FillChar представлен в листинге Ж.59.

Листинг Ж.59. Программа `ProcFChr.pas`

```
program ProcFChr;
uses Crt;
var
```

Окончание листинга Ж.59

```
  s: string;
  i: integer;
begin
  ClrScr;  {Очищаем экран}
  for i := 10 downto 1 do
  begin
    {Заполняем первые i символов строки}
    FillChar(s, i+1, Chr(i+64));
    s[0] := Chr(i); {Устанавливаем длину строки}
    Writeln(s);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура FreeMem

Освобождает память, занятую динамической переменной указанного размера.

Синтаксис: FreeMem(P, Size)

Параметр-переменная P — переменная любого указательного типа.

Параметр Size — значение типа Word, определяющее размер динамической переменной в байтах.

Пример использования функции FreeMem представлен в листинге Ж.60.

Листинг Ж.60. Программа ProcFMem.pas

```
program ProcFMem;
uses Crt;
type
  MyRec =- record
    Num: Longint;
    Name: array[1..3] of string;
  end;
var
  P: ^MyRec;
begin
  ClrScr;  {Очищаем экран}
  GetMem(P, SizeOf(MyRec)); {Распределяем память}
  with P^ do
  begin
    Num := 1;
  . Name[1] := 'Шпак';
    Name[2] := 'Юрий';
    Name[3] := 'Алексеевич';
    Writeln(Num,' - ',Name[1],' ',Name[2],' ' ,Name[3]);
  end;
  FreeMem(P, SizeOf(MyRec)); {Освобождаем память}
  ReadKey; (Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура GetMem

Создает динамическую переменную указанного размера и помещает ее адрес в переменную-указатель. При вызове этой процедуры в куче должно быть добтаточно мес-

та для размещения динамической переменной, в противном случае во время выполнения программы возникнет ошибка.

◀◀ см. о куче в примечании из раздела о функции MaxAvail.

Синтаксис: `GetMem(P, Size)`

Параметр-переменная P — переменная любого указательного типа.

Параметр Size — значение типа Word, определяющее размер динамической переменной в байтах.

Пример использования функции GetMem представлен в листинге Ж.61.

Листинг Ж.61. Программа `ProcGMem.pas`

```
program ProcGMem;
uses Crt;
var
  P: ^Integer;
begin
  ClrScr;  {Очищаем экран}
  if MaxAvail >= SizeOf(Integer) then {Если в куче
    достаточно места для размещения переменной}
  begin
    GetMem(P, SizeOf(Integer)); {Распределяем память}
    P^ := 1;
    Writeln(P^);
    FreeMem(P, SizeOf(Integer)); {Освобождаем память}
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура New

Создает новую динамическую переменную и устанавливает на нее указатель.

Синтаксис: `New(P, Constructor)`

Параметр-переменная P — переменная любого указательного типа.

Параметр Conctructor — вызов конструктора при создании динамических объектов.

Пример использования функции New представлен в листинге Ж.62.

Листинг Ж.62. Программа `ProcNew.pas`

```
program ProcNew;
uses Crt;
type
  PObj = ^MyObj;
  MyObj = Object
    constructor Init(n: integer);
    procedure ShowX;
    private
      X: integer;
  end;

  constructor MyObj.Init(n: integer);
  begin
```

Окончание листинга Ж.60

```
    X := n;
  end;

  procedure MyObj.ShowX;
  begin
    Writeln(X);
  end;

var
  P: PObj;
begin
  ClrScr;   {Очищаем экран}
  New(P, Init(10)); {Создаем динамический объект}
  P^.ShowX;
  Dispose(P); {Уничтожаем динамическкий объект}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Mark

Записывает состояние кучи в указатель.

◀◀ см. о куче в примечании из раздела о функции MaxAvail.

Синтаксис: Mark(P)

Параметр-переменная P — переменная любого указательного типа.

Пример использования функции Mark представлен в листинге Ж.63.

Листинг Ж.63. Программа ProcMark.pas

```
program ProcMark;
var
  P : Pointer;
  P1, P2, P3 : ^Integer;
begin
  New(P1); {Распределяем память под число типа Integer}
  Mark(P); {Сохраняем состояние кучи}
  New(P2); {Распределяем память под еще два числа типа Integer}
  New(P3);
  Release(P); {Память, резервированная для P2^ и P3^ освобождается,
               а память для P1^ все еще может быть использована}
end.
```

Процедура Move

Копирует содержимое указанной области памяти в другую область памяти.

Синтаксис: Move(Source, Dest, Count)

Параметр-переменная Source — источник.

Параметр-переменная Dest — получатель.

Параметр Count — значение типа Word, определяющее количество копируемых байтов.

Пример использования функции Move представлен в листинге Ж.64.

Листинг Ж.64. Программа ProcMove.pas

```
program ProcMove;
uses Crt;
var
  s1, s2: string;
  c: Word;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(s1);
  Write('Сколько символов скопировать? ');
  Readln(c);
  Move(s1,s2,c+1); {Копируем строку}
  s2[0] := Chr(c); {Устанавливаем длину строки s2}
  Write('Скопированная строка: ');
  Write(s2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Release

Возвращает кучу в заданное состояние.

◀◀ см. о куче в примечании из раздела о функции MaxAvail.

Синтаксис: Release(P)

Параметр-переменная P — переменная любого указательного типа.

Пример использования функции Release представлен в листинге Ж.63 (◀◀ см. раздел, по священный функции Mark).

Работа со значениями простых типов

Функции

Функция Odd

Возвращает значение True, если принятый параметр четный.

Синтаксис: Odd(X)

Параметр X — выражение типа Longint.

Тип возвращаемого результата: Boolean.

Пример использования функции Odd представлен в листинге Ж.65.

Листинг Ж.65. Программа FuncOdd.pas

```
program FuncOdd;
uses Crt;
var
  i: Word;
begin
  ClrScr;  {Очищаем экран}
  for i := 1 to 5 do
  begin
    Write(i,' - ');
```

Окончание листинга Ж.65

```
    if Odd(i)
    then Writeln('нечетное')
    else Writeln('четное');
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Pred

Возвращает значение, предшествующее значению принятого параметра для данного типа.

Синтаксис: `Pred(X)`
Параметр X — выражение перечислимого типа.

Тип возвращаемого результата: совпадает с типом принятого параметра.

Пример использования функции Pred представлен в листинге Ж.66.

Листинг Ж.66. Программа `FuncPred.pas`

```
program FuncPred;
uses Crt;
var
  i: Word;
  c: char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите целое число: ');
  Readln(i);
  Writeln('Предшественник ',i,' - ',Pred(i));
  Write('Введите символ: ');
  Readln(c);
  Writeln('Предшественник ',c,' - ',Pred(c));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Succ

Возвращает значение, следующее для данного типа после значения принятого параметра.

Синтаксис: `Succ(X)`
Параметр X — выражение перечислимого типа.

Тип возвращаемого результата: совпадает с типом принятого параметра.

Пример использования функции Succ представлен в листинге Ж.67.

Листинг Ж.67. Программа `FuncSucc.pas`

```
program FuncSucc;
uses Crt;
var
  i: Word;
  c: char;
begin
```

Окончание листинга Ж.67

```
  ClrScr;  {Очищаем экран}
  Write('Введите целое число: ');
  Readln(i);
  Writeln('После ',i,' следует ',Succ(i));
  Write('Введите символ: ');
  Readln(c);
  Writeln('После ',c,' следует ',Succ(c));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция UpCase

Преобразует символ в верхний регистр.

Синтаксис: UpCase (Ch)
Параметр Ch — значение типа Char.

Тип возвращаемого результата: Char.

Пример использования функции UpCase представлен в листинге Ж.68.

Листинг Ж.68. Программа FuncUpCs.pas

```
program FuncUpCs;
uses Crt;
var
  i: Byte;
  s: string;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Введите строку:');
  Readln(s);
  for i := 1 to Length(s) do s[i] := UpCase(s[i]);
  Writeln('Результирующая строка:');
  Writeln(s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура Dec

Уменьшает принятый параметр на указанное число.

Синтаксис: Dec(X, N)
Параметр-переменная X — переменная перечислимого типа или типа PChar, если используется расширенный синтаксис.
Необязательный параметр N — значение типа Longint, определяющее, на сколько должно быть уменьшено значение X (по умолчанию параметр N = 1).

Пример использования процедуры Dec представлен в листинге Ж.69.

Листинг Ж.69. Программа ProcDec.pas

```
program ProcDec;
uses Crt;
var
```

Окончание листинга Ж.69

```
  i: Byte;
begin
  ClrScr;  {Очищаем экран}
  i := 10;
  {Вывод на экран десяти цифр в порядке убывания с 10 по 1}
  while i > 0 do
  begin
    Writeln(i);
    Dec(i);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Inc

Увеличивает принятый параметр на указанное число.

Синтаксис: Inc (X, N)

Параметр-переменная X — переменная перечислимого типа или типа PChar, если используется расширенный синтаксис.

Необязательный параметр N — значение типа Longlnt, определяющее, на сколько должно быть увеличено значение X (по умолчанию параметр N = 1).

Пример использования процедуры Inc представлен в листинге Ж.70.

Листинг Ж.70. Программа `ProcInc.pas`

```
program Proclnc;
uses Crt;
var
  i: Byte;
begin
  ClrScr;  {Очищаем экран}
  i := 0;
  {Вывод на экран десяти цифр в порядке возрастания с 0 по 9}
  while i < 10 do
  begin
    Writeln(i);
    Inc(i);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Работа со строками

Функции

Функция Concat

Объединяет последовательность строк в одну строку.

Синтаксис: Concat (S1, S2, ...)

Параметры S1, S2 и т.д. — значения типа String.

Тип возвращаемого результата: string.

Пример использования функции Concat представлен в листинге Ж.71.

Листинг Ж.71. Программа FuncConc.pas

```
program FuncConc;
uses Crt;
var
  A: array[1..3] of string;
  i: byte;
begin
  ClrScr;  {Очищаем экран}
  for i := 1 to 3 do
  begin
    Write(i,'-е слово: ');
    Readln(A[i]);
  end;
  Write('Полученная строка: ');
  Write(Concat(A[1],' ',A[2],' ',A[3]));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Copy

Возвращает подстроку.

Синтаксис; Copy(S, Index, Count)

Параметр S — значение типа String.

Параметр Index — значение типа Integer, определяющее номер первого символа подстроки в строке S.

Параметр Count — значение типа Integer, определяющее длину подстроки.

Тип возвращаемого результата: String.

Пример использования функции Copy представлен в листинге Ж.72.

Листинг Ж.72. Программа FuncCopy.pas

```
program FuncCopy;
uses Crt;
var
  A: array[1..3] of string;
  i: byte;
  s: string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку из трех слов: ');
  Readln(s);
  for i := 1 to 2 do
  begin
    A[i] := copy(s,1,pos(' ',s)-1);
    Delete(s,1,pos(' ',s)); {Удаляем i-е слово из строки}
  end;
  A[3] := s; {Последнее слово в строке}
  for i := 1 to 3 do
    Writeln(i,'-е слово: ',A[i]);
```

Окончание листинга Ж.72

```
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Length

Возвращает длину строки.

Синтаксис: Length(S)
Параметр S — значение типа String.

Тип возвращаемого результата: Integer.

Пример использования функции Length представлен в листинге Ж.73.

Листинг Ж.73. Программа FuncLeng.pas

```
program FuncLeng;
uses Crt;
var
  s: string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(s);
  Write('Строка состоит из ',Length(s),' символов.');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Pos

Возвращает позицию первого вхождения подстроки в строку.

Синтаксис: Pos(Substr, S)
Параметр Substr — искомая подстрока типа String.
Параметр S — строка, в которой выполняется поиск, типа string.

Тип возвращаемого результата: Byte.

Пример использования функции Pos представлен в листинге Ж.74.

Листинг Ж.74. Программа FuncPos.pas

```
program FuncPos;
uses Crt;
var
  s: string;
  CurChar: Char;
  c: Word;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(s);
  while Length(s) > 0 do
  begin
    CurChar := s[1]; {Получаем первый символ строки}
    c := 0;
    while (pos(CurChar,s) > 0) do {Пока в строке}
```

Окончание листинга Ж.74

```
    begin                          {встречается данный символ)
      Inc(c); {Увеличиваем счетчик символа и}
      Delete(s,pos(CurChar,s),1); {удаляем символ}
    end;
    Writeln('Символов "',CurChar,'" - ',c);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура Delete

Удаляет из строки подстроку.

Синтаксис: `Delete(S, Index, Count)`

Параметр-переменная S — строковая переменная типа String.

Параметр Index — значение типа Integer, определяющее номер первого символа удаляемой подстроки.

Параметр Count — значение типа Integer, определяющее количество удаляемых символов.

Пример использования процедуры Delete представлен в листинге Ж.75.

Листинг Ж.75. Программа ProcDele.pas

```
program ProcDele;
uses Crt;
var
  s: string;
begin
  ClrScr;   {Очищаем экран}
  Write('Введите строку: ');
  Readln(s);
  while (pos(' ',s) > 0) do {Пока в строке есть пробелы}
    Delete(s,pos(' ',s),1); {Удаляем первый пробел}
  Write('Получена строка: ');
  Write(s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Insert

Вставляет подстроку в строку.

Синтаксис: `Insert(Source, S, Index)`

Параметр Source — вставляемое значение типа String. ,

Параметр-переменная S — строковая переменная типа String, в которую вставляется подстрока.

Параметр Index — значение типа Integer, определяющее позицию первого символа вставляемой подстроки в строке.

Пример использования процедуры Insert представлен в листинге Ж.76.

Листинг Ж.76. Программа ProcInse.pas

```
program ProcInse;
uses Crt;
var
  s: string;
  Source: string;
  Index: Integer;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Введите строку:');
  Readln(s);
  Writeln('Введите вставляемую подстроку:');
  Readln(Source);
  Writeln('Укажите позицию вставки:');
  Readln(Index);
  Insert(Source,s,Index);
  Writeln('Получена строка:');
  Writeln (s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Преобразования

Функции

Функция Chr

Возвращает символ по его коду ASCII.

Синтаксис: Chr(X)

Параметр X — значение типа Byte.

Тип возвращаемого результата: Char.

Пример использования функции Chr представлен в листинге Ж.77.

Листинг Ж.77. Программа FuncChr.pas

```
program FuncChr;
uses Crt;
var
  i: byte;
begin
  ClrScr;  {Очищаем экран}
  {Выводим на экран символы с кодами от 48 до 57}
  for i := 48 to 57 do
    Writeln('"',Chr(i),'" - код ';i);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Ord

Возвращает порядковое значение выражения перечислимого типа.

Синтаксис: Ord(X)

Параметр X — значение перечислимого типа.

Тип возвращаемого результата: Longint.

Пример использования функции Ord представлен в листинге Ж.78.

Листинг Ж.78. Программа FuncOrd.pas

```
program FuncOrd;
uses Crt;
var
  c: Char;
begin
  ClrScr;  {Очищаем экран}
  for c := '0' to '9' do {Выводим на экран коды цыфр от 0 до 9}
    Writeln('"',c, '" - код ',Ord(c));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Round

Округляет вещественное число до целого значения.

Синтаксис: Round(X)
Параметр X — значение типа Real.

Тип возвращаемого результата: Longint.

Пример использования функции Round представлен в листинге Ж.79.

Листинг Ж.79. Программа FuncRoun.pas

```
program FuncRoun;
uses Crt;
var
  r: Real;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите вещественное число: ');
  Readln(r);
  Writeln (r:10:3,', округленное до целого = ',Round(r));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Trunc

Усекает вещественное число до целочисленного значения.

Синтаксис: Trunc (X)
Параметр X — значение типа Real.

Тип возвращаемого результата: Longint.

Пример использования функции Trunc представлен в листинге Ж.80.

Листинг Ж.80. Программа FuncTrun.pas

```
program FuncTrun;
uses Crt;
var
  r: Real;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите вещественное число: ');
```

Окончание листинга Ж.80

```
  Readln(r);
  Writeln(r:10:3,', усеченное до целого = ',Trunc(r)); .
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура Str

Преобразует числовое значение в строку.

Синтаксис: Str (X:Width:Decimals, S)

Параметр X — числовое значение.

Необязательные поля Width и Decimals определяют длину полученной строки разрядность и количество знаков после десятичной точки (в случае преобразования вещественного числа).

Параметр-переменная S — строковая переменная типа String, в которой сохраняется преобразованное значение.

Пример использования процедуры Str представлен в листинге Ж.81.

Листинг Ж.81. Программа ProcStr.pas

```
program ProcStr;
uses Crt;
var
  r: Real;
  i: Integer;
  s: string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите вещественное число: ');
  Readln(r);
  Str(r:10:3,s); {Преобразование в строку}
  Writeln('Его строковое представление: ',s);
  Write('Введите целое число: ');
  Readln(i);
  Str(i,s); {Преобразование в строку}
  Writeln('Его строковое представление: ',s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Val

Преобразует строковое значение в его числовое представление.

Синтаксис: Val (S, V, Code)

Параметр S — значение строкового типа, содержащая последовательность символов, формирующих знаковое число.

Параметр-переменная V — вещественная или целочисленная переменная.

Параметр-переменная Code — значение типа Integer, в которой возвращается позиция *в* строке, в которой произошла ошибка, или ноль, если ошибки не было.

Пример использования процедуры Val представлен в листинге Ж.82.

Листинг Ж.82. Программа ProcVal.pas

```
program ProcVal;
uses Crt;
var
  s: string;
  n: Real;
  Error: Integer;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите число: ');
  Readln(s);
  Val(s,n,Error);
  if Error > 0
  then Writeln('Ошибка преобразования в позиции ',Error)
  else if pos('.',s) = 0 {Целое число}
       then Writeln('Преобразовано в число ',n:10:0)
       else Writeln('Преобразовано в число ',n:10:3);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Другие процедуры и функции

Функции

Функция High

Возвращает наибольшее значение для типа принятого параметра.

Синтаксис: High(X)

Параметр X — идентификатор переменной или имя простого типа.

Тип возвращаемого результата: совпадает с типом принятого параметра.

Пример использования функции High представлен в листинге Ж.83.

Листинг Ж.83. Программа FuncHigh.pas

```
program FuncHigh;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  Write('Наибольшее значение для типа Integer = ');
  Writeln(High(Integer));
  Write('Наибольшее значение для типа Char = #');
  Writeln(Ord(High(Char)));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Low

Возвращает наименьшее значение для типа принятого параметра.

Синтаксис: Low(X)

Параметр X — идентификатор переменной или имя простого типа.

Тип возвращаемого результата: совпадает с типом принятого параметра.

Пример использования функции Low представлен в листинге Ж.84.

Листинг Ж.84. Программа FuncLow.pas

```
program FuncLow;
uses Crt;
begin
  ClrScr;  {Очищаем экран)
  Write('Наименьшее значение для типа Integer = ');
  Writeln(Low(Integer));
  Write('Наименьшее значение для типа Char = #');
  Writeln(Ord(Low(Char)));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция ParamCount

Возвращает количество переданных в программу параметров при запуске в режиме командной строки. При запуске программы после ее названия в командной строке можно указывать параметры запуска, отделенные от названия программы и друг от друга пробелами. Пример использования параметров командной строки — запуск компилятора tpc.exe, когда в качестве первого передаваемого компилятору параметра указывается имя компилируемого файла, а затем через пробелы — параметры компиляции.

« см. об использовании компилятора tpc.exe в разделе "Компиляция в режиме командной строки DOS" главы 13.

Синтаксис: ParamCount

Тип возвращаемого результата: Word. Если программа была запущена без параметров (то есть в командной строке было введено только название выполняемого файла), то возвращается значение 0.

Пример использования функции ParamCount представлен в листинге Ж.85.

Листинг Ж.85. Программа FuncParC.pas

```
program FuncParC;
uses Crt;
begin
  ClrScr; {Очищаем экран}
  if ParamCount = 0
  then Writeln('Программа запущена без параметров')
  else Writeln('Количество параметров - ', ParamCount);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция ParamStr

Возвращает параметр командной строки с указанным номером.

Синтаксис: ParamStr (Index)

Параметр Index — номер параметра командной строки. Значению Index = 0 соответствует полный путь и имя программы.

Тип возвращаемого результата: String.

Пример использования функции ParamStr представлен в листинге Ж.86. Для проверки работы этой программы откомпилируйте ее, а затем запустите ее на выполнение в режиме командной строки, указав через пробел несколько параметров запуска.

В противном случае на экран будет выведено только название программы с указанием полного пути к ней.

Листинг Ж.86. Программа FuncParSpas

```
program FuncParS;
uses Crt;
var
  i: Word;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Командная строка:');
  {Выводим на экран все параметры командной строки}
  for i := 0 to ParamCount do
    Write(ParamStr(i),' ');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция Random

Возвращает случайное число в указанном диапазоне. Перед вызовом этой функции необходимо активизировать генератор случайных чисел, вызвав процедуру Randomize.

Синтаксис: Random(Range)

Необязательный параметр Range — значение типа Word. Если параметр передан, то случайное число генерируется в диапазоне от 0 (включительно) до Range. Если параметр не передан, то возвращается случайное вещественное число в диапазоне от О (включительно) до 1.

Тип возвращаемого результата: Longint или Real.

Пример использования функции Random представлен в листинге Ж.87.

Листинг Ж.87. Программа FuncRndm.pas

```
program FuncRndm;
uses Crt;
begin
  ClrScr;   {Очищаем экран}
  Randomize;  {Активизируем генератор случайных чисел}
  Write('Случайное число в диапазоне от 0 до 100: ');
  Writeln(Random(100):8);
  Write('Случайное число в диапазоне от 0 до 1: ');
  Writeln(Random:10:3);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура Exclude

Исключает элемент из множества.

Синтаксис: Exclude (S, I)

Параметр-переменная S — переменная типа множество.

Параметр I — выражение типа, совместимого с исходным типом для S.

Пример использования процедуры Exclude представлен в листинге Ж.88.

Листинг Ж.88. Программа ProcExcl.pas

```
program ProcExcl;
uses Crt;
var
  S: set of Byte;
  i: Byte;
  procedure ShowSet;
  {Вывод элементов множества на экран,
   если они найдены в диапазоне о 0 до 255}
  begin
    for i := 0 to 255 do
      if i in S then Write(i,' ');
    Writeln;
  end;
begin
  ClrScr;   {Очищаем экран}
  S := [1,2,4,5,200,201,255]; {Определяем множество}
  Writeln('Множество до исключения четных чисел:');
  ShowSet;
  for i := 1 to 255 do
    if (i in S) and ((i mod 2) = 0) {Если найден элемент множества
                                     и он делится без остатка на 2,}
    then Exclude(S,i);              {тогда исключаем его из множества}
  Writeln('Множество после исключения четных чисел:');
  ShowSet;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Include

Включает элемент в множество.

Синтаксис: Include(S, I)

Параметр-переменная S — переменная типа множество.

Параметр I — выражение типа, совместимого с исходным типом для S.

Пример использования процедуры Include представлен в листинге Ж.89.

Листинг Ж.89. Программа ProcIncl.pas

```
program ProcIncl;
uses Crt;
var
  S1,S2: set of Byte;
  i: Byte;

  procedure ShowSet;
  {Вывод элементов множества на экран,
   если они найдены в диапазоне о 0 до 255}
  begin
    for i := 0 to 255 do
      if i in S1 then Write(i,' ');
```

Окончание листинга Ж.89

```
    Writeln;
  end;

begin
  ClrScr;  {Очищаем экран}
  {Определяем множества}
  S1 := [1,5,201,255];
  S2 := [2,4,200];
  Writeln('Множество до включения четных чисел:');
  ShowSet;
  for i := 1 to 255 do
    {Если найден элемент множества S2 в
     диапазоне от 1 до 255, тогда включаем его в множество s1}
    if i in S2 then Include(S1,i);
  Writeln('Множество после включения четных чисел:');
  ShowSet;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Randomize

Активизирует генератор случайных чисел.

Синтаксис: Randomize

Пример использования процедуры Randomize представлен в листинге Ж.87 (◀◀ см. раздел, посвященный функции Random).

Модуль Crt

Функции

Функция KeyPressed

Возвращает значение True, если была нажата какая-либо клавиша на клавиатуре.

Синтаксис: KeyPressed

Тип возвращаемого результата: Boolean.

Пример использования функции KeyPressed представлен в листинге Ж.90.

Листинг Ж.90. Программа FuncKPrs.pas

```
program FuncKPrs;
{Программа вращения курсора ввода}
uses Crt;
var
  c: byte;
begin
  ClrScr;  {Очищаем экран)
  c := 1; {Устанавливаем начальное значение переменной c)
  while not KeyPressed do {Пока не нажата клавиша}
  begin
```

Окончание листинга Ж.90

```
    GotoXY(1,1); {Перемещаемся на экране в позицию (1,1)}
    {Выводим на экран символ в зависимости от значения c}
    case c of
      1: Write('|');
      2: Write('/');
      3: Write('-');
      4: Write('\');
    end;
    Delay (2000); {Задержка на 2 секунды}
    Inc(c); {Увеличиваем значение переменной c на 1}
    if c > 4 then c := 1; {Проверяем, значение переменной c,}
                          {чтобы оно не стало больше 4}
  end;
end.
```

Функция ReadKey

Считывает символ с клавиатуры.

Синтаксис: ReadKey

Тип возвращаемого результата: Char.

Пример использования функции ReadKey представлен в листинге Ж.91.

Листинг Ж.91. Программа FuncRKey.pas

```
program FuncRKey;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Нажмите какую-нибудь клавишу');
  Writeln('Код нажатой клавиши - ',Ord(ReadKey));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция WhereX

Возвращает координату X текущей позиции курсора.

Синтаксис: WhereX

Тип возвращаемого результата: Integer.

Пример использования функции WhereX представлен в листинге Ж.92.

Листинг Ж.92. Программа FuncWhrX.pas

```
program FuncWhrX;
uses Crt;
var
  x: Integer;
begin
  ClrScr;  {Очищаем экран}
  Write('Текущая координата X курсора = ');
  x := WhereX;
  Writeln(x);
```

Окончание листинга Ж.92

```
  GotoXY(x,1);
  {Перемещаем курсор в позицию, которая была у него
   до выполнения последнего оператора вывода на экран}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция WhereY

Возвращает координату Y текущей позиции курсора.

Синтаксис: WhereY

Тип возвращаемого результата: Integer.

Пример использования функции WhereY представлен в листинге Ж.93.

Листинг Ж.93. Программа FuncWhrY.pas

```
program FuncWhrY;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  Write('Текущая координата Y курсора = ', WhereY);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура AssignCrt

Связывает текстовый файл с окном CRT. Процедуры вывода в текстовый файл, назначенный при помощи процедуры AssignCrt, выводят информацию на экран монитора. Физически текстовый файл, назначенный при помощи этой процедуры, на диске не создается, как в случае использования процедуры Assign. Таким образом, процедуры записи Write и Writeln выводят данные в некий виртуальный файл, который сопоставлен с экраном монитора.

Синтаксис: AssignCrt(F)
Параметр-переменная F — файловая переменная типа Text.

Пример использования процедуры AssignCrt представлен в листинге Ж.94.

Листинг Ж.94. Программа ProcACRT.pas

```
program ProcACRT;
uses Crt;
var
  F: Text;
begin
  ClrScr;   {Очищаем экран}
  Assign(F,'test.txt'); {Эту команду можно и не использовать}
  AssignCrt(F);
  Rewrite(F); {Создаем файл}
  {Вывод на экран монитора}
  Write(F, 'Пример использования процедуры AssignCrt');
```

Окончание листинга Ж.94

```
  Close(F);  {Закрываем файл}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура ClrEol

Удаляет (заменяет пробелами) все символы, начиная от текущей позиции курсора до конца строки.

Синтаксис: ClrEol

Пример использования процедуры ClrEol представлен в листинге Ж.95.

Листинг Ж.95. Программа ProcCEol.pas

```
program ProcCEol;
uses Crt;
var
  i,j: integer;
begin
  ClrScr;  {Очищаем экран}
  {Вывод на экран матрицы чисел: 5 строк на 75 цыфр в строке}
  for i := 1 to 5 do
  begin
    for j := 1 to 75 do Write(i);
    Writeln;
  end;
  Writeln('Нажмите любую клавишу... ');
  ReadKey;
  {Усечение строк матрицы при помощи процедуры ClrEol}
  for i := 1 to 5 do
  begin
    GotoXY(i+1,i);
    ClrEol;
  end;
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура ClrScr

Очищает экран и перемещает курсор в левый верхний угол экрана.

Синтаксис: ClrScr

Пример использования процедуры ClrScr представлен в листинге Ж.96.

Листинг Ж.96. Программа ProcCScr.pas

```
program ProcCScr;
uses Crt;
var
  c: char;
begin
  ClrScr;  {Очищаем экран}
  Randomize; {Активизируем генератор случайных чисел}
```

Окончание листинга Ж.96

```
  repeat
    c := #0;
    while not KeyPressed do  {Пока не нажата клавиша}
    begin
      {Перемещаемся в произвольную позицию на экране}
      GotoXY(Random(80),Random(25));
      {Выводим выбранный случайным образом символ}
      Write(Chr(33+Random(60)));
      Delay(3000);  {Задержка на 3 секунды}
    end;
    ClrScr;  {Очищаем экран после нажатия клавиши}
    c := ReadKey;  {Определяем, какая была нажата клавиша}
  until c = #27;  {Выходим из цикла, если была нажата Esc}
end.
```

Процедура Delay

Приостанавливает выполнение программы на указанное количество миллисекунд.

Синтаксис: Delay (MS)

Параметр MS — значение типа Word, определяющее количество миллисекунд, на которое должно быть приостановлено выполнение программы.

Пример использования процедуры Delay представлен в листинге Ж.97.

Листинг Ж.97. Программа ProcDely.pas

```
program ProcDely;
uses Crt;
var
  DelayTime: Word;
  c: Char;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Ускорение - "+", Замедление - "-", Выход - Esc');
  DelayTime := 4000;  {Начальное время задержки - 4 с.}
  repeat
    c := #0;
    while not KeyPressed do  {Пока не нажата клавиша}
    begin
      Write('*');
      Delay(DelayTime);  {Задержка}
    end;
    c := ReadKey;  {Определяем, какая была нажата клавиша}
    {Увеличиваем или уменьшаем время задержки}
    case c of
      '+','=': if DelayTime > 2000 then Dec(DelayTime,1000);  {Ускорение}
      '-','_': Inc(DelayTime,1000);  {Замедление}
    end;
  until c = #27;  {Выходим из цикла, если была нажата Esc}
end.
```

Процедура DelLine

Удаляет строку, в которой расположен курсор.

Синтаксис: DelLine

Пример использования процедуры DelLine представлен в листинге Ж.98.

Листинг Ж.98. Программа ProcDelL.pas

```
program ProcDelL;
{Удаление строк на экране при помощи клавиши <Del>}
uses Crt;
var
  i,j: integer;
begin
  ClrScr;  {Очищаем экран}
  {Заполняем экран строками}
  for i := 1 to 25 do Writeln ('Книга "Turbo Pascal 7.0 на примерах.
                               Автор Ю.А. Шпак. Издательство "Юниор"');
  GotoXY(1,1);
  repeat
    case ReadKey of
      #27: Break; {Выход из цикла - по клавише Esc}
      #72: if WhereY > 1 then GotoXY(WhereX,WhereY-1); {Стрелка вверх}
      #80: if WhereY < 25 then GotoXY(WhereX,WhereY+1); {Стрелка вниз}
      #83: DelLine; {нажата клавиша Delete}
    end;
  until False;
end.
```

Процедура GotoXY

Перемещает курсор в позицию с заданными координатами.

Синтаксис: GotoXY(X,Y)

Параметры X и Y — значения типа Integer, определяющие экранные координаты X и Y.

Пример использования процедуры GotoXY представлен в листинге Ж.99.

Листинг Ж.99. Программа ProcGoXY.pas

```
program ProcGoXY;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Перемещение курсора - клавиши со стрелками. ', 'Выход - Esc');
  repeat
    case ReadKey of
      #27: Break; {Выход из цикла - по клавише Esc}
      #72: if WhereY > 1 then GotoXY(WhereX,WhereY-1); {Стрелка вверх}
      #75: if WhereX > 1 then GotoXY(WhereX-1,WhereY); {Стрелка влево}
      #77: if WhereX < 80 then GotoXY(WhereX+1,WhereY); {Стрелка вправо}
      #80: if WhereY < 25 then GotoXY(WhereX,WhereY+1); {Стрелка вниз}
    end;
  until False;
end.
```

Процедура HighVideo

Устанавливает высокую яркость свечения символов.

Синтаксис: HighVideo

Пример использования процедуры HighVideo представлен в листинге Ж. 100.

Листинг Ж. 100. Программа ProcHVid.pas

```
program ProcHVid;
uses Crt;
begin
  ClrScr;   {Очищаем экран)
  HighVideo;
  Writeln ( 'Высокая яркость символов');
  NormVideo;
  Writeln ( 'Обычная яркость символов' ) ;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура InsLine

Вставляет пустую строку в месте расположения курсора.

Синтаксис: InsLine

Пример использования процедуры InsLine представлен в листинге Ж. 101.

Листинг Ж.101. Программа ProcInsL.pas

```
program ProcInsL;
{Вставка строк на экране при помощи клавиши <Insert>}
uses Crt;
var
  i,j: integer;
begin
  ClrScr;   {Очищаем экран}
  {Заполняем экран строками)
  for i := 1 to 25 do Writeln ('Книга "Turbo Pascal 7.0 на примерах.
                               Автор Ю.А. Шпак. Издательство "Юниор"');
  GotoXY(1,1);
  repeat
    case ReadKey of
      #27: Break; {Выход из цикла - по клавише Esc}
      #72: if WhereY > 1 then GotoXY(WhereX,WhereY-1); {Стрелка вверх)
      #80: if WhereY < 25 then GotoXY(WhereX,WhereY+1); {Стрелка вниз}
      #82: InsLine; {Нажата клавиша Insert)
    end;
  until False;
end.
```

Процедура LowVideo

Устанавливает низкую яркость свечения символов.

Синтаксис: LowVideo

Пример использования процедуры LowVideo представлен в листинге Ж. 102.

Листинг Ж. 102. Программа ProcLVid.pas

```
program ProcLVid;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  LowVideo;
  Writeln('Низкая яркость символов');
  NormVideo;
  Writeln('Обычная яркость символов');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура NormVideo

Устанавливает нормальную яркость свечения символов.

Синтаксис: NormVideo

Пример использования процедуры NormVideo представлен в листинге Ж. 103.

Листинг Ж. 103. Программа ProcNVid.pas

```
program ProcNVid;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  LowVideo;
  Writeln('Низкая яркость символов');
  HighVideo;
  Writeln('Высокая яркость символов');
  NormVideo;
  Writeln('Обычная яркость символов');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура NoSound

Отключает встроенный динамик компьютера.

Синтаксис: NoSound

Пример использования процедуры NoSound представлен в листинге Ж. 104.

Листинг Ж.104. Программа ProcNSnd.pas

```
program ProcNSnd;
uses Crt;
begin
  ClrScr;  {Очищаем экран}
  Sound(1000); {Звук с частотой 1000 Гц}
  Writeln('Чтобы отключить динамик, нажмите любую клавишу');
  ReadKey;
  NoSound;
  Writeln('Динамик отключен');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Sound

Генерирует звук заданной частоты.

Синтаксис: Sound (Hz)

Параметр Hz — значение типа Word, определяющее частоту звука в герцах.

Пример использования процедуры Sound представлен в листинге Ж. 105.

Листинг Ж.105. Программа ProcSoun.pas

```
program ProcSoun;
uses Crt;
var
  i: Word;
begin
  ClrScr;          {Очищаем экран}
  i := 1000;
  Write('Для завершения нажмите любую клавишу...');
  while i <= 5000 do
  begin
    Sound(i);     {Генерация звука с частотой i Гц}
    Delay(5000);  {Задержка на 5 секунд}
    Inc(i,100);   {Увеличиваем частоту на 100 Гц}
    if KeyPressed then break;
  end;
  NoSound;         {Отключаем динамик}
end.
```

Процедура TextBackGround

Устанавливает цвет фона символов.

Синтаксис: TextBackGround (Color)

Параметр Color — значение типа Byte, определяющее цвет фона выводимых символов. Вместо числового значения можно использовать соответствующие константы модуля Crt (табл. Ж.1).

Таблица Ж.1. Соответствие констант модуля Crt цветам, выводимым на экран

Константа	Цвет	Константа	Цвет
0	Черный (Black)	8	Темно-серый (DarkGray)
1	Синий (Blue)	9	Светло-синий (LightBlue)
2	Зеленый (Green)	10	Светло-зеленый (LightGreen)
3	Бирюзовый (Cyan)	11	Светло-бирюзовый (LightCyan)
4	Красный (Red)	12	Светло-красный (LightRed)
5	Малиновый (Magenta)	13	Светло-малиновый (LightMagenta)
6	Коричневый (Brown)	14	Желтый (Yellow)
7	Светло-серый (LightGray)	15	Белый (White)

Пример использования процедуры TextBackGround представлен в листинге Ж. 106.

Листинг Ж.106. Программа ProcTBgr.pas

```
program ProcTBgr;
uses Crt;
var
  i: Byte;
begin
  ClrScr;  {Очищаем экран}
  for i := 0 to 15 do
  begin
    TextBackGround(i);
    Writeln ('Книга "Turbo Pascal 7.0 на примерах.
             Автор Ю.А. Шпак. Издательство "Юниор"');
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура TextColor

Устанавливает цвет символов.

Синтаксис: TextColor (Color)

Параметр Color — значение типа Byte, определяющее цвет выводимых символов. Вместо числового значения можно использовать соответствующие константы модуля Crt (см. табл. Ж.1).

Пример использования процедуры TextColor представлен в листинге Ж. 107.

Листинг Ж.107. Программа ProcTClr.pas

```
program ProcTClr;
uses Crt;
var
  i: Byte;
begin
  ClrScr;  {Очищаем экран}
  for i := 0 to 15 do
  begin
    TextColor(i);
    Writeln ('Книга "Turbo Pascal 7.0 на примерах.
             Автор Ю.А. Шпак. Издательство "Юниор"');
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура TextMode

Устанавливает указанный текстовый режим.

Синтаксис: TextMode (Mode)

Параметр Mode — значение типа Integer, определяющее текстовый режим. Вместо числового значения можно использовать соответствующие константы модуля Crt (табл. Ж.2).

Пример использования процедуры TextMode представлен в листинге Ж. 108.

Таблица Ж.2. Соответствие констант модуля Crt текстовым режимам монитора

Константа	Текстовый режим	Константа	Текстовый режим
0	Черно-белый 40x25 (BW40)	3	Цветной 80x25 (CO80)
1	Цветной 40x25 (CO40)	7	Монохромный 80x25 (Mono)
2	черно-белый 80x25 (BW80)	256	43/50 строк для адаптеров EGA/VGA (Font8x8)

Листинг Ж.108. Программа ProcTMod.pas

```
program ProcTMod;
uses Crt;
var
  CurMode: Integer;
  procedure ShowStr(Mode: Integer; s: string);
  begin
    TextMode(Mode); {Устанавливаем режим}
    TextBackGround(Blue); {Цвет фона - синий}
    TextColor(Yellow); {Цвет символов - желтый}
    Writeln(s);
    Writeln ('Книга "Turbo Pascal 7.0 на примерах.
             Автор Ю.А. Шпак. Издательство "Юниор"');
    ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  end;
begin
  ClrScr;  {Очищаем экран}
  CurMode := LastMode; {Запоминаем текущий режим}
  ShowStr(BW40,'BW40');
  ShowStr(CO40,'CO40');
  ShowStr(BW80,'BW80');
  ShowStr(CO80,'CO80');
  ShowStr(Mono,'Mono');
  ShowStr(Font8x8,'Font8x8');
  TextMode(CurMode); {Восстанавливаем исходный режим}
end.
```

Процедура Window

Определяет на экране текстовое окно. По умолчанию в программе всегда используется одно текстовое окно размером во весь экран. Все экранные координаты отсчитываются относительно левого верхнего угла текущего окна.

Синтаксис: Window(X1,Y1,X2,Y2)

Параметры X1, У1, X2, Y2 — значения типа Byte, определяющие координаты левого верхнего и правого нижнего угла текстового окна.

Пример использования процедуры Window представлен в листинге Ж. 109.

Листинг Ж.109. Программа ProcWind.pas

```
program ProcWind;
uses Crt;
  procedure ShowWindow(x1,y1,x2,y2,Color: Byte);
  begin
    Window(x1,y1,x2,y2); {Определяем окно}
```

Окончание листинга Ж. 109

```
    TextBackGround(Color); {Цвет фона}
    TextColor(White); {Цвет символов - белый}
    ClrScr; {Очищаем текущее окно}
    GotoXY(1,1); {Перемещаемся в левый верхний угол окна}
    Writeln(Color); {Отображаем номер цвета}
  end;

begin
  ShowWindow(1,1,40,12,Blue);
  ShowWindow(41,1,80,12,Green);
  ShowWindow(1,13,40,24,Cyan);
  ShowWindow(41,13,80,24,Red);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Результат работы программы ProcWind представлен на рис. Ж.7.

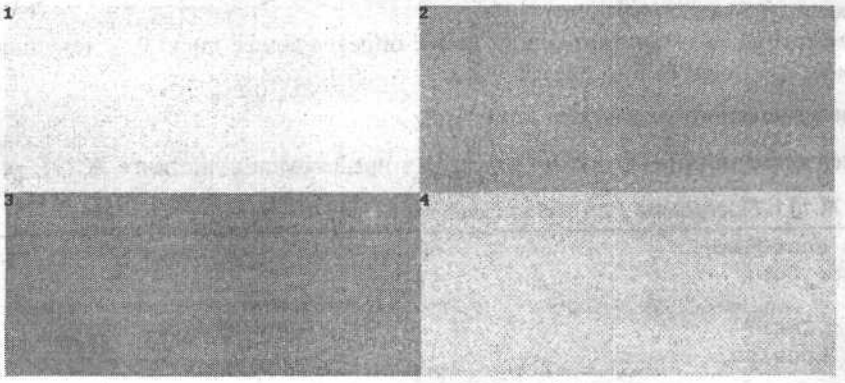


Рис. Ж.7. Четыре окна с разным фоном

Модуль Dos

Функции

Функция DiskFree

Возвращает количество свободных байтов на указанном диске.

Синтаксис: DiskFree(Drive)

Параметр Drive — значение типа Byte, определяющее диск: 0 — текущий диск; 1 — диск A; 2 — диск B; 3 — диск C и т.д.

Тип возвращаемого результата: LongInt.

Пример использования функции DiskFree представлен в листинге Ж.110.

Листинг Ж.110. Программа FuncDFre.pas

```
program FuncDFre;
uses Crt, Dos;
var
```

Окончание листинга Ж.110

```
  Drive: Char;
  FreeSpace: LongInt;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите символ диска: ');
  Readln(Drive);
  Drive := UpCase(Drive); {Переводим в верхний регистр}
  FreeSpace := DiskFree(Ord(Drive)-64);
  Write('На диске ',Drive,' свободно ');
  Write(FreeSpace div 1024,' КБайт и ');
  Writeln(FreeSpace mod 1024,' байт');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция DiskSize

Возвращает общий объем указанного диска в байтах.

Синтаксис: DiskSize (Drive)

Параметр Drive — значение типа Byte, определяющее диск: 0 — текущий диск; 1 — диск A; 2 — диск B; 3 — диск C и т.д.

Тип возвращаемого результата: Longlnt.

Пример использования функции DiskSize представлен в листинге Ж.111.

Листинг Ж.111. Программа FuncDSiz.pas

```
program FuncDSiz;
uses Crt, Dos;
var
  Drive: Char;
  Size: Longlnt;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите символ диска: ');
  Readln(Drive);
  Drive := UpCase(Drive); {Переводим в верхний регистр}
  Size := DiskSize(Ord(Drive)-64);
  Write('Объем диска ',Drive,' составляет ');
  Write(Size div 1024,' КБайт и');
  Writeln(Size mod 1024,' байт');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция DosExitCode

Возвращает код завершения процесса.

Синтаксис: DosExitCode

Тип возвращаемого результата: Word. Младший байт содержит код завершения процесса; старший байт определяет характер завершения процесса: 0 — нормальное; 1 — по <Ctrl+C>; 2 — ошибка устройства; 3 — процедура Keep.

Пример использования функции DosExitCode представлен в листинге Ж.112.

Листинг Ж.112. Программа FuncDsEC.pas

```
program FuncDsEC;
uses Crt,Dos;
var
  ProgramName, CmdLine: String;
begin
  ClrScr;  {Очищаем экран}
  Write('Путь доступа и имя программы для запуска: ');
  Readln(ProgramName);
  Write('Параметры командной строки ', ProgramName, ': ');
  Readln(CmdLine);
  Writeln('Пробую запустить...');
  SwapVectors;  {Запоминаем текущее состояние векторов прерываний}
  Exec(ProgramName, CmdLine); {Выполняем внешнюю программу}
  SwapVectors;  {Восстанавливаем состояние векторов прерываний}
  WriteLn('... вернулся из Exec');
  if DosError <> 0 {Если была ошибка DOS}
  then Writeln('Ошибка DOS #', DosError)
  else Writeln('Запуск был удачным. Код выхода = ', DosExitCode);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция DosVersion

Возвращает номер версии DOS.

Синтаксис: DosVersion

Тип возвращаемого результата: Word. Младший байт содержит номер версии, а старший — номер подверсии.

Пример использования функции DosVersion представлен в листинге Ж.113.

Листинг Ж.113. Программа FuncDosV.pas

```
program FuncDosV;
uses Crt,Dos;
var
  Ver: Word;
begin
  ClrScr;   {Очищаем экран}
  Ver := DosVersion;
  WriteLn('Версия DOS: ',Lo(Ver),'.',Hi(Ver));
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция EnvCount

Возвращает количество установленных переменных окружения DOS.

Синтаксис: EnvCount

Тип возвращаемого результата: Integer.

ПРИМЕЧАНИЕ

Окружение (enviroment) — совокупность ресурсов, предоставляемых в распоряжение пользователя системы, хранящихся в определенной области памяти системы. Окружение

ОКОНЧАНИЕ ПРИМЕЧАНИЯ

операционной системы MS-DOS составляют переменные вида: . *имя переменной=значение*, представляющее собой текстовую строку. Набор таких текстовых строк называется переменными окружения MS-DOS (системная информация). Переменные окружения MS-DOS предназначены для выполнения системных или прикладных программ. Например, с их помощью указывают путь поиска выполняемых файлов или хранят системное приглашение, указывают на местонахождение командного процессора системы и т.п. Значения одних переменных устанавливаются системой по умолчанию, другие могут быть установлены прикладными программами или пользователем. Например, переменная COMSPEC по умолчанию установлена в значение 'C:\' — путь к каталогу, в котором находится командный процессор MS-DOS — файл COMMAND.COM. Например, переменная PATH, содержащая путь поиска выполняемых файлов, по умолчанию имеет пустое значение, то есть все выполняемые файлы, указываемые в командной строке без явного указания пути к каталогу их расположений в системе, ищутся только в текущем каталоге. Для изменения значений переменных окружения в MS-DOS предназначена команда SET переменная=значение. Обычно команды такого вида выполняются автоматически при запуске системы в файлах AUTOEXEC.BAT (см. рис. Ж.6) и CONFIG.SYS. Например, при помощи команды SET PATH=C:\;C:\DOS;C:\NC, которую можно расположить в файле AUTOEXEC.BAT, можно определить те каталоги, в которых будет искаться системой выполняемый файл, если он был указан в командной строке без явного указания каталога, в котором он расположен.

Пример использования функции EnvCount представлен в листинге Ж. 114.

Листинг Ж.114. Программа FuncEnvC.pas

```
program FuncEnvC;
uses Crt,Dos;
begin
  ClrScr;  {Очищаем экран}
  WriteLn('Установлено ',EnvCount,' переменных окружения DOS');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция EnvStr

Возвращает указанную строку с переменной окружения DOS.

◄◄ см. примечание из предыдущего раздела.

Синтаксис: EnvStr (Index)

Параметр Index — значение типа Integer, определяющее номер переменной окружения DOS.

Тип возвращаемого результата: String.

Пример использования функции EnvStr представлен в листинге Ж. 115.

Листинг Ж.115. Программа FuncEnvS.pas

```
program FuncEnvS;
uses Crt,Dos;
var
  i: integer;
begin
  ClrScr;  {Очищаем экран}
```

Окончание листинга Ж.115

```
  WriteLn('Установленные переменные окружения DOS: ');
  for i := 1 to EnvCount do Writeln(EnvStr(i));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция FExpand

Расширяет имя файла до полного имени (с указанием пути).

Синтаксис: FExpand(Path)

Параметр Path — значение типа PathStr (строковый тип, объявленный в модуле Dos), содержащее имя файла.

Тип возвращаемого результата: PathStr.

Пример использования функции FExpand представлен в листинге Ж. 116.

Листинг Ж.116. Программа FuncFExp.pas

```
program FuncFExp;
uses Crt,Dos;
begin
  ClrScr;   {Очищаем экран}
  Writeln('Полное имя файла FuncFExp.exe:');
  Writeln(FExpand('FuncFExp.exe'));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши)
end.
```

Функция FSearch

Выполняет поиск файла в списке каталогов.

Синтаксис: FSearch(Path, DirList)

Параметр Path — значение типа PathStr (строковый тип, объявленный в модуле Dos), содержащее имя файла.

Параметр DirList — значение типа String, содержащее перечень каталогов для поиска, в котором отдельные элементы отделены друг от друга точкой с запятой.

Тип возвращаемого результата: PathStr.

Пример использования функции FSearch представлен в листинге Ж. 117.

Листинг Ж.117. Программа FuncFSrc.pas

```
program FuncFSrc;
uses Crt,Dos;
var
  s: PathStr;
 nf: String;
begin
  ClrScr;   {Очищаем экран}
  Writeln('Поиск файла в каталогах,')
  Writeln('указанных в списке переменной окружения PATH')
  Write('Введите имя файла: ')}
  Readln(nf);
  s := FSearch(nf, GetEnv('PATH')); {Поиск файла}
```

Окончание листинга Ж.117

```
  if s = ''
  then Writeln(nf,' не найден!')
  else Writeln('Найден как ', FExpand(s));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

◀◀ см. о переменных окружения DOS в примечании из раздела посвященного функции EnvCount.

Функция GetEnv

Возвращает значение указанной переменной окружения DOS.

Синтаксис: GetEnv(EnVar)

Параметр EnVar — значение типа String, содержащее имя переменной окружения DOS.

Тип возвращаемого результата: string.

Пример использования функции GetEnv представлен в листинге Ж.118.

Листинг Ж.118. Программа FuncGEnv.pas

```
program FuncGEnv;
uses Crt,Dos;
var
  Name: string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите имя переменной окружения DOS: ');
  Readln(Name);
  Writeln ('Значение переменной ',Name,':');
  Writeln(GetEnv(Name));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура Exec

Выполняет указанную программу с указанной командной строкой. Код возникающих ошибок сохраняется в переменной DosError.

Синтаксис: Exec (Path, CmdLine)

Параметры Path и CmdLine — значения типа String, определяющие имя программы и ее командную строку.

Пример использования процедуры Exec представлен в листинге Ж.112 (◀◀ см. раздел, посвященный функции DosExitCode).

Процедура FindFirst

Ищет в указанном каталоге первый файл по заданному условию.

Синтаксис: FindFirst(Path, Attr, F)

Параметр Path — значение типа String, определяющее имя искомого файла, в котором допускается использование символов шаблона * и ?.

Параметр Attr — значение типа Word, определяющее атрибут искомого файла. Вместо числового значения в качестве этого параметра можно передавать одну из следующих констант модуля Dos (или их сумму): 1 — Readonly (только для чтения); 2 — Hidden (скрытый); 4 — System (системный); 8 — VolumeID (идентификатор каталога); 16 — Directory (каталог); 32 — Archive (архивный); 63 — AnyFile (любой файл).

Параметр-переменная F — значение определенного в модуле Dos типа SearchRec. В этой переменной сохраняется информация о найденном файле в виде следующей структуры:

```
type
  SearchRec = Record
    Fill: array[1..21] of Byte; {Не используется}
    Attr: Byte;        {Атрибут файла}
    Time: LongInt;     {Дата и время}
    Size: Longlnt;     {Размер в байтах}
    Name: String[12]; {Имя}
  end;
```

» Для распаковки даты и времени, хранимых в поле Time, используется процедура UnpackTime.

Пример использования процедуры FindFirst представлен в листинге Ж. 119.

Листинг Ж.119. Программа ProcF1st.pas

```
program ProcF1st;
uses Crt, Dos;
var
  F: SearchRec;
  s:, string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  FindFirst(s,AnyFile,F);
  Writeln('Информация о файле ',s);
  Writeln('Имя: ',F.Name);
  Writeln('Размер: ',F.Size,' байт');
  Writeln('Атрибут: ',F.Attr);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура FindNext

Ищет следующий файл в соответствии с именем и набором атрибутов, определенных в предшествующем вызове процедуры FindFirs't. Для того чтобы определить, есть ли еще в заданном каталоге файлы, удовлетворяющие улосвию поиска, следует после каждого вызова процедуры FindNext проверять значение объявленной в модуле Dos переменной DosError. Если файлов больше нет, то эта переменная примет значение 18.

Синтаксис: FindNext (F)

Параметр-переменная F — значение определенного в модуле Dos типа SearchRec. В этой переменной сохраняется информация о найденном файле в виде следующей структуры:

```
type
  SearchRec = Record
    Fill: array[1..21] of Byte; {Не используется}
    Attr: Byte;        {Атрибут файла}
    Time: Longlnt;     {Дата и время}
    Size: Longlnt;     {Размер в байтах}
    Name: String[12]; {Имя}
  end;
```

- Для распаковки даты и времени, хранимых в поле Time, используется процедура UnpackTime.

Пример использования процедуры FindNext представлен в листинге Ж. 120.

Листинг Ж.120. Программа ProcFNxt .pas

```
program ProcFNxt;
uses Crt, Dos;
var
  F: SearchRec;
  s: string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  FindFirst(s,AnyFile,F);
  Writeln('Перечень файлов ',s,' в текущем каталоге:');
  while DosError = 0 do
  begin
    Writeln(F.Name);
    FindNext(F);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура FSplit

Разбивает имя файла на составляющие.

Синтаксис: FSplit(Path, Dir, Name, Ext)

Параметр Path — значение определенного в модуле Dos типа PathStr, содержащее имя файла.

Параметр-переменная Dir — переменная определенного в модуле Dos типа DirStr, в которой сохраняется имя каталога.

Параметр-переменная Name — переменная определенного в модуле Dos типа NameStr, в которой сохраняется имя файла (без расширения).

Параметр-переменная Ext — переменная определенного в модуле Dos типа ExtStr, в которой сохраняется расширение файла.

Пример использования процедуры FSplit представлен в листинге Ж. 121.

Листинг Ж.121. Программа ProcFSpl.pas

```
program ProcFSpl;
uses Crt, Dos;
var
  Path: PathStr;
```

Окончание листинга Ж.121

```
  Dir: DirStr;
  Name: NameStr;
  Ext: ExtStr;
begin
  ClrScr;  {Очищаем экран}
  Writeln('Введите имя файла с указанием пути:');
  Readln(Path);
  FSplit(Path, Oir, Name, Ext);
  Writeln('Каталог: ',Dir);
  Writeln('Имя: ',Name);
  Writeln('Расширение: ',Ext);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура GetCBreak

Извлекает результат проверки системой DOS нажатия комбинации клавиш <Ctrl+Break>.

Синтаксис: GetCBreak (Break)

Параметр-переменная Break — переменная типа Boolean. Если после вызова процедуры GetCBreak эта переменная содержит False, то DOS проверяет нажатие комбинации клавиш <Ctrl+Break> только во время операций ввода/вывода на консоль или принтер; если же эта переменная содержит True, то проверка осуществляется при каждом системном вызове.

Пример использования процедуры GetCBreak представлен в листинге Ж. 122.

Листинг **Ж.122.** Программа ProcGCBr.pas

```
program ProcGCBr;
uses Crt, Dos;
const
  OffOn: array [Boolean] of String[4] = ('выкл','вкл');
var
  Cb: Boolean;
begin
  ClrScr;  {Очищаем экран}
  GetCBreak(Cb);
  Writeln('Проверка на <Ctrl+Break>: ', OffOn[Cb]);
  Cb := Not(Cb);
  Writeln('Переключение проверки на: ', OffOn[Cb]);
  SetCBreak(Cb);
  GetCBreak(Cb);
  Writeln('Проверка на <Ctrl+Break>: ', OffOn[Cb]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура GetDate

Извлекает текущую дату, установленную в операционной системе.

Синтаксис: GetDate(Year, Month, Day, DayOfWeek)

Параметры-переменные Year, Month, Day и DayOfWeek — переменные типа Word, в которых после вызова процедуры GetDate сохраняются значения года, месяца, числа и дня недели: 0 — воскресенье, 1 — понедельник и т.д.

Пример использования процедуры GetDate представлен в листинге Ж. 123.

Листинг Ж.123. Программа ProcGDat.pas

```
program ProcGDat;
uses Crt, Dos;
const
  Months: array[1..12] of String[8] =
          ('Январь','Февраль','Март','Апрель',
           'Май','Июнь','Июль','Август',
           'Сентябрь','Октябрь','Ноябрь','Декабрь');
  Days: array[0..6] of String[11] =
        ('Воскресенье','Понедельник','Вторник',
         'Среда','Четверг','Пятница','Суббота');
var
  Year, Month, Day, DayOfWeek: Word;
begin
  ClrScr;  {Очищаем экран}
  GetDate(Year, Month, Day, DayOfWeek);
  Writeln('Год: ',Year);
  Writeln('Месяц: ',Months[Month]);
  Writeln('Число: ',Day);
  Writeln('День недели: ',Days[DayOfWeek]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура GetFAttr

Извлекает атрибуты файла.

Синтаксис: GetFAttr(F, Attr)

Параметр-переменная F — переменная любого файлового типа.

Параметр-переменная Attr — переменная типа Word, в которой сохраняются атрибуты файла. Значению Attr соответсвует сумма констант модуля Dos: 1 — Readonly (только для чтения), 2 — Hidden (скрытый), 4 — System (системный), 8 — VolumeID (идентификатор каталога), 16 — Directory (каталог), 32 — Archive (архивный), 63 — AnyFile (любой файл).

Пример использования процедуры GetFAttr представлен в листинге Ж. 124.

Листинг Ж.124. Программа ProcGFAt.pas

```
program ProcGFAt;
uses Crt, Dos;
var
 F: File;
Attr: Word;
 s: string;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
```

Окончание листинга Ж.124

```
  Assign(F, s); {Связываем имя файла с файловой переменной}
  GetFAttr(F, Attr);
  {Определяем атрибуты, используя константы модуля Dos}
  if (Attr and Readonly) <> 0 then Writeln('Только для чтения');
  if (Attr and Hidden) <> 0 then Writeln('Скрытый');
  if (Attr and SysFile) <> 0 then Writeln('Системный');
  if (Attr and VolumeID) <> 0 then Writeln('ID тома');
  if (Attr and Directory) <> 0 then Writeln('Имя каталога');
  if (Attr and Archive) <> 0 then Writeln('Архивный (нормальный)');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура GetFTime

Извлекает дату и время последней модификации файла.

Синтаксис: GetFTime(F, Time)

Параметр-переменная F — переменная любого файлового типа.

Параметр-переменная Time — переменная типа Longint, в которой в упакованном виде сохраняется дата и время последней модицикации файла F.

- Для распаковки этого значения Time используется процедура UnpackTime.

Пример использования процедуры GetFTime представлен в листинге Ж. 125.

Листинг Ж.125. Программа ProcGFTm.pas

```
program ProcGFTm;
uses Crt, Dos;
var
 s: string;
 F: File;
Time: Longint;
 DT: DateTime;  {Тип DateTime объявлен в модуле Dos}
begin
  ClrScr;  {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  Assign(F, s); {Связываем имя файла с файловой переменной}
  GetFTime(F,Time); {Извлекаем упакованные дату и время}
  {Распаковываем дату и время в формат DateTime}
  UnpackTime(Time, DT);
  Writeln('Последняя модификация файла ',s,' выполнена ');
  Write(DT.Day,'.',DT.Month,'.',DT.Year);
  Writeln(' в ',DT.Hour,':',DT.Min,':',DT.Sec);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Тип DateTime, объявленный в модуле Dos, имеет следующее объявление:

```
type
  DateTime = Record
    Year, Month, Day: Word;
      Hour, Min, Sec: Word;
  end;
```

Процедура GetIntVec

Извлекает адрес, хранимый в указанном векторе прерывания,

Синтаксис: `GetIntVec(IntNo, Vector)`

Параметр IntNo — значение типа Byte, определяющее номер вектора прерывания.

Адрес указанного вектора прерывания возвращается в параметр-переменную указательного типа Vector.

Пример использования процедуры GetIntVec представлен в листинге Ж. 126.

Листинг Ж.126. Программа `ProcGIVc.pas`

```
program ProcGIVc;
uses Crt, Dos;
var
  IntlCSave: Pointer;
  procedure TimerHandler; interrupt;
  begin {В данном примере - пустая процедура}
  end;
begin
  ClrScr;   {Очищаем экран}
  {Получаем старый вектор прерывания $1C - (системный таймер)}
  {и сохраняем его в переменной Int1CSave}
  GetIntVec($1C, Int1CSave);
  {Переназначаем его на нашу процедуру обработки прерывания}
  SetIntVec($1C, @TimerHandler);
  WriteLn('Нажмите любую клавишу для снятия обработчика...');
  repeat
  until KeyPressed;  {Пока не будет нажат клавиша}
  {Возвращаем обработчик прерывания в исходное состояние}
  SetIntVec($1C, Int1CSave);
end.
```

В данном примере вектору прерывания `$1C` назначется пользовательская процедура обработки TimerHandler. В заголовке подобных процедур используется директива `interrupt`. В начале программы при помощи процедуры GetIntVec текущий адрес вектора прерывания сохраняется в переменной IntlCSave, затем при помощи процедуры `SetInVec` этому вектору прерывания назначается новая процедура обработки, а в конце программы, после нажатия любой клавиши, обработчик вектора прерывания возвращается в исходное состояние.

Процедура GetTime

Извлекает текущее время, установленное в операционной системе.

Синтаксис: `GetTime(Hour, Minute, Second, SeclOO)`

Параметры-переменные Hour, Minute, Second, SeclOO — переменные типа Word, в которых сохраняется текущие час, минута, секунда и сотая доля секунды.

Пример использования процедуры GetTime представлен в листинге Ж. 127.

Листинг Ж.127. Программа `ProcGtTm.pas`

```
program ProcGtTm;
uses Crt, Dos;
```

Окончание листинга Ж.127

```
var
  H1,H2,M1,M2,S1,S2,S100: Word;
begin
  ClrScr;   {Очищаем экран}
  Writeln('Нажмите любую клавишу...');
  ReadKey;
  GetTime(H1,M1,S1,S100);
  Writeln('Нажмите любую клавишу еще раз...');
  ReadKey;
  GetTime(H2,M2,S2,S100);
  Writeln('Промежуток времени между двумя');
  Write('нажатиями составляет: ',
        H2-H1,' часов, ',M2-M1,' минут, ', S2-S1, ' секунд');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура GetVerify

Извлекает состояние флажка проверки записи на диск в DOS.

Синтаксис: `GetVerify(Verify)`

Параметр-переменная Verify — переменная типа Boolean. Если Verify имеет значение False, то запись на диск не проверяется.

Пример использования процедуры GetVerify представлен в листинге Ж. 128.

Листинг Ж.128. Программа ProcGVrf.pas

```
program ProcGVrf;
uses Crt, Dos;
var
  IsVerify: Boolean;
begin
  ClrScr;   {Очищаем экран}
  GetVerify(IsVerify);
  Write('Запись на диск ');
  if not IsVerify then Write('не ');
  Writeln('проверяется');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Intr

Выполняет указанное программное прерывание.

Синтаксис: `Intr(IntNo, Regs)`

Параметр IntNo — значение типа Byte, определяющее номер выполняемого программного прерывания.

Параметр-переменная Regs — переменная типа Registers.

Пример использования процедуры Intr представлен в листинге Ж. 129.

Листинг Ж.129. Программа ProcIntr.pas

```
program ProcIntr;
uses Crt, Dos;
```

Окончание листинга Ж.129

```
var
  Year, Month, Day: String;
  Regs : Registers;
begin
  ClrScr;              {Очищаем экран}
  Regs.AH:=$2A;        {Функция считывания текущей даты}
  Intr($21, Regs);     {Вызываем прерывание 21H}
  with Regs do         {Считываем содержимое регистров}
  begin
    Str(CX, Year) ;    {CX - год}
    Str(DH, Month);    {DH - месяц}
    Str(DL, Day);      {DL - день}
  end;
  Writen('Сегодня: ',Day,'.',Month,'.',Year);
  ReadKey;  {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура Keep

Прерывает выполнение программы и оставляет ее резидентной в памяти.

Синтаксис: Keep(ExitCode)

Параметр ExitCode — значение типа Word, определяющее код завершения программы.

Пример использования процедуры Keep представлен в листинге Ж.130.

Листинг Ж.130. Программа ProcKeep.pas

```
program ProcKeep;
{$M $800, 0, 0 }   {Стек 2КБайта, без кучи}
uses Crt, Dos;
var
  KbdIntVec: Procedure;  {Переменная процедурного типа}

  {$F+} {Включаем дальний (FAR) тип вызова}
  procedure KeyClick; interrupt;
  begin \
    {Порт $60 - порт клавиатуры (также $64 )}
    if Port[$60] < $80 then
    begin  {Генерируем звук при нажатии клавиши}
      Sound(5000);
      Delay(1);
      NoSound;
    end;
    InLine($9C); {PUSHF - запоминаем флаги в стеке}
    {Теперь нужно вернуть событие клавиатуры в цепь обработки
     с использованием сохраненного вектора}
    KbdlntVec;
  end;
  {$F-}

begin
  {Устанавливаем новый обработчик событий клавиатуры}
```

Окончание листинга Ж.130

```
  GetIntVec($9, @KbdIntVec); {9 - прерывание клавиатуры}
  {Переназначаем стандартный обработчик на процедуру KeyClick}
  SetIntVec($9, Addr(KeyClick));
  {Оставляем программу в памяти}
  Keep(0);
end.
```

Эта программа включает встроенный динамик компьютера каждый раз при нажатии клавиши на клавиатуре.

Процедура MsDos

Выполняет вызов функции DOS (прерывание $21).

Синтаксис: MsDos (Regs)

Параметр-переменная Regs — переменная типа Registers, в которой после вызова процедуры сохраняется содержимое регистров.

Пример использования процедуры MsDos представлен в листинге Ж. 131.

Листинг Ж.131. Программа ProcMsDs.pas

```
program ProcMsDs;
uses Crt, Dos;
var
  Year, Month, Day: String;
  Regs : Registers;
begin
  ClrScr;              {Очищаем экран}
  Regs.AH:=$2A;        {Функция считывания текущей даты}
  MsDos(Regs);         {Вызываем прерывание 21H}
  with Regs do         {Считываем содержимое регистров}
  begin
    Str(CX, Year);     (CX - год)
    Str(DH, Month);    (DH - месяц}
    Str(DL, Day);      {DL - день}
  end;
  WriteLn('Сегодня:',Day,'.',Month,'.',Year);
  ReadKey; (Программа ожидает нажатия какой-нибудь клавиши}
end.
```

◀◀ см. также листинг Ж. 129.

Процедура PackTime

Преобразует запись типа DateTime в значение типа Longint.

Синтаксис: PackTime (DT, Time)

Параметр-переменная DT — переменная типа DateTime.

Параметр-переменная Time — переменная типа Longint, в которой сохраняются в упакованном виде дата и время.

- **Пример использования** процедуры PackTime представлен в листинге Ж. 132.

Листинг Ж.132. Программа ProcPkTm.pas

```
program ProcPkTm;
uses Crt, Dos;
var
  nf: string;
  DT: DateTime;
  Time: LongInt;
  F: File;
  Y,Mo,D,DW,H,Mi,S,S100: Word;
begin
  ClrScr; {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(nf);
  Assign(F, nf); {Связываем имя файла с файловой переменной}
  {Ивзлекаем текущие дату и время}
  GetDate(Y,Mo,D,DW);
  GetTime(H,Mi,S,S100);
  with DT do
  begin
    Year := Y;
    Month := Mo;
    Day := D;
    Hour := H;
    Min := Mi;
    Sec := S;
  end;
  PackTime(DT,Time); {Упаковываем дату и время}
  Reset(F);          {Открываем файл F}
  SetFTime(F,Time); {Изменяем дату и время модификации файла F}
  Close(F);          {Закрываем файл F}
end.
```

СОВЕТ

Для тестирования программы ProcPkTm выбирайте пользовательские файлы, например, с расширением PAS, чтобы случайно не изменить атрибуты системных файлов.

Процедура SetCBreak

Устанавливает состояние проверки DOS нажатия комбинации клавиш <Ctrl+Break>.

Синтаксис: SetCBreak (Break)

Параметр Break — значение типа Boolean. Если этот параметр имеет значение True, то проверка нажатия комбинации клавиш <Ctrl+Break> выполняется при каждом системном вызове. В противном случае проверка выполняется только при выполнении операций ввода/вывода на консоль и принтер.

Пример использования процедуры SetCBreak представлен в листинге Ж. 133.

Листинг Ж.133. Программа ProcSCBr.pas

```
program ProcSCBr;
uses Crt, Dos;
const
```

Окончание листинга Ж.133

```
  OffOn: array [Boolean] of String[4] = ('выкл','вкл');
var
  Cb: Boolean;
begin
  ClrScr;  {Очищаем экран}
  GetCBreak(Cb);
  Writeln('Проверка на <Ctrl+Break>: ', OffOn[Cb]);
  Cb := Not(Cb);
  Writeln('Переключение проверки на: ', OffOn[Cb]);
  SetCBreak(Cb);
  GetCBreak(Cb);
  Writeln('Проверка на <Ctrl+Break>: ', OffOn[Cb]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура SetDate

Устанавливает текущую дату в операционной системе.

Синтаксис: SetDate(Year, Month, Day)

Параметры Year, Month и Day — значение типа Word, определяющие год, месяц и день новой даты.

Пример использования процедуры SetDate представлен в листинге Ж. 134.

Листинг Ж.134. Программа ProcSDat.pas

```
program ProcSDat;
uses Crt, Dos;
var
  Y,M,D: Word;
begin
  ClrScr;  (Очищаем экран}
  Write('Укажите год: ');
  Readln(Y);
  Write('Укажите месяц: ');
  Readln(M);
  Write('Укажите день: ');
  Readln(D);
  SetDate(Y,M,D); {Устанавливаем в системе указанную дату}
end.
```

Процедура SetFAttr

Устанавливает атрибуты файла.

Синтаксис: SetFAttr(F, Attr)

Параметр-переменная F — переменная любого файлового типа.

Параметр Attr — значение типа Word, определющее атрибуты файла. Параметру Attr соответсвует сумма констант модуля Dos: 1 — Readonly (только для чтения), 2 — Hidden (скрытый), 4 — System (системный), 8 — VolumeID (идентификатор каталога), 16 — Directory (каталог), 32 — Archive (архивный), 63 — AnyFile (любой файл).

Пример использования процедуры SetFAttr представлен в листинге Ж.135.

Листинг Ж.135. Программа ProcSFAt.pas

```
program ProcSFAt;
uses Crt, Dos;
var
 s: string;
 F: File;
 Attr: Word;
begin
  ClrScr;   {Очищаем экран}
  Write('Введите имя файла: ');
  Readln(s);
  Assign(F, s); {Связываем имя файла с файловой переменной}
  {Определяем атриубуты "только для чтения" и "скрытый файл"}
  Attr := Readonly + Hidden;
  Reset(F); {Открываем файл}
  SetFAttr(F, Attr);
  Close(F); {Закрываем файл}
end.
```

◀◀ см. совет из раздела посвященного процедуре PackTime.

Процедура SetFTime

Устанавливает дату и время последней модификации файла.

Синтаксис: SetFTime(F, Time)

Параметр-переменная F — переменная любого файлового типа.

Параметр Time — значение типа Longint, определяющее дату и время последней модицикации файла F в упакованном виде.

▶▶ Для упаковки значения типа DateTime используется процедура PackTime.

Пример использования процедуры SetFTime представлен в листинге Ж.132 (◀◀ см. раздел, посвященный процедуре PackTime).

Процедура SetIntVec

Устанавливает указанный вектор прерывания на указанный адрес.

Синтаксис: SetIntVec(IntNo, Vector)

Параметр IntNo — значение типа Byte, определяющее номер вектора прерывания.

Параметр Vector — адрес (указатель).

Пример использования процедуры SetIntVec представлен в листинге Ж.126 (◀◀ см. раздел, посвященный процедуре GetIntVec).

Процедура SetTime

Устанавливает текущее время в операционной системе.

Синтаксис: SetTime(Hour, Minute, Second, Sec100)

Параметры Hour, Minute, Second и Sec100 — значения типа Word, определяющие час, минуту, секунду и сотую долю секунды нового времени.

Пример использования процедуры SetTime представлен в листинге Ж.136.

Листинг Ж.136. Программа ProcStTm.pas

```
program ProcStTm;
uses Crt, Dos;
var
  H,M,S: Word;
begin
  ClrScr;   {Очищаем экран}
  Write('Укажите час: ');
  Readln(H);
  Write('Укажите минуту: ');
  Readln(M);
  Write('Укажите секунду: ');
  Readln(S);
  SetTime(H,M,S,0) ; {Устанавливаем в системе указанное время}
end.
```

Процедура SetVerify

Устанавливает состояние флажка проверки DOS записи на диск.

Синтаксис: SetVerify (Verify)

Параметры Verify — значения типа Boolean, определяющие состояние флажка проверки записи на диск: Значению True соответствует установка флажка.

Пример использования процедуры SetVerify представлен в листинге Ж.137.

Листинг Ж.137. Программа ProcSVrf.pas

```
program ProcSVrf;
uses Crt, Dos;
const
  OffOn: array [Boolean] of string[4] = ('выкл', 'вкл');
var
  v: Boolean;
begin
  ClrScr;   {Очищаем экран}
  GetVerify(V);
  WriteLn('Проверка записи : ', OffOn[V]);
  V := not V ;
  WriteLn('Проверка переключена на ', OffOn[V]);
  SetVerify(V);
  GetVerify(V);
  WriteLn('Проверка записи : ', OffOn[V]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура SwapVectors

Обменивает указатели SaveIntXX, объявленные в модуле System, с текущими векторами.

Синтаксис: SwapVectors

Пример использования процедуры SwapVectors представлен в листинге Ж.112 (◄◄ см. раздел, посвященный функции DosExitCode).

Процедура UnpackTime

Преобразует значение типа LongInt в запись типа DateTime.

Синтаксис: UnpackTime(Time, DT)

Параметр Time — значение типа Longint, определяющее дату и время в упакованном виде.

Параметр-переменная DT — переменная типа DateTime, в которой сохраняется значение времени в распакованном виде.

Пример использования процедуры UnpackTime представлен в листинге Ж.125 (◄◄ см. раздел, посвященный процедура GetFTime).

Модуль Graph

Во многих примерах, рассматриваемых в данном подразделе, выполняется инициализация графического режима и проверка на наличие ошибки. Для того чтобы не повторять сответствующий фрагмент программного кода в каждом примере, он был вынесен в отдельный модуль IniGraph.pas, представленаный в листинге Ж.138.

Листинг Ж.138. Модуль IniGraph.pas

```
unit IniGraph;
interface
uses Graph;

   procedure GraphInit;

implementation

  procedure Graphlnit;
  var
    DriverVar, ModeVar, ErrorCode: integer;
  begin
    DriverVar := EGA;
    ModeVar := EGAHI;
    InitGraph(DriverVar,ModeVar,'\tp\bgi');
    ErrorCode := GraphResult;
    if ErrorCode <> grOK then
    begin
      Writeln(GraphErrorMsg(ErrorCode));
      Halt(1);
    end;
  end;

end.
```

Для использования функции Graphlnit в программе необходимо указать в разделе uses ссылку на модуль IniGraph, при этом путь к откомпилированному модулю IniGraph.tpu должен быть указан в поле **Unit directories** диалогового окна **Directories** (см. рис. 1.4).

◄◄ см. примечание в начале раздела "Арифметические вычисления" этого приложения.

Функции

Функция GetBkColor

Возвращает текущий цвет фона как значение в диапазоне от 0 до 15.

Синтаксис: GetBkColor

Тип возвращаемого результата: Word.

Пример использования функции GetBkColor представлен в листинге Ж.139.

Листинг Ж.139. Программа FuncGBkC.pas

```
program FuncGBkC;
uses Crt, Graph, IniGraph;
const
  Colors: array[0..15] of String =
          ('Черный','Синий','Зеленыйn','Бирюзовый','Красный',
           'Малиновый','Коричневый','Светло-серый','Темно-серый',
           'Светло-синий','Светло-зеленый','Светло-бирюзовый',
           'Светло-красный','Светло-малиновый','Желтый','Белый');
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  {Выбираем случайным образом цвет фона}
  SetBkColor(Random(16));
  {Устанавливаем контрастный цвет вывода текста}
  SetColor(15-GetBkColor);
  {Выводим текст}
  OutText('Цвет фона - ' + Colors[GetBkColor]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetColor

Возвращает текущий цвет рисования как значение в диапазоне от 0 до 15.

Синтаксис: GetColor

Тип возвращаемого результата: Word.

Пример использования функции GetBkColor представлен в листинге Ж. 140.

Листинг Ж.140. Программа FuncGetC.pas

```
program FuncGetC;
uses Crt, Graph, IniGraph;
const
  Colors: array[0..15] of String =
          ('Черный','Синий','Зеленыйn','Бирюзовый','Красный',
           'Малиновый','Коричневый','Светло-серый','Темно-серый',
           'Светло-синий','Светло-зеленый','Светло-бирюзовый',
           'Светло-красный','Светло-малиновый','Желтый','Белый');
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
```

Окончание листинга Ж.140

```
  {Выбираем случайным образом цвет рисования}
  SetColor(Random(16));
  {Устанавливаем контрастный цвет фона}
  SetBkColor(15-GetColor);
  {Выводим текст}
  OutText('Цвет рисования - ' + Colors[GetColor]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetDriverName

Возвращает строку с именем текущего драйвера.

Синтаксис: GetDriverName

Тип возвращаемого результата: String.

Пример использования функции GetDriverName представлен в листинге Ж. 141.

Листинг Ж.141. Программа FuncGDrN.pas

```
program FuncGDrN;
uses Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  OutText(GetDriverName); {Выводим имя драйвера}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetGraphMode

Возвращает номер текущего графического режима.

Синтаксис: GetGraphMode

Тип возвращаемого результата: Integer.

Пример использования функции GetGraphMode представлен в листинге Ж. 142.

Листинг Ж.142. Программа FuncGGrM.pas

```
program FuncGGrM;
uses Crt, Graph, IniGraph;
var
  OldGrMode: Integer;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  OutText('Графический режим EGAHI');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  OldGrMode := GetGraphMode;
  SetGraphMode(0);
  OutText('Графический режим EGALO');
  ReadKey;
  SetGraphMode(OldGrMode);
  OutText('Возврат в режим EGAHI');
```

Окончание листинга Ж.142

```
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetMaxColor

Возвращает максимальный номер цвета, который может быть передан в процедуру SetColor.

Синтаксис: GetMaxColor

Тип возвращаемого результата: Word.

Пример использования функции GetMaxColor представлен в листинге Ж. 143.

Листинг Ж.143. Программа FuncGMxC.pas

```
program FuncGMxC;
uses Crt, Graph, IniGraph;
const
  Colors: array[0..15] of String =
          ('Черный','Синий','Зеленыйn','Бирюзовый','Красный',
           'Малиновый','Коричневый','Светло-серый','Темно-серый',
           'Светло-синий','Светло-зеленый','Светло-бирюзовый',
           'Светло-красный','Светло-малиновый','Желтый','Белый');
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  OutText('Цвет с максимальным номером - '+Colors[GetMaxColor]);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetMaxMode

Возвращает максимальный номер доступного в данный момент видеорежима.

Синтаксис: GetMaxMode

Тип возвращаемого результата: Integer.

Пример использования функции GetMaxMode представлен в листинге Ж. 144.

Листинг Ж.144. Программа FuncGMxM.pas

```
program FuncGMxM;
uses Crt, Graph, IniGraph;
var
  S: string;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Str(GetMaxMode,s);
  OutText('Максимальный номер доступного режима - '+s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetMaxX

Возвращает максимальную координату по горизонтали.

Синтаксис: GetMaxX

Тип возвращаемого результата: Integer.

Пример использования функции GetMaxX представлен в листинге Ж. 145.

Листинг Ж.145. Программа FuncGMxX.pas

```
program FuncGMxX;
uses Crt, Graph, IniGraph;
var
  S: string;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Str(GetMaxX,s);
  OutText('Максимальная ккордината по горизонтали - '+s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetMaxY

Возвращает максимальную координату по вертикали.

Синтаксис: GetMaxY

Тип возвращаемого результата: Integer.

Пример использования функции GetMaxY представлен в листинге Ж. 146.

Листинг Ж.146. Программа FuncGMxY.pas

```
program FuncGMxY;
uses Crt, Graph, IniGraph;
var
  S: string;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Str(GetMaxY,s);
  OutText('Максимальная ккордината по вертикали - '+s);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetModeName

Возвращает строку с именем указанного графического режима.

Синтаксис: GetModeName (ModeNumber)

Параметр ModeNumber — значение типа Integer, определяющее номер графического режима.

Тип возвращаемого результата: String.

Пример использования функции GetModeName представлен в листинге Ж. 147.

Листинг Ж.147. Программа FuncGMdN.pas

```
program FuncGMdN;
uses Crt, Graph, IniGraph;
var
  i: integer;
  S: string;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 0 to GetMaxMode do
  begin
    Str(i,s);
    MoveTo(1,i*20);
    OutText('Графический режим с номером '+s+' - '+GetModeName(i));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetPaletteSize

Возвращает размер палитры.

Синтаксис: GetPaletteSize

Тип возвращаемого результата: Integer.

Пример использования функции GetPaletteSize представлен в листинге Ж. 148.

Листинг Ж. 148. Программа FuncGPSz.pas

```
program FuncGPSz;
uses Crt, Graph, IniGraph;
var
  i: integer;
  S: string;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 1 to GetPaletteSize do
  begin
    SetColor(i);
    MoveTo(1,i*10);
    LineTo(100,i*10);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetPixel

Возвращает цвет точки с указанными координатами.

Синтаксис: GetPixel(X,Y)

Тип возвращаемого результата: Word.

Пример использования функции GetPixel представлен в листинге Ж.149.

Листинг Ж.149. Программа FuncGPix.pas

```
program FuncGPix;
uses Crt, Graph, IniGraph;
const
  Colors: array[0..15] of String =
          ('Черный', 'Синий', 'Зеленыйп', 'Бирюзовый', 'Красный',
           'Малиновый','Коричневый','Светло-серый','Темно-серый',
           'Светло-синий','Светло-зеленый','Светло-бирюзовый',
           'Светло-красный','Светло-малиновый','Желтый','Белый');

  procedure DetectColor(Color: Byte);
  begin
    ClearDevice;
    SetFillStyle(SolidFill,Color);
    FillEllipse(100,50,30,23);
    OutText(Colors[GetPixel(100,50)]);
    ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  end;

begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  DetectColor(Blue);
  DetectColor(Red);
  CloseGraph; {Выходим из графического режима}
end.
```

Функция GetX

Возвращает координату X текущей позиции на экране.

Синтаксис: GetX

Тип возвращаемого результата: Integer.

Пример использования функции GetX представлен в листинге Ж.150.

Листинг Ж.150. Программа FuncGetX.pas

```
program FuncGetX;
uses Crt, Graph, IniGraph;
var
  i: integer;
  S: string;
begin
  Graphlnit;        {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 1 to 5 do
  begin
    Str(GetX,S);    {Преобразуем координату X в строку тектса S}
    OutText(S+' '); {Вывод на экран координаты X}
                     со смещением позиции вправо}
    ReadKey;        {Программа ожидает нажатия какой-нибудь клавиши}
  end;
  CloseGraph;       {Выходим из графического режима}
end.
```

Функция GetY

Возвращает координату Y текущей позиции на экране.

Синтаксис: GetY

Тип возвращаемого результата: Integer.

Пример использования функции GetY представлен в листинге Ж.151.

Листинг Ж.151. Программа FuncGetY.pas

```
program FuncGetY;
uses Crt, Graph, IniGraph;
var
  i: integer;
  S: string;
begin
  GraphInit;          {Процедура из модуля IniGraph (листинг Ж. 138)}
  for i := 1 to 5 do
  begin
    MoveTo(1,i*10); {Переносим текущую позицию в
                     точку на экране с координатами (1,i*10)}
    Str(GetY,S);    {Преобразуем координату Y в строку тектса S}
    OutText(S+' '); {Вывод на экран координаты Y}
                     со смещением позиции вниз}
    ReadKey;        {Программа ожидает нажатия какой-нибудь клавиши}
  end;
  CloseGraph;       {Выходим из графического режима}
end.
```

Функция GraphErrorMsg

Возвращает строку с сообщением об ошибке.

Синтаксис: GraphErrorMsg.

Тип возвращаемого результата: string.

Пример использования функции GraphErrorMsg представлен в листинге Ж.152.

Листинг Ж.152. Программа FuncGrEM.pas

```
program FuncGrEM;
uses Crt, Graph;
var
  DriverVar, ModeVar, ErrorCode: integer;
begin
  DriverVar := EGA;
  ModeVar := EGAHI;
  InitGraph(DriverVar,ModeVar,'\tp\bgi'); {Инициализация
                                           графического режима}
  ErrorCode := GraphResult; {Возврат кода ошибки
                             последней графической операции}
  if ErrorCode <> grOK then
  begin
    Writeln(GraphErrorMsg(ErrorCode));
    Halt(1);  {Прекращение выполнения программы}
  end else OutText('Инициализация выполнена успешно');
```

Окончание листинга Ж.152

```
  ReadKey;  ,  {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

ПРИМЕЧАНИЕ

Путь к каталогу bgi в процедуре InitGraph указывайте в соответствии с его размещением на конкретном компьютере.

◄◄ см. примечание в начале раздела "Арифметические вычисления" этого приложения.

Функция GraphResult

Возвращает код ошибки последней графической операции.

Синтаксис: GraphResult

Тип возвращаемого результата: Integer.

Пример использования функции GraphResult представлен в листинге Ж.152 (◄◄ см. предыдущий раздел).

Функция ImageSize

Возвращает число байтов, требуемое для заполнения прямоугольной области экрана.

Синтаксис: ImageSize (X1,Y1,X2,Y2)

Параметры X1, Y1, X2, Y2 — значения типа Integer, определяющие координаты верхнего левого и нижнего правого углов прямоугольной области экрана.

Тип возвращаемого результата: Word.

Пример использования функции ImageSize представлен в листинге Ж.153.

Листинг Ж.153. Программа FuncImgS.pas

```
program FuncImgS;
uses Crt, Graph, IniGraph;
var
  P: Pointer;
  Size: Word;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  {Заполняем экран}
  SetFillStyle(XHatchFill, Cyan);
  Bar(0, 0, GetMaxX, GetMaxY);
  {Определяем размер области экрана}
  Size := ImageSize(10,20,30,40);
  GetMem(P, Size);   {Распределяем память в куче}
  {Сохраняем область экрана в памяти}
  GetImage(10,20,30,40,P^); {Сохраняем битовый образ
                             указанной части экрана в памяти}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  ClearDevice; {Очищаем экран}
  {Извлекаем область экрана из памяти}
  PutImage(100,100,P^,NormalPut); ,
  ReadKey;
```

Окончание листинга Ж.153

```
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы FuncImgS представлен на рис. Ж.8.

Функция InstallUserFont

Устанавливает новый шрифт, не встроенный в систему BGI (Borland Graphical Interface).

Синтаксис: InstallUserFont(FontFileName)

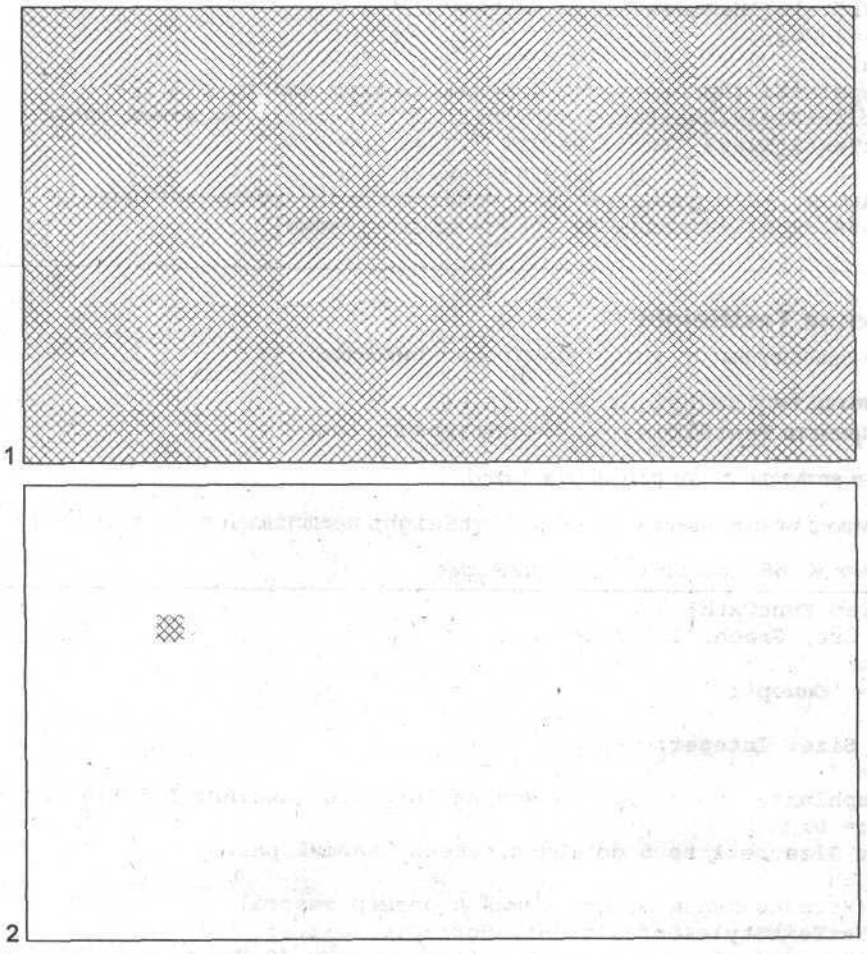


Рис. Ж.8. Сохранение области на экране в памяти (1) и вывод ее на экран (2)

Параметр FontFileName — значение типа String, определяющее имя файла шрифта.

Тип возвращаемого результата: Integer. Возвращается номер добавленного шрифта, который затем можно указать при вызове процедуры SetTextstyle.

Пример использования функции InstalluserFont представлен в листинге Ж.154.

Листинг Ж.154. Программа FuncInUF.pas

```
program FuncInUF;
uses Crt, Graph, IniGraph;
var
  UserFont: Integer;
begin
  UserFont := InstallUserFont('user.chr');
  if GraphResult <> grOk then
  begin
    Writeln('Ошибка установки шрифта (используется DefaultFont)');
    Writeln('Нажмите любую клавишу...');
    ReadKey;
  end;
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  SetTextStyle(UserFont, HorizDir, 2); {Используем новый шрифт}
  OutText('Книга "Turbo Pascal 7.0 на примерах".
          Автор Ю.А.Шпак. Издательство "Юниор"');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Функция TextHeight

Возвращает высоту указанной строки в пикселях.

Синтаксис: TextHeight(TextString)
Параметр TextString — значение типа String.

Тип возвращаемого результата: Word.

Пример использования функции TextHeight представлен в листинге Ж.155.

Листинг Ж.155. Программа FuncTxtH.pas

```
program FuncTxtH;
uses Crt, Graph, IniGraph;
const
  S = 'Юниор';
var
  Y, Size: Integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Y := 0;
  for Size := 1 to 5 do {Цикл, увеличивающий размер текста}
  begin
    {Устанавливаем шрифт, стиль и размер текста}
    SetTextStyle(DefaultFont, HorizDir, Size);
    {Выводим текст в позиции с координатами (0,Y)}
    OutTextXY(0, Y, S);
    {Увеличиваем координату Y позиции на экране на высоту
     текущей строки — для смещения позиции на экране вниз}
    Inc(Y, TextHeight(S));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
```

Окончание листинга Ж.155

```
  CloseGraph;  {Выходим из графического режима}
end.
```

Результат работы программы FuncTxtH представлен на рис. Ж.9.

Функция TextWidth

Возвращает ширину указанной строки в пикселях.

Синтаксис: TextWidth (TextString)
Параметр TextString — значение типа String.

Тип возвращаемого результата: Word.

Пример использования функции TextWidth представлен в листинге Ж.156.

Листинг Ж.156. Программа FuncTxtW.pas

```
program FuncTxtW;
uses Crt, Graph, IniGraph;
const
  S = 'Юниор';
var
  Row, Size: Integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Row:=0;
  Size:=1;
  while TextWidth(S) < GetMaxX do {Пока ширина строки не}
  begin                           {превышает границы экрана}
    {Выводим текст в позицию с координатами (0,Row)}
    OutTextXY(0, Row, S);
    {Увеличиваем координату Y (переменная Row) позиции на экране на
     высоту строки 'M' — для смещения позиции на экране вниз}
    Inc(Row,TextHeight('M'));
    Inc(Size); {Увеличиваем размер текста на 1}
    {Устанавливаем шрифт, стиль и размер текста}
    SetTextStyle(DefaultFont, HorizDir, Size);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы FuncTxtW представлен на рис. Ж.10.

Процедуры

Процедуры рисования

Процедура Arc

Рисует дугу.

Синтаксис: Arc(X, Y, StAngle, EndAngle, Radius)
Параметры X и Y — значения типа Integer, определяющие координаты центра дуги.

Рис. Ж.9. Вывод на экран строк текста разного размера

Рис. Ж.10. Результат работы программы FuncTxtW

Параметры StAngle и EndAngle — значения типа Word, определяющие начальный и конечный угол (в градусах).

Параметр Radius — значение типа Word, определяющее радиус дуги.

Пример использования процедуры Arc представлен в листинге Ж. 157.

Листинг Ж.157. Программа ProcArc.pas

```
program ProcArc; »
uses ,Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Arc(100,100,0,90,50);   {Четверть окружности}
  Arc(300,100,0,180,50);  {Половина окружности}
  Arc(500,100,0,360,50);  {Полная окружность}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcArc представлен на рис. Ж.11.

Рис. Ж.11. Создание дуг и эллипсов (окружностей)

Процедура Bar

Рисует закрашенный прямоугольник.

Синтаксис: Bar(X1, Y1, X2, Y2)

Параметры X1, Y1, X2 и Y2 — значения типа Integer, определяющие координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов прямоугольника.

Пример использования процедуры Bar представлен в листинге Ж. 158.

Листинг Ж.158. Программа ProcBar.pas

```
program ProcBar;
uses Crt, Graph, IniGraph;
var
  i: byte;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 0 to 15 do
  begin
    {Устанавливаем стиль заполнения прямоугольника: сплошной заливкой
     цвета- I.                    ◀◀ см. о соответствии констант}
    SetFillStyle (SolidFill,i) ;  {от 0 до 15 цвету в табл. Ж.1}
    Bar(0, i*(GetMaxY div 16), GetMaxX, (i+1)*(GetMaxY div 16));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcBar представлен на рис. Ж. 12.

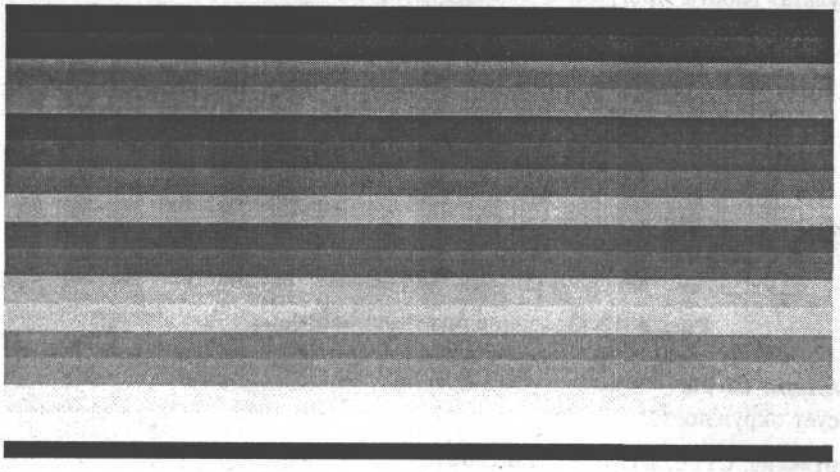


Рис. Ж.12. Создание закрашенных прямоугольников

Процедура Bar3D

Рисует параллелепипед.

Синтаксис: Bar**3D**(X1, Y1, X2, Y2, Depth, Top)

Параметры X1, Y1, X2 и Y2 — значения типа **Integer**, определяющие координаты левого верхнего (X1, Y1) и правого нижнего (X2, Y2) углов прямоугольника.

Параметр Depth — значение типа Word, определяющее глубину параллелепипеда.

Параметр Top — значение типа Boolean, определяющее, рисуется ли верхняя грань параллелепипеда.

Пример использования процедуры Bar3D представлен в листинге Ж. 159.

Листинг Ж.159. Программа ProcBar3.pas

```
program ProcBar3;
uses Crt, Graph, IniGraph;
var
  i: byte;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 0 to 15 do
  begin
    {Устанавливаем стиль заполнения прямоугольника: сплошной заливкой
     цвета I.                    ◄◄ см. о соответствии констант}
    SetFillStyle(SolidFill,i); {от 0 до 15 цвету в табл. Ж.1}
    Bar3D(i*(GetMaxX div 16), 10, i*(GetMaxX div 16)+20, 200, 5, True);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcBar3 представлен на рис. Ж.13.

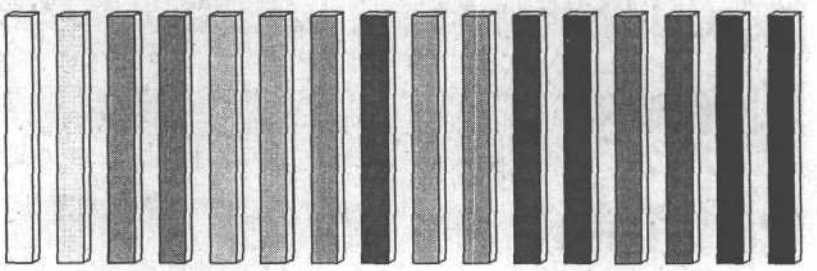


Рис. Ж.13. Создание разноцветных параллелепипедов

Процедура Circle

Рисует окружность.

Синтаксис: Circle(X, Y, Radius)

Параметры X и Y — значения типа Integer, определяющие координаты центра окружности.

Параметр Radius — значение типа Word, определяющее радиус окружности.

Пример использования процедуры circle представлен в листинге Ж.160.

Листинг Ж.160. Программа ProcCrcl.pas

```
program ProcCrcl;
uses Crt, Graph, IniGraph;
var
  i: byte;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  for i := 1 to 10 do
  begin
```

Окончание листинга Ж.160

```
    {Устанавливаем случайный цвет рисования}
    SetColor(1+Random(15));
    {Создаем 10 окружностей со случайными параметрами}
    Circle(Random(GetMaxX),Random(GetMaxY),Random(100));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcCrcl представлен на рис. Ж. 14.

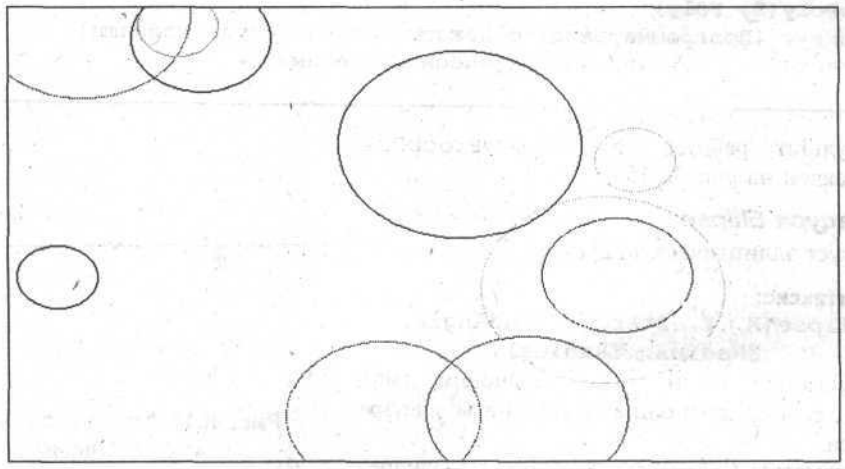


Рис. Ж.14. Хаотическое создание разноцветных окружностей на экране

Процедура DrawPoly

Рисует многоугольник.

Синтаксис: DrawPoly(NumPoints, PolyPoints)

Параметр NumPoints — значение типа Word, определяющие количество отображаемых вершинв, заданных параметром PolyPoints. Каждая вершина многоугольника определяется двумя значениями — X и Y.

Параметр-переменная PolyPoint — массив значений типа PointType, объявление которого имеет следующий вид:

```
type
  PointType = Record
    X, Y: Integer;
  end;
```

Пример использования процедуры DrawPoly представлен в листинге Ж. 161.

Листинг Ж.161. Программа ProcDPol.pas

```
program ProcDPol;
uses Crt, Graph, IniGraph;
var
  i: byte;
```

Окончание листинга Ж.161

```
  Poly: array[1..5] of PointType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  {Выбираем случайным образом координаты вершин}
  for i := 1 to 5 do
  begin
    Poly[i].X := Random(GetMaxX);
    Poly[i].Y := Random(GetMaxY);
  end;
  DrawPoly(5, Poly);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcDPol представлен на рис. Ж.15.

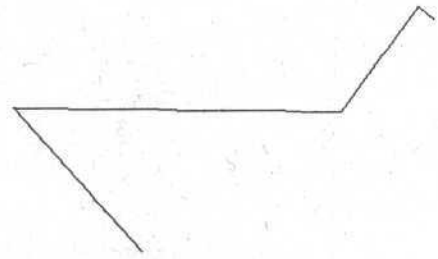


Рис. Ж.15. Процесс создания многоугольника

Процедура Ellipse

Рисует эллиптическую дугу.

Синтаксис:

```
Ellipse(X, Y, StAngle, EndAngle,
        XRadius, YRadius)
```

Параметры X и Y — значения типа Integer, определяющие координаты центра эллипса.

Параметры StAngle и EndAngle — значения типа Word, определяющие начальный и конечный угол дуги эллипса.

Параметры XRadius и YRadius — значения типа Word, определяющие радиусы эллипса по оси X и по оси Y.

Пример использования процедуры Ellipse представлен в листинге Ж.162.

Листинг Ж.162. Программа ProcElps.pas

```
program ProcElps;
uses Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  {Эллипсы}
  Ellipse(100,100,0,90,20,80);
  Ellipse(300,100,0,180,20,80);
  Ellipse(500,100,0,360,20,80);
  Ellipse(100,220,0,90,80,20);
  Ellipse(300,220,0,180,80,20);
  Ellipse(500,220,0,360,80,20);
  {Окружность}
  Ellipse(300,300,0,360,30,21);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcElps представлен на рис. Ж. 16.

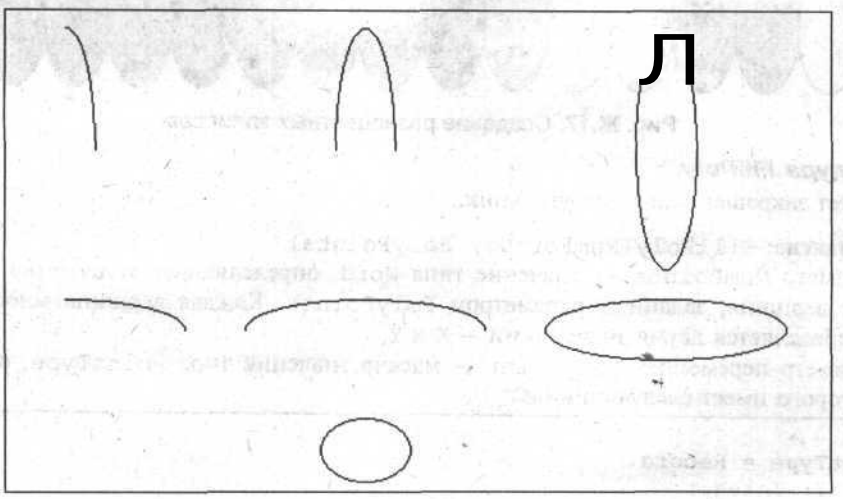


Рис. **Ж.16.** Создание дуг и эллипсов на экране

Процедура FillEllipse

Рисует заполненный эллипс.

Синтаксис: `FillEllipse(X, Y, XRadius, YRadius)`

Параметры X и Y — значения типа Integer, определяющие координаты центра эллипса.

Параметры XRadius и YRadius — значения типа Word, определяющие радиусы эллипса по оси X и по оси Y.

Пример использования процедуры FillEllipse представлен в листинге Ж.163.

Листинг Ж.163. Программа ProcFEll.pas

```
program ProcFEll;
uses Crt, Graph, IniGraph;
var
  i: integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 0 to 15 do
  begin
    {Устанавливаем стиль заполнения прямоугольника: сплошной заливкой
     цвета I.                    см. о соответствии констант}
    SetFillStyle(SolidFill, i) ;  {от 0 до 15 цвету в табл. Ж.1}
    FillEllipse(i*(GetMaxX div 16)+20,100,20,40);
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcFEll представлен на.рис. Ж.17.

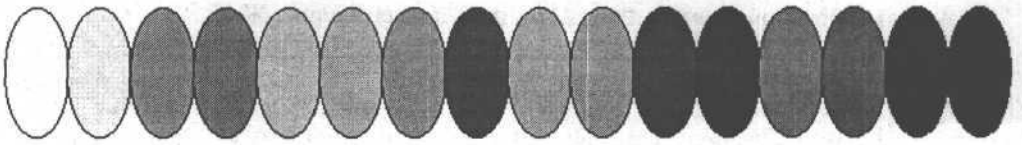


Рис. Ж.17. Создание разноцветных эллипсов

Процедура FillPoly

Рисует закрашенный многоугольник.

Синтаксис: `FillPoly(NumPoints, PolyPoints)`

Параметр NumPoints — значение типа Word, определяющее количество отображаемых вершинв, заданных параметром PolyPoints. Каждая вершина многоугольника определяется двумя значениями — X и Y.

Параметр-переменная PolyPoint — массив значений типа PointType, объявление которого имеет следующий вид:

```
type
  PointType = Record
    X, Y: Integer;
  end;
```

Пример использования процедуры FillPoly представлен в листинге Ж. 164.

Листинг Ж.164. Программа `ProcFPol.pas`

```
program ProcFPol;
uses Crt, Graph, IniGraph;
var
  i: byte;
  Poly: array[1..5] of PointType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  {Выбираем случайным образом координаты вершин}
  for i := 1 to 5 do
  begin
    Poly[i].X := Random(GetMaxX);
    Poly[i].Y := Random(GetMaxY);
  end;
  SetFillStyle(SolidFill,Green); {Устанавливаем стиль
            лнения многоугольника: сплошной заливкой зеленого цвета.
            Рисуем зеленый многоугольник, закрашенный а области
            между случайно выбранными вершинами (от 0 до 5 вершин)}
  FillPoly(Random(6), Poly);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcFPol представлен на рис. Ж. 18.

Процедура FloodFill

Закрашивает замкнутую область.

Синтаксис:
`FloodFillPoly(X, Y, Border)`

Параметры X и Y — значения типа Integer, определяющие координаты начальной точки закраски.

Параметр Border — значение типа Word, определяющее цвет границы, до которой выполняется закраска.

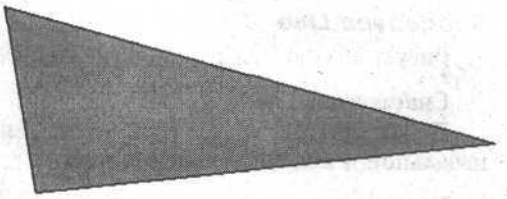


Рис. Ж.18. Закрашенный многоугольник

Пример использования процедуры FloodFill представлен в листинге Ж.165.

Листинг Ж.165. Программа ProcFFil.pas

```
program ProcFFil;
uses Crt, Graph, IniGraph;
var
  i: integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  SetColor(Red); {Устанавливаем цвет рисования}
  Circle(100,100,50); {Окружность красного цвета}
  SetColor(Yellow); {устанавливаем цвет рисования}
  Circle(100,100,80); {Окружность желтого цвета}
  {Устанавливаем стиль заполнения замкнутой области: сплошной
   заливкой зеленого цвета}
  SetFillStyle(SolidFill,Green);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  FloodFill(100,100,Red); {Заполняем внутреннюю окружность}
  ReadKey;
  FloodFill(100,100,Yellow); {Заполняем внешнюю окружность}
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcFFil представлен на рис. Ж.19.

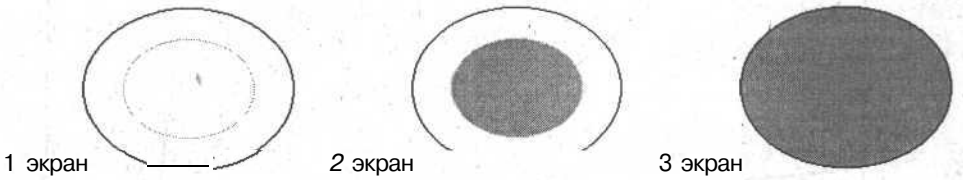


Рис. Ж.19. Закрашивание областей экрана (1) разными цветами (2 и 3)

Процедура GetImage

Сохраняет битовый образ указанной части экрана в памяти.

Синтаксис: GetImage(X1, Y1, X2, Y2, BitMap)

Параметры X1, Y1, X2 и Y2 — значения типа Integer, определяющие координаты верхнего левого и правого нижнего углов сохраняемой области.

Параметр-переменная BitMap — переменная-указатель на область памяти.

Пример использования процедуры Get Image, представлен в листинге Ж.153 (◀◀ см. раздел, посвященный функции ImageSize).

Процедура Line

Рисует линию между двумя указанными точками.

Синтаксис: Line(X1, Y1, X2, Y2)

Параметры X1, Y1, X2 и Y2 — значения типа Integer, определяющие координаты начальной и конечной точек линии.

Пример использования процедуры Line представлен в листинге Ж.166.

Листинг Ж.166. Программа ProcLine.pas

```
program ProcLine;
uses Crt, Graph, IniGraph;
var
  i: integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  for i := 1 to 20 do
  begin
    {Устанавливаем случайный цвет рисования}
    SetColor(1+Random(15));
    {Создаем 20 линий со случайными параметрами}
    Line(Random(GetMaxX),Random(GetmaxY),
         Random(GetMaxX),Random(GetmaxY));
  end;
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcLine представлен на рис. Ж.20.

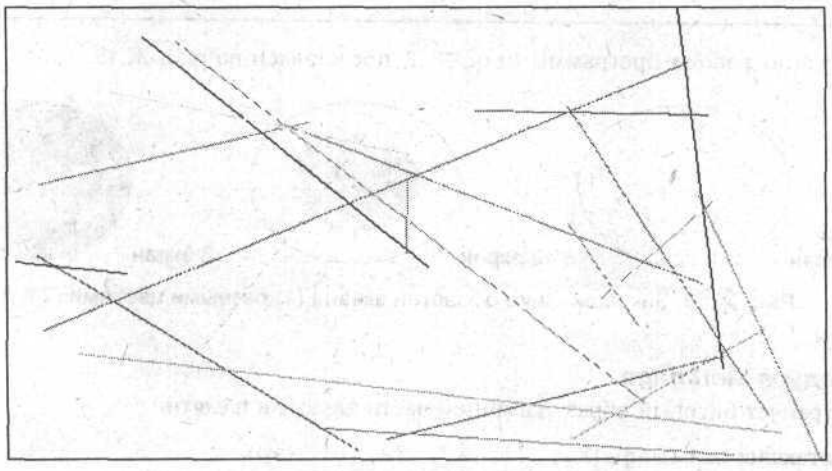


Рис. Ж.20. Хаотическое создание разноцветных линий на экране

Процедура LineRel

Рисует линию от текущей позиции к точке, лежащей на заданном расстоянии.

Синтаксис: LineRel(DX, DY)

Параметры DX и DY — значения типа Integer, определяющие смещение от текущей позиции на экране до конечной точки линии.

Пример использования процедуры LineRel представлен в листинге Ж. 167.

Листинг Ж.167. Программа ProcLnRl.pas

```
program ProcLnRl;
uses Crt, Graph, IniGraph;
var
  c: char;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  while True do
  begin
    {Программа ожидает нажатия какой-нибудь клавиши}
    c := ReadKey;
    case c of
      #27: Break; {Выход по Esc}
      #72: LineRel(0,-10); {Стрелка вверх}
      #75: LineRel(-10,0); {Стрелка влево}
      #77: LineRel(10,0);  {Стрелка вправо}
      #80: LineRel(0,10);  {Стрелка вниз}
    end;
  end;
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcLnRl представлен на рис. Ж.21.

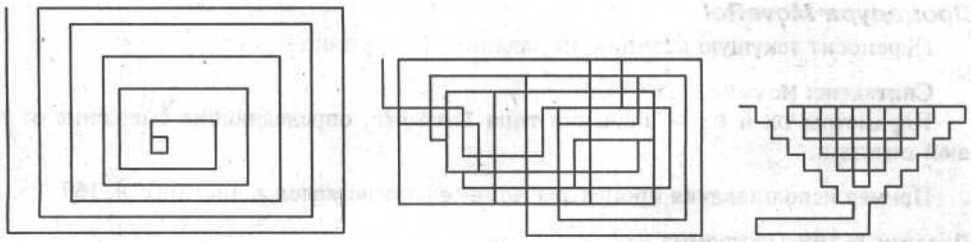


Рис. Ж.21. Фигуры, которые можно создать при помощи программы ProcLnRl

Процедура LineTo

Рисует линию от текущей позиции к указанной точке.

Синтаксис: LineTo(X, Y)

Параметры X и Y — значения типа Integer, определяющие координаты конечной точки.

Пример использования процедуры LineTo представлен в листинге Ж. 168.

Листинг Ж.168. Программа ProcLnTo.pas

```
program ProcLnTo;
uses Crt, Graph, IniGraph;
var
  c: char;
```

Окончание листинга Ж.168

```
  NextX, NextY: Integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  NextX := 0;
  NextY := 0;
  while True do
  begin
    {Программа ожидает нажатия какой-нибудь клавиши}
    c := ReadKey;
    case c of
      #27: Break; {Выход по Esc}
      #72: Dec(NextY,10); {Стрелка вверх: уменьшение
                           координаты Y (переменная NextY) на 10}
      #75: Dec(NextX,10) ; {Стрелка влево}
      #77: Inc(NextX,10); {Стрелка вправо: увеличение
                           координаты X (переменная NextX) на 10}
      #80: Inc(NextY,10); {Стрелка вниз}
    end;
    LineTo(NextX,NextY);
  end;
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcLnTo представлен на рис. Ж.21. Результат работы этой программы аналогичен результату работы программы ProcLnRl (листинг Ж.167).

Процедура MoveRel

Переносит текущую позицию на заданное расстояние.

Синтаксис: MoveRel(DX, DY)

Параметры DX и DY — значения типа Integer, определяющие смещение от текущей позиции.

Пример использования процедуры MoveRel представлен в листинге Ж.169.

Листинг Ж.169. Программа ProcMvRl.pas

```
program ProcMvRl;
uses Crt, Graph, IniGraph;
var
  c: char;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  while True do
  begin
    {Программа ожидает нажатия какой-нибудь клавиши}
    c := ReadKey;
    case c of
      #27: Break; {Выход по Esc}
      #72: begin  {Стрелка вверх}
             MoveRel(0,-5);
             LineRel(0,-10);
           end;
```

Окончание листинга Ж.169

```
      #75: begin  {Стрелка влево}
             MoveRel(-5,0);
             LineRel(-10,0);
           end;
      #77: begin  {Стрелка вправо}
             MoveRel(5,0);
             LineRel(10,0);
           end;
      #80: begin  {Стрелка вниз}
             MoveRel(0,5);
             LineRel(0,10);
           end;
    end;
  end;
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcLnTo представлен на рис. Ж.22. Сравните этот рисунок с результатом работы программы ProcLnRl (листинг Ж.167 — см. рис. 21).

Процедура MoveTo

Переносит текущую позицию в указанную точку.

Синтаксис: MoveTo (X, Y)

Параметры X и Y — значения типа Integer, определяющие координаты новой позиции.

Пример использования процедуры MoveTo представлен в листинге Ж.170.

Листинг Ж.170. Программа ProcMvTo.pas

```
program ProcMvTo;
uses Crt, Graph, IniGraph;
var
  i: integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 1 to 15 do {Рисуем 15 линий}
  begin
    SetColor(i); {Устанавливаем цвет рисования}
    {◄◄ см. о соответствии констант от 0 до 15 цвету в табл. Ж.1}
    MoveTo(10,i*10); {Смещаем позицию вниз}
    LineRel(100,0);  {Рисуем линию}
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcMvTo представлен на рис. Ж.23.

Процедура OutText

Выводит текст в текущей позиции.

Синтаксис: OutText (TextString)

Параметр TextString — значение типа String.

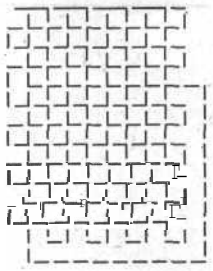


Рис. Ж.22. Рисование в программе ProcLnTo

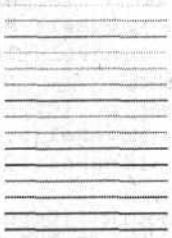


Рис. Ж.23. Создание "зебры" из цветных линий

Пример использования процедуры OutText представлен в листинге Ж.171.

Листинг Ж.171. Программа ProcOutT.pas

```
program ProcOutT;
uses Crt, Graph, IniGraph;
var
  c: char;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  while True do
  begin
    {Программа ожидает нажатия какой-нибудь клавиши}
    c := ReadKey;
    if c = #27 then Break {Выход по Esc}
    else begin
           OutText(c); {Выводим символ}
           {Если позиция выходит за пределы экрана}
           if GetX > GetMaxX then
             MoveTo(0,GetY+15); {Переходим к новой строке}
         end;
  end;
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура OutTextXY

Выводит текст в указанной позиции.

Синтаксис: OutTextXY(X, Y, TextString)

Параметры X и Y — значения типа Integer, определяющие координаты позиции вывода строки.

Параметр TextString — значение типа String.

Пример использования процедуры OutTextXY представлен в листинге Ж. 172.

Листинг Ж.172. Программа ProcOTXY.pas

```
program ProcOTXY;
uses Crt, Graph, IniGraph;
var
  c: char;
begin
```

Окончание листинга Ж.171

```
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  while True do
  begin
    {Программа ожидает нажатия какой-нибудь клавиши}
    c := ReadKey;
    if c = #27 then Break {Выход по Esc}
    else begin
           OutTextXY(GetX,GetY,c); {Выводим символ}
           MoveRel(10,0); {Смещаем позицию влево}
           {Если позиция выходит за пределы экрана}
           if GetX > GetMaxX then
             MoveTo(0,GetY+15); {Переходим к новой строке}
         end;
  end;
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура PieSlice

Рисует и заполняет сектор круга.

Синтаксис: `PieSlice(X, Y, StAngle, EndAngle, Radius)`

Параметры X и Y — значения типа Integer, определяющие координаты вершины сектора.

Параметры StAngle и EndAngle — значения типа Word, определяющие начальный и конечные углы сектора.

Параметр Radius — значение типа Word, определяющее радиус окружности, частью которой является данный сектор.

Пример использования процедуры PieSlice представлен в листинге Ж. 173.

Листинг Ж.173. Программа `ProcPieS.pas`

```
program ProcPieS;
uses Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  SetFillStyle(SolidFill,Blue); {Устанавливаем стиль заполнения дуг
                                 круга: сплошной заливкой синего цвета}
  PieSlice(100,100,0,90,50); {Четверть круга}
  PieSlice(250,100,0,180,50); {Половина круга}
  PieSlice(400,100,0,360,50); {Полный круг}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcPieS представлен на рис. Ж.24.

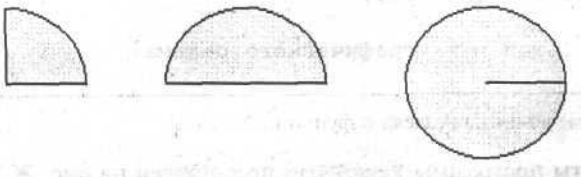


Рис. Ж.24. Создание заполненных секторов круга

Процедура PutImage

Выводит на экран битовый образ.

Синтаксис: PutImage (X, Y, BitMap, BitBIt)

Параметры X и Y — значения типа Integer, определяющие координаты верхнего левого угла области экрана, в которой будет отображен битовый образ.

Параметр BitMap — указатель на область памяти, в которой размещен битовый образ.

Параметр **BitBIt** — значение типа Word, определяющее тип битовой операции. Каждому из возможных значений параметра **BitBIt** соответствует константа модуля Graph: 0 (NormalPut) — обычное отображение; 1 (XORPut) — отображение с применением к каждому байту битового образа операции XOR; 2 (ORPut) — отображение с применением к каждому байту битового образа операции OR; 3 (ANDPut) — отображение с применением к каждому байту битового образа операции AND; 4 (NOTPut) — отображение с применением к каждому байту битового образа операции NOT.

Пример использования процедуры PutImage представлен в листинге Ж.174.

Листинг Ж.174. Программа ProcPImg.pas

```
program ProcPImg;
uses Crt, Graph, IniGraph;
var
  P: Pointer;
  Size: Word;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  {Заполняем экран}
  SetFillStyle(XHatchFill, Cyan); {Устанавливаем стиль
  заполнения прямоугольника: частой штриховкой бюризового цвета}
  Bar(0, 0, GetMaxX, GetMaxY); {Создаем заполненный штриховкой
                                прямоугольник размером в экран}
  {Определяем размер области экрана}
  Size := ImageSize(10,20,30,40);
  GetMem(P, Size);   {Распределяем память в куче}
  {Сохраняем область экрана в памяти}
  GetImage(10,20,30,40,P^);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  {Заполняем экран}
  SetFillStyle(SolidFill, Yellow); {Устанавливаем стиль заполнения
                    прямоугольника: сплошной заливкой желтого цвета}
  Bar(0, 0, GetMaxX, GetMaxY);
  {Извлекаем область экрана из памяти}
  PutImage(100,100,P^,NormalPut);
  PutImage(150,100,P^,XORPut);
  PutImage(200,100,P^,ORPut);
  PutImage(250,100,P^,ANDPut);
  PutImage(300,100,P^,NOTPut);
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

◀◀ см. о куче в примечании из раздела о функции MaxAvail.

Результат работы программы ProcPImg представлен на рис. Ж.25.

Рис. Ж.25. Вывод на экран битового образа при помощи процедуры PutImage с разными значениями параметра **BitBlt** (от 0 до 4)

Процедура PutPixel

Рисует точку.

Синтаксис: `PutPixel(X, Y, Color)`

Параметры X и Y — значения типа Integer, определяющие координаты точки.

Параметр Color — значение типа Word, определяющее цвет точки. Вместо числового значения Color можно использовать соответствующие константы модуля Crt (см. табл. Ж.1).

Пример использования процедуры PutPixel представлен в листинге Ж.175.

Листинг Ж.175. Программа ProcPPxl.pas

```
program ProcPPxl;
uses Crt, Graph, IniGraph;
var
  x,y: Word;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  for x := 1 to GetMaxX do
    for y := 1 to GetMaxY do
      PutPixel(x,y,Random(16));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcPPxl представлен на рис. Ж.26.

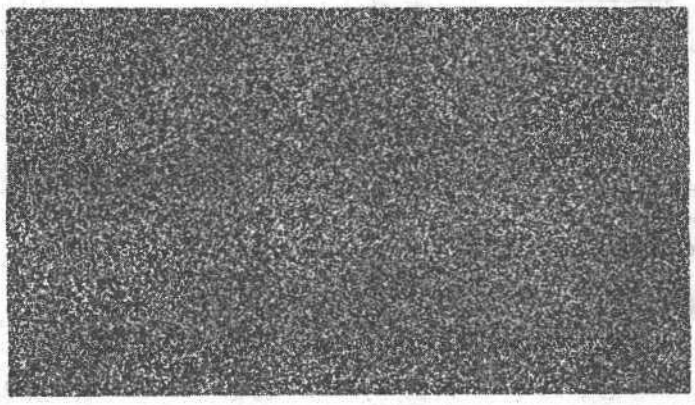


Рис. Ж.26. Создание изображения из хаотически расположенных на экране цветных точек

Процедура Rectangle

Рисует прямоугольник.

Синтаксис: Rectangle(X1, Y1, X2, Y2)

Параметры X1, Y1, X2 и Y2— значения типа Integer, определяющие координаты верхнего левого (X1, Y1) и правого нижнего (X2, Y2) углов прямоугольника.

Пример использования процедуры Rectangle представлен в листинге Ж. 176.

Листинг Ж.176. Программа ProcRect.pas

```
program ProcRect;
uses Crt, Graph, IniGraph;
var
  i: Byte;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  for i := 1 to 10 do {Рисуем 10 прямоугольников}
  begin
    SetColor(1+Random(15)); {Устанавливаем цвет рисования}
    {◄◄ см. о соответствии констант от 0 до 15 цвету в табл. Ж.1}
    Rectangle(Random(GetMaxX),Random(GetMaxY),
              Random(GetMaxX),Random(GetMaxY));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcRect представлен на рис. Ж.27.

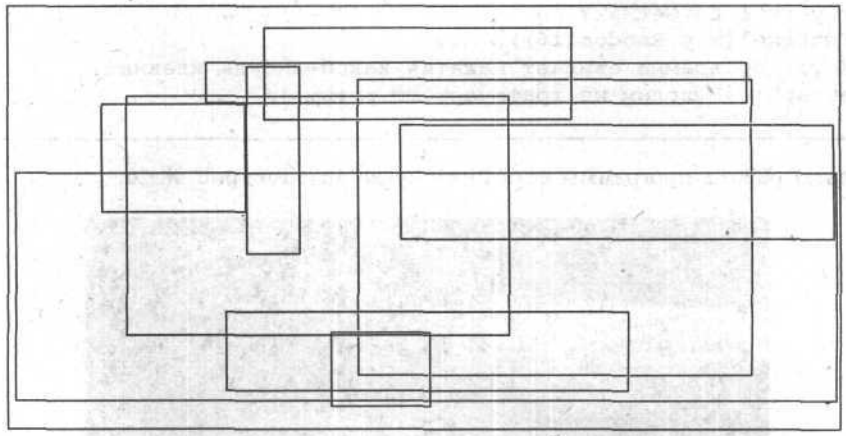


Рис. Ж.27. Создание хаотически расположенных прямоугольников

Процедура Sector

Рисует и заполняет сектор эллипса.

Синтаксис: Sector(X, Y, StAngle, EndAngle, XRadius, YRadius)

Параметры X и Y — значения типа Integer, определяющие координаты вершины сектора.

Параметры StAngle и EndAngle — значения типа Word, определяющие начальный и конечные углы сектора.

Параметры XRadius и YRadius — значения типа Word, определяющие радиусы эллипса по горизонтали и по вертикали.

Пример использования процедуры Sector представлен в листинге Ж.177.

Листинг Ж.177. Программа ProcSect.pas

```
program ProcSect;
uses Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж. 138)}
  SetFillStyle(SolidFill,Blue); {Устанавливаем стиль
        заполнения сектора: сплошной заливкой синего цвета}
  Sector(100,100,0,90,30,40); {Четверть эллипса}
  Sector(250,100,0,180,30,40); {Половина эллипса}
  Sector(400,100,0,360,30,40); {Полный эллипс}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcSect представлен на рис. Ж.28.

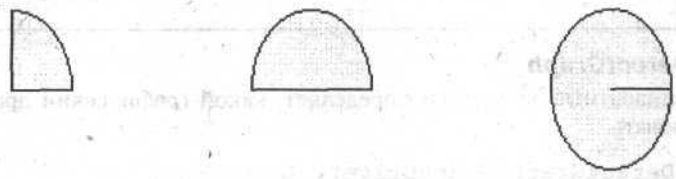


Рис. Ж.28. Создание заполненных секторов эллипса

Управляющие процедуры

Процедура ClearViewPort

Очищает текущее окно.

Синтаксис: ClearViewPort

Пример использования процедуры ClearViewPort представлен в листинге Ж.178.

Листинг Ж.178. Программа ProcClVP.pas

```
program ProcClVP;
uses Crt, Graph, IniGraph;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  {Рисуем прямоугольник}
  Rectangle(19,19,GetMaxX-19,GetMaXY-19);
  {Определяем окно в рамках прямоугольника}
  SetViewPort(20,20,GetMaxX-20,GetMaxY-20,True);
  'OutTextXY(0, 0, 'Нажмите любую клавишу, чтобы очистить окно');
  ReadKey;
  ClearViewPort;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
```

Окончание листинга Ж.178

```
  CloseGraph;  {Выходим из графического режима}
end.
```

Процедура CloseGraph

Выход из графического режима.

Синтаксис: CloseGraph

Пример использования процедуры CloseGraph представлен в листинге Ж.179.

Листинг Ж.179. Программа ProcClsG.pas

```
program ProcClsG;
uses Crt, Graph, IniGraph;
begin
  Writeln('Текстовыйрежим');
  ReadKey;
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  OutTextXY(0, 0, 'Графическийрежим');
  ReadKey;
  CloseGraph; {Выходим из графического режима}
  Writeln('Опять текстовый режим');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедура DetectGraph

Тестирует аппаратные средства и определяет, какой графический драйвер и режим можно использовать.

Синтаксис: DetectGraph(GraphDriver, GraphMode)

Пример использования процедуры DetectGraph представлен в листинге Ж. 180.

Листинг Ж.180. Программа ProcDetG.pas

```
program ProcDetG;
uses Crt, Graph;
var
  DriverVar, ModeVar: integer;
begin
  DetectGraph(DriverVar,ModeVar);
  InitGraph(DriverVar,ModeVar,'\tp\bgi');
  OutTextXY(1,1,'Драйвер - ' + GetDriverName);
  OutTextXY(1,10,'Режим - ' + GetModeName(ModeVar));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

ПРИМЕЧАНИЕ

Путь к каталогу bgi в процедуре InitGraph указывайте в соответствии с его размещением на конкретном компьютере.

◀◀ см. примечание в начале раздела "Арифметические вычисления" этого приложения.

Процедура GetArcCoords

Извлекает координаты процедуры Arc.

Синтаксис: GetArcCoords (ArcCoords)

Параметр-переменная ArcCoords — переменная типа ArcCoordsType, который объявлен в модуле Graph следующим образом:

```
type
  ArcCoordsType = Record
                    X, Y: Integer;
    XStart, YStart: Integer;
        XEnd, YEnd: Integer;
  end;
```

Где поля: X и Y — координаты центра дуги;
XStart и YStart — координаты начальной точки дуги;
XEnd и YEnd — координаты конечной точки дуги.

Пример использования процедуры GetArcCoords представлен в листинге Ж. 181.

Листинг Ж.181. Программа ProcGtAC.pas

```
program ProcGtAC;
uses Crt, Graph, IniGraph;
var
  ArcCoords: ArcCoordsType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Arc(100,100,0,270,75); {Рисуем дугу}
  GetArcCoords(ArcCoords);
  with ArcCoords do
    Line(XStart,YStart,XEnd,YEnd); {Рисуем линию}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcGtAC представлен на рис. Ж.29.

Рис. Ж.29. Совмещение конечных точек дуги и линии при помощи процедуры GetArcCoords

Процедура GetAspectRatio

Извлекает два значения, на основании которых может быть вычислено отношение разрешения по горизонтали к разрешению по вертикали (так называемый *коэффициент сжатия).*

Синтаксис: GetAspectRatio(XAsp, YAsp)

Параметры-переменные XAsp и YAsp — переменные типа Word, на основании которых вычисляется коэффициент сжатия по формуле XAsp/YAsp.

Пример использования процедуры GetAspectRatio представлен в листинге Ж.182.

Листинг Ж.182. Программа ProcGtAR.pas

```
program ProcGtAR;
uses Crt, Graph, IniGraph;
```

Окончание листинга Ж.182

```
var
  XAsp, YAsp: Word; ,
  XAspS,YAspS,S: String;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetAspectRatio(XAsp,YAsp);
  Str(XAsp,XAspS);       {Преобразование чисел XAsp, Yasp и }
  Str(YAsp,YAspS);       {XAsp/Yasp в строки XAspS ,YAspS,}
  Str(XAsp/YAsp:5:2,S); {S, соответственно}
  OutText('Коэффициент сжатия = ' + XAspS + '/' + YAspS + ' = ' + S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура GetDefaultPalette

Определяет палитру, заданную по умолчанию.

Синтаксис: `GetDefaultPalette(Palette)`

Параметр-переменная `Palette` — переменная типа `PaletteType`, который объявлен в модуле Graph следующим образом:

```
const
  MaxColors = 15;
type
  PaletteType = Record
      Size: Byte;
    Colors: array[0..MaxColors] of ShortInt;
  end;
```

Где поле: Size — количество размер палитры;
Colors — массив номеров цветов.

Пример использования процедуры GetDefaultPalette представлен в листинге Ж.183.

Листинг Ж.183. Программа `ProcGtDP.pas`

```
program ProcGtDP;
uses Crt, Graph, IniGraph;
var
  i: integer;
  MyPal, OldPal: PaletteType;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetDefaultPalette(OldPal); {Сохраняем старую палитру}
  MyPal:=OldPal;    {Копируем палитру}
  {Выводим текст}
  for i := 0 to MyPal.Size - 1 do
  begin
    SetColor(i); {Устанавливаем цвет рисования}
    {◄◄ см. о соответствии констант от 0 до 15 цвету в табл. Ж.1}
    OutTextXY(10, i*10, 'Нажмите любую клавишу');
  end;
  repeat    {Меняем палитру}
```

Окончание листинга Ж.183

```
    with MyPal do
      {Подставляем случайным образом цвета в палитре}
      Colors[Random(Size)] := Random(Size+1);
    SetAllPalette(MyPal); {Изменяем отображение цветов}
  until KeyPressed; {Пока не нажата клавиша}
  SetAllPalette(OldPal); {Восстанавливаем старую палитру}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура GetFillPattern

Извлекает текущий шаблон заполнения.

Синтаксис: GetFillPattern(FillPattern)

Параметр-переменная FillPattern — переменная типа FillPatternType, который объявлен в модуле Graph следующим образом:

```
type
  FillPatternType = array[1..8] of Byte;
```

Пример использования процедуры GetFillPattern представлен в листинге Ж.184.

Листинг Ж.184. Программа ProcGtFP.pas

```
program ProcGtFP;
uses Crt, Graph, IniGraph;
const
  Gray50: FillPatternType = ($AA, $55, $AA, $55, $AA, $55, $AA, $55);
var
  OldPattern: FillPatternType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetFillPattern(OldPattern);
  {Устанавливаем шаблон и цвет заполнения}
  SetFillPattern(Gray50, White); {Устанавливаем стиль заполнения
                прямоугольника: сплошной заливкой серого цвета(50%)}
  Bar(0,0,100,100); {Рисуем прямоугольник}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  {Восстанавливаем шаблон и цвет заполнения}
  SetFillPattern(OldPattern, White);
  {Рисуем полосу старым стилем (по умолчанию, состоящим из байт $FF)}
  Bar(0,0,100,100);
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура GetFillSettings

Извлекает текущий шаблон и цвет заполнения.

Синтаксис: GetFillSettings(FillInfo)

Параметр-переменная FillInfo — переменная типа FillSettingsType, который объявлен в модуле Graph следующим образом:

```
type
  FillSettingsType = Record
    Pattern: Word;
      Color: Word;
  end;
```

Где поле: Pattern — номер шаблона заполнения;
Color — номер цвета заполнения.

Пример использования процедуры GetFillSettings представлен в листинге Ж.185.

Листинг Ж.185. Программа ProcGtFS.pas

```
program ProcGtFS;
uses Crt, Graph, IniGraph;
var
  FillInfo: FillSettingsType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetFillSettings(FillInfo); {Сохраняем стиль и цвет заливки}
  Bar(0,0,100,100); {Рисуем прямоугольник}
  SetFillStyle(XHatchFill, GetMaxColor); {Устанавливаем стиль
                          заполнения прямоугольника: частой штриховкой
                          цветом с максимальным номером в палитре)}
  Bar(100,0,200,100);
  with Fillnfo do SetFillStyle(Pattern, Color);
  {Восстанавливаем старый стиль заполнения }
  Bar(200,0,300,100);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcGtFS представлен на рис. Ж.30.

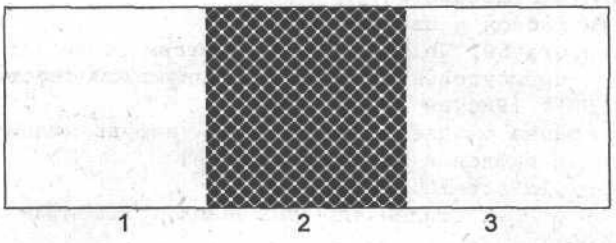


Рис. Ж.30. Пример сохранения стиля и цвета заливки (1) и их восстановления (3), после установки новых параметров заливки (2)

Процедура GetLineSettings

Извлекает текущие стиль, шаблон и толщину линии.

Синтаксис: GetLineSettings (LineInfo)

Параметр-переменная Linelnfo — переменная типа LineSettingsType, который объявлен в модуле Graph следующим образом:

```
type
  LineSettingsType = Record
```

```
    LineStyle: Word;
      Pattern: Word;
    Thickness: Word;
  end;
```

Где поле: LineStyle — номер стиля линии;
Pattern — номер шаблона линии, если в качестве LineStyle указано значение UserBitLn (4);
Thickness — толщина линии.

Пример использования процедуры GetLineSettings представлен в листинге Ж.186.

Листинг Ж.186. Программа ProcGtLS.pas

```
program ProcGtLS;
uses Crt, Graph, IniGraph;
var
  OldStyle: LineSettingsType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Line(0,0,300,0); {Рисуем линию с параметрами по умолчанию}
  GetLineSettings(OldStyle);
  SetLineStyle(DottedLn, 0, ThickWidth); {Устанавливаем стиль
                                рисования линии: жирная точечная линия}
  Line(0,10,300,10);
  {Восстанавливаем прежний стиль}
  with OldStyle do
    SetLineStyle(LineStyle,Pattern,Thickness);
  Line(0,20,300,20);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcGtLS представлен на рис. Ж.31.

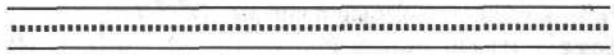


Рис. Ж.31. Пример сохранения стиля и толщины линии (1) и их восстановления (3), после установки новых параметров линии (2)

Процедура GetModeRange

Извлекает наименьший и наибольший номера графических режимов для указанного драйвера.

Синтаксис: GetModeRange(GraphDriver, LoMode, HiMode)

Параметр GraphDriver — значение типа Integer, определяющее номер графического драйвера, вместо которого можно использовать одну из констант модуля Graph: -128 — CurrentDriver (текущий драйвер); 1 — CGA; 2 — MCGA; 3 — EGA; 4 — EGA64; 5 — EGAMono; 6 — IBM8514; 7 — HercMono; 8 — ATT400; 9 — VGA; 10 — PC3270.

Параметры-переменные LoMode и HiMode — переменные типа Integer, в которых сохранются наименьший и наибольший номера графического режима для драйвера с номером GraphDriver.

Пример использования процедуры GetModeRange представлен в листинге Ж. 187.

Листинг Ж.187. Программа ProcGtMR.pas

```
program ProcGtMR;
uses Crt, Graph, IniGraph;
var
  Lowest, Highest: Integer;
  S: string;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetModeRange(CurrentDriver, Lowest, Highest);
  Str(Lowest,S); {Преобразуем число из Lowest в
                  строку и сохраняем ее в переменной s}
  OutTextXY(1,1,'Наименьший номер режима = ' + S) ;
  Str(Highest,S);
  OutTextXY(1,10,'Наибольший номер режима = ' + S) ;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура GetPalette

Извлекает текущую палитру и ее размер.

Синтаксис: GetPalette (Palette)

Параметр-переменная Palette — переменные типа PaletteType, который объявлен в модуле Graph следующим образом;

```
const
  MaxColors = 15;
type
  PaletteType = Record
      Size: Byte;
    Colors: array[0..MaxColors] of ShortInt;
  end;
```

Где поле: Size — количество размер палитры;
Colors — массив номеров цветов.

Пример использования процедуры GetPalette представлен в листинге Ж.188.

Листинг Ж.188. Программа ProcGtPa.pas

```
program ProcGtPa;
uses Crt, Graph, IniGraph;
var
  Color: Word;
  Palette: PaletteType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetPalette(Palette); {Извлекаем текущую палитру}
  if Palette.Size <> 1 then
  {Если в палитре больше одного цвета}
    for Color := 0 to Pred(Palette.Size) do
    begin {Перебираем все цвета палитры}
```

Окончание листинга Ж.188

```
      SetColor(Color); {Устанавливаем цвет рисования (44 см. о
                        соответствии констант от 0 до 15 цвету в табл. Ж.1)}
      Line(0, Color*5, 300, Color*5); {Рисуем линию}
    end
  else Line(0,0,300,0);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcGtPa представлен на рис. Ж.32.

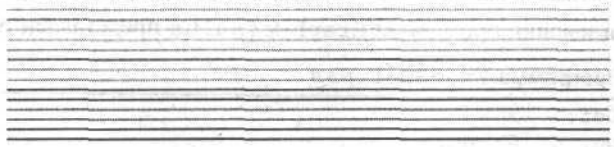


Рис. Ж.32. Число цветных линий соответствует числу цветов в палитре

1

Процедура GetViewSettings

Извлекает текущие параметры окна и отсечения.

Синтаксис: `GetViewSettings(ViewPort)`

Параметр-переменная ViewPort — переменная типа ViewPortType, который объявлен в модуле Graph следующим образом:

```
type
  ViewPortType = Record
    X1, Y1: Integer;
    X2, Y2: Integer;
      Clip: Boolean;
  end;
```

Поля X1, Y1, X2, Y2 — координаты верхнего левого и нижнего правого углов окна.

Поле Clip указывает, отсекается изображение, выходящее за пределы окна (значение True), или нет (значение False).

Пример использования процедуры GetViewSettings представлен в листинге Ж.189.

Листинг Ж.189. Программа ProcGtVS.pas

```
program ProcGtVS;
uses Crt, Graph, IniGraph;
var
  ViewPort: ViewPortType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetViewSettings(ViewPort);
  with ViewPort do
  begin
    Rectangle(0, 0, X2-X1, Y2-Y1);
    If Clip
    then OutText('Используется отсечение')
```

Окончание листинга Ж.189

```
    else OutText('Отсечение не используется');
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура GraphDefaults

Устанавливает текущую позицию на экране в току с координатами (0,0) и присваивает параметрам графического режима значения по умолчанию.

Синтаксис: GraphDefaults.

Пример использования процедуры GraphDefaults представлен в листинге Ж. 190.

Листинг Ж.190. Программа ProcGDef.pas

```
program ProcGDef;
uses Crt, Graph, IniGraph;
var
  Viewport: ViewPortType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  SetColor(1); {Устанавливаем цвет рисования (◄◄ см. о соответствии
                констант от 0 до 15 цвету в табл. Ж.1)}
  OutText('Рисуем синим цветом');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  ClearViewPort; {Очищаем экран}
  GraphDefaults; {Установки по умолчанию}
  OutText('Рисуем цветом по умолчанию');
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура InitGraph

Инициализирует графический режим.

Синтаксис: InitGraph(GraphDriver, GraphMode, PathToDriver)

Параметры-переменные GraphDriver и GraphMode — переменные типа Integer, в которых сохраняются номера графического драйвера и графического режима.

Параметр PathToDriver — значение типа String, определяющее путь к каталогу, где хранится файл указанного графического драйвера (*.bgi). Если в качестве этого параметра передается пустая строка, то требуемый файл *.bgi ищется в текущем каталоге.

Пример использования

Процедура InitGraph часто используется в программах этого приложения, например, в двух ключевых модулях приложения Ж: CoordSys, который представлен в листинге Ж.1; и в модуле IniGraph, который представлен в листинге Ж. 138.

Процедура RestoreCRTMode

Восстанавливает текстовый режим.

Синтаксис: RestoreCRTMode

Пример использования процедуры RestoreCRTMode представлен в листинге Ж.191.

Листинг Ж.191. Программа ProcRCRM.pas

```
program ProcRCRM;
uses Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  OutText('Графический режим');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  RestoreCrtMode; {Переходим в текстовый режим}
  WriteLn('Текстовый режим');
  ReadKey;
  SetGraphMode(GetGraphMode); {Переходим в графический режим}
  OutText('Опять графический режим');
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура SetActivePage

Устаналивает активную видеостраницу.

Синтаксис: SetActivePage (Page)

Параметр Page — значение типа Word, определяющее номер видеостраницы, которая становится активной (начиная с 0).

Пример использования процедуры SetActivePage представлен в листинге Ж. 192.

Листинг Ж.192. Программа ProcStAP.pas

```
program ProcStAP;
uses Crt, Graph, IniGraph;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  SetVisualPage(0); {Видимая страница- 1-я видеостраница}
  SetActivePage(1); {Активная страница - 2-я видеостраница}
  {Рисуем прямоугольник на активной видеостранице}
  Rectangle(10, 20, 30, 40);
  SetVisualPage(1); {Видимая страница - 2-я видеостраница}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура SetAllPalette

Изменяет все цвета в палитре на заданные.

Синтаксис: SetAIIPalette (Palette)

Параметр Palette — значение типа PaletteType, который объявлен в модуле Graph следующим образом:

```
const
  MaxColors = 15;
type
  PaletteType = Record
      Size: Byte;
```

```
    Colors: array[0..MaxColors] of ShortInt;
  end;
```

Где поле: Size — количество размер палитры;
Colors — массив номеров цветов.

Пример использования процедуры SetAllPalette представлен в листинге Ж.193.

Листинг Ж.193. Программа ProcSAlP.pas

```
program ProcSAlP;
uses Crt, Graph, IniGraph;
var
  i: integer;
  MyPal, OldPal: PaletteType;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  GetDefaultPalette(OldPal); {Сохраняем старую палитру}
  MyPal:=OldPal;    {Копируем палитру}
  {Выводим текст}
  for i := 0 to MyPal.Size - 1 do
  begin
    SetColor(i); {Устанавливаем цвет рисования}
    OutTextXY(10, i*10, 'нажмите любую клавишу');
  end;
  repeat    {Меняем палитру}
    with MyPal do
      {Подставляем случайным образом цвета в палитре}
      Colors[Random(Size)] := Random(Size+1);
    SetAllPalette(MyPal); {Изменяем отображение цветов}
  until KeyPressed;        {пока не нажата клавиша}
  SetAllPalette(OldPal);  {Восстанавливаем старую палитру}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура SetAspectRatio

Изменяет заданный по умолчанию коэффициент сжатия.

Синтаксис: SetAspectRatio (XAsp, YAsp)

Параметры XAsp и YAsp — значения типа Word, на основании которых вычисляется коэффициент сжатия по формуле XAsp/Yasp — отношение разрешения по горизонтали к разрешению по вертикали.

Пример использования процедуры SetAspectRatio представлен в листинге Ж. 194.

Листинг Ж.194. Программа ProcStAR.pas

```
program ProcStAR;
uses Crt, Graph, IniGraph;
var
  XAsp, YAsp: Word;
  S: String;
begin
  Graphlnit;  {Процедура из модуля IniGraph (листинг Ж.138)}
```

Окончание листинга Ж.194

```
  GetAspectRatio(XAsp, YAsp); {Получить разрешения экрана по
                               горизонтали и по вертикали}
  if XAsp = YAsp then YAsp := 5*XAsp;
  {Изменяем в цикле коэффициент сжатия}
  while Xasp < Yasp do {Повторять пока разрешение по горизонтали
                        меньше разрешения по вертикали}
  begin
    ClearDevice; {Очищаем экран}
    Str(XAsp,S); {Преобразуем число из XAsp в
                  строку и сохраняем ее в переменной s}
    OutTextXY(1,1,'XAsp = ' + S) ; {Выводим на экран
                                    разрешение по горизонтали}
    Str(YAsp,S);
    OutTextXY(1,15,'YAsp = ' + S);
    {Изменяем коэффициент сжатия}
    SetAspectRatio(XAsp, YAsp);
    {Рисуем окружность, которая при каждом шаге
     цикла while растягивается в эллипс по вертикали}
    Circle(GetMaxX div 2, GetMaxY div 2, 50);
    Inc(XAsp, 100); {Увеличиваем разрешение по горизонтали на 100}
    if ReadKey = #27 then Break; {Выход - по Esc}
  end;
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура SetBkColor

Устанавливает цвет фона, используя палитру.

Синтаксис: SetBkColor (ColorNum)

Параметр ColorNum — значение типа Word, определяющее номер цвета в палитре.

Пример использования процедуры SetBkColor представлен в листинге Ж.195.

Листинг Ж.195. Программа ProcStBC.pas

```
program ProcStBC;
uses Crt, Graph, IniGraph;
var
  i: Integer;
  S: String;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 1 to 15 do
  begin
    SetBkColor(i); {Устанавливаем (изменяем 15 раз) цвет фона}
    {Устанавливаем контрастный цвет рисования}
    if i=15 then SetColor(1) else SetColor (15-i); {◄◄ см. о
                  соответствии констант от 0 до 15 цвету в табл. Ж.1}
    Str(i,S);    {Преобразуем номер цвета в строку s}
    ClearDevice; {Очищаем экран}
    OutText(S);  {Выводим на экран номер цвета фона}
    if ReadKey = #27 then Break; {Прерываем цикл по Esc}
  end;
```

Окончание листинга Ж. 195

```
  CloseGraph; {Выходим из графического режима}
end.
```

Процедура SetColor

Устанавливает цвет рисования.

Синтаксис: `SetColor(ColorNum)`

Параметр ColorNum — значение типа Word, определяющее номер цвета. Вместо числового значения можно использовать соответствующие константы из модуля Crt, которые представлены в табл. Ж. 1 (◄◄ см. раздел, посвященный процедуре TextBackGround).

Пример использования

Процедура `SetColor` часто используется в программах этого приложения, например, в листинге Ж. 170 (◄◄ см. раздел, посвященный процедуре `MoveTo`).

Процедура SetFillPattern

Устанавливает шаблон заполнения.

Синтаксис: `SetFillPatter(Pattern, Color)`

Параметр Pattern — значение типа FillPatternType, который объявлен в модуле Graph следующим образом:

```
type
  FillPatternType = array[1..8] of Byte;
```

Параметр Color — значение типа Word, определяющее цвет заполнения. Вместо числового значения можно использовать соответствующие константы из модуля Crt, которые представлены в табл. Ж.1 (◄◄ см.. раздел, посвященный процедуре `TextBackGround`).

Пример использования

Процедура SetFillPattern часто используется в программах этого приложения, например, в листинге Ж.185 (◄◄ см. раздел, посвященный процедуре `GetFillSettings`).

Процедура SetFillStyle

Устанавливает стиль и цвет заполнения.

Синтаксис: `SetFillStyle(Pattern, Color)`

Параметр Pattern — значение типа Word, определяющее стиль заполнения. Вместо числового значения можно использовать соответствующие константы:

0 (`EmptyFill`) — заполнение фоновым цветом;
1 (SolidFill) — сплошное заполнение;
2 (`LineFill`) — заполнение линиями;
3 (LtSlashFill) — заполнение линиями "///";
4 (SlashFill) — заполнение жирными линиями "///";
5 (BkSlashFill) — заполнение жирными линиями "\\\";
6 (`LtBkSlashFill`) — заполнение линиями "\\\";
7 (HatchFill) — заполнение разреженной штриховкой;
8 (XHatchFill) — заполнение частой штриховкой;
9 (InterleaveFill) — заполнение прерывистыми линиями;

10 (WideDotFill) — заполнение линиями из разреженных точек;
11 (CloseDotFill) — заполнение линиями из частых точек;
12 (UserFill) — заполнение, определенное пользователем.

Параметр Color — значение типа Word, определяющее цвет заполнения. Вместо числового значения можно использовать соответствующие константы из модуля Crt, которые представлены в табл. Ж.1 (◄◄ см. раздел, посвященный процедуре TextBackGround).

Пример использования процедуры SetFillstyle представлен в листинге Ж.196.

Листинг Ж.196. Программа ProcStFS.pas

```
program ProcStFS;
uses Crt, Graph, IniGraph;
var
  i: byte;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  for i := 1 to 15 do
  begin
    SetFillStyle(Random(13),i); {Устанавливаем стиль и
                                 цвет заливки прямоугольника}
    Bar(0, (i-1)*(GetMaxY div 15), GetMaxX, i*(GetMaxY div 15));
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcStFS представлен на рис. Ж.33.

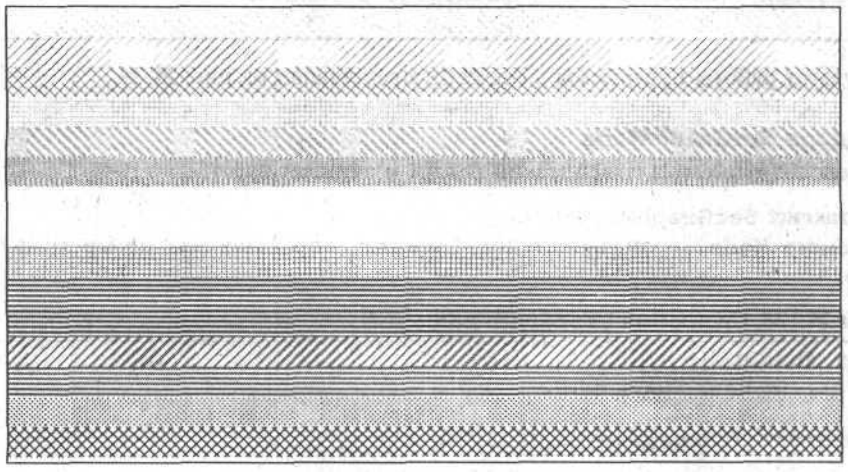


Рис. Ж.33. Прямоугольники, заполненные заливками разного стиля и цвета

Процедура SetGraphBufSize

Изменяет заданный по умолчанию размер графического буфера, используемого при заполнении областей экрана.

Синтаксис: SetGraphBufSize (BufSize)

Параметр BufSize — значение типа Word, определяющее размер буфера.

Пример использования процедуры SetBufSize представлен в листинге Ж.197.

Листинг Ж.197. Программа ProcStGS.pas

```
program ProcStGS;
uses Crt, Graph, IniGraph;
var
  I, R: Integer;
  BigPoly: array [0..1256] of PointType;
begin
  Randomize; {Активизируем генератор случайных чисел}
  {Распределяем в куче буфер на 8Кбайт, чтобы закрасить}
  SetGraphBufSize(8*1024); {многоугольник с 1256-ю вершинами}
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  R := GetMaxY div 2;
  {Определяем в массиве координаты точек для построения
   многоугольника в виде окружности с 1256-ю вершинами и радиусом R}
  for I := 0 to 1256 do
    with BigPoly[I] do
    begin
      X := GetMaxX div 2 + Round(R * Sin(I/200));
      Y := GetMaxY div 2 + Round(R * Cos (I/200));
    end;
  {Закрашиваем многоугольник}
  SetFillStyle(HatchFill,Yellow) ; {Устанавливаем стиль заполнения
                многоугольника: разряженной штриховкой желтого цвета}
  FillPoly(SizeOf(BigPoly) div SizeOf(PointType), BigPoly);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcStGS представлен на рис. Ж.34.

Процедура SetGraphMode

Переводит систему в графический режим и очищает экран.

Синтаксис: SetGraphMode (Mode)

Параметр Mode — значение типа Integer, определяющее номер графического режима.

Листинг Ж.198. Программа ProcStGM.pas

```
program FuncStGM;
uses Crt, Graph, IniGraph;
var
  OldGrMode: Integer;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
  OutText('Графический режим EGAHI');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  {Сохраняем параметры графического режима}
  OldGrMode := GetGraphMode;
  SetGraphMode(0);
```

Окончание листинга Ж.198

```
  OutText('Графический режим EGALO');
  ReadKey;
  SetGraphMode(OldGrMode);
  OutText('Возврат в режим EGAHI');
  ReadKey;
  CloseGraph; {Выходим из графического режима}
end.
```

Пример использования процедуры SetGraphMode представлен в листинге Ж.198.

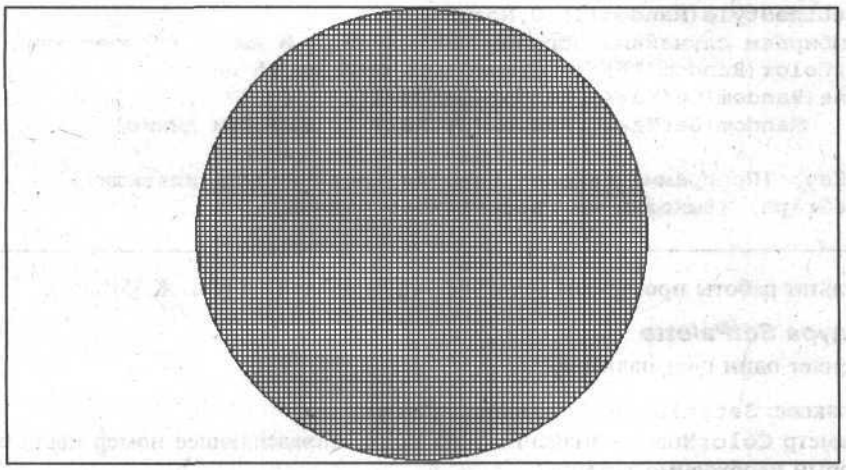


Рис. Ж.34. Закрашенный многоугольник с 1256-ю вершинами

Процедура SetLineStyle

Устанавливает толщину и стиль линии.

Синтаксис: SetLineStyle(LineStyle, Pattern, Thickness)

Параметр LineStyle — значение типа Word, определяющее стиль линии. Вместо числового значения, можно использовать одну из констант модуля Graph:

0 (SolidLn) — сплошная линия;
1 (DottedLn) — точечная линия;
2 (CenterLn) — штрих-пунктирная линия;
3 (DashedLn) — пуктирная линия;
4 (UserBitLn) — тип линии, определенный пользователем.

Параметр Pattern — значене типа Word, определяющее 16-разрядный битовый шаблон для рисования линии в случае, если параметр LineStyle имеет значение UserBitLn.

Параметр Thickness — значение типа Word, определяющее толщину линии в пикселях. Может принимать значения 1 (NormWidth), соответствующее обычной линии, или 3 (ThickWidth), соответствующее жирной линии.

Пример использования процедуры SetLineStyle представлен в листинге Ж.199.

Листинг Ж.199. Программа ProcStLS.pas

```
program ProcStLS;
uses Crt, Graph, IniGraph;
var
  i: Integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  for i := 1 to 20 do {Рисуем 20 разных линий}
  begin
    {Выбираем случайным образом стиль линии}
    SetLineStyle(Random(4),0,NormWidth);
    {Выбираем случайным образом цвет линии, 44 см. о соответствии}
    SetColor(Random(16));   {константа от 0 до 15 цвету в табл. Ж.1}
    Line(Random(GetMaxX),Random(GetMaxY),
         Random(GetMaxX),Random(GetMaxY)); {Рисуем линию}
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcStLS представлен на рис. Ж.35.

Процедура SetPalette

Заменяет один цвет палитры.

Синтаксис: SetPalette(ColorNum, Color)

Параметр ColorNum — значение типа Word, определяющее номер цвета в палитре, который необходимо заменить.

Параметр Color — значение типа ShortInt, определяющее цвет. Вместо числового значения можно использовать соответствующие константы из модуля Crt, которые представлены в табл. Ж.1 (◀◀ см. раздел, посвященный процедуре TextBackGround).

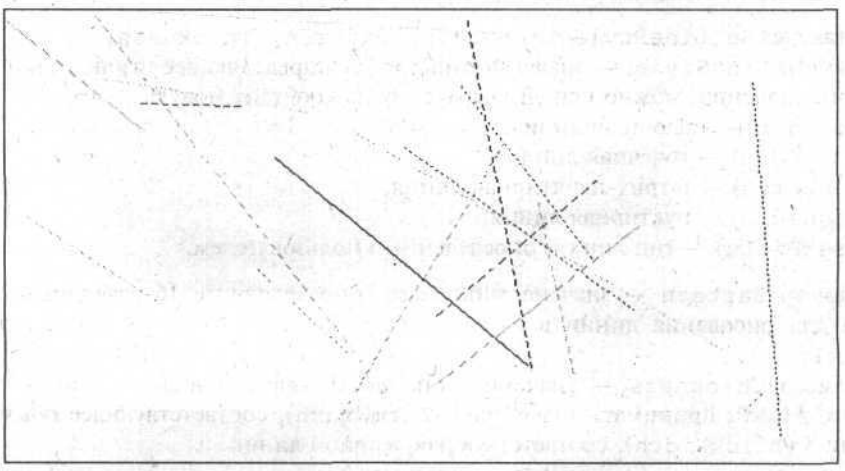


Рис. Ж.35. Создание 20 хаотически расположенных линий разного стиля и цвета

Пример использования процедуры SetPalette представлен в листинге Ж.200.

Листинг Ж.200. Программа ProcSetP.pas

```
program ProcSetP;
uses Crt, Graph, IniGraph;
var
  i: integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i := 0 to 15 do (Рисуем 15 разных параллелепипедов}
  begin
    {Устанавливаем стиль заполнения параллелепипеда: сплошной заливкой}
    SetFillStyle(SolidFill,i); {цвета i. ◄◄ см. о соответствии констант
                                от 0 до 15 цвету в табл. Ж.1.}
    Bar3D(i*20+10, 1, (i+1)*20, 200, 3, True); {Рисуем параллелепипед}
  end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  (Изменяем цвета в палитре}
  for i := 1 to 14 do SetPalette(i,15-i);
  ReadKey;
  CloseGraph; (Выходим из графического режима}
end.
```

Результат работы программы ProcSetP представлен на рис. Ж.36.

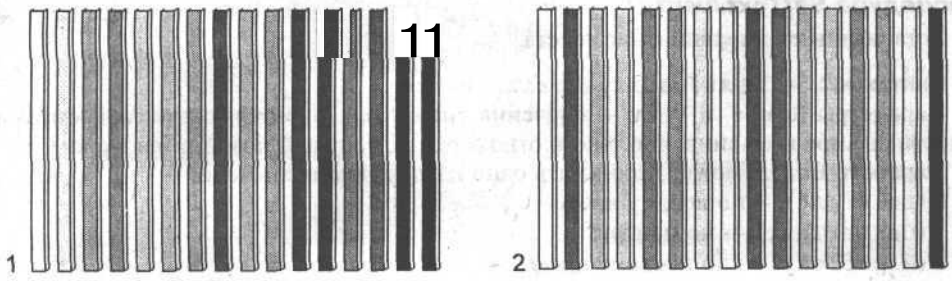


Рис. Ж.36. Цвет параллелепипедов из первой группы (1) изменяется в последовательности расположения цветов в палитре по умолчанию, а цвет параллелепипедов из второй группы (2) — в новой последовательности цветов палитре, переопределенной в программе

Процедура SetRGBPalette

Изменяет один цвет палитры для драйверов VGA и IBM8514.

Синтаксис:

SetRGBPalette(ColorNum, RedValue, GreenValue, BlueValue)

Параметр ColorNum — значение типа Integer, определяющее номер цвета в палитре, который необходимо изменить (для VGA — от 0 до 15; для IBM8514 — от О до 255).

Параметры RedValue, GreenValue и BlueValue — значения типа Integer (от О до 63), определяющие красную, зеленую и синюю составляющие цвета.

Пример использования процедуры SetRGBPalette представлен в листинге Ж.201.

Листинг Ж.201. Программа ProcSRGB.pas

```
program ProcSRGB;
uses Crt, Graph, IniGraph;
var
  x,y: integer;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  {Заполняем экран цветными точками}
  for x := 1 to GetMaxX do
    for y := 1 to GetMaxY do
      PutPixel(x,y,Random(16));
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  repeat {Изменяем цвета в палитре}
    SetRGBPalette(Random(16), Random(64),Random(64),Random(64));
  until KeyPressed; {Пока не будет нажата клавиша}
  CloseGraph; {Выходим из графического режима}
end.
```

Первая половина программы ProcSRGB создает на экране что-то подобное рис. Ж.26 (◀◀ см. раздел, посвященный процедуре PutPixel), а вторая ее половина изменяет (в цикле repeat...until) цвета в палитре, что приводит к изменению сочетания цветных точек на экране.

Процедура SetTextJustify

Устанавливает выравнивание текста.

Синтаксис: SetTextJustify(Horiz, Vert)

Параметры Horiz и Vert — значения типа Word, определяющие горизонтальное и вертикальное выравнивание текста относительно текущей позиции на экране.

Параметр Horiz может принимать одно из следующих значений:

0 (LeftText) — по левому краю;
1 (CenterText) — по центру;
2 (RightText) — по правому краю.

Параметр Vert может принимать одно из следующих значений:

0 (BottomText) — по нижней линии;
1 (CenterText) — по центру;
2 (TopText) — по верхней линии.

Пример использования процедуры SetTextJustify представлен в листинге Ж.202.

Листинг Ж.202. Программа ProcStTJ.pas

```
program ProcStTJ;
uses Crt, Graph, IniGraph;
var
  i,j: byte;
  a: array[1..3,1..3] of string;
begin
  a[1,1]:='Left&Bottom';
  a[1,2]:='Left&Center';
  a[1,3]:='Left&Top';
```

Окончание листинга Ж.202

```
  a[2,1]:='Center&Bottom';
  a[2,2]:='Center&Center';
  a[2,3]:='Center&Top';
  a[3,1]:='Right&Bottom';
  a[3,2]:='Right&Center';
  a[3,3]:='Right&Top';
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  for i:= 0 to 2 do
    for j := 0 to 2 do
    begin
      SetTextJustify(i,j);
      OutTextXY((j+1)*120,(i+1)*20,a[i+1,j+1]);
    end;
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcStTJ представлен на рис. Ж.37.

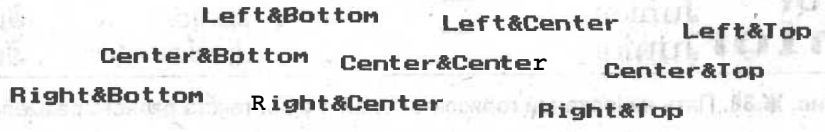


Рис. Ж.37. Выравнивание строки текста при различных сочетаниях параметров выравнивания по горизонтали&вертикали

Процедура SetTextStyle

Устанавливает шрифт, стиль и размер текста.

Синтаксис: SetTextStyle(Font, Direction, CharSize)

Параметр Font — значение типа Word, определяющее шрифт (начертание). Может принимать одно из следующих значений: 0 (DefaultFont) — растровый шрифт 8x8; 1 (TriplexFont), 2 (SmallFont), 3 (SansSerifFont), 4 (GothicFont) — векторные шрифты.

Параметр Direction — значение типа Word, **определяющее** направление шрифта. Может принимать одно из двух значений:

0 (HorizDir) — горизонтальное направление;

1 (VertDir) — вертикальное направление.

Параметр CharCase — значение типа Word, определяющее размер шрифта.

Пример использования процедуры SetTextStyle представлен в листинге Ж.203.

Листинг Ж.203. Программа ProcStTS .pas

```
program ProcStTS;
uses Crt, Graph, IniGraph;
var
  i,h,j: byte;
begin
  Graphlnit; {Процедура из модуля IniGraph (листинг Ж.138)}
```

Окончание листинга Ж.203

```
  for j:=0 to 1 do {Два направления текста}
  begin
    for i:= 0 to 4 do {Пять начертаний текста (шрифтов)}
      for h := 1 to 3 do {Три размера шрифта}
      begin
        SetTextStyle(i,j,h);
        {Выводим строку текста на экран в указанной позиции}
        OutTextXY(i*(140-j*100)+1,(h-j)*(20+50*j)+j,'Junior');
      end;
    ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  end;
  CloseGraph; {Выходим из графического режима}
end. •"
```

Результат работы программы ProcStTS представлен на рис. Ж.38 и Ж.39.

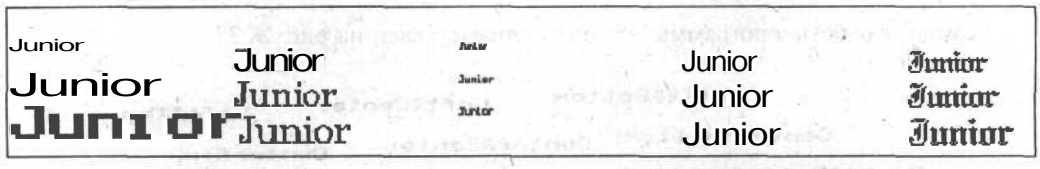


Рис. Ж.38. Пять начертаний горизонтальной строки текста разного размера

Как видно на рис. 39, не существует шрифта пр умолчанию, когда первый параметр процедуры SetTextStyle равен 0 (DefaultFont), вертикального направления, когда второй параметр этой процедуры равен 1 (VertDir) — пустое место слева.

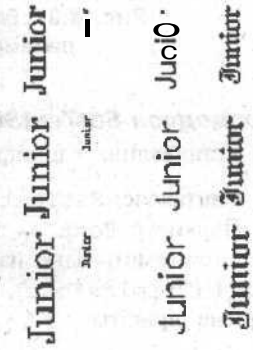


Рис. Ж.39. Четыре начертания вертикальной строки текста разного **размера**

Процедура SetUserCharSize

Изменяет ширину и высоту символов для векторных шрифтов.

Синтаксис:

`SetUserCharSize(MultX, DivX, MultY, DivY)`

Параметры MultX, DivX, MultY и DivY — значения типа Word, которые используются для вычисления коэффициента умножения по ширине (MultX/DivX) и высоте (MultY/DivY) активного шрифта.

Пример использования процедуры SetUserCharSize представлен в листинге Ж.204.

Листинг Ж.204. Программа `ProcSUCS.pas`

```
program ProcSUCS;
uses Crt, Graph, IniGraph;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  {Устанавливаем шрифт, стиль и размер текста}
  SetTextStyle(TriplexFont, HorizDir, 4);
  OutTextXY(1,1,'Junior'); {Выводим обычный текст}
```

Окончание листинга Ж.204

```
  SetUserCharSize(1,2,1,1);
  OutTextXY(130,1,'Junior'); {Выводим узкий текст}
  SetUserCharSize(2,1,1,1);
  OutTextXY(200,1,'Junior'); {Выводим широкий текст}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  CloseGraph; {Выходим из графического режима}
end.
```

Результат работы программы ProcSUCS представлен на рис. Ж.40.

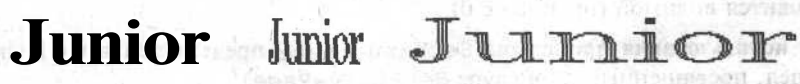


Рис. Ж.40. Пример надписи с разной шириной и высотой

Процедура SetViewPort

Устанавливает текущее окно.

Синтаксис: `SetViewPort(X1, Y1, X2, Y2, Clip)`

Параметры `X1`, Yl, X2 и Y2 — значения типа Integer, определяющие координаты левого верхнего (`X1`, Y1) и нижнего правого (X2, `Y2`) угла окна.

Параметр Clip — значение типа Boolean, определяющее, будет ли отсекаться изображение, выходящее за пределы окна (значение True), или нет (значение False).

Пример использования процедуры SetViewPort представлен в листинге Ж.205.

Листинг Ж.205. Программа `ProcStVP.pas`

```
program ProcStVP;
uses Crt, Graph, IniGraph;
const
  VPWidth = 400;   {Ширина окна}
  VPHeight = 150;  {Высота окна}
var
  i: byte;
  procedure DrawLines(Clip: Boolean);
  begin
    ClearDevice; {Очищаем экран}
    {Определяем окно}
    SetViewPort(100,100,100+VPWidth,100+VPHeight,Clip);
    {Рисование в окне}
    Rectangle(0,0,VPWidth,VPHeight);
    for i := 1 to 20 do
      Line(Random(GetMaxX),Random(GetMaxY),
           Random(GetMaxX),Random(GetMaxY));
    ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
  end;
begin
  GraphInit; {Процедура из модуля IniGraph (листинг Ж.138)}
  Randomize; {Активизируем генератор случайных чисел}
  DrawLines(True);  {Отсечение активизировано}
```

Окончание листинга Ж.203

```
  DrawLines(False); {Отсечение не используется}
  CloseGraph;       {Выходим из графического режима}
end.
```

Результат работы программы ProcStVP представлен на рис. Ж.41.

Процедура SetVisualPage

Устаналивает видимую видеостраницу.

Синтаксис: `SetVisualPage(Page)`

Параметр Page — значение типа Word, определяющее номер видеостраницы, которая становится видимой (начиная с 0).

Пример использования процедуры SetVisualPage представлен в листинге Ж.192 (◄◄ см. раздел, посвященный процедуре SetActivePage).

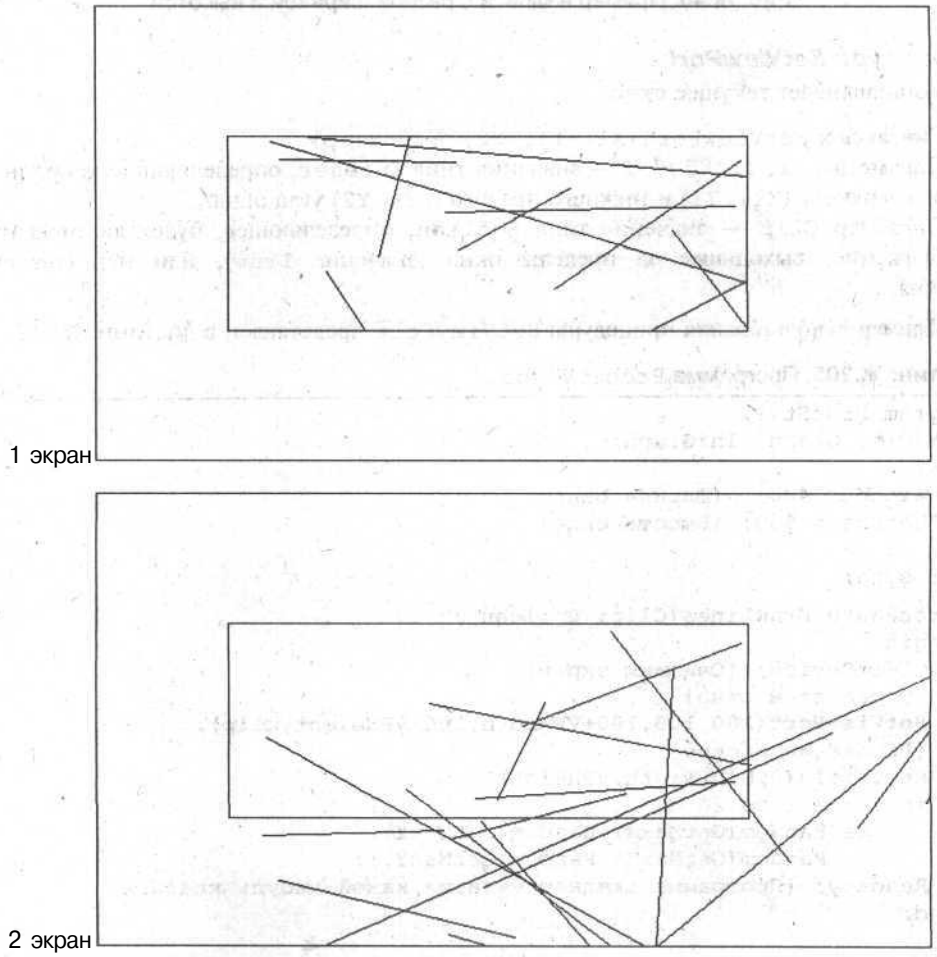


Рис. Ж.41. Пример, когда изображение в окне отсекается (1), и выводится на экран без отсекания по размеру окна (2)

Модуль Strings

Этот модуль содержит функции системы Turbo Pascal позволяющие работать со строками типа PChar — строки, оканчивающиеся символом #0.

Функции

Функция StrCat

Добавляет копию одной строки к концу другой и возвращает полученную строку.

Синтаксис: `StrCat(Dest, Source)`
Параметр Dest — значение типа PChar, определяющее строку, к которой присоединяется строка PChar, переданная в параметре Source. При этом строка Dest должна иметь длину, достаточную для добавления строки Source.

Тип возвращаемого результата: PChar.

Пример использования функции StrCat представлен в листинге Ж.206.

Листинг Ж.206. Программа `FuncSCat.pas`

```
program FuncSCat;
uses Crt, Strings;
const
  Header: PChar = 'Книга "Turbo Pascal 7.0 на примерах"';
  Author: PChar = 'Автор Ю.А. Шпак';
  Publisher: PChar = 'Издательство "Юниор"';
var
  S: array [0..75] of Char;
begin
  ClrScr;   {Очищаем экран}
  StrCopy(S,Header);    {Сохраняем строку Header в S}
  StrCat(S,'. ');       {Добавляем к S точку с пробелом}
  StrCat(S,Author);     {Добавляем к S строку Author}
  StrCat(S,'. ');       {Добавляем к S точку с пробелом}
  StrCat(S,Publisher); {Добавляем к S строку Publisher}
  WriteLn(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrComp

Сравнивает две строки.

Синтаксис: `StrComp(Str1, Str2)`
Параметры Str1 и Str2 — сравниваемые значения типа PChar.

Тип возвращаемого результата: Integer.
Результат меньше 0, если Str1 < Str2.
Результат равен 0, если Str1 = Str2.
Результат больше 0, если Str1 > Str2.

Пример использования функции StrComp представлен в листинге Ж.207.

Листинг Ж.207. Программа FuncSCmp.pas

```
program FuncSCmp;
uses Crt, Strings;
var
 c: integer;
 Result: PChar;
 S1,S2: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Первая строка: ');
  Readln(S1);
  Write('Вторая строка: ');
  Readln(S2);
  C := StrComp(S1,S2);
  if C < 0 then Result := ' меньше, чем ' else
  if C > 0 then Result := ' больше, чем ' else
                Result := ' равно ';
  Writeln(S1,Result,S2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrCopy

Копирует одну строку в другую.

Синтаксис: StrCopy(Dest, Source)

Параметр Dest — значение типа PChar, определяющее строку назначения.

Параметр Source — значение типа PChar, определяющее копируемую строку. При этом строка Dest должна иметь длину, достаточную для размещения строки Source.

Тип возвращаемого результата: PChar.

Пример использования функции StrCopy представлен в листинге Ж.206 (◀◀ см. раздел, посвященный функции StrCat).

Функция StrECopy

Копирует одну строку в другую и возвращает указатель на конец полученной строки.

Синтаксис: StrECopy(Dest, Source)

Параметр Dest — значение типа PChar, определяющее строку назначения.

Параметр Source — значение типа PChar, определяющее копируемую строку. При этом строка Dest должна иметь длину, достаточную для размещения строки Source.

Тип возвращаемого результата: PChar.

Пример использования функции StrECopy представлен в листинге Ж.208.

Листинг Ж.208. Программа FuncSECp.pas

```
program FuncSECp;
uses Crt, Strings;
Const
  Header: PChar = ' Книга "Turbo Pascal 7.0 на примерах" ';
```

Окончание листинга Ж.208

```
  Author: PChar = 'Автор Ю.А. Шпак';
  Publisher: PChar = 'Издательство "Юниор"';
var
  S: array [0..75] of Char;
begin
  ClrScr;  {Очищаем экран}
  StrECopy(StrECopy(StrECopy(StrECopy(StrECopy(S,Header),
           '. '),Author),'. '),Publisher);
  Writeln(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrEnd

Возвращает указатель на конец строки — на символ #0.

Синтаксис: StrEnd(Str)
Параметр Str — значение типа PChar.

Тип возвращаемого результата: PChar.

Пример использования функции StrEnd представлен в листинге Ж.209.

Листинг Ж.209. Программа FuncSEnd.pas

```
program FuncSEnd;
uses Crt, Strings;
var
  S: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  Write ('Длина строки = ', StrEnd(S) - S, ' символов');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrIComp

Сравнивает две строки без учета регистра символов.

Синтаксис: StrIComp(Str1, Str2)
Параметры Str1 и Str2 — сравниваемые значения типа PChar.

Тип возвращаемого результата: Integer.
Результат меньше 0, если Str1 < Str2.
Результат равен 0, если Str1 = Str2.
Результат больше 0, если Str1 > Str2.

Пример использования функции StrIComp представлен в листинге Ж.210.

Листинг Ж.210. Программа FuncSICm.pas

```
program FuncSICm;
uses Crt, Strings;
var
```

Окончание листинга Ж.210

```
  S1,S2: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S1);
  StrCopy(S2,S1); {Копируем S1 в S2}
  StrUpper(S2); {Преобразуем S2 в верхний регистр}
  if StrIComp(S1,S2) = 0
  then Writeln(S1,' = ',S2)
  else Writeln(S1,' <> ',S2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrLCat

Добавляет символы из одной строки в конец другой строки и возвращает полученную строку.

Синтаксис: StrLCat(Dest, Source)

Параметр Dest — значение типа PChar, определяющее строку, к которой присоединяется строка PChar, переданная в параметре Source. При этом строка Dest должна иметь длину, достаточную для добавления строки Source.

Тип возвращаемого результата: PChar.

Пример использования функции StrLCat представлен в листинге Ж.211.

Листинг Ж.211. Программа FuncSLCt.pas

```
program FuncSLCt;
uses Crt, Strings;
var
  S: array [0..70] of Char;
begin
  ClrScr;  {Очищаем экран}
  StrLCopy(S, 'Книга "Turbo Pascal 7.0 на примерах"', SizeOf(S) - 1);
  StrLCat(S, '. ', SizeOf(S) - 1);
  StrLCat(S, 'Автор Ю. А. Шпак', SizeOf(S) - 1);
  StrLCat(S, '. ', SizeOf(S) - 1);
  StrLCat(S, 'Издательство "Юниор"', SizeOf(S) - 1);
  WriteLn(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

В результате выполнения программы FuncSLCt будет получена обрезанная строка: "Книга "Turbo Pascal 7.0 на примерах". Автор Ю. А. Шпак. Издательство "Ю". Это произойдет из-за того, что переменная s объявлена строкой в 70 символов, а для отображения всего предложения требуется 75 символов.

Функция StrLComp

Сравнивает две строки до определенной позиции.

Синтаксис: StrLComp(Str1, Str2, MaxLen)

Параметры Str1 и Str2 — сравниваемые значения типа PChar.

Параметр MaxLen — значение типа Word, определяющее позицию, до которой сравниваются строки.

Тип возвращаемого результата: Integer.
Результат меньше 0, если Str1 < Str2.
Результат равен 0, если Str1 = Str2.
Результат больше 0, если Str1 > Str2.

Пример использования функции StrLComp представлен в листинге Ж.212.

Листинг Ж.212. Программа FuncSLCm.pas

```
program FuncSLCm;
uses Crt, Strings;
var
  Result: PChar;
  S1,S2: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Первая строка: ');
  Readln(S1);
  Write('Вторая строка: ');
  Readln(S2);
  if StrLComp(S1,S2,5) = 0 then Result:='одинаковые'
                           else Result:='разные';
  Writeln('Первые пять символов ', Result, '.');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrLCopy

Копирует указанное количество символов одной строки в другую строку.

Синтаксис: StrLCopy(Dest, Source, MaxLen)

Параметр Dest — значение типа PChar, определяющее строку назначения.

Параметр Source — значение типа PChar, определяющее копируемую строку. При этом строка Dest должна иметь длину, достаточную для размещения строки Source.

Параметр MaxLen — значение типа Word, определяющее количество копируемых символов.

Тип возвращаемого результата: PChar.

Пример использования функции StrLCopy представлен в листинге Ж.213.

Листинг Ж.213. Программа FuncSLCp.pas

```
program FuncSLCp;
uses Crt, Strings;
var
  S: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  StrLCopy(S, 'Книга "Turbo Pascal 7.0 на примерах"', 23);
  Writeln(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrLen

Возвращает количество символов в строке.

Синтаксис: StrLen(Str)
Параметр Str — значение типа PChar.

Тип возвращаемого результата: Word.

Пример использования функции StrLen представлен в листинге Ж.214.

Листинг Ж.214. Программа FuncSLen.pas

```
program FuncSLen;
uses Crt, Strings;
var
  S: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  Writeln('Длина строки - ',StrLen(S),' символов');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrLIComp

Сравнивает две строки до определенной позиции без учета регистра символов.

Синтаксис: StrLIComp(Str1, Str2, MaxLen)
Параметры Strl и Str2 — сравниваемые значения типа PChar.
Параметр MaxLen — значение типа Word, определяющее позицию, До которой сравниваются строки.

Тип возвращаемого результата: Integer.
Результат меньше 0, если Strl < Str2.
Результат равен 0, если Strl = Str2.
Результат больше 0, если Strl > Str2.

Пример использования функции StrLIComp представлен в листинге Ж.215.

Листинг Ж.215. Программа FuncSLIC.pas

```
program FuncSLIC;
uses Crt, Strings;
var
  Result: PChar;
  S1,S2: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Первая строка: ');
  Readln(S1);
  Write('Вторая строка: ');
  Readln(S2);
  StrUpper(S2); {Преобразуем S2 в верхний регистр}
  if StrLIComp(S1,S2,5) = 0 then Result:='одинаковые'
                            else Result:='разные';
  Writeln('Первые пять символов ', Result, '.');
```

Окончание листинга Ж.215

```
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrLower

Преобразовывает символы строки в нижний регистр.

Синтаксис: StrLower(Str)

Параметр Str — значение типа PChar.

Тип возвращаемого результата: PChar.

Пример использования функции StrLower представлен в листинге Ж.216.

Листинг Ж.216. Программа FuncSLow.pas

```
program FuncSLow;
uses Crt, Strings;
var
  S: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  StrLower(S);
  Writeln('Строка в нижнем регистре: ');
  Writeln(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrMove

Копирует из одной строки в другую указанное количество символов.

Синтаксис: StrMove(Dest, Source, Count)

Параметр Dest — значение типа PChar, определяющее строку назначения.

Параметр Source — значение типа PChar, определяющее копируемую строку.

Параметр Count — значение типа Word, определяющее количество копируемых символов.

Тип возвращаемого результата: PChar.

Пример использования функции StrMove представлен в листинге Ж.217.

Листинг Ж.217. Программа FuncSMov.pas

```
program FuncSMov;
uses Crt, Strings;
var
  S1,S2: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S1);
  StrMove(S2,S1,StrLen(S1));
  Writeln(S2);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrNew

Выделяет для строки память в куче.

◄◄ см. о куче в примечании из раздела о функции MaxAvail.

Синтаксис: StrNew(Str)
Параметр Str — значение типа PChar.

Тип возвращаемого результата: PChar.

Пример использования функции StrNew представлен в листинге Ж.218.

Листинг Ж.218. Программа FuncSNew.pas

```
program FuncSNew;
uses Crt, Strings;
var
  P: PChar;
  S: array [0..79] of Char;
begin
  ClrScr;   {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  P := StrNew(S);
  Writeln(P);
  StrDispose(P); {Освобождает память, выделенную под строку}
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrPas

Преобразовывает строку типа PChar в строку типа string.

Синтаксис: StrPas (Str)
Параметр Str — значение типа PChar.

Тип возвращаемого результата: String.

Пример использования функции StrPas представлен в листинге Ж.219.

Листинг Ж.219. Программа FuncSPas.pas

```
program FuncSPas;
uses Crt, Strings;
var
  A: array [0..79] of Char;
  S: String[79];
begin
  ClrScr;   {Очищаем экран}
  Write('Введите строку: ');
  Readln(A);
  S := StrPas(A);
  Writeln(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrPCopy

Преобразовывает строку типа String в строку типа PChar.

Синтаксис: StrPCopy(Dest, Source)
Параметр Dest — значение типа PChar, определяющее строку назначения.
Параметр Source — значение типа String, определяющее копируемую строку.

Тип возвращаемого результата: PChar.

Пример использования функции StrPCopy представлен в листинге Ж.220.

Листинг Ж.220. Программа FuncSPCp.pas

```
program FuncSPCp;
uses Crt, Strings;
var
  A: array [0..79] of Char;
  S: String[79];
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  StrPCopy(A,S);
  Writeln(A);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrPos

Возвращает указатель на первое вхождение строки в другую строку.

Синтаксис: StrPos(Str1, Str2)
Параметры Str1 и Str2 — значения типа PChar.

Тип возвращаемого результата: PChar.

Пример использования функции StrPos представлен в листинге Ж.221.

Листинг Ж.221. Программа FuncSPos.pas

```
program FuncSPos;
uses Crt, Strings;
var
  P: PChar;
  S, SubStr: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  Write('Введите подстроку: ');
  Readln(SubStr);
  P := StrPos(S, SubStr);
  if P = nil
  then WriteLn('Подстрока не найдена.')
  else WriteLn('Подстрока найдена в позиции ', P - S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrRScan

Возвращает указатель на последнее вхождение символа в строку.

Синтаксис: StrRScan(Str, Chr)
Параметр Str — значение типа PChar.
Параметр Chr — значение типа Char, определяющее искомый символ

Тип возвращаемого результата: PChar.

Пример использования функции StrRScan представлен в листинге Ж.222.

Листинг Ж.222. Программа FuncSRSc.pas

```
program FuncSRSc;
uses Crt, Strings;
var
  P: PChar;
  S: array [0..79] of Char;
begin <
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  P := StrRScan(S,' ');
  if P = nil
  then WriteLn('Строка состоит из одного слова')
  else WriteLn('Последнее слово в строке - "', P+1, '"');
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrScan

Возвращает указатель на первое вхождение символа в строку.

Синтаксис: StrScan (Str, Chr)
Параметр Str — значение типа PChar.
Параметр Chr — значение типа Char, определяющее искомый символ.

Тип возвращаемого результата: PChar.

Пример использования функции StrScan представлен в листинге Ж.223.

Листинг Ж.223. Программа FuncSScn.pas

```
program FuncSScn;
uses Crt, Strings;
var
  P: PChar;
  S,W: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  P := StrScan(S,' ');
  if P = nil
  then WriteLn('Строка состоит из одного слова')
  else WriteLn('Первое слово в строке - "', StrLCopy(W,S,P-S), '"');
```

Окончание листинга Ж.223

```
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Функция StrUpper

Преобразовывает символы строки в верхний регистр.

Синтаксис: StrUpper (Str)
Параметр `Str` — значение типа PChar.

Тип возвращаемого результата: PChar.

Пример использования функции StrUpper представлен в листинге Ж.224.

Листинг Ж.224. Программа `FuncSUpp.pas`

```
program FuncSUpp;
uses Crt, Strings;
var
  S: array [0..79] of Char;
begin
  ClrScr;  {Очищаем экран}
  Write('Введите строку: ');
  Readln(S);
  StrUpper(S);
  Writeln('Строка в верхнем регистре: ');
  Writeln(S);
  ReadKey; {Программа ожидает нажатия какой-нибудь клавиши}
end.
```

Процедуры

Процедура StrDispose

Освобождает память, выделенную под строку.

Синтаксис: StrDispose (Str)
Параметр Str — значение типа PChar.

Пример использования функции StrDispose представлен в листинге Ж.218 (◀◀ см. раздел, посвященный функции StrNew).

Приложение 3

Сообщения об ошибках

В этом приложении представлены сообщения об ошибках, выдаваемые на этапе компиляции, а также ошибки времени выполнения программ.

Ошибки при компиляции

Номер ошибки	Сообщение	Значение
1	Out of memory	Выход за границы памяти
2	Identifier expected	Не указан идентификатор. В этом месте должен находиться идентификатор
3	Unknown identifier	Неизвестный идентификатор. Идентификатор не был описан
4	Duplicate identifier	Повторный идентификатор. Идентификатор уже представляет имя программы, модуля, константы, переменной, типа, процедуры или функции, описанных в текущем блоке
5	Syntax error	Синтаксическая ошибка. В исходном тексте найден неверный символ
6	Error in real constant	Ошибка в вещественной константе
7	Error in integer constant	Ошибка в целой константе
8	String constant exceeds line	Строковая константа превышает размеры строки
9	Too many nested files	Слишком много вложенных файлов
10	Unexpected end of file	Некорректный конец файла
11	Line too long	Строка слишком длинная
12	Type identifier expected	Требуется идентификатор типа
13	Too many open files	Слишком много открытых файлов
14	Invalid file name	Неверное имя файла. Имя файла неверно или указывает на несуществующий путь
15	File not found	Файл не найден
16	Disk full	Диск заполнен
17	Invalid compiler directive	Неверная директива компилятора
18	Too many files	Слишком много файлов. В компиляции программы или программного модуля задействовано слишком много файлов

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
19	Undefined type in pointer definition	Неопределенный тип в определении указателя
20	Variable identifier expected	Требуется идентификатор переменной
21	Error in type	Ошибка в определении типа. Определение типа не может начинаться с этого символа
22	Structure too large	Слишком большая структура. Максимально допустимый размер структурного типа — 65535 байт
23	Set base type out of range	Базовый тип множества нарушает границы. Базовый тип множества должен представлять собой отрезок типа с границами в пределах от 0 до 255 или перечисляемый тип с не более чем 256 значениями
24	File components may not be files or objects	Компоненты файла не могут быть файлами или объектами
25	Invalid string length	Неверная длина строки. Максимальная описываемая длина строки должна находиться в диапазоне от 1 до 255
26	Type mismatch	Несоответствие типов
27	Invalid subrange base type	Неправильный базовый тип диапазона
28	Lower bound greater than upper bound	Нижняя граница больше верхней границы
29	Ordinal type expected	Требуется порядковый тип. Действительные, строковые, структурные и указательные типы в данном случае не допускаются
30	Integer constant expected	Требуется целая константа
31	Constant expected	Требуется константа
32	Integer or real constant expected	Требуется целая или вещественная константа
33	Pointer type identifier expected	Требуется идентификатор типа указателя
34	Invalid function result type	Неверный тип результата функции. Правильными типами результата функции являются все простые типы, строковые типы и ссылочные типы
35	Label identifier expected	Требуется идентификатор метки
36	BEGIN expected	Требуется слово begin
37	END expected	Требуется слово end
38	Integer expression expected	Требуется выражение типа Integer
39	Ordinal expression expected	Требуется выражение порядкового типа. Выражение должно иметь порядковый тип

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
40	Boolean expression expected	Требуется выражение типа Boolean
41	Operand types do not match operator	Типы операндов не соответствуют оператору
42	Error in expression	Ошибка в выражении
43	Illegal assignment	Неверное присваивание
44	Field identifier expected	Требуется идентификатор поля. Данный идентификатор не определяет поле предшествующей переменной типа запись
45	Object file too large	Объектный файл слишком большой. Turbo Pascal не может компоновать файлы .obj размером больше 64 КБайт
46	Undefined external	Не определена внешняя процедура. Внешняя процедура или функция не содержит соответствующего определения public в объектном файле
47	Invalid object file record	Неправильная запись объектного файла. Файл .obj содержит неверную объектную запись
48	Code segment too large	Сегмент кода слишком большой. Максимальный размер кода программы модуля равняется 65520 байтам
49	Data segment too large	Сегмент данных слишком большой. Максимальный размер сегмента данных программы равен 65520 байтам, включая данные, описываемые используемыми программными модулями
50	DO expected	Требуется слово do
51	Invalid PUBLIC definition	Неверное определение public
52	Invalid EXTRN definition	Неправильное определение extrn
53	Too many EXTRN definition	Слишком много определений extrn. Turbo Pascal не может обрабатывать файлы .obj при более чем 256 определениях extrn
54	OF expected	Требуется слово of
55	INTERFACE expected	Требуется интерфейсный раздел
56	Invalid relocatable reference	Недопустимая переместимая ссылка
57	THEN expected	Требуется слово then
58	TO or DOWNTO expected	Требуется слово to или downto
59	Undefined forward	Не определено опережающее описание
61	Invalid typecast	Неверное приведение типа
62	Division by zero	Деление на ноль
63	Invalid file type	Неверный файловый тип
64	Cannot Read or Write variables of this type	Нельзя считать или записать переменные данного типа

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
65	Pointer variable expected	Требуется использовать переменную-указатель
66	String variable expected	Требуется строковая переменная
67	String expression expected	Требуется выражение строкового типа
68	Circular unit reference	Циклическая ссылка на модуль. В разделе interface два модуля не могут ссылаться друг на друга
69	Unit name mismatch	Несоответствие имен программных модулей. Имя программного модуля, найденное в файле .tpu, не соответствует имени, указанному в операторе uses
70	Unit version mismatch	Несоответствие версий программных модулей. Один или несколько программных модулей используемых данной программой, были изменены после их компиляции
71	Internal stack overflow	Переполнение внутреннего стека. Внутренний стек компилятора исчерпан из-за слишком большого уровня вложенности операторов
72	Unit file format error	Ошибка формата файла программного модуля. Файл .tpu некорректен
73	Implementation expected	Требуется раздел реализации. Отсутствует ключевое слово implementation
74	Constant and case types do not match	Типы констант и тип выражения оператора case не соответствуют друг другу
75	Record variable expected	Требуется переменная типа запись
76	Constant out of range	Константа выходит за границы допустимых значений
77	File variable expected	Требуется файловая переменная
78	Pointer expression expected	Требуется выражение типа указатель
79	Integer or real expression expected	Требуется выражение типа real или integer
80	Label not within current block	Метка не находится внутри текущего блока. Оператор goto не может ссылаться на метку, находящуюся вне текущего блока
81	Label already defined	Метка уже определена
82	Undefined label in processing statement part	Неопределенная метка в обрабатываемом разделе операторов
83	Invalid @ argument	Недействительный аргумент оператора @. Действительными аргументами являются ссылки на переменные и идентификаторы процедур или функций
84	Unit expected	Требуется ключевое слово unit

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
85	";" expected	Требуется ";"
86	":" expected	Требуется ":"
87	"," expected	Требуется ","
88	"(" expected	Требуется "("
89	")" expected	Требуется ")"
90	"=" expected	Требуется "="
91	":=" expected	Требуется ":="
92	"[" or "(" expected	Требуется "[" или "("
93	"]" or ")" expected	Требуется "]" или ")"
94	"." expected	Требуется "."
95	".." expected	Требуется ".."
96	Too many variables	Слишком много переменных
97	Invalid FOR control variable	Недопустимая управляющая переменная оператора for. Управляющая переменная оператора for должна быть перечисляемого типа, определенной в разделе описаний текущей подпрограммы
98	Integer variable expected	Требуется переменная целого типа
99	Files are not allowed here	Здесь не допускаются файлы
100	String length mismatch	Несоответствие длины. Длина строковой константы не соответствует количеству элементов символьного массива
101	Invalid ordering of fields	Неверный порядок полей. Поля в константе типа запись должны записываться в порядке их описания
102	String constant expected	Требуется константа строкового типа
103	Integer or real variable expected	Требуется переменная типа integer или real
104	Ordinal variable expected	Требуется переменная порядкового типа
105	INLINE error	Ошибка в операторе inline
106	Character expression expected	Предшествующее выражение должно иметь символьный тип
107	Too many relocation items	Слишком много переместимых элементов. Размер раздела таблицы перемещения файла .EXE превышает 64 КБайта
108	Overflow in arithmetic operation	Переполнение в арифметической операции. Результат операции не находится в диапазоне Longint (-2147483648...2147483647)
109	No enclosing FOR, WHILE or REPEAT statement	Нет включающего оператора for, while или repeat. Стандартные процедуры Break и Continue не могут использоваться вне операторов for, while или repeat

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
112	CASE constant out of range	Константа case вне диапазона. Для целочисленных операторов case константы должны находиться в диапазоне от -32768 до 32767
113	Error in statement	Ошибка в операторе. Данный идентификатор не может начинать оператор
114	Cannot call an interrupt procedure	Нет возможности вызвать процедуру прерывания. Непосредственно вызвать процедуру прерывания невозможно
116	Must be in 8087 mode tp compile this	Для компиляции необходим режим 8087. Данная конструкция может компилироваться только в режиме {$N+}
117	Target address not found	Адрес назначения не найден
118	Include files are not allowed here	В такой ситуации включаемые файлы не допускаются. Каждый блок операторов должен целиком размещаться в одном файле
119	No inherited methods are accessible here	Наследуемые методы здесь недоступны. Используется ключевое слово inherited вне метода или в методе, или объектном типе, не имеющем предка
121	Invalid qualifier	Неверный квалификатор
122	Invalid variable reference	Недопустимая ссылка на переменную. Предыдущая конструкция удовлетворяет синтаксису ссылки на переменную, но она не указывает на адрес памяти
123	Too many symbols	Слишком много идентификаторов. Программа или программный модуль описывает более 64 КБайт идентификаторов
124	Statement part too large	Слишком большой раздел операторов. Turbo Pascal ограничивает размер раздела операторов до величины примерно 24 КБайта
126	Files must be var parameters	Параметры файлового типа должны быть параметрами-переменными
127	Too many conditional symbols	Слишком много символов условий
128	Misplaced conditional directive	Пропущена условная директива. Компилятор обнаружил директиву {$ELSE} ИЛИ {$ENDIF} без соответствующих директив {$IFDEF}, {$IFNDEF} ИЛИ {$IFOPT}
129	ENDIF directive missing	Пропущена директива {$ENDIF}. В исходном файле должно быть равное количество директив {$IFxxx} и {$ENDIF}
130	Error in initial conditional defines	Ошибка в начальных условных определениях

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
131	Header does not match previous definition	Заголовок не соответствует предыдущему определению. Заголовок процедуры или функции, указанный в интерфейсной секции или описании `forward`, не соответствует самому заголовку процедуры или функции
132	Cannot evaluate this expression	Невозможно вычислить данное выражение. В выражении-константе или в отладочном выражении имеет место попытка использования неподдерживаемых средств
134	Expression incorrectly terminated	Некорректное завершение выражения (только для встроенного отладчика). Turbo Pascal ожидает в данном месте конец выражения или операцию, но не находит ни того, ни другого
135	Invalid format specifier	Неверный спецификатор формата (только для встроенного отладчика). Используется неверный спецификатор формата или числовой аргумент спецификатора формата выходит за допустимые границы
136	Invalid indirect reference	Недопустимая косвенная ссылка. Оператор пытается осуществить недопустимую косвенную ссылку
137	Structured variable are not allowed here	Не допускается использование структурной переменной
138	Cannot evaluate without System unit	Нельзя вычислить без блока System (только для встроенного отладчика). Чтобы отладчик смог вычислить выражение, файле `.tpl` должен содержаться модуль System
139	Cannot access this symbol	Доступ к данному идентификатору отсутствует (только для встроенного отладчика). Как только программа скомпилирована, все множество ее идентификаторов становится доступным. Однако к отдельным идентификаторам (например, к переменным) нельзя получить доступ, пока программа не запущена
140	Invalid floating-point operation	Недопустимая операция с плавающей точкой. При операции с двумя действительными значениями было получено переполнение или деление на ноль
141	Cannot compile overlay to memory	Нельзя выполнить компиляцию оверлеев в память. Программа, использующая оверлеи, должна компилироваться на диск
142	Procedure or function variable expected	Должна использоваться процедурная или функциональная переменная. Стандартная процедура Assigned требует аргумент типа переменной-указателя или процедурного типа
143	Invalid procedure or function reference	Недопустимая ссылка на процедуру или функцию

Ошибки при компиляции. Продолжение

Номер ошибки	Сообщение	Значение
144	Cannot overlay this unit	Этот модуль не может использоваться в качестве оверлейного. Попытка использовать в качестве оверлейного модуль, который не был скомпилирован с директивой {$O+}
145	Too many nested scopes	Слишком большая вложенность. На уровень вложенности влияют: каждый модуль в разделе uses, каждая запись, имеющая вложенность, а также вложенность операторов with
146	File access denied	Файл недоступен. Файл не может быть открыт или создан
147	Object type expected	Требуется объектный тип. Идентификатор не определяет объектный тип
148	Local object type are not allows	Описание локальных объектных типов не допускается
149	VIRTUAL expected	Требуется ключевое слово virtual
150	Method identifier expected	Требуется идентификатор метода
151	Virtual constructor are not allowed	Виртуальный конструктор не допускается. Метод конструктора должен быть статическим
152	Constructor identifier expected	Требуется идентификатор конструктора
153	Destructor identifier expected	Требуется идентификатор деструктора
154	Fail only allowed within constructors	Стандартная процедура Fail может использоваться только внутри конструктора
155	Invalid combination of opcode and operands	Недопустимая комбинация кода операции и операндов. Код операции ассемблера не воспринимает данное сочетание операндов
156	Memory reference expected	Требуется ссылка на память. Операнд ассемблера не является ссылкой на память
157	Cannot add or substract relocatable symbols	Нельзя складывать или вычитать переместимые идентификаторы. Единственная операция, которую допускается выполнять с перемещаемыми идентификаторами в операнде ассемблера — это сложение с константой или вычитание константы
158	Invalid register combination	Недопустимое сочетание регистров. Допустимыми сочетаниями индексных регистров являются [BX], [BP], [SI], [DI], [BX+SI], [BX+DI], [BP+SI] и [BP+DI]
159	286/287 Instructions not allowed	Инструкции процессоров 286/287 не разрешены. Для разрешения кодов операций указанных процессоров нужно использовать директиву компилятора {$G+}, однако, результирующий код не сможет работать на машинах с процессорами 8086 и 8088
160	Invalid symbol reference	Недопустимая ссылка на идентификатор. Данный идентификатор в операнде ассемблера недоступен

Ошибки при компиляции. Окончание

Номер ошибки	Сообщение	Значение
161	Code generation error	Ошибка генерации кода. Предшествующая часть оператора содержит инструкции LOOPNE, LOOPE, LOOP или JCXZ, которые не могут достичь целевой метки
162	ASM expected	Требуется ключевое слово asm
163	Duplicate dynamic method index	Дублируется индекс динамического метода. Этот индекс динамического метода уже используется другим методом
164	Duplicate resource identifier	Дублирование идентификатора ресурса (только в Windows или защищенном режиме). Данный файл ресурса содержит ресурс с именем или идентификатором, который уже используются для другого ресурса
165	Duplicate or invalid export index Начало	Дублирующийся или недопустимый индекс экспорта (только в Windows или защищенном режиме). Порядковый номер, заданный в операторе Index, не находится в диапазоне 1..32767, или уже используется другой экспортируемой подпрограммой
166	Procedure or function identifier expected	Требуется идентификатор процедуры или функции (только в Windows или защищенном режиме). Оператор export допускает экспорт только процедур и функций
167	Cannot export this symbol	Этот идентификатор экспортировать нельзя (только в Windows или защищенном режиме). Процедура или функция не может экспортироваться, если она не описана в операторе процедуры export
168	Duplicate export name	Дублирование экспортируемого имени (только в Windows или защищенном режиме). Имя, заданное в операторе name, уже используется для другой экспортируемой подпрограммы
169	Executable file header too large	Слишком велик заголовок выполняемого файла (только в Windows или защищенном режиме). Генерируемый заголовок файла . exe превышает 64 КБайта (верхний предел для компоновщика)

Ошибки времени выполнения

Сообщение об ошибке времени выполнения программы имеют следующий формат:

```
Runtime error NNN at XXXX:YYYY
```

где NNN — номер ошибки, XXXX и YYYY — сегмент и смещение адреса, по которому возникла ошибка.

Ошибки системы MS-DOS

Номер ошибки	Сообщение	Значение
1	Invalid function number	Недопустимый номер функции. Обращение к несуществующей функции DOS
2	File not found	Файл не найден
3	Path not found	Путь к файлу не найден
4	Too many open files	Слишком много открытых файлов
5	File access denied	Нет доступа к файлу
6	Invalid file handle	Недопустимый дескриптор файла
12	Invalid file access code	Некорректный код доступа к файлу
15	Invalid driver number	Некорректный номер дискового устройства
16	Cannot remove current directory	Невозможно удалить текущий каталог
17	Cannot rename across drives	Недопустимо указывать различные дисковые устройства при переименовании файла
18	No more files	Нет файлов

Ошибки ввода-вывода

При выключенной директиве `{$I-}` номер ошибки ввода-вывода возвращается функцией `IOResult`.

Номер ошибки	Сообщение	Значение
100	Disk read error	Ошибка чтения диска
101	Disk write error	Ошибка записи на диск
102	File not assigned	Файл не назначен. Файловой переменной не поставлено в соответствие имя файла
103	File not open	Файл не открыт
104	File not open for input	Файл не открыт для ввода
105	File not open for output	Файл не открыт для вывода
106	Invalid numeric format	Неверный числовой формат

Критические ошибки

Номер ошибки	Сообщение	Значение
150	Disk is write protected	Диск защищен от записи
151	Unknown unit	Неизвестный модуль
152	Drive not ready	Дисковое устройство не готово к работе
153	Unknown command	Неопознанная команда
154	CRC error in data	Ошибка в данных, обнаруженная при помощи кодов CRC
155	Bad drive request structure length	Указана неверная длина структуры запроса к диску

Критические ошибки. Окончание

Номер ошибки	Сообщение	Значение
156	Disk seek error	Ошибка при установке головок на диске
157	Unknown media type	Неизвестный тип носителя
158	Sector not found	Сектор не найден
159	Printer out of paper	В принтере закончилась бумага
160	Device write fault	Ошибка при записи на устройство
161	Device read fualt	Ошибка при чтении с устройства
162	Hardware failure	Сбой в оборудовании

Фатальные ошибки

Номер ошибки	Сообщение	Значение
200	Division by zero	Деление на ноль
201	Range check error	Ошибка при проверке попадания значения в диапазон допустимых значений
202	Stack overflow error	Переполнение стека
203	Heap overflow error	Переполнение динамически распределяемой области памяти
204	Invalid pointer operation	Некорректная операция ссылки
205	Floating point overflow	Переполнение при выполнении операции с плавающей точкой
206	Floating point underflow	Исчезновение порядка при выполнении операции с плавающей точкой
207	Invalid floating point operation	Недопустимая операция с плавающей точкой
208	Overlay manager not installed	Не установлена подсистема управления оверлеями
209	Overlay file read error	Ошибка чтения оверлейного файла
210	Object not initialized	Объект не инициализирован
211	Call to abstract method	Вызов абстрактного метода
212	Stream registration error	Ошибка регистрации потока
213	Collection index out of range	Индекс набора типа TCollection вышел за пределы диапазона допустимых значений
214	Collection overflow error	Переполнение набора типа TCollection
215	Arithmetic overflow error	Арифметическое переполнение. Эта ошибка может возникнуть только при включенной директиве {$G+}
216	General protection fault	Общее нарушение защиты (только в защищенном режиме)

Книга являє собою навчальний посібник, у якому матеріал викладається за схемою "від простого до складного". Посібник розраховано на початкових програмістів в середовищі Turbo Pascal 7.0. Особисте Місго в книзі було приділено прикладам, що ілюструють різні можливості мови Pascal і бібліотечних програмних модулів. В окрему частицу відокремлені приклади найбільш складних програм, наприклад, для робота з базами даних.

Особисте Місго в книзі займають додатки, у які були включені короткі довідники по командах мови асемблера і по перериваннях. Один з додатків являє собою повний довідник по процедурам і функціям мови Pascal із прикладами їхнього використання.

Навчальне видання

Шпак Юрій Олексійович

Turbo Pascal 7.0 на прикладах

За редакцією Ю. Є. Ковтанюка

Підп. до друку 05.09.2003. Формат 70 х 100 1/16.
Папір газетний. Гарнітура Тайме. Друк офсетний.
Ум. др. арк. 19. Обл.-вид. арк. 22,10.
Тираж 2000 пр. Зам. № 100903

Видавництво "Юшор", Украша, 03142, м. Ки'ш, вул. Радгоспна, 35-37, оф. 111
тел./ф.: (044) 452-82-22; e-mail: office@junior.com.ua; http://www.junior.com.ua
Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготівників і розповсюджувачів видавничої продукци серія ДК, № 368 вщ 20.03.2001 р.

Надруковано з готових діапозитивів у ТзОВ “ Дизайн-студія Папуга”
79054, м. Львів, вул. Любінська, 92

Ю. О. Кулаков, Г. М. Луцький

Комп'ютерні мережі

ISBN: 966-7323-27-7; обсяг: 400 с., іл.; формат: 170x240 мм (70x100 1/16); обкладинка м'яка, плівка

Цей підручник, рекомендований Міністерством Освіти Украши, в першу чергу призначається студентам вищих закладів комп'ютерних спеціальностей. В основу даної книги покладений курс лекции з комп'ютерних мереж, що читається авторами протягом останніх двадцяти років студентам спеціальності "Комп'ютерні системи і мережі" Національного технічного університету Украши "Київський політехнічний інститут". Матеріал книги відповідає програмі курсу "Комп'ютерні мережі", затвердженої Міністерством Освіти Украши для технічних університетів, що, безсумнівно, зробить корисною дану книгу при вивченні вищезгаданого курсу.

Ця книга буде корисна і доступна всім бажаючим одержати або поглибити свої знання в області комп'ютерних мереж. Автори даної книги спробували викласти матеріал таким чином, щоб він був доступний читачеві, який вперше приступив до знайомства з комп'ютерними мережами і володіє мінімальними знаннями в області комп'ютерної техніки.

В. А. Хорошко, А. А. Чекатков

Методы и средства защиты информации

ISBN: 966-7323-29-3; объем: 504 с., ил.; формат: 170x240 мм (70x100 1/16); брошюра, пленка

Широкое применение электроники и вычислительной техники во всех сферах человеческой деятельности является в настоящее время приоритетным. Масштабы и сферы применения этой техники таковы, что возникают проблемы обеспечения безопасности циркулирующей в ней информации.

В предлагаемой читателю книге сделана попытка системного изложения всей совокупности вопросов, составляющих проблему методов и средств защиты информации в современных условиях. Эта проблема рассматривается в книге с единых позиций системно-концептуального подхода, который заключается в формулировке двух основных положений:

- все факторы, являющиеся значительными, должны рассматриваться как система;
- итогом должна быть совокупность взглядов для общего случая на сущность проблем и общих решений.

Большое внимание в книге уделено систематизации и обоснованию создания условий, необходимых для оптимальной реализации концепции защиты.

Книга "Методы и средства защиты информации" представляет собой учебное пособие, в первую очередь предназначенное для студентов и преподователей высших учебных заведений. Однако, собранные в ней сведения будут полезны специалистам и всем интересующимся людям, которые хотят расширить свой кругозор. Для последней категории читателей особенно интересной будет первая часть книга, которая содержит обширные сведения об истории мировой разведки.

Академик В.М. Глушков — пионер кибернетики

Составитель и автор биографического очерка В. П. Деркач

ISBN: 966-7323-31-5; объем: 384 с., ил.; цветные вклейки: 36 с.;
формат: 170x240 мм (70x100 1/16); книга, суперобложка (подарочный вариант)

Эта книга об академике Викторе Михайловиче Глушкове — выдающемся ученом и незаурядной личности. Издание содержит много фотографий из семейного архива ученого. Книга уникальна тем, что позволяет ознакомиться с неизвестными фактами из биографии Виктора Михайловича Глушкова из воспоминаний родных и друзей.

В.М. Глушков стоял у истоков кибернетической науки и внес неоценимый вклад в ее становление и развитие. Книга содержит много работ В.М. Глушкова, которые актуальны и сегодня. Они знакомят читателя с историей развития Института кибернетики НАН Украины.

Читателям будут интересны впервые публикуемые выдержки из дневника дочери Виктора Михайловича Ольги о ходе его болезни и лечения, о его мужественном поведении, когда был прикован к постели и за несколько дней до кончины в тяжелейшем состоянии полушепотом диктовал на магнитофон свои заветные мысли.

"Для тех, кто остается", так называется раздел, в котором эти мысли приведены с несущественными сокращениями и сносками, содержащими данные о людях, фамилии которых Виктор Михайлович упоминает.

Заключительной является глава "Глазами современников", в которой собраны статьи о В. М. Глушкове тех, кто его хорошо знал и мог судить о нем как об ученом, организаторе науки и о личности.

Ю. С. Ковтанюк, С. В. **Соловьян**

Самоучитель работы на ПК

ISBN: 966-7323-32-3; объем: 704 с., ил.; формат: 170x240 мм (70x100 1/16); брошюра, пленка

Книга представляет собой учеьное пособие, адресованное широкому кругу читателей, желающих самостоятельно научиться работать на персональном компьютере. Пособие построено таким образом, что читателю необязательно иметь специальную подготовку в области компьютерных технологий. Читатель ознакамливается с основними приемами работы на компьютере и общини понятиями пои мере зложения материала.

Книга будет полезна и тем пользователям, которые интересуются вопросами эксплуатации аппаратных средств и программного обеспечения современных персональных компьютеров, а также студентам и школьникам старших классов, изучающих принципы работы IBM PC-совместимых компьютеров в рамках курса общей информатики.

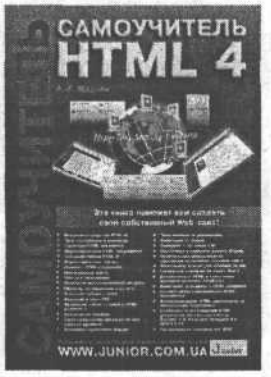
С. Э. Зелинский
MicrosoftWindowsXP
Вопросы и ответы. Русская версия

ISBN: 966-7323-25-0; объем: 528 с., ил.; формат: 170x240 мм (70x100/16); брошюра, ламинирование

Это простое и понятное руководство, изучив которое пользователи смогут не только приступить к использованию Windows XP, но и самостоятельно применять новые возможности при работе в Internet, с электронными сообщениями и мультимедийным содержимым, а также эффективно настраивать систему, оптимизировать производительность и устранять неполадки. Книга охватывает темы, связанные с настройкой, оптимизацией работы устройств компьютера и установленного на нем программного обеспечения. Подробно рассмотрены практически все темы, необходимые для успешной повседневной работы, пользователей на компьютере. Среди них — настройка пользовательского интерфейса, установка устройств и программ и их обслуживание, работа с файлами и папками, работа в локальных сетях и Internet, подключение к различным сетям. Отдельно рассмотрены вопросы использования Internet Explorer 6, Outlook express 6, Messenger, Media Player, Movie Maker, а также системных служб из состава Windows XP. В дополнение к основному материалу приводятся сочетания клавиш для выполнения основных задач в Windows XP. Книга рассчитана на широкий круг пользователей, как начинающих, так и имеющих навыки работы в предыдущих версиях операционной системы Windows.

А. Г. Жадаев
Самоучитель HTML 4

ISBN: 966-7323-19-6; объем: 296 с., ил.; формат: 170x240 мм (70x100/16); с дискетой



Книга "Самоучитель HTML 4" предназначена для пользователей начального и среднего уровня, и позволяет быстро приступить к созданию собственных HTML-документов на языке HTML 4.01. Хорошая структурированность материала и широкий охват разделов языка HTML превращают это издание в отличный настольный справочник по HTML 4 для пользователя любого уровня.

Материал книги рассматривается с учетом последних рекомендаций консорциума W3C, а многочисленные примеры HTML-страниц рассматриваются с учетом особенностей их отображения в самых популярных программах: Microsoft Internet Explorer 5.5, Natscape Navigator 6 и Opera 5.02.

А. Н. Моргун
Решение задач средствами языка Turbo Pascal 7.0

ISBN: 966-7323-22-6; объем: 216 с., ил.; формат: 170x240 мм (70x100/16); брошюра, ламинирование

В этой книге очень подробно и последовательно изложены основы алгоритмизации и программирования на языке Паскаль. Обучение программированию, в отличие от традиционных подходов, построено по принципу "от задачи к программе". Автор показывает, как целесообразно строить программы, начиная с решения простейших задач. Средства языка Паскаль вводятся в сферу их применения по мере появления необходимости в них для решения все усложняющихся задач.

Книга предназначена, в первую очередь, для учащихся средней школы и студентов начальных курсов ВУЗов, а также для всех, кто желает научиться правильному программированию, не ограничиваясь просто записью операторов на Паскале. Книга может быть полезна особенно тем студентам, которым трудно преодолеть вузовские барьеры на базе существующего школьного уровня подготовки по информатике. Никаких специальных знаний при изучении материала книги не требуется.

Ю. А. Шпак
Delphi 7 на примерах

ISBN: 966-7323-28-5; объем: 384 с., ил.; формат: 170x240 мм (70x100/16); с дискетой

Книга написана специалистом-разработчиком систем управления базами данных. Разработанные им системы успешно эксплуатируются в разных организациях, например, в подразделениях Таможенной службы Украины. Автор делится с разработчиками секретами создания приложений средствами Delphi 7, предназначенных для работы с базами данных разных структур: локальными базами данных и базами данных, расположенными на SQL-серверах.

Изучение материала построено на примерах реальных проектов, которые можно применять в повседневной работе или использовать, как основу, для создания собственных приложений. Начинающий разработчик может легко изменить представленные в книге проекты, руководствуясь пошаговыми инструкциями по созданию баз данных, программного кода и интерфейса приложений.

Кроме того, книга содержит полезную справочную информацию по языку Object Pascal, соглашениям и компонентам Delphi, процедурам и функциям, и основам языка HTML.

Уважаемый читатель!

- Понравилась ли вам эта книга?
- Интересуют ли вас другие книги по этой тематике?
- Нуждаетесь ли вы в другой литературе?
- Хотите ли вы регулярно получать информацию о перспективных планах и новинках нашего издательства?

Если ваш ответ — "Да", давайте познакомимся!

С помощью приведенной ниже карточки Вы сможете стать нашим зарегистрированным читателем и оперативно получать информацию о новых книгах нашего издательства и приобретать их на выгодных условиях. Мы же, со своей стороны, сможем лучше и быстрее с помощью наших книг оказывать вам помощь в решении ваших проблем.

РЕГИСТРАЦИОННАЯ КАРТОЧКА читателя издательства "ЮНИОР"

Фамилия _____ Имя _____ Отчество _____

ул. _____ дом, корпус, кв. _____

нас. пункт _____ район _____

область _____ страна _____

контактный телефон и код населенного пункта _____

E-mail: _____

возраст _____ место работы или учебы _____

параметры домашнего компьютера _____

Индекс

Turbo Pascal 7.0 на примерах

Пожалуйста, заполняйте разборчиво

Почтовый адрес издательства:

03142 г. Киев, ул. Совхозная, 35, офис 111. Издательство "Юниор"

Издательство "ЮНИОР" приглашает к сотрудничеству авторов компьютерных книг по следующим темам

- Программирование на языке C/C++
- Программирование на языке Ассемблер
- Разработка приложений в среде Visual Basic
- Разработка приложений в среде Visual C
- Разработка приложений в среде Delphi
- Разработка приложений в среде Visual FoxPro
- Компьютерная безопасность и защита информации
- Web-программирование: создание серверных сценариев на языке JavaScript, Perl и PHP
- Операционные системы Windows и Linux
- Adobe PhotoShop и Adobe Illustrator
- 3ds max
- Аппаратное обеспечение ПК
- Разработка клиент-серверных приложений

С предложениями обращаться по электронному адресу: edit@junior.com.ua

Turbo Pascal 7.0 на примерах

- В книге представлены основные сведения для работы в среде Turbo Pascal 7.0. Одно из приложений содержит справочник команд интегрированной среды
- Подробно рассмотрены основы программирования на языке Turbo Pascal: идентификаторы, константы, переменные и операторы; типы данных; ввод и вывод данных; операторы ветвления и циклов; процедуры и функции
- На примере программ учета, обработки и преобразования данных рассмотрены структурированные типы данных: строки и массивы; множества и записи; работа с файлами, указателями и объектами
- Подробно рассмотрены вопросы создания модулей, компиляции и отладки программ. Отдельное приложение посвящено описанию сообщений об ошибках
- Графика, звук и работа с мышью рассматриваются на примерах законченных приложений: игра "Охотник"; программа, превращающая клавиатуру компьютера в клавиатуру пианино; простейший графический редактор
- Представлена реализация основных методов сортировки массивов с наглядным описанием работы программ сортировки
- На примере приложения, позволяющего вести учет музыкальных компакт-дисков, рассматриваются вопросы создания баз данных и управления ими средствами Turbo Pascal
- Справочники по основным прерываниям BIOS/DOS, командам процессоров 8x86 и готовые примеры с использованием подпрограмм на языке ассемблер позволяют применять эти средства для расширения возможностей программиста на языке Pascal
- В книге уделено внимание разным методам установки Turbo Pascal на компьютер и использованию Turbo Pascal с разной конфигурацией
- Завершает книгу подробный справочник с отдельными работающими примерами по каждой процедуре и функции языка Turbo Pascal

Дискета содержит программные коды примеров, рассматриваемых в книге

Изучите Turbo Pascal 7.0 на примерах реальных программ

Изучение материала построено на примерах реальных программ, которые можно применять в повседневной работе, учебном процессе или использовать как основу для создания собственных приложений

Начинающий программист может легко изменить представленные в книге программы, руководствуясь пошаговыми инструкциями по разработке программного кода и интерфейса приложений

Об авторе

Шпак Юрий Алексеевич — специалист по компьютерным и интеллектуальным системам и сетям, разработчик систем управления базами данных, которые с успехом использовались в различных подразделениях Таможенной службы Украины. Является автором книги "Delphi 7 на примерах". В 1996 году закончил с отличием Киевский международный университет Гражданской авиации. На Turbo Pascal программирует с 1991 года

Личный сайт автора
http://www.yshpak.net

ISBN 966-7323-30-7

ваться комбинацией клавиш <Alt+N>, где N — номер окна. Кроме того, для перехода к одному из окон можно выполнить команду меню **Window | List.**

Закрыть окно интегрированной среды Turbo Pascal, например, окно **Help,** можно щелкнув мышью на его закрывающей кнопке, или при помощи комбинации клавиш <Alt+F3>.

Строка состояния

Строка состояния расположена вдоль нижнего края экрана (см. рис. 1.1). Она выполняет следующие функции.

- Отображает основные комбинации клавиш для быстрого выполнения команд меню, *применимых* в данный момент.
- Позволяет выполнять команды при помощи щелчка мышью или нажатия соответствующей комбинации клавиш. ,
- Отображает информацию о выполняемой в данный момент операции.
- Отображает краткую подсказку о выбранной команде меню или каком-нибудь элементе диалогового окна.

Простейшая программа

После краткого рассмотрения основных элементов интегрированной среды Turbo Pascal, приступим к созданию первой простейшей программы.

Имя файла с исходным текстом программы

Выполните команду **File | Save** (клавиша <F2>) или команду **File | Save As.** В результате на экране появится одноименное диалоговое окно, представленное на рис. 1.3.

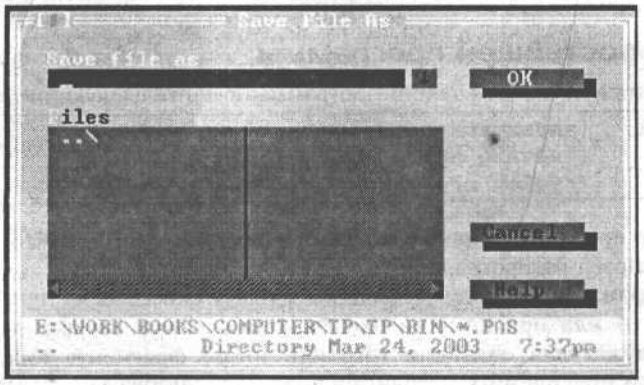


Рис. 1.3. Диалоговое окно для сохранения файла

В списке **Files** (Файлы) выберите каталог, в котором будет сохранен файл с текстом программы, а в поле **Save file as** (Сохранить файл как) укажите имя этого файла (например, `Prog01.pas`). Щелкните мышью на кнопке **OK** или нажмите клавишу <Enter>. В результате в заголовке окна текстового редактора отобразится новое название файла.

Определение каталогов

Выполните команду меню **Options | Directories.** В результате на экране появится диалоговое окно, представленное на рис. 1.4.

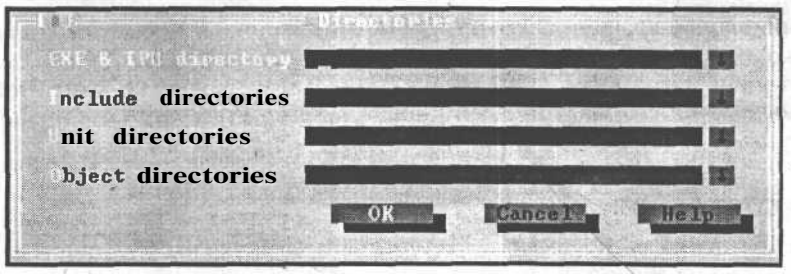


Рис. 1.4. Диалоговое окно **Directories**

В этом диалоговом окне можно определить пути поиска файлов, используемых средой Turbo Pascal при компиляции программ.

- **EXE & TPU directory.** В этом поле указывается каталог размещения результирующего выполняемого файла .exe, а также откомпилированных модулей .tpu.
- **Include directories.** В этом поле указываются каталоги, содержащие стандартные включаемые файлы. Имена каталогов отделяются друг от друга символом ";".
- **Unit directories.** В этом поле указываются каталоги, содержащие файлы подключаемых модулей .tpu. В системе Turbo Pascal такие файлы размещаются в каталоге ..\UNITS.
- **Object directories.** В этом поле указываются каталоги, содержащие объектные файлы подпрограмм, написанных на языке ассемблера (файлы с расширением .obj).

» Язык ассемблера рассматривается в главе 15.

Если поле **EXE & TPU directory** будет пустым, то файл .exe будет создан в том же каталоге, что и файл .pas. Для того чтобы сохранить настройки, сделанные в диалоговом окне **Directories,** закройте это окно щелчком мыши на кнопке **OK.**

Структура простейшей программы

В окне редактора исходного текста программы введите следующие строки:

```
program  Prog01;   {заголовок программы}
begin              (*начало программы*)
end;               (*конец программы*)
```

Prog01 — это простейшая программа на языке Pascal, которая ничего не выполняет.

Выражения справа от программного кода — это *комментарии,* которые игнорируются компилятором. Они выделяются одним из двух способов — при помощи фигурных скобок {...} или при помощи пары круглых скобок со звездочками (*...*). Комментарии служат для описания того или иного фрагмента текста программ, а также для вставки пояснений к исходному коду. Использование комментариев значительно облегчает анализ больших и сложных программ. Комментарии, которые используются в примерах, рассматриваемых в данной книге, можно не вводить — это не повлияет на работу программ.

Обратите внимание на то, что в окне редактирования различные элементы текста программы выделены различными цветами. Например, слова program, begin и end выделены белым цветом, название программы Prog01 — желтым цветом, а комментарии — серым цветом. Подобные цветовые установки выбираются интегрированной средой программирования по умолчанию, и при желании их можно изменить в специальном диалоговом окне, открываемом при помощи команды меню **Options | Environment | Colors.**

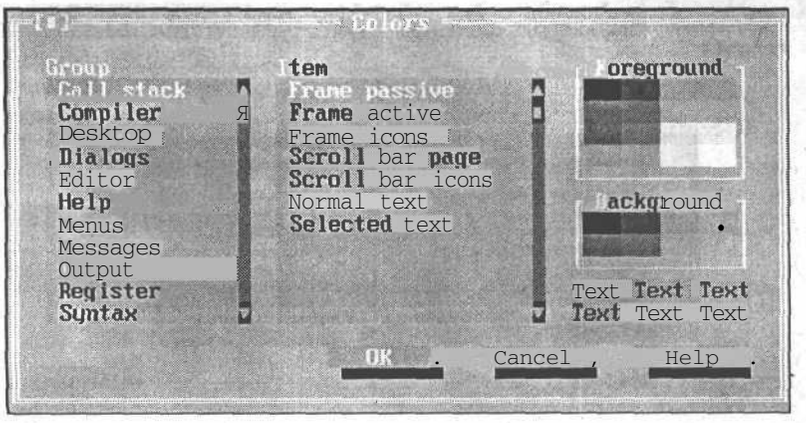


Рис. 1.5. Диалоговое окно **Colors**

Слова, по умолчанию выделяемые белым цветом, называются *зарезервированными*. Зарезервированные слова — это составная часть языка Pascal, и потому они не могут использоваться программистом в качестве имен переменных, констант и т.п. Зарезервированными являются следующие слова: absolute, and, array, asm, begin, case, const, constructor, div, go to, do, downto, destructor, else, end, exports, external, file, for, forward, function, if, implementation, in, inline, interrupt, interface, inherited, label, library, mod, nil", not, or, of, object, packed, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.

В программе ProgOl содержится три зарезервированных слова, которые присутствуют в любой программе, написанной на языке Pascal. Слово program используется для определения заголовка программы. После него указывается имя программы. Рекомендуется, чтобы имя программы совпадало с именем файла .pas, в котором хранится текст этой программы. Компилятор языка Pascal не учитывает регистр символов, поэтому, имена P'rogOl и PROG01 — это одно и то же.

Отдельные структурные элементы программы отделяются друг от друга символом ";". В конце программы ставится точка (символ "."). В данном случае программа состоит из двух структурных элементов: заголовка и тела программы. Тело программы определенно парой слов begin и end.

Компиляция программы

Для компиляции программы (то есть трансляции ее текста в машинные коды) необходимо выполнить команду Compile | Compile, или нажать комбинацию клавиш <Alt+F9>, или щелкнуть мышью на этой комбинации клавиш в строке состояния. Если текст пограммы не содержит синтаксических ошибок, то на экране появится сообщение об успешном выполнении компиляции (рис. 1.6).

Для того чтобы закрыть это окно, нажмите любую клавишу. А что произойдет, если текст программы будет содержать какую-нибудь ошибку? В качестве примера, удалите завершающую точку программы после слова end и откомпилируйте программу еще раз. В результате в верхней строке редактора отобразится сообщение об ошибке, а курсор будет установлен в строку, в которой эта ошибка была обнаружена (рис. 1.7).

» Подробно сообщения об ошибках рассматриваются в приложении 3.

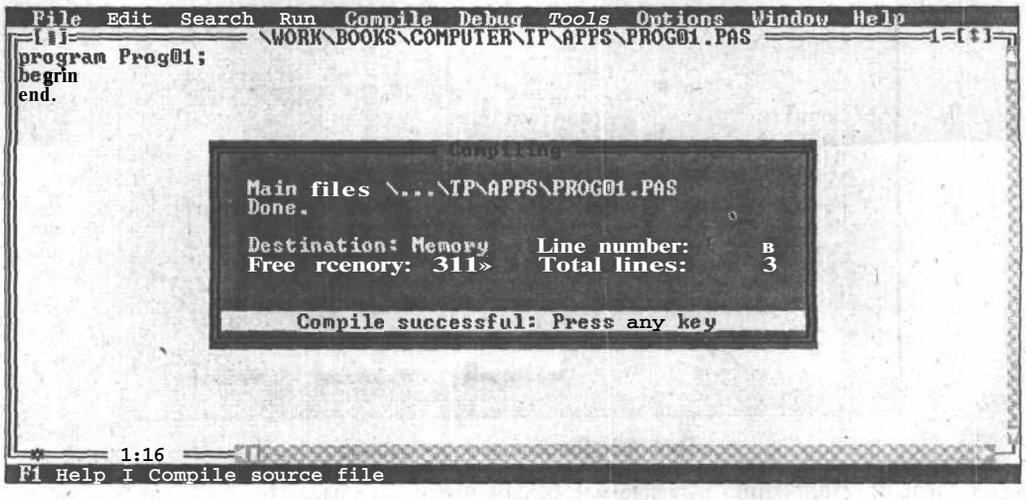


Рис. 1.6. Сообщение о результатах компиляции

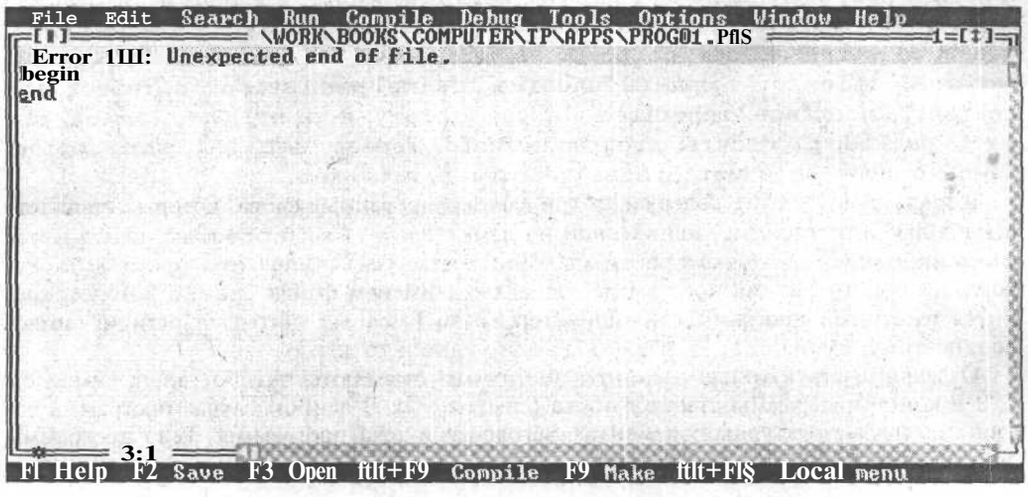


Рис. 1.7. Обнаружена ошибка в тексте программы

Выполнение программы

Откомпилированный выполняемый код может сохраняться либо в оперативной памяти, либо на жестком диске. Это определяется значением параметра **Destination** (Назначение) выбираемое при помощи соответствующего пункта меню **Compile.** Если этот параметр имеет значение **Memory,** то откомпилированный код будет сохраняться в оперативной памяти и выполнить его будет невозможно. Если же параметр **Destination** имеет значение **Disk,** то откомпилированный код будет сохраняться на жестком диске в виде выполняемого файла с расширением `.exe`. Для изменения значения параметра **Destination** необходимо просто выбрать пункт меню **Compile | Destination.**

Вставьте в текст программы точку после слова end, выберите пункт меню **Compile | Destination** и откомпилируйте программу. В результате в каталоге, определенном для

хранения файлов .exe (см. рис. 1.4) будет создан файл Prog01.exe. Этот файл можно выполнять непосредственно из интегрированной среды программирования Turbo Pascal при помощи команды Run | Run или комбинации клавиш <Ctrl+F9>.

СОВЕТ

При запуске программы из интегрированной среды предварительно выполняется компиляция, поэтому выполнение команды **Compile | Compile** или нажатие комбинации клавиш <Alt+F9>, перед запуском программы, является излишним.

Выполнять программу ProgOl пока не имеет смысла, так как она ничего не делает. Обычно в книгах по программированию в качестве простейшего примера рассматривается программа "Привет, мир", которая отображает на экране одну текстовую строку. Не будем отходить от этой традиции, и реализуем в программе Prog01 вывод строки на экран. Вставьте в программу текст, имеющий полужирное начертание в листинге 1.1.

СОВЕТ

К данной книге прилагается дискета со всеми, рассматриваемыми в ней, примерами программ. Текст программ можно вводить в интегрированной среде Turbo Pascal при помощи клавиатуры, а можно открыть любую программу из книги, найдя ее на дискете. Примеры располагаются в папках, имена которых совпадают с номерами глав.

см. об открытии файлов в разделе "Открытие файлов с текстами программ" этой главы.

Листинг 1.1. Программа Prog01.pas

```
program ProgOl;
begin
  Writeln('Привет, мир!');
end.
```

Запустите эту программу на выполнение, нажав комбинацию клавиш <ctrl+F9>. На первый взгляд кажется, что ничего не происходит. Дело в том, что эта программа выполняется настолько быстро, что нельзя успеть заметить ее действие. Для того чтобы увидеть результаты работы программы, выполните команду **Debug | User screen** или нажмите комбинацию клавиш <Alt+F5>. Для возврата в интегрированную среду программирования нажмите любую клавишу или щелкните кнопкой мыши.

По результатам работы программы ProgOl можно догадаться, что ее фрагмент Writeln('Привет, мир!'); выводит на экран соответствующую строку текста. В данном случае для вывода строки вызывается процедура Writeln. *Процедуры* и *функции* — это особые конструкции языка Pascal, представляющие собой именованные фрагменты программного кода. Подробно процедуры и функции рассматриваются в главе 7, здесь же отметим только отличие между ними. Процедуры не возвращают никакого результата, а функции — возвращают. При вызове процедур и функций в них могут передаваться *параметры.* Например, в процедуру Writeln передается параметр, содержащий текстовую строку, которая должна быть выведена на экран монитора.

Справочная информация

Разнообразную справочную информацию можно просматривать при помощи команд меню **Help.** Кроме того, для просмотра общей справочной информации можно нажать клавишу <F1>. Для вызова подсказки по использованию какого-нибудь диалогового окна можно также щелкнуть мышью на его кнопке **Help.**

Если необходимо выполнить поиск информации по предметному указателю (в алфавитном порядке), нажмите комбинацию клавиш <Ctrl+Fl>. Если при этом курсор будет установлен в окне редактора на имени какой-либо процедуры, функции или другой синтаксической конструкции, то будет отображена информация, соответствующая этому зарезервированному слову. Например, установите курсор на имени процедуры `Writeln` из листинга 1.1, и нажмите комбинацию клавиш <Ctrl+Fl>. На экране раскроется окно справки, содержащее информацию о процедуре Writeln (рис. 1.8).

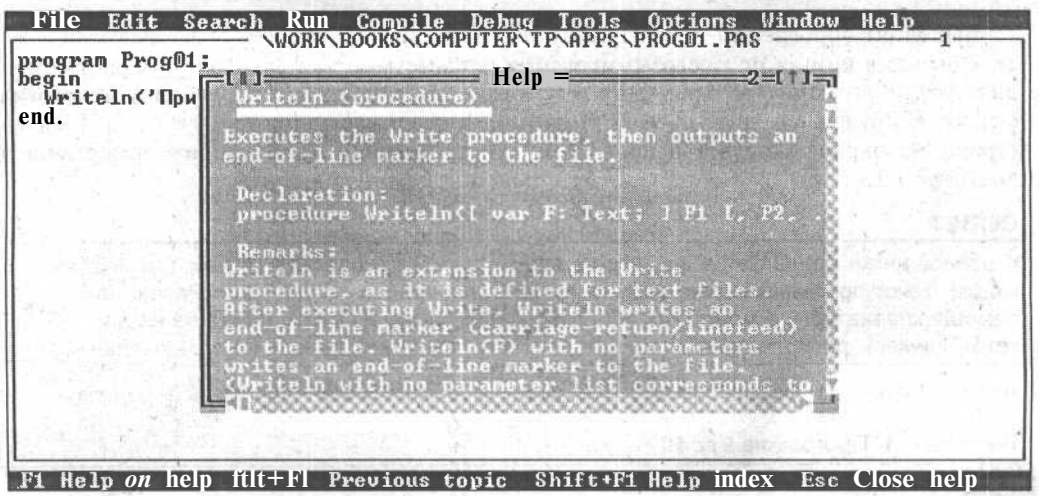


Рис. 1.8. Просмотр тематической справочной информации

Закрытие окна текстового редактора

Как уже было указано в начале этой главы, закрыть окно интегрированной среды Turbo Pascal можно при помощи закрывающей кнопки, расположенной в левой части заголовка окна. Если закрыть окно текстового редактора, содержащее программу, которую изменяли и забыли сохранить, то на экране компьютера появится окно **Information**, показанное на рис. 1.9. Это окно позволяет программисту избежать потери изменений программы, которую он мог забыть сохранить.

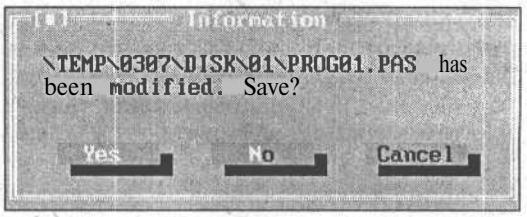


Рис. 1.9. Диалоговое окно **Information**

Окно **Information** содержит запрос на сохранение файла с расширением `.pas`. При помощи кнопки **Yes** можно сохранить измененный файл `.pas`. Щелчок мыши на кнопке ***Но*** закрывает файл с программой, но без сохранения внесенных в него изменений. Кнопка **Cancel** (Отмена) позволяет вернуться к программе, отменяя закрытие окна текстового редактора.

Выход из среды Turbo Pascal

Выйти из среды Turbo Pascal можно, выполнив одно из следующих действий: команду **File | Exit** или нажав комбинацию клавиш <Alt+x>. Если во время завершения работы с интегрированной средой программирования Turbo Pascal