Министерство образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

В.В. Коробова

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ ДАННЫХ

Учебное пособие

2000

Коробова В.В.
Информационные технологии обработки данных: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2000. – 88 с.

Учебное пособие предназначено для изучения языка Турбо-Паскаль. Пособие содержит краткое описание языка и основных приемов программирования.

Пособие может быть использовано для студентов заочной и дневной форм обучения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Обработка данных предполагает создание некоторой программы, выполняющей над этими данными определенные действия. В настоящее время существует множество различных специализированных программ, предназначенных для работы с какими-либо конкретными данными, например, для проведения экономических расчетов, для обработки статистических исследований, для расчетов различных технических параметров, для обработки звука или графических изображений и т.д.

Несмотря на обилие существующих программ, потребность в создании новых программ только возрастает.

Учиться программировать легче всего на языке Паскаль. Этот язык изначально был создан именно для обучения. Как и другие языки программирования высокого уровня (например, Си, который Вы изучите позже), он предоставляет множество возможностей, но при этом Паскаль гораздо проще, что делает его просто незаменимым для начинающих программистов. В рамках данного курса будет использоваться видоизмененная версия языка Паскаль – Турбо-Паскаль.

В течениии первого семестра студентам будет предложено выполнить две контрольные работы, во втором семестре – курсовую работу.

1. ПИШЕМ ПЕРВУЮ ПРОГРАММУ

Система программирования Турбо-Паскаль представляет собой не только сам язык, но и некоторую среду, с помощью которой создаются и компилируются (преобразуются в машинные коды) исходные тексты программ, а также запускаются на выполнение и отлаживаются готовые программы. Сама система (а это тоже программа) находится в файле `turbo.exe` в каталоге `tp`. На Вашем компьютере Турбо-Паскаль может находиться и в другом каталоге. В любом случае, найдите файл `turbo.exe` и запустите его.

Итак, Вы запустили Турбо-Паскаль. И увидели на экране что-то похожее на рис. 1.1.

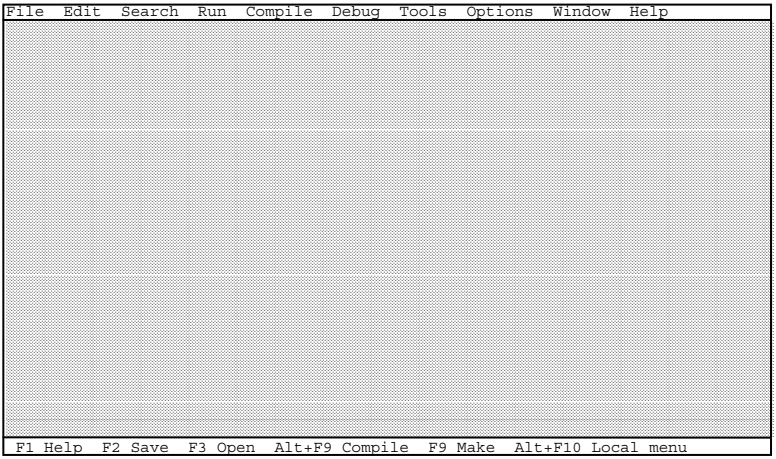

Рис. 1.1. Вид экрана после запуска программы Турбо-Паскаль

Обратите внимание на верхнюю строку экрана, она представляет собой меню системы Турбо-Паскаль. С помощью меню Вы можете выполнять все действия, которые только возможны в данной системе. Нажмите `F10` и выбирайте, что бы Вы хотели сейчас сделать. Например, Вы можете создать новый файл, содержащий текст программы на Паскале. Для этого нужно:

- выбрать пункт меню `File` (поместите на него курсор и нажмите `Enter` или просто щелкните мышью);
- в появившемся выпадающем меню выбрать пункт `New`.

Если Вы проделали все эти действия, то вид Вашего экрана изменился (рис. 1.2). Теперь большую часть экрана занимает окно, в котором отображается содержимое только что созданного Вами файла – это окно редактора Турбо-Паскаль. Как видите, в окне ничего нет. Наша задача – чем-нибудь его заполнить. Что ж, приступим к написанию нашей первой программы. Но сначала рассмотрим поподробнее вид окна.

В центре самой первой строки окна расположен его заголовок – имя файла, содержимое которого отображено в окне. В нашем случае файл называется NONAME00.PAS. Такое имя автоматически присваивается вновь создаваемому файлу.

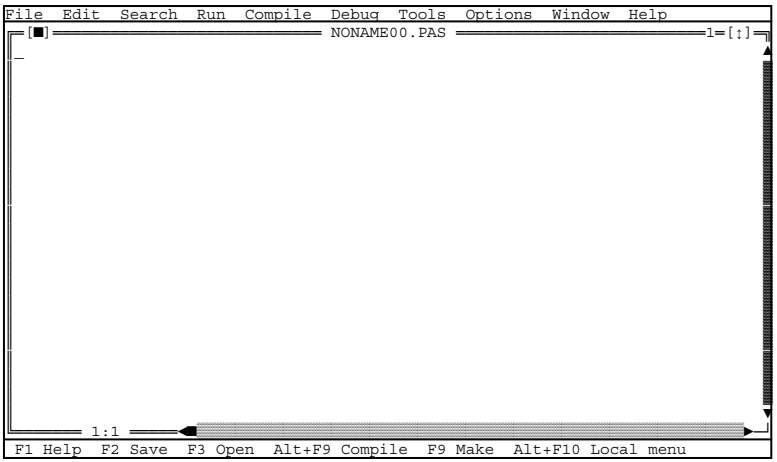

Рис. 1.2. Общий вид окна редактора Турбо-Паскаль

На верхней строке слева расположена кнопка закрытия окна [■]. Попробуйте щелкнуть на ней мышкой – окно исчезнет (не забудьте потом снова открыть его). Можно закрыть окно и другим способом – нажатием клавиши Alt+F3.

При работе с программой Турбо-Паскаль Вы будете использовать множество различных окон. И размеры окон могут быть самыми разными. Если в верхней строке окна справа имеется кнопка [↑] или [↕], то, щелкнув мышкой на этой кнопке, можно увеличить окно до максимального размера или вернуть прежний размер, соответственно. К сожалению, наше окно уже распахнуто, насколько возможно, поэтому для того, чтобы испробовать эту кнопку откройте еще одно. Его размер чуть-чуть отличается от первого, это позволяет продемонстрировать возможности кнопок [↑] и [↕].

Обратите внимание на число рядом с кнопкой – это номер окна. Следует отметить, что для перехода из одного окна в другое нужно, нажать Alt+*номер окна*.

Теперь перейдем к самой нижней строке окна.

Слева отображаются координаты курсора – номер строки и столбца, в которых он находится. Попробуйте подвигать его и убедитесь в том, что координаты в нижней строке меняются. Скорее всего, Вы не смогли подвигать курсор по вертикали, так как для перехода на новую строку необходимо нажать Enter. Если Вы нажмете Enter, то сможете перемещаться уже по двум строкам и т.д.

Большую часть самой нижней строки окна самый правый столбец занимают ***полосы скроллинга***. Курсор на каждой из них показывает, текущее положение текста в окне относительно всего текста. В этом Вы убедитесь, когда напишите программу побольше – не помещающуюся на экране целиком. Вот тогда и попробуйте пощелкать на полосах скроллинга мышкой.

Вот мы и разобрались с окном. Теперь уделим внимание самой нижней строке экрана. Эта строка постоянно напоминает Вам о том, как можно выполнить самые важные действия. Со временем Вы все это запомните, а пока – достаточно глянуть на строку подсказки и сразу ясно:

- чтобы посмотреть файл помощи, нужно нажать `F1`;
- чтобы записать файл (сохранить изменения в файле), находящийся в активном окне (в окне, в котором Вы работаете), нужно нажать `F2`;
- открыть существующий файл – `F3`;
- откомпилировать программу (из текста создать файл, готовый к запуску) – `Alt+F9`;
- откомпилировать программу в режиме Make – `F9` (отличие этого режима от обычного для Вас пока неважно – можете использовать и тот, и другой);
- попасть в локальное меню – `Alt+F10`.

Итак, все, что нужно для работы, Вы уже знаете. Можно начинать писать программу. С возникающими вопросами будем разбираться по ходу дела.

Пока наш файл не имеет даже имени. Запишите его на диск. Для этого нажмите `F2`. На экране появится диалоговое окно `Save File As` (рис. 1.3).

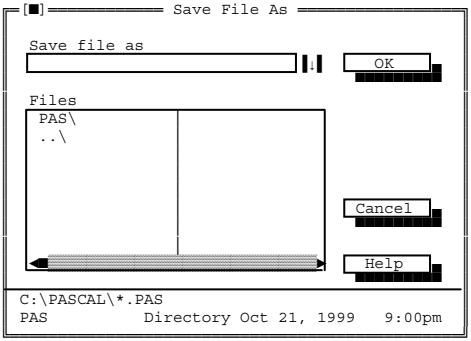

Рис. 1.3. Диалоговое окно сохранения файла

В верхней части этого окна расположено поле для ввода имени файла. Если Вы наберете нужное имя и нажмете `Enter` (или щелкните мышкой кнопку `OK`), файл с таким именем и с расширением `.PAS` появится в каталоге, указанном внизу окна – в данном случае, в каталоге `C:\PASCAL`. Если Вы хотите сохранить файл в каком-нибудь другом каталоге, то перейдите с помощью клавиши `Tab` в поле `Files` и укажите нужный каталог.

Сохраните Ваш файл под каким-нибудь именем, например, под именем Prog1. В следующий раз, когда Вы нажмете F2, файл будет сохранен автоматически под этим же именем. Если же Вы захотите сохранить его под другим именем, выберите File|Save as... (в меню – пункт File, затем в выпадающем меню – пункт Save as...) и получите на экране диалоговое окно, которое мы только что рассмотрели.

Текст Вашей первой программы приведен ниже (пример 1). Наберите его в окне, стараясь не допускать ошибок.

Пример 1

```
PROGRAM Prog1;
uses Crt;
var
   Name: string[20];
BEGIN
ClrScr;
write('Введите, пожалуйста, Ваше имя: ');
readln(Name);
writeln('Здравствуйте, ', Name);
readln;
END.
```

Что же делает эта программа?

Для того чтобы узнать это, нужно откомпилировать ее и запустить. Посмотрите на строку подсказки и Вы сразу же вспомните, что откомпилировать программу можно нажав клавишу F9 (или Alt+F9). Откомпилируйте сей шедевр, сохраните его на всякий случай (F2) и запустите на выполнение. Запустить программу можно через меню Run|Run или нажав клавишу Ctrl+F9.

Если программа была набрана правильно, то после запуска она попросит Вас ввести Ваше имя. Наберите имя и нажмите Enter. Теперь программа поприветствует Вас лично. В данный момент экран Вашего компьютера должен выглядеть примерно так, как изображено на рис. 1.4.

Чтобы вернуться к окну с текстом программы, нажмите Enter.

```
Введите, пожалуйста, Ваше имя_: Александр
Здравствуйте, Александр
_
```

Рис. 1.4. Результаты работы программы примера 1

Попробуем разобраться в скрытом смысле каждой строки нашей программы.

Первая строка программы начинается ***зарезервированным словом*** `PROGRAM` и содержит объявление имени программы: наша программа имеет имя `Prog1`. На самом деле, объявление имени программы необязательно, поэтому в дальнейшем эту строку мы будем опускать.

Первая строка заканчивается ***разделителем*** - точкой с запятой. Этот разделитель в языке Турбо-Паскаль показывает конец ***оператора*** или ***описания***. Использование разделителя позволяет помещать в одной строке несколько различных операторов, но делать это не рекомендуется во избежание лишних ошибок.

Вторая строка программы содержит объявление используемых модулей. ***Модуль*** – это дополнительный файл, который содержит процедуры, выполняющие действия, не предусмотренные стандартными операторами языка. В данной программе, например, для очистки экрана используется процедура `ClrScr` из модуля `Crt`. Позднее мы будем использовать и другие процедуры из этого модуля, а также процедуры из других модулей. Их описание будет рассматриваться далее.

Обратите внимание на то, что зарезервированное слово `uses` должно следовать сразу же за объявлением имени программы или, если таковое отсутствует, в самом начале программы.

В третьей строке Вы видите единственное слово `var`, означающее, что далее будут описаны одна или несколько переменных. ***Переменными*** в языке называют «ячейки» памяти, которые могут хранить какое-нибудь значение. Описать переменную в Турбо-Паскале – значит указать ее имя и тип.

В следующей строке программы описана переменная по имени `Name`. После двоеточия указан ее тип: `string[10]`. Это означает, что в программе будет использоваться переменная `Name` является строкой из десяти символов.

Как видите, все четыре первые строки программы не связаны с какими-либо действиями: они только содержат всякую информацию о самой программе и использующихся в ней объектах. Эта часть программы называется ***разделом описаний***.

Слово `BEGIN`, которое содержится в следующей строке программы, указывает на начало следующей части программы – ***раздела операторов***. В этом разделе описываются последовательно все действия, которые должна выполнять программа. Раздел операторов является обязательным. В нашей программе этот раздел состоит из следующих пяти операторов.

Оператор

```
ClrScr;
```

на самом деле является процедурой, которая находится в модуле `Crt`. Как было сказано выше, эта процедура выполняет очистку экрана.

Оператор

```
write('Введите, пожалуйста, Ваше имя: ');
```

выводит на экран монитора сообщение, указанное в скобках. Обратите внимание на то, что сообщение должно быть заключено в одинарные кавычки.

В следующей строке осуществляется ввод значения переменной Name с помощью оператора readln:

```
readln(Name);
```

При выполнении этого оператора программа ожидает, пока пользователь введет какое-нибудь значение, а затем запоминает его в переменной, имя которой указано в скобках.

Следующий оператор

```
writeln('Здравствуйте, ', Name);
```

также как и описанный выше оператор write, выводит сообщение на экран монитора. Но в данном случае сообщение состоит уже из двух частей (вообще-то, их может быть сколько угодно): текста 'Здравствуйте, ' и того текста, который ввел пользователь в предыдущей строке программы (содержимого переменной Name). В нашем случае переменная Name содержит строку символов 'Александр', и в результате выполнения этого оператора на экран выводится сообщение 'Здравствуйте, Александр'.

Последний оператор программы

```
readln;
```

является маленькой хитростью. Как Вы, вероятно, заметили, этот оператор аналогичен оператору, с помощью которого мы вводим значение переменной. Но в данном случае переменная не указана. Чего же тогда ожидает этот оператор? Все очень просто: этот оператор ждет, когда пользователь введет хоть что-нибудь, ну или просто нажмет Enter. Попробуйте удалить его. И запустите программу. Сразу же после ввода имени Вы окажетесь в окне редактора: программа отработала, сделала все, что должна была и вернула нас туда, откуда мы вышли. Чтобы просмотреть теперь результаты работы программы, нужно нажать Alt+F5. А можно сделать проще: заставить программу не заканчиваться, пока мы не нажмем Enter. Тут-то нам и помог этот замечательный оператор.

Последняя строка программы содержит слово END с точкой, которое, также как и BEGIN, является обязательным для каждой программы. Точка оповещает компилятор о конце программы.

Осталось только пояснить одну тонкость: отличие оператора write от оператора writeln: при выводе на экран какой-либо информации с помощью оператора writeln курсор автоматически переходит на новую строку, при использовании оператора write – остается в том месте, где закончился вывод. Посмотрите на рис. 1.5: так будет выглядеть экран при работе нашей программы, если оператор write заменить на оператор writeln.

```
Введите, пожалуйста, Ваше имя_:
Александр
Здравствуйте, Александр
_
```

Рис. 1.5. Результаты работы измененной программы примера 1

Итак, запомните, что любая программа содержит два раздела: раздел описаний и раздел операторов.

```
PROGRAM Prog1;
  { Раздел описаний }
BEGIN
  { Раздел операторов }
END.
```

Слова PROGRAM, BEGIN и END выделяют эти две части в программе. При этом объявление имени программы в первой строке не является обязательным. Чего нельзя сказать о словах BEGIN и END.

Такая структура обязательна для любой программы, такие уж требования у Турбо-Паскаля.

Все объекты, используемые в программе (переменные, модули и т.д.) должны быть сначала описаны в разделе описаний. Если в программе не используются никакие объекты, то раздел описаний будет отсутствовать. Получается, что самая короткая программа на языке Турбо-Паскаль выглядит так:

```
BEGIN
END.
```

Вот мы и разобрали нашу первую программу. Если Вам что-то показалось сложным или непонятным, не расстраивайтесь – дальше будет подробно рассмотрено множество примеров. Главное – не бойтесь, пробуйте творить сами, изменяйте приведенные примеры и смотрите, к каким результатам это приведет...

Задания для самостоятельной работы

1. В конце какого-нибудь оператора удалите точку с запятой. Попытайтесь откомпилировать программу. Посмотрите, как отреагирует на это компилятор. Исправьте ошибку.
2. В четвертой строке программы попробуйте изменить количество символов в строковой переменной, например, на два. Посмотрите, что из этого получится.
3. Измените программу так, чтобы после приветствия на экран выводился восклицательный знак.

2. ТИПЫ ДАННЫХ. ПЕРЕМЕННЫЕ. ВЫРАЖЕНИЯ

Наиболее важными элементами программы являются переменные. Именно они влияют на ход событий в программе во время ее выполнения. Например, если бы мы не указали значение переменной `Name` в примере 1, кому было бы адресовано приветствие, выведенное программой?

Переменные могут содержать совершенно различные данные. Например, в одной переменной может храниться чье-то имя, в другой – год рождения, в – третьей – рост и т.д. Такие разные данные и представляются компьютером поразному. Имя – это строка символов, год рождения – целое число, рост – вещественное число (например, рост равен 1.72 м).

Способ представления данных компьютером определяется их ***типом***. Кроме того, тип данных определяет, какие действия разрешается выполнять над этими данными.

Ниже перечислены основные стандартные типы данных языка Турбо-Паскаль:

`INTEGER` - целочисленные данные в диапазоне от –32768 до 32767, в памяти занимают два байта;

`REAL` - вещественные числа в диапазоне от 2.9×10^{-39} (2.9E-39) до 1.7×10^{38} (1.7E38), занимают шесть байт;

`CHAR` - отдельный символ, один байт;

`STRING` - строка символов, количество символов в строке (длина строки) ограничивается числом N в квадратных скобках, занимает N+1 байт (если число N не указано, то максимальная длина строки равна 255 символов);

`BOOLEAN` – логический тип, имеет два значения: FALSE (ложь) и TRUE (истина), один байт.

Заметим, что типы `INTEGER`, `CHAR`, и `BOOLEAN` относятся к ***порядковым*** типам (ordinal types).

Как Вы, наверное, помните, при описании переменной после ее имени ставится двоеточие, а затем указывается тип. Если несколько переменных имеют одинаковый тип, их имена можно перечислить через запятую. Пример описания переменных различных типов:

```
var
   a, b, c : integer;
   sum : real;
   Alpha, Beta : char;
   S : string[25];
   S_1 : string;
   t : boolean;
```

Заметьте, что переменная `S_1` является строкой символов, но при ее описании не указывается длина. В таком случае компилятор сам устанавливает максимально возможную длину - 255 символов.

Для хранения целых и вещественных чисел существуют и другие предопределенные типы данных. Их характеристики приведены в таблицах 2.1 и

2.2. Сравните эти типы с типами INTEGER и REAL, также приведенными в таблицах.

Таблица 2.1

Целые типы данных

Тип	Диапазон	Размер в байтах
SHORTINT	-128 .. 127	1
INTEGER	-32768 .. 32767	2
LONGINT	-2147483648 .. 2147483647	4
BYTE	0 .. 255	1
WORD	0 .. 65535	2

Таблица 2.2

Вещественные типы данных

Тип	Диапазон	Число значащих цифр	Размер в байтах
REAL	2.9×10^{-39} .. 1.7×10^{38}	11-12	6
SINGLE	1.5×10^{-45} .. 3.4×10^{38}	7-8	4
DOUBLE	5.0×10^{-324} .. 1.7×10^{308}	15-16	8
EXTENDED	3.4×10^{-4932} .. 1.1×10^{4932}	19-20	10
COMP	$-2^{63}+1$ .. $2^{63}-1$	19-20	8

Столько разных типов, скажете Вы, и какой же из них использовать?

Это зависит от поставленной перед Вами задачи. Например, Вам нужна переменная, в которой Вы будете хранить рост некоторого человека (вещественное значение): в этом случае достаточно использовать тип SINGLE. Если какая-то переменная используется у Вас для подсчета количества определенных объектов (целое положительное значение), то прикиньте, может ли быть это число больше 255, если нет – используйте BYTE, если же может – Вам не обойтись без WORD, а в некоторых случаях может понадобиться и LONGINT.

Чтобы узнать о различных типах побольше, нажмите Shift+F1 в среде Турбо-Паскаль (появится окно индекса помощи), а затем выбирайте интересующий Вас объект (например, наберите ‘type’ или ‘real’).

А теперь рассмотрим еще одну программу (пример 2). Эта программа выводит на экран отношение двух целых чисел, введенных с клавиатуры.

Пример 2

```
{Программа вычисления частного от деления
 двух целых чисел, введенных с клавиатуры}
uses Crt;
var
   n1, n2 : integer;
   x : real;
```

```
BEGIN
ClrScr;
write('n1=');
readln(n1);
write('n2=');
readln(n2);
x:=n1/n2;
writeln('n1/n2=',x);
END.
```

Наверное, Вы сразу же заметили появление ***комментариев*** – пояснений к программе, заключенных в { и }. Комментарии обладают одним замечательным свойством: в них можно писать все, что вздумается, и компилятор не обратит на это внимания. Используйте их для пояснения действий программы, и Вам не нужно будет каждый раз вспоминать, что делает та или иная ее часть, тот или иной оператор. Комментарии можно также заключать в пары скобка-звездочка. Например:

```
(* Это комментарий *)
```

Необходимо лишь помнить о том, что комментарии одного типа нельзя включать друг в друга. Например, вполне допустимы следующие комментарии:

```
(* Комментарий { Продолжение комментария } *)
{ Комментарий (* Продолжение комментария *) }
```

но совершенно недопустимы следующие:

```
(* Комментарий (* Продолжение комментария *) *)
{ Комментарий { Продолжение комментария } }
```

Вернемся к нашей программке. Если Вы хорошо разобрались с предыдущим примером, то и эта программка не покажется Вам сложной. Но, все же, опишем ее вкратце.

Раздел описаний программы включает объявление модуля `Crt` и описание переменных. Переменные `n1` и `n2` представляют собой целые числа, а переменная `x` – результат их деления, вещественное число.

Первая строка раздела операторов Вам уже знакома, в этом месте вызывается процедура очистки экрана из модуля `Crt`. В последующих четырех строках программы осуществляется ввод значений переменных `n1` и `n2`, это Вам тоже знакомо. А вот что же делает следующий оператор? Наверное, Вы догадались: оператор

```
x:=n1/n2;
```

заносит в переменную `x` значение, полученное при делении `n1` на `n2`. Такой оператор называется ***оператором присваивания***. В общем виде его можно записать так:

переменная := *выражение*

При этом *переменная* должна иметь тот же тип, что и *выражение*. В нашей программе, например, при делении одного целого числа на другое может получиться вовсе не целое значение. Поэтому мы и использовали переменную типа `real` для хранения результата.

Ну и последний оператор программы, как Вы, конечно, знаете, выводит на экран полученный результат.

Как видите, эта программа не представляет особой сложности.

Результаты работы программы должны выглядеть так, как показано на рис. 2.1.

```
n1=5
n2=2
n1/n2= 2.5000000000E+00
_
```

Рис. 2.1. Результаты работы программы примера 2

Результат деления, выведенный программой, может показаться странным. На самом же деле, ничего странного в этом нет: именно в таком формате обычно представляются вещественные числа (вспомните, что тип `real` подразумевает 11 значащих цифр – столько и отображается на экране). Мы же привыкли к другому их представлению. В данном случае хотелось бы вместо такой громоздкой записи числа увидеть привычную: `2.5`. К счастью, это возможно, - нужно просто использовать форматированный вывод данных.

Форматированный вывод определяет количество позиций на экране, отведенных под вывод переменной, а также количество знаков после запятой (для вещественных переменных). Эти значения записываются через двоеточие после имени переменной при ее выводе. При этом значение переменной сдвигается к правому краю отведенного под него поля.

Программа примера 3 демонстрирует возможности форматированного вывода.

Пример 3

```
{Программа демонстрации возможностей
 форматированного вывода
 значений переменных}
uses Crt;
var
   i, k : integer;
   s, t : real;
BEGIN
ClrScr;
i:=2;           { переменным присваиваются}
k:=778;         { некоторые значения}
s:=2.5e-1;
t:=4e2;
writeln(i:5);     { значения переменных отображаются}
writeln(k:15);    { в указанном формате}
writeln(s:5:2);
writeln(t:15:2);
END.
```

Вид экрана после выполнения программы примера 3 представлен на рис. 2.2. Действительно, как и было указано в программе, под переменную `i` на экране выделено поле размером 5 позиций, под `k` – 15 позиций, для представ-

ления вещественной переменной s выделено поле из 5 позиций, причем 2 из них – для знаков после запятой и т.д.

Поэкспериментируйте с форматированным выводом чисел. Например, замените операторы writeln на операторы write и посмотрите, что из этого получится. Или попробуйте указать другое число позиций для вывода.

```
    2
          778
 0.25
       400.00
_
```

Рис 2.2. Результаты выполнения программы примера 3

Теперь вспомним программу примера 2. При ее рассмотрении мы столкнулись с оператором присваивания и отметили, что правой его частью является выражение. Давайте разберемся с этим понятием.

Выражение – это правило, с помощью которого получается новое значение переменной. Выражение составляется из имен переменных и вызовов функций с помощью знаков операций над переменными и скобок.

Какие же операции над переменными можно выполнять? Это зависит от их типа. Для переменных типа real и integer, конечно же, существуют все четыре арифметические операции:

+ - сложение;
- - вычитание;
* - умножение;
/ - деление.

Обратите внимание на то, что в результате деления целых чисел получается вещественное число.

Кроме вышеперечисленных, существуют также специальные операции деления для данных целого типа: div и mod. Результат применения операции div представляет собой целую часть от деления. Например, при выполнения оператора

```
x:=5 div 2;
```

переменной x будет присвоено значение выражения 5 div 2: число 2.

Операция mod вычисляет остаток от деления. Таким образом, в результате выполнения оператора

```
x:=5 mod 2;
```

переменная x получит значение 1.

К сожалению, в Турбо-Паскале отсутствует операция возведения в степень. Хотя для вычисления квадрата числа есть функция sqr(x), а для вычисления квадратного корня – функция sqrt(x). При использовании этих функций само число указывается в качестве параметра – в скобках после имени функции. Например:

```
t:=sqr(5)-sqrt(16); { t=25-4=21 }
```

Заметьте, что результат функции `sqrt(x)` имеет вещественный тип.

В Турбо-Паскале существуют также другие арифметические функции. Их перечень приведен в таблице 2.3.

Применяя знания из области математики, можно легко получить недостающие арифметические функции. Например, для вычисления значения некоторого числа `a` в степени `x` нужно использовать выражение `exp(x*ln(a))`.

Таблица 2.3

Арифметические функции языка Турбо-Паскаль

Функция	Назначение	Тип результата
`abs(x)`	Абсолютное значение `x`	Совпадает с типом `x`
`arctan(x)`	Арктангенс `x`	Вещественный
`cos(x)`	Косинус `x`	Вещественный
`exp(x)`	`e` в степени `x`	Вещественный
`frac(x)`	Дробная часть числа `x`	Вещественный
`int(x)`	Целая часть числа `x`	Вещественный
`ln(x)`	Натуральный логарифм `x`	Вещественный
`pi`	Значение числа π	Вещественный
`sin(x)`	Синус `x`	Вещественный
`sqr(x)`	Квадрат числа `x`	Совпадает с типом `x`
`sqrt(x)`	Квадратный корень числа `x`	Вещественный

При работе с символьными и целочисленными переменными могут оказаться полезными следующие две процедуры:

`dec(x,[n])` – уменьшает содержимое переменной `x` на значение выражения `n` (`n` – необязательный параметр), если `n` не задано – содержимое переменной уменьшается на единицу;

`inc(x,[n])` – увеличивает содержимое переменной `x` на значение выражения `n`.

Например:

```
inc(s);     { значение увеличивается на 1 }
dec(s,6);   { значение уменьшается на 6}
```

А какие же существуют операции для работы со строками, спросите Вы. Для строк и символов определена одна единственная операция – ***сцепление***. Она обозначается символом +. Например, программа

```
var
   s : string;
BEGIN
s:='Турбо-'+'Паскаль';
writeln(s);
END.
```

напечатает строку

```
Турбо-Паскаль
```

Другие действия со строками можно выполнять, используя встроенные функции, которые будут подробно рассмотрены далее.

Кроме описанных выше операций существуют также операции отношения (сравнения):

`=` - равно;
`<>` - не равно;
`<` - меньше;
`>` - больше;
`<=` - меньше или равно;
`>=` - больше или равно.

Очевидно, что сравнивать можно лишь переменные одного типа. Результат применения операции отношения к переменным любого типа будет иметь логический тип (`boolean`).

Как сравниваются числа, Вам понятно, а вот как сравнить две строки? Какая же из них «больше»?

Сравнение двух строк происходит следующим образом. Символы строк сравниваются попарно друг с другом так, что первый символ первой строки сравнивается с первым символом второй строки, второй символ первой строки – со вторым символом второй строки и т.д. При сравнении символов на самом деле сравниваются их коды (см. приложение 2). Но, вообще-то, порядок следования кодов символов совпадает с алфавитным (если используются символы лишь одного алфавита). Если одна строка короче другой, то во время их сравнения недостающий символы заменяются нулями.

Например, справедливы следующие отношения:

```
'aaa'<'baa'
'aaa'<'aab'
'aa'<'aaa'
```

При сравнении данных типа `BOOLEAN` учитывается внутреннее соглашение Турбо-Паскаля, в соответствии с которым `false` есть нулевой байт, а `true` – единичный. Заметим, что функция `ord` преобразует к целому не только символы, но и логические величины, поэтому

```
ord(false)=0,
ord(true)=1.
```

Для логических данных в Турбо-Паскале определены следующие операции:

`not` – логическое НЕ;
`and` – логическое И;
`or` – логическое ИЛИ;
`xor` – ИСКЛЮЧАЮЩЕЕ ИЛИ.

Рассмотрим их подробнее.

Логическое выражение `not a` равно `true`, если `a` равно `false`, и наоборот, равно `false`, если `a` равно `true`.

Выражение `a and b` равно `true` в единственном случае, когда и `a`, и `b` равны `true`. Во всех остальных случаях результат этого выражения равен `false`.

Выражение `a or b` равно `false` только в случае, когда и `a`, и `b` равны `false`. Во всех остальных случаях результат этого выражения равен `true`.

Выражение `a xor b` равно `true` только тогда, когда один из операндов `a` или `b` равен `true`. Если значения операндов одинаковы, то результат этого выражения равен `false`.

Итак, в выражении может быть использовано множество различных переменных, знаков операций и функций. Порядок выполнения операций определяется их приоритетом (табл. 2.4). Чтобы изменить заданный порядок, нужно просто правильно расставить скобки.

Таблица 2.4

Приоритеты операций

Приоритет	Операции
1	`not`
2	`*, /, div, mod, and`
3	`+, −, or, xor`
4	`=, <>, <=, >=, <, >`

Например, Вам необходимо вычислить значение выражения

$$a = arctg(x) - \frac{3}{5}e^{xy} + \frac{|x+y|}{x^y + y}.$$

Для этого достаточно записать оператор присваивания вида

```
a:=arctan(x)-3/5*exp(x*y)+abs(x+y)/(exp(y*ln(x))+y);
```

Теперь полученное значение хранится в переменной `a` и Вы можете как угодно его использовать: например, можно вывести его на экран или подставить в другое выражение.

Как видите, все очень просто. Главное, не бойтесь пробовать, и тогда у Вас все получится.

До сих пор мы обсуждали лишь данные, изменяющиеся в процессе работы программы – переменные. Кроме них в языке Турбо-Паскаль существуют также ***константы*** – неизменные данные. Их использование ничем не отличается от использования переменных. Описываются константы с помощью зарезервированного слова `const`: после него указывается имя константы, знак равенства и ее значение. При этом тип константы не указывается, он определяется компилятором по ее записи. Например:

```
const
   a=16;            { константа целого типа }
   b=2.7;           { константа вещественного типа }
```

```
c='f';          { константа символьного типа }
st='Строка';    { константа строкового типа }
```

Еще раз отметим, что константы не могут изменяться в процессе работы программы. В частности, они никогда не должны стоять в левой части оператора присваивания.

Строгое описание переменных загоняет нас в рамки типов. А как же быть, если какое-то значение нужно использовать в качестве содержимого переменной, тип которой не совпадает с указанным?

На этот случай в Турбо-Паскале существует возможность преобразования данных. Для этого имеются встроенные функции, которые в качестве параметра получают значение одного типа, а возвращают – значение другого типа. В частности, для преобразования типа `real` в тип `integer` используются две функции:

`round` – округляет вещественное число до ближайшего целого;

`trunc` – отсекает дробную часть вещественного числа (положительное число округляет в меньшую сторону, отрицательное – в большую).

Например:

```
x:=round(4.7);  { x=5 }
x:=trunc(4.7);  { x=4 }
```

Для преобразования данных типа `char` в целое число предназначена функция `ord`. Обратное преобразование `integer` в `char` осуществляет функция `chr` (кроме того, для этой цели существует операция #). Например, с помощью следующей программы можно узнать внутренний код произвольного символа.

```
var
   ch : char;
BEGIN
write('Введите любой символ: ');
readln(ch);
writeln('Код символа ',ch,' равен ',ord(ch));
readln;
END.
```

Существуют также и другие функции преобразования типов. В дальнейшем Вы с ними еще встретитесь. А пока можете почитать об этом в каких-нибудь книжках.

Задания для самостоятельной работы

1. Измените в примере 3 значение переменной `s`, например, на 2352.3792, оставив формат вывода прежним. Посмотрите, что из этого получится.
2. Измените программу примера 2 так, чтобы результат от деления выводился в удобном виде.
3. Измените программу примера 2 так, чтобы результат от деления округлялся до ближайшего целого; до второго знака после запятой.
4. Напишите программу, которая запрашивает у пользователя радиус круга и вычисляет его площадь. Площадь круга вычисляется по формуле:

$$S = \pi \times r^2,$$

где r – радиус. Используйте функцию `Pi`. Выведите полученное значение в удобном виде.

3. ОПЕРАТОРЫ ЯЗЫКА ТУРБО-ПАСКАЛЬ

3.1. Операторы: простые и сложные

Вы уже не раз встречались с понятием ***оператор***. Именно так мы называем ту частичку программы, которая выполняет какое-то действие. Например, оператор присваивания заносит в определенную переменную некоторое значение, оператор `readln` ожидает ввода каких-либо данных и т.д. Но тех операторов, которые Вы знаете, еще не хватает, чтобы написать серьезную программу. Для этого нужно знать операторы и посложнее.

Итак, все операторы можно разделить на ***простые*** и ***сложные***.

С простыми операторами Вы уже работали. К ним относятся:

- оператор присваивания (`a:=b; x:=sin(y);` и пр.);
- вызов процедуры (`ClrScr;` и пр.);
- операторы ввода и вывода (`readln; write;` и пр.).

Сложные операторы Вы изучите в дальнейшем, к ним относятся:

- условный оператор;
- оператор выбора;
- операторы цикла.

Существует еще один замечательный оператор, который трудно назвать простым или сложным. Этот оператор называется ***составным***.

Составной оператор представляет собой последовательность любых операторов, заключенную в операторные скобки – зарезервированные слова `begin` и `end`. В составной оператор могут входить любые операторы языка Турбо-Паскаль, в том числе и другие составные операторы. Причем глубина их вложенности может быть любой:

```
begin
  write(x);
  . . .
  begin
    t:=x;
    . . .
    begin
      x:=x*t;
      . . .
      . . .
    end;
    . . .
  end;
  . . .
end;
```

И вот что интересно: на самом деле, весь раздел операторов в программе представляет собой один составной оператор.

Позднее Вы, конечно, оцените всю важность этого нелепого, на первый взгляд, оператора.

3.2. Условный оператор IF

Приступаем к изучению сложных операторов, самый первый из них – ***условный***.

Условный оператор позволяет проверить некоторое условие и в зависимости от результата проверки выполнить то или иное действие.

Структура условного оператора имеет следующий вид:

```
IF условие THEN
   оператор1
[ELSE
   оператор2]
```

где `IF`, `THEN`, `ELSE` – зарезервированные слова;

условие - любое логическое выражение;

оператор1, *оператор2* - любые операторы языка Турбо-Паскаль (операторы могут быть и составными).

Как происходит выполнение условного оператора?

В первую очередь осуществляется проверка истинности `условия`. Если *условие* истинно (равно `true`), выполняется *оператор1*, если же условие ложно (равно `false`) - *оператор2*. И в том и в другом случае выполняется только один из операторов *оператор1* и *оператор2*, другой же – просто игнорируется.

Как видите, часть `else` условного оператора может отсутствовать. Тогда, если условие истинно, то выполняется `оператор1`, в противном же случае весь оператор `if` пропускается.

Применение условного оператора проиллюстрируем на следующем примере.

Пусть значение *y* зависит от значения *x*. График зависимости приведен на рис. 3.1. Требуется по заданному *x* определить значение *y*.

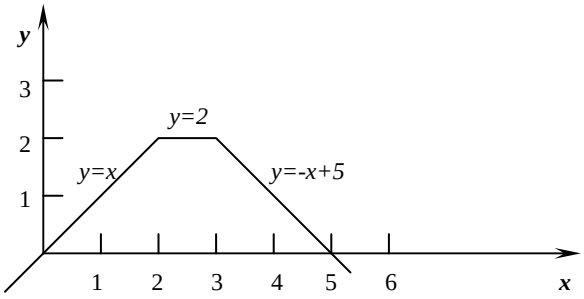

Рис. 3.1

Для того, чтобы по заданному *x* определить значение *y*, нужно выяснить, в пределах какого из трех интервалов лежит значение *x*, а затем уже подставлять его в нужное выражение.

Пример 4 представляет программу, которая вычисляет значение *y* по заданному значению *x*.

Пример 4

```
uses Crt;
var x, y : real;
BEGIN
ClrScr;
write('Введите x: ');
readln(x);
if x<2 then       {если x попадает в 1-й интервал}
   y:=x
else
   if x<3 then    {если x попадает во 2-й интервал}
      y:=2
   else           {если x не попадает ни в 1-й, ни во 2-й}
      y:=-x+5;
writeln('При x=',x:6:2,' y=',y:6:2);
END.
```

Обратите внимание на то, что точка с запятой перед `else` не ставится.

Поскольку любой из операторов `оператор1` и `оператор2` может быть условным, и, в то же время, не каждый из вложенных условных операторов может иметь часть `else`, оператор может быть неоднозначно истолкован. Подобная неоднозначность в Турбо-Паскале решена так: любая встретившаяся часть `else` соответствует ближайшему сверху `if...then`.

Давайте попробуем написать еще одну программу с применением условного оператора `IF`. Эта программа должна определять, попадает ли точка с заданными координатами (*x*,*y*) в кольцо, больший радиус которого равен *r1*, а меньший – *r2*. Центр кольца совпадает с началом координат.

В этой программе нам придется проверять два условия: необходимо, чтобы точка попала в больший круг и в то же время – не попала в меньший. Чтобы записать такое двойное условие нужно использовать логическую операцию `AND`. Все остальное в этой программе – достаточно просто (пример 5).

Пример 5

```
uses Crt;
var x, y, r1, r2 : real;
BEGIN
ClrScr;
write('Введите координату x: ');
readln(x);
write('Введите координату y: ');
readln(y);
write('Введите больший радиус: ');
readln(r1);
write('Введите меньший радиус: ');
```

```
readln(r2);
if (sqrt(x*x+y*y)<r1) AND (sqrt(x*x+y*y)>r2) then
   writeln('Точка (',x:5:2,';',y:5:2:,
        ') попадает в указанное кольцо');
else
writeln('Точка (',x:5:2,';',y:5:2:,
        ') не попадает в указанное кольцо');
END.
```

Задания для самостоятельной работы

1. Напишите программу, определяющую максимальное из трех заданных чисел.

2. Напишите программу, определяющую, попадает ли точка (x,y) хотя бы в один из двух кругов радиусом r с центрами в точках (x_1,y_1) и (x_2,y_2), соответственно. Используйте логическую операцию OR.

3.3. Оператор выбора CASE

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Параметром, по которому осуществляется выбор, служит ***ключ выбора*** – выражение любого типа, кроме вещественного и строкового.

Структура оператора выбора такова:

```
CASE ключ выбора OF
   константа1: оператор1;
   константа2: оператор2;
   . . . . . .
   константаN: операторN;
[ELSE
   оператор]
END
```

где CASE, OF, ELSE, END – зарезервированные слова;

ключ-выбора – ключ выбора (см. выше);

константа1, константа2, константаN – константы или перечни констант, соответствующие типу ключа выбора;

оператор1, *оператор2, операторN, оператор* - любые операторы языка Турбо-Паскаль (операторы могут быть и составными).

Оператор выбора работает так. Вначале вычисляется значение выражения *ключ-выбора*, а затем среди *констант* отыскивается равная вычисленному значению. *Оператор*, следующий за ней, выполняется, после чего оператор выбора завершается. Если в списке не обнаруживается константа, соответствующая вычисленному значению, то выполняется *оператор*, стоящий после слова else.

Применение оператора выбора рассмотрим на примере программы, реализующей простейшее меню (пример 6).

Пример 6

```
uses Crt;
var
   ch : char;
   x : real;
BEGIN
writeln('Введите вещественное число: ');
readln(x);
writeln('Укажите функцию, которую нужно вычислить:');
writeln('S – синус,  C - косинус ');
readln(ch);
case ch of
   'S','s': writeln('Синус равен ',sin(x):5:2);
   'C','c': writeln('Косинус равен ',cos(x):5:2);
else
 writeln('Буква указана неправильно');
end;
writeln('Нажмите Enter . . . ');
readln;
END.
```

Ту же самую программу можно было написать и с использованием оператора `IF`. Но в этом случае пришлось бы использовать вложенный `IF`, а также операцию `OR`. Как видите, оператор `CASE` позволяет избежать использования слишком сложных условных операторов.

Также как и в условном операторе, в операторе выбора часть `else` можно опускать. Тогда при отсутствии нужной константы в списке ничего не произойдет, и оператор просто завершит свою работу.

Задания для самостоятельной работы

1. Напишите программу, которая для заданного целого числа *k* печатает фразу «`Мы съели` *k* `яблок`», согласуя окончание слова «яблоко» с числом *k* (*k*<10).
2. Напишите программу-калькулятор, которая запрашивает два числа и знак операции (`+` , `-` , `*` , `/`), а затем выполняет указанное действие над введенными числами.

3.4. Операторы цикла

Часто одно и то же действие нужно повторить несколько раз. Для этого используются циклы.

В языке Турбо-Паскаль есть три различных оператора, с помощью которых можно организовать цикл.

Оператор FOR

Оператор FOR имеет следующую структуру:

```
FOR пар-цикла := нач-знач TO кон-знач DO
   оператор
```

где FOR, TO, DO – зарезервированные слова;

пар-цикла – параметр цикла, переменная типа integer;

нач-знач, *кон-знач* – начальное и конечное значения цикла, любые выражения целого типа;

оператор - любой оператор языка Турбо-Паскаль.

При выполнении оператора FOR вначале вычисляется выражение *нач-знач* и осуществляется присваивание *пар-цикла*:=*нач-знач*. После этого циклически повторяется:

- проверка условия *пар-цикла*<=*кон-знач*, если условие не выполняется, оператор FOR завершает свою работу;
- выполнение оператора *оператор*;
- увеличение переменной *пар-цикла* на единицу.

Этот оператор используется, когда какое-либо действие нужно повторить заданное число раз. Например, чтобы получить значение x^n нужно ровно n раз умножить число x само на себя:

```
. . .
x_n:=1;          {устанавливаем начальное значение}
for i:=1 to n do
  x_n:=x_n*n;
. . .
```

В качестве иллюстрации применения оператора FOR рассмотрим программу, осуществляющую ввод с клавиатуры произвольного целого числа n и вычисление суммы всех целых чисел от 1 до n (пример 7).

Пример 7

```
uses Crt;
var
   i, n, s : integer;
BEGIN
ClrScr;
write('Введите N: ');
readln(n);
s:=0;        {устанавливаем начальное значение}
for i:=1 to n do
   s:=s+i;
writeln('Сумма равна: ',s)
END.
```

Существует и другая форма оператора FOR:

```
FOR пар-цикла := нач-знач DOWNTO кон-знач DO
   оператор
```

Замена зарезервированного слова TO на DOWNTO означает, что переменная *пар-цикла* на каждом шаге уменьшается на единицу, а управляющее условие приобретает вид *пар-цикла*>=*кон-знач*.

В случае использования оператора FOR ... DOWNTO программа примера 6 примет следующий вид (пример 8).

Пример 8

```
uses Crt;
var
   i, n, s : integer;
BEGIN
ClrScr;
write('Введите N: ');
readln(n);
s:=0;
for i:=n downto 1 do
   s:=s+i;
writeln('Сумма равна: ',s)
END.
```

Обратите внимание на оператор

```
s:=s+i;
```

Переменной s присваивается сумма ее же значения и значения другой переменной. Такой прием называется ***накоплением суммы*** и является одним из основных приемов программирования. Существует также прием ***накопления произведения***:

```
p:=p*i;
```

Необходимо отметить, что перед использованием этих приемов соответствующим переменным должны быть присвоены начальные значения.

Оператор WHILE

Оператор WHILE называют также оператором цикла с предпроверкой условия. Он имеет следующую структуру:

```
WHILE условие DO
   оператор
```

где WHILE, DO – зарезервированные слова;

оператор - любой оператор языка Турбо-Паскаль.

Если *условие* истинно, то выполняется *оператор*, после чего проверка условия повторяется. Если условие ложно, оператор WHILE прекращает свою работу.

Оператор WHILE используется в тех случаях, когда указанное действие необходимо выполнять до тех пор, пока не выполнится некоторое условие.

Рассмотрим следующий пример.

Заданы числа a и b (a>1). Необходимо получить последовательность a, a^2, a^3 и т.д., ограниченную числом b. В этом случае вычисления будут продол-

жаться до тех пор, пока очередной член последовательности не превысит значение *b*.(пример 9).

Пример 9

```
uses Crt;
var
   a, b, c : real;
BEGIN
ClrScr;
write('Введите числа A и B:');
readln(a,b);
c:=a;
while c<b do
 begin
   writeln(c);
   c:=c*a;
 end;
END.
```

Обратите внимание на использование приема накопления произведения, описанного выше.

Оператор REPEAT

Оператор `REPEAT` часто называют оператором цикла с постпроверкой условия. Он имеет следующую структуру:

```
REPEAT
   оператор
UNTIL условие
```

где `REPEAT`, `UNTIL` – зарезервированные слова;

`оператор` - любой оператор языка Турбо-Паскаль (если используется составной оператор, то слова `begin` и `end` можно опустить).

`Оператор` выполняется хотя бы один раз, после чего вычисляется истинность выражения *`условие`*. Если условие ложно, оператор повторяется, в противном случае работа оператора `REPEAT` заканчивается.

Для иллюстрации применения оператора `REPEAT` приведем пример программы, которая считает сумму цифр заданного числа (пример 10).

Пример 10

```
uses Crt;
var
   n,s: integer;
BEGIN
ClrScr;
write('Введите целое число: ');
readln(n);
```

```
s:=0;
repeat
   s:=s+n mod 10;
   n:=n div 10;
until n=0;
writeln('Сумма цифр этого числа равна: ',s);
readln;
END.
```

В отличие от оператора WHILE, при использовании REPEAT указанное действие обязательно выполняется один раз, и только после этого проверяется условие.

Задания для самостоятельной работы

1. Напишите программу, выводящую на экран таблицу умножения для чисел от 1 до 9. Используйте форматированный вывод.
2. Напишите программу, выводящую цифры заданного целого числа в обратном порядке.

4. МАССИВЫ

Рассмотренные ранее простые типы данных позволяют использовать в программе одиночные объекты – числа, символы, строки и т.д. В Турбо-Паскале могут использоваться также объекты, содержащие множество однотипных элементов. Это ***массивы*** – объединение нескольких однотипных объектов, рассматриваемое как единое целое. К необходимости применения массивов мы приходим всякий раз, когда требуется связать и использовать целый ряд родственных величин. Например, результаты многократных замеров температуры воздуха в течение года удобно рассматривать как совокупность вещественных чисел, объединенных в массив.

При описании массива необходимо указать общее число входящих в него элементов и их тип. Например:

```
var
   a: array [1..10] of real; { 10 вещественных чисел }
   b: array [1..33] of char; { 33 символа (например, для
                               букв русского алфавита) }
   c: array [-12..3] of boolean; { 16 значений логического
                                   типа }
```

Как видно из примеров, при описании массива используются ключевые слова `array` и `of`. За словом `array` в квадратных скобках указывается диапазон индексов массива, определяющий число элементов массива. Диапазон задается левой и правой границами изменения индекса. За словом `of` указывается тип элементов, образующих массив.

Доступ к каждому элементу массива в программе осуществляется с помощью индекса – целого числа, которое представляет собой своеобразное имя элемента массива (если левая граница диапазона равна 1, то индекс элемента совпадает с его порядковым номером). Индекс указывается в квадратных скобках после имени массива. Например:

```
a[7]:=7.6;
b[1]:='a';
c[-6]:=true;
```

Во избежание ошибок во время выполнения программы, индекс элемента массива не должен выходить за границы, указанные при описании. Например, если массивы `a`,`b` и `c` описаны так, как приведено выше, то нельзя использовать элементы `a[14]`, `b[0]`, `c[-20]` и т.п.

Для иллюстрации приемов работы с массивами приведем программу (пример 11), которая создает массив случайных целых чисел, подсчитывает их среднее арифметическое, а также определяет и выводит на экран минимальное и максимальное из этих чисел.

Пример 11

```
var
   x : array [1..100] of integer;
   i : integer;
   max, min : integer;
   s : real;
```

```
BEGIN
{ Создание массива из случайных чисел }
randomize;
for i:=1 to 100 do
   x[i]:=random(101);

s:=0;
max:=x[1];
min:=x[1];

{ Вычисление суммы всех случайных чисел и поиск
  максимума и минимума }
for i:=1 to 100 do
 begin
   s:=s+x[i];
   if x[i]<min then
      min:=x[i]
   else
      if x[i]>max then
         max:=x[i]
 end;

{ Вычисление среднего и вывод результата }
writeln('Среднее арифметическое равно: ',s/100:4:1);
writeln('Минимальное значение равно: ',min:3);
writeln('Максимальное значение равно: ',max:3);

END.
```

В приведенном примере для создания массива случайных чисел использовалась встроенная функция `random(max)`, которая генерирует случайное число, равномерно распределенное в диапазоне от `0` до `max-1`. Функция `randomize` используется для начальной инициализации датчика случайных чисел.

Сортировка массива

Часто данные, хранящиеся в массиве необходимо расположить в порядке возрастания или убывания – отсортировать.

При сортировке представляется вполне естественным выбрать из всего массива минимальный элемент и поставить его на первое место. Затем – выбрать из всех оставшихся элементов минимальный и поставить его на второе место и т.д., пока весь массив не будет отсортирован. Такой метод сортировки называется ***методом выбора*** (в данном случае рассматривается сортировка по возрастанию).

Программа примера 12 реализует сортировку массива методом выбора.

Пример 12

```
uses Crt;
const
```

```
   n=10;
var
   x   : array [1..n] of integer;
   i,j : integer;
   t   : integer;
BEGIN
ClrScr;
{ Ввод массива }
writeln('Введите элементы массива');
for i:=1 to n do
 begin
   write(' Элемент ',i:2,': ');
   readln(x[i])
 end;

{ Сортировка методом выбора }
for i:=1 to n-1 do
   for j:=n downto i+1 do
      if x[i]>x[j] then
       begin
         t:=x[i];     { x[i] и x[j] меняются местами }
         x[i]:=x[j];
         x[j]:=t;
       end;

Write('Отсортированный массив:');
for i:=1 to n do
   write(x[i]:4);
END.
```

Кроме рассмотренного выше, существует множество других методов сортировки, различающихся быстродействием.

Приведем еще один метод – ***метод пузырька***. Этот метод заключается в следующем. Сначала сравниваются последний элемент с предпоследним (`x[n]` с `x[n-1]`). Если предпоследний больше, то они меняются местами. Далее сравниваются следующие два элемента массива (`x[n-1]` с `x[n-2]`), и так до тех пор, пока не будет достигнуто начало массива. Теперь самый “легкий” элемент находится на первом месте (всплывет, как пузырек). Снова выполняем все те же действия, но уже не до начала массива, а до его второго элемента. Затем – до третьего и т.д. – до предпоследнего элемента. Фрагмент программы, реализующей сортировку методом пузырька приведен ниже (пример 13).

Пример 13

```
. . .
{ Сортировка методом пузырька }
for i:=n downto 2 do
   for j:=1 to i-1 do
      if x[j]>x[j+1] then
       begin
```

```
            t:= x[j];
            x[j]:= x[j+1];
            x[j+1]:= t;
         end;
. . .
```

Строки

Переменные типа `string` на самом деле являются массивами символов. Таким образом, обращение к отдельному символу строки осуществляется так же, как и к элементу массива. Например:

```
s[7]:='g';
```

В языке Турбо-Паскаль существует ряд стандартных функций для работы со строками. Ниже приведены некоторые из них.

`Length(s)` – вычисляет текущую длину строки `s`;

`Copy(s,x,y)` – выделяет из строки `s` подстроку длиной `y`, начиная с `x`;

`Concat(s1,…,sN)` – соединяет строки `s1`, `s2`, … , `sN`;

`Pos(s1,s2)` – ищет подстроку `s1` в строке `s2`.

Работу со строками рассмотрим на следующем примере (пример 14).

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Вывести все слова, имеющие среднюю для заданной строки длину.

Пример 14

```
uses Crt;
var
   s   : string;
   i,j   : integer;
   l,n,len_sr : integer;
   word: boolean;
BEGIN
ClrScr;
{ Ввод строки }
writeln('Введите строку');
readln(s);

{ Определение средней длины слова в строке }
l:=0;  { суммарная длина слов в строке }
n:=0;  { количество слов в строке }
word:=false; { слово еще не началось }
for i:=1 to length(s) do
 begin
   if ord(s[i])<>32 then
    begin
      l:=l+1;
      word:=true; { слово началось }
    end
   else
```

```
      if word then
       begin
         word:=false;
         n:=n+1;
       end
 end;
if word then n:=n+1;
len_sr:=round(l/n);    { вычисление средней длины слова }

writeln;
writeln('Общее количество символов в словах: ',l:3);
writeln('Количество слов в строке:',n:3);
writeln('Средняя длина слова в строке:',len_sr:3);

l:=0;  { длина текущего слова }
n:=0;  { начало слова - номер элемента строки }
word:=false;  { слово еще не началось }
for i:=1 to length(s) do
 begin
   if ord(s[i])<>32 then
    begin
      l:=l+1;
      if not (word) then
       begin
         word:=true;    { слово началось }
         n:=i;
       end
    end
   else
      if word then
       begin
         word:=false;
         if l=len_sr then
         begin
            for j:=1 to l do
              write(s[n+j-1]);
            writeln;
          end;
         l:=0;
       end
 end;
if word then
   if l=len_sr then
    begin
      for j:=1 to l do
         write(s[n+j-1]);
      writeln;
    end;

readln;
END.
```

Матрицы

Допустим, нам нужно хранить в массиве координаты n точек на плоскости: (x_1,y_1), (x_2,y_2),…,(x_n,y_n). В этом случае мы можем использовать ***двумерный массив*** (возьмем количество точек равным 100):

```
var
   coord: array [1..100, 1..2] of integer;
```

Такой массив представляет собой матрицу следующего вида:

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \multicolumn{2}{c}{\dots} \\ x_n & y_n \end{bmatrix}.$$

Примеры обращения к элементам матрицы:

```
x:=coord[2,1];
coord[95,2]:=73;
```

Конечно, массивы могут быть и трехмерными, и четырехмерными и т.д. Но принципы работы с ними не отличаются от принципов работы с матрицами. Поэтому рассмотрим лишь работу с матрицами. Приведем программу для решения следующей задачи (пример 15).

Задана матрица целых чисел A размерности $n \times n$. Вычесть поэлементно последний столбец из всех столбцов, кроме последнего. Размерность матрицы и значения ее элементов ввести с клавиатуры.

Пример 15

```
uses Crt;
var
   x   : array [1..20,1..20] of integer;
   i,j : integer;
BEGIN
ClrScr;
write('Введите размерность матрицы: ');
readln(n);

{ Ввод матрицы }
writeln('Введите элементы матрицы');
for i:=1 to n do
 begin
   write(' Строка ',i:2,': ');
   for j:= 1 to n do
      read(x[i,j]);
   writeln
 end;

{ Вычитание последнего столбца }
for j:=1 to n-1 do
   for i:=1 to n do
```

```
      x[i,j]:=x[i,j]-x[i,n] ;

{ Вывод на экран полученной матрицы }
writeln('Полученная матрица:');
for i:=1 to n do
 begin
   for j:=1 to n do
      write(x[i,j]:4);
   writeln;
 end;
readln;
END.
```

Задания для самостоятельной работы

1. Напишите программу для решения следующей задачи. Дано натуральное число *N*, действительные числа X_1, ... , X_n. Выяснить, является ли последовательность X_1, ..., X_n упорядоченной по убыванию.

2. Перепишите программу примера 13 с использованием стандартных функций `Pos` и `Copy`.

3. Напишите программу для решения следующей задачи. Задан массив, содержащий координаты точек на плоскости. Найти пару точек, расстояние между которыми минимально. Вывести их координаты на экран.

5. ПРОЦЕДУРЫ И ФУНКЦИИ

Часто в различных местах программы требуется выполнить очень похожие либо абсолютно одинаковые действия. Например, если требуется задать с клавиатуры два (или более) однотипных массивов, то в программе будет дважды встречаться следующий фрагмент кода:

```
. . .
for i:=1 to N do    { N – количество элементов массива A }
  readln(A[i]);
. . .
```

При этом различными будут лишь имена массивов. В данном примере повторяющийся фрагмент состоит всего из двух строчек, но представьте себе, что вам нужно отсортировать эти массивы или ввести с клавиатуры несколько матриц. В таком случае размер повторяющегося кода будет заметно больше. Но можно сократить и размер программы, и время, затрачиваемое на ее создание. Для этого и существуют процедуры.

Процедурой в Турбо-Паскале называется фрагмент программы, имеющий собственное имя. При упоминании имени процедуры в тексте программы происходит ее вызов, – начинают выполняться входящие в нее операторы. После выполнения последнего из операторов управление возвращается обратно в основную программу, и выполняются операторы, следующие непосредственно за вызовом процедуры (рис. 5.1).

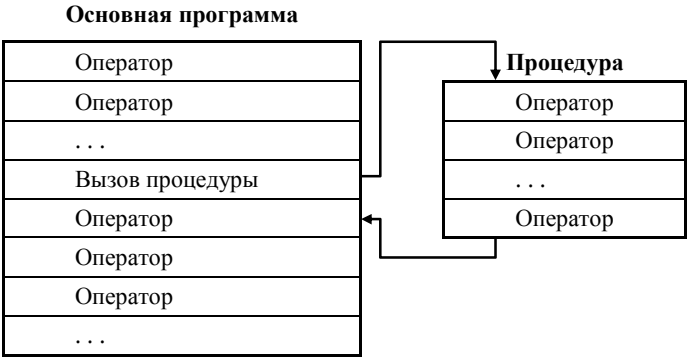

Рис 5.1. Взаимодействие основной программы и процедуры

Для обмена информацией между основной программой и процедурой используются ***параметры вызова***, которые перечисляются в круглых скобках за именем процедуры. В случае, когда для выполнения процедуры не нужно никакой дополнительной информации, параметры вызова отсутствуют.

Процедура – это маленькая программа. Она также имеет раздел описаний и раздел операторов, заключенный в операторные скобки `begin-end`. Поскольку в одной программе может быть множество процедур, заголовок про-

цедуры (в отличие от заголовка программы) является обязательным. Заголовок процедуры состоит из зарезервированного слова PROCEDURE, имени процедуры и следующими за ним в круглых скобках параметрами вызова. Например, процедура для ввода массива целых чисел может выглядеть следующим образом.

```
PROCEDURE InputIntArray(N:byte; var A:IntArray);
var
   i : byte;
BEGIN
for i:=1 to N do
  readln A[i];
END;
```

При этом в программе должен быть описан тип IntArray:

```
type IntArray=array [1..50] of integer;
```

Тогда можно вызвать процедуру для ввода двадцати значений массива D указанного типа:

```
InputIntArray(20, D);
```

Попробуем написать программу с использованием процедур. Эта программа осуществляет ввод с клавиатуры трех массивов целых чисел A, B и C, и вывод на экран того из них, сумма элементов которого минимальна (пример 16).

Пример 16

```
const
   N=10;
type
   IntArray=array [1..N] of integer;
var
   A,B,C: IntArray;          { Заданные массивы }
   SA,SB,SC: integer;        { Суммы элементов массивов }

{ Процедура ввода с клавиатуры массива целых чисел }
PROCEDURE InputArray(var Arr: IntArray);
var
   i: integer;
begin
   for i:=1 to N do
      readln(Arr[i]);
end;

{ Процедура подсчета суммы элементов массива целых чисел }
PROCEDURE SumArray(Arr: IntArray; var Sum: integer);
var
   i: integer;
begin
  Sum:=0;
  for i:=1 to N do
```

```
     Sum:=Sum+Arr[i];
end;

{ Процедура вывода на экран массива целых чисел }
PROCEDURE OutputArray(Arr: IntArray);
var
   i: integer;
begin
   for i:=1 to N do
      writeln(Arr[i]:6);
end;

BEGIN
InputArray(A);
InputArray(B);
InputArray(C);
SumArray(A,SA);
SumArray(B,SB);
SumArray(C,SC);
if SA<=SB and SA<=SC then
   OutputArray(A)
else
 if SB<=SA and SB<=SC then
    OutputArray(B)
 else
    OutputArray(C);
END.
```

Обратите внимание на различное описание параметров вызова в используемых процедурах: перед некоторыми из них ставится зарезервированное слово `var` – такие параметры называются ***параметрами-переменными***, а остальные – ***параметрами-значениями***. Параметры-значения не могут изменяться внутри процедуры, параметры-переменные – наоборот, используются специально для того, чтобы вернуть программе измененное процедурой значение. Например, в процедуре `InputArray` изменяются значения элементов массива. Для того, чтобы можно было использовать измененный массив далее в программе, нужно описать его как параметр-переменную. В процедуре `OutputArray` значения элементов массива не изменяются, поэтому он описывается как параметр-значение.

Параметры, указанные в описании заголовка процедуры, называются ***формальными***. Например, при описании процедуры `OutputArray` указан формальный параметр `Arr`.

Параметры, которые указываются при вызове процедуры из основной программы, называются ***фактическими***. Например, оператор

```
OutputArray(C);
```

вызывает процедуру вывода на экран для массива `C`. `C` – фактический параметр. При обращении к процедуре формальные параметры заменяются на соответствующие значения фактических параметров. В данном случае формаль-

ный параметр `Arr` процедуры `OutputArray` заменяется фактическим параметром `C`.

Как уже отмечалось, в каждой процедуре могут быть описаны собственные «внутренние» переменные. Эти переменные создаются при вызове процедуры и уничтожаются по окончании ее работы. Главной программе они недоступны. Такие переменные называются ***локальными***.

В процедуре могут также использоваться и переменные, описанные в главной программе – ***глобальные***.

Кроме понятия процедуры, в языке Турбо-Паскаль существует также понятие функции. ***Функция*** предназначена для вычисления какого-либо параметра. Она имеет два основных отличия от процедуры.

Первое отличие функции в ее заголовке. Он состоит из слова `FUNCTION`, за которым следует имя функции, далее в скобках – список формальных параметров, а затем через двоеточие записывается тип функции – тип возвращаемого параметра.

Второе отличие заключается в том, что внутри функции ее имени обязательно должно быть присвоено значение указанного типа.

В примере 16 вместо процедуры `SumArray` можно использовать такую функцию:

```
FUNCTION SumArray(Arr: IntArray) : integer;
var
   i: integer;
begin
  SumArray:=0;
  for i:=1 to N do
     SumArray:=SumArray+Arr[i];
end;
```

В этом случае основная часть программы изменится так:

```
BEGIN
InputArray(A);
InputArray(B);
InputArray(C);
SA:=SumArray(A);
SB:=SumArray(B);
SC:=SumArray(C);
if SA<SB and SA<=SC then
   OutputArray(A)
else
 if SB<=SA and SB<=SC then
    OutputArray(B)
 else
    OutputArray(C);
END.
```

Задания для самостоятельной работы

1. Напишите программу для отыскания минимального и максимального элемента массива `IntArray`. Используйте функции.
2. Напишите и используйте процедуру пузырьковой сортировки массива.

6. МОДУЛИ

6.1. Создание модулей

Часто одну и ту же процедуру нужно использовать в различных программах. Например, Вам нужно написать несколько программ, каждая из которых выполняет какие-либо операции с мссивами целых чисел. Неужели в каждой программе нужно будет писать заново процедуры ввода и вывода массива или процедуру его сортировки? Можно, конечно, так и поступить, но лучше просто объединить эти процедуры в отдельный файл, оформить его как модуль, и подключать, если необходимо. Вспомните, как во многих программах Вы подключали модуль `Crt`, когда нужно было вызвать процедуру очистки экрана.

Итак, Вам требуется создать свой собственный модуль. Для этого нужно знать, как же он должен быть устроен. Давайте разберемся со всем по порядку.

Модуль состоит из следующих частей:

- заголовок модуля;
- интерфейсная часть;
- исполняемая часть;
- инициирующая часть.

Причем первые три части обязательны, а инициирующая часть может и отсутствовать.

Заголовок модуля состоит из зарезервированного слова `UNIT` и имени модуля, например:

```
UNIT MyModule;
```

Имя модуля должно совпадать с именем файла, в который Вы его помещаете. Т.е. в данном случае исходный текст модуля размещается в файле с именем `mymodule.pas`.

Интерфейсная часть модуля открывается зарезервированным словом `INTERFACE`. Далее указываются константы, типы, переменные, процедуры и функции, которые могут быть использованы любой программой при вызове этого модуля. В этой части и происходит взаимодействие модуля с другими программами. Например, если интерфейсная часть Вашего модуля выглядит следующим образом:

```
UNIT MyModule;

INTERFACE
const Number=100;
type IntArray=array[1..Number] of integer;

procedure InputIntArray(var A:IntArray);
procedure SortIntArray(var A:IntArray);
function SumIntArray(A:IntArray):longint;
```

то в любой программе достаточно лишь написать в самом начале

```
uses MyModule;
```

и можно будет дальше использовать константу Number, тип IntArray, процедуры ввода и сортировки массива InputIntArray и SortIntArray и функцию SumIntArray, не утруждая себя их описанием.

С константами, переменными и типами все понятно, а вот где находятся сами процедуры и функции, спросите Вы. Сами процедуры и функции, а также многое другое находятся в исполняемой части модуля.

Исполняемая часть начинается зарезервированным словом IMPLEMENTATION и содержит описание всех процедур и функций, объявленных в интерфейсной части. Кроме того, здесь же могут объявляться локальные для модуля константы, типы и переменные. А еще здесь могут быть различные вспомогательные процедуры и функции, не объявленные в интерфейсе.

Инициирующая часть завершает модуль. Она может отсутствовать вместе с начинающим ее словом BEGIN, или быть пустой – тогда за BEGIN сразу следует слово END с точкой. В инициирующей части содержится некоторый фрагмент программы. Он выполняется до передачи управления основной программе и обычно используется для подготовки ее работы (например, именно в этой части переменным присваиваются начальные значения).

Посмотрим, как выглядит наш модуль для работы с массивами (пример 17).

Пример 17

```
UNIT MyModule;

INTERFACE
{ Интерфейсная часть }
const Number=100;
type IntArray=array[1..Number] of integer;

procedure InputIntArray(var A:IntArray);
procedure SortIntArray(var A:IntArray);
function SumIntArray(A:IntArray):longint;

IMPLEMENTATION
{ Исполняемая часть }
{ Процедура ввода массива }
PROCEDURE InputIntArray(var A: IntArray);
var
   i: byte;
begin
   for i:=1 to Number do
      readln(A[i]);
end;

{ Процедура сортировки массива методом пузырька }
PROCEDURE SortIntArray;
var
   i,j: byte;
```

```
   t:integer;
begin
  for i:=Number downto 2 do
     for j:=1 to i-1 do
        if A[j]>A[j+1] then
         begin
            t:= A[j];
            A[j]:= A[j+1];
            A[j+1]:= t;
         end;
end;

{ Функция подсчета суммы элементов массива }
FUNCTION SumIntArray(A: IntArray):longint;
var
   i: byte;
begin
  SumIntArray:=0;
  for i:=1 to Number do
     SumIntArray:=SumIntArray+A[i];
end;

{ Инициирующая часть отсутствует }
END.
```

Обратите внимание на описание процедуры `SortIntArray` в исполняемой части – параметры вызова здесь опущены. Так можно было поступить со всеми процедурами и функциями, объявленными в интерфейсной части, ведь там уже все указано. Поэтому теперь заголовок процедуры (функции) можно либо сократить, либо описать только точно так же.

После компиляции этого модуля Вы получите файл `mymodule.tpu`. Вот теперь Ваш модуль готов к употреблению. Можно смело подключать его к любой программе также как и любые другие, используя предложение `USES`:

```
USES MyModule;
```

Конечно, Вы можете расширить свой модуль, добавив туда необходимые процедуры и функции, только не забудьте его перекомпилировать.

Задания для самостоятельной работы

1. Добавьте в Ваш модуль процедуру вывода на экран массива `IntArray`.
2. Напишите собственный модуль для работы с матрицами. Он должен включать в себя ввод матрицы и вывод ее на экран, сложение и умножение матриц, транспонирование матрицы.

6.2. Стандартные модули

В Турбо-Паскале есть восемь стандартных модулей: System, Dos, Crt, Printer, Graph, Overlay, Turbo3, Graph3. В этих модулях содержится множество типов, констант, процедур и функций, использование которых позволяет облегчить создание разнообразных программ.

Модули Graph, Turbo3 и Graph3 находятся в одноименных tpu-файлах, а все остальные содержатся в библиотечном файле turbo.tpl. Модуль System подключается к любой программе автоматически. Для подключения любого другого модуля необходимо указать его имя в предложении uses.

Полное описание стандартных модулей можно найти в специальной литературе. Здесь же будут рассмотрены лишь некоторые из них.

6.2.1. Модуль System

Модуль System содержит все процедуры и функции стандартного Паскаля, а также встроенные процедуры и функции Турбо-Паскаля, которые не вошли в другие стандартные модули. Поскольку этот модуль подключается автоматически, его константы, переменные, процедуры и функции можно считать встроенными в Турбо-Паскаль.

6.2.2. Модуль Crt

Модуль Crt используется для работы с экраном в текстовом режиме.

Текстовый режим характеризуется количеством строк на экране и количеством символов в каждой из них. По умолчанию устанавливается режим 25×80 (25 строк по 80 символов), при этом и строки, и символы нумеруются с единицы (рис. 6.1).

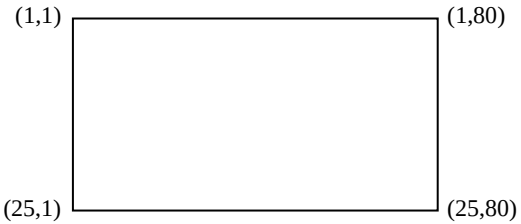

Рис. 6.1.

При выводе на экран каждого символа можно задать его цвет, яркость и цвет фона. Модуль Crt содержит следующие процедуры, позволяющие управлять этими параметрами:

LowVideo – устанавливает режим минимальной яркости выводимых на экран символов;

NormVideo – устанавливает режим нормальной яркости выводимых на экран символов;

`HighVideo` – устанавливает режим максимальной яркости выводимых на экран символов;

`TextColor(c)` – устанавливает цвет символов `c`, все возможные значения приведены в табл. 6.1;

`TextBackGround(c)` – устанавливает цвет фона `c`, который может принимать значения от 0 до 7 (табл. 6.1).

Таблица 6.1

Цветовые константы

Цвет	Константа	Значение
Черный	`Black`	0
Синий	`Blue`	1
Зеленый	`Green`	2
Голубой	`Cyan`	3
Красный	`Red`	4
Фиолетовый	`Magenta`	5
Коричневый	`Brown`	6
Светло-серый	`LightGray`	7
Темно-серый	`DarkGray`	8
Ярко-синий	`LightBlue`	9
Ярко-зеленый	`LightGreen`	10
Ярко-голубой	`LightCyan`	11
Розовый	`LightRed`	12
Малиновый	`LightMagenta`	13
Желтый	`Yellow`	14
Белый	`White`	15
Мерцание символа	`Blink`	128

Каждый следующий символ выводится с заданными параметрами до тех пор, пока они не будут изменены повторным применением описанных процедур.

Для того чтобы вывести следующий символ не прямо за предыдущим (как обычно и происходит), а в любой указанной позиции экрана, необходимо переместить курсор в эту позицию. Для этого используется процедура `GotoXY`:

`GotoXY(x,y)` – перемещает курсор в позицию с координатами (`x`,`y`).

Как Вы уже знаете, очистку экрана осуществляет функция `ClrScr`.

`ClrScr` – выполняет очистку экрана в текстовом режиме и устанавливает курсор в левый верхний угол.

Кроме того, в модуле `Crt` содержатся процедуры и функции для управления чтением клавиатуры и звуковым генератором.

`KeyPressed` – функция, которая возвращает значение `true`, если была нажата какая-нибудь клавиша, и `false` в противном случае;

ReadKey – функция, которая считывает код символа с клавиатуры;

Delay(t) – вызывает задержку выполнения программы на заданное число миллисекунд t;

Sound(f) – включает звук с частотой f;

NoSound – выключает звук.

Работа со звуком происходит по схеме Sound-Delay-NoSound.

В модуле Crt также находятся некоторые служебные переменные, в которых содержится информация, необходимая для управления выводом на экран. Например, для нас представляет интерес переменная TextAttr, которая содержит цветовые атрибуты текста – цвет фона и цвет символов:

TextAttr:=<цвет-фона>*16+<цвет символа>

Программа примера 18 демонстрирует реализацию меню средствами модуля Crt.

Реализация меню подразумевает следующие этапы:

- отображение меню на экране;
- установка одного из пунктов в активное состояние;
- ожидание нажатия клавиш;
- анализ нажатой клавиши (если стрелки – изменение активного пункта, если Enter – выполнение активного пункта).

Пример 18

```
{ Программа реализует работу меню }
uses Crt;
const               { описание названий пунктов меню }
  Items : array [0..2] of string =
          (' Пункт 1 ',' Пункт 2 ',' Пункт 3 ');
var
  xpos,ypos,curPos,i: integer;
  Key: char;
  NormCol,SelCol,StatCol:byte;
BEGIN
ClrScr;             { установка начальных значений }
NormCol:=$71;       { значения переменных указаны в }
SelCol:=$8F;        { шестнадцатеричной системе счисления}
StatCol:=$0F;
xpos:=10;
ypos:=10;
TextAttr:=NormCol;
for i:=0 to 2 do    { вывод списка меню }
 begin
   GotoXY(xpos,ypos+i);
   write(Items[i]);
 end;
TextAttr:=SelCol;   { выделение 1-го пункта меню }
GotoXY(xpos,ypos);
write(Items[1]);

repeat              { реакция на нажатие клавиш }
```

```
    Key:=readkey;
    if Key=chr(0) then
       key:=readkey;
    TextAttr:=StatCol;
    GotoXY(2,24);
    ClrEol;
    TextAttr:=NormCol;
    GotoXY(xpos,ypos+CurPos);
    write(Items[CurPos]);
    case ord(key) of
       72:if CurPos>0 then  { изменение номера текущего }
             dec(curPos);   {  пункта меню }
       80:if CurPos<2 then
             inc(curPos);
       13:begin               { выбор пункта меню }
            TextAttr:=StatCol;
            GotoXY(2,24);
            Sound(300);
            Delay(100);
            NoSound;
            write('Выбран ',Items[CurPos]);
          end
    end;
    TextAttr:=SelCol;
    GotoXY(xpos,ypos+CurPos);
    write(Items[CurPos]);
until key=#27;
TextAttr:=$0F;      { установка первоначальных параметров }
ClrScr;
END.
```

6.2.3. Модуль Graph

Наверно, Вы знаете, что компьютер часто используется для создания каких-нибудь картинок или мультипликации. Но каким же образом они создаются? В стандартном языке Турбо-Паскаль для этого нет никаких процедур и функций. Но, на наше счастье, существует модуль `Graph`. С его-то помощью мы и попытаемся создать какие-нибудь примитивные картинки.

Вы помните, что вся информация, которая выводилась не экран в рассмотренных ранее программах, была представлена в символьном виде. Такой режим работы дисплея называется текстовым. Для создания графических изображений используется графический режим. В графическом режиме экран дисплея представляет собой прямоугольную сетку, состоящую из множества маленьких элементов – пикселов. ***Пиксел*** – это точка, имеющая свои координаты и свой цвет. Из таких точек и складывается изображение.

Работу графического режима поддерживает специальная программа – драйвер. Эта программа находится в файле с расширением `.bgi` (в нашем случае – в файле `egavga.bgi`).

При переходе в графический режим необходимо проинициализировать его - загрузить драйвер. Для этого нужно выполнить процедуру `InitGraph`. Для перехода из графического режим а текстовый используется процедура `CloseGraph`.

Шаблон программы для работы в графическом режиме выглядит следующим образом:

```
uses Graph;
var
    Gd, Gm : integer;
BEGIN
Gd:=Detect;
InitGraph(Gd, Gm, '');

. . .  { в этом месте выполняются любые
         графические процедуры и функции}
CloseGraph;
END.
```

Оператор присваивания

```
Gd:=Detect;
```

означает, что в данном случае используется процедура автоматического обнаружения графической аппаратуры и загрузки драйвера.

В строке

```
InitGraph(Gd, Gm, '');
```

осуществляется инициализация графического режима. При этом в кавычках указывается путь к файлу `egavga.bgi`. В данном случае, драйвер находится в том же каталоге, что и сама программа, поэтому путь не указан.

В результате инициализации будет выбран режим, в котором на экране по горизонтали помещается 640 пикселов, а по вертикали – 480. При этом верхний левый угол экрана имеет координаты (0,0), а нижний правый – координаты (639, 479) (рис. 6.2).

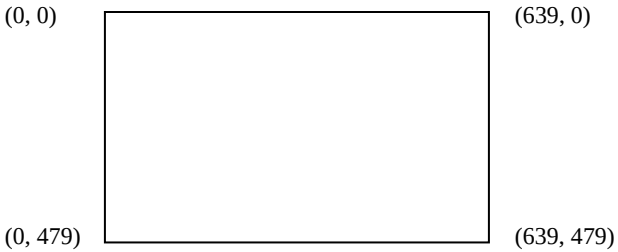

Рис 6.2. Координатная сетка экрана.

В этом режиме возможно использование шестнадцати цветов. Каждому цвету соответствует константа целого типа, которые совпадают с константами, приведенными в табл. 6.1.

Количество пикселов на экране может отличаться от указанного. Для определения размерности экрана используются функции GetMaxX и GetMaxY. Эти функции возвращают, соответственно, максимальную координату по горизонтали и максимальную координату по вертикали. В приведенном фрагменте программы переменным x и y присваиваются координаты центра экрана:

```
uses Graph;
var
   x, y : integer;
. . .
BEGIN
. . .
x:=GetMaxX div 2;
y:=GetMaxY div 2;
. . .
END.
```

Рассмотрим кратко основные процедуры и функции модуля Graph.

ClerDevice – очищает экран и устанавливает начальные значения графических праметров;

SetColor (c) – устанавливает цвет выводимого изображения, c – номер заданного цвета (см. табл. 2.1);

SetBkColor (c) – устанавливает цвет фона, c – номер заданного цвета;

PutPixel(x,y,c) – закрашивает пиксел с координатами (x,y) цветом с номером c;

MoveTo(x,y) – перемещает текущий указатель в точку с координатами (x,y);

LineTo(x,y) – проводит прямую из точки, где находится указатель в точку с координатами (x,y) текущим цветом. Текущий указатель перемещается в точку (x,y);

Line(x1,y1,x2,y2) – проводит прямую из точки (x1,y1) в точку (x2,y2) текущим цветом. Положение текущего казателя не изменяется;

Rectangle(x1,y1,x2,y2) – рисует прямоугольник с координатами (x1,y1) – верхний левый угол, и (x2,y2) – нижний правый угол;

Bar(x1,y1,x2,y2) – рисует закрашенный прямоугольник с координатами (x1,y1) – верхний левый угол, и (x2,y2) – нижний правый угол, используя установленный цвет закраски;

Circle(x,y,r) – рисует окружность с центром в точке (x,y) радиусом r текущим цветом;

OutText(text) – выдает на экран строку text, начиная с текущей позиции указателя. Текущий указатель при этом перемещается в конец строки;

OutTextXY(x,y,text) - выдает на экран строку text, начиная с точки (x,y). Положение текущего указателя не изменяется.

Модуль Graph содержит также некоторые другие процедуры и функции, описание которых здесь не рассматривается. Для их изучения рекомендуется обратиться к специальной литературе.

Приведенный ниже пример иллюстрирует использование основных процедур и функций модуля Graph.

Пример 19

```
uses Graph, Crt;
var
   Gd,Gm   : integer;
   x,y,r,c : integer;
   i       : integer;

BEGIN
{ Инициализация графического режима }
Gd:=Detect;
InitGraph(Gd,Gm,'c:\lang\bp\bgi\');
if GraphResult <> grOk then Halt(1);

ClearDevice;         { очистка экрана }
SetBkColor(15);      { установка цвета фона }
SetColor(14);        { установка цвета изображения }
x:=GetMaxX div 2;    { определение координат центра экрана }
y:=GetMaxY div 2;

{ Рисование разноцветных окружностей }
c:=0;
r:=1;
repeat
   SetColor(c);      { Установка нового цвета }
   if c>13 then      { Вычисление следующего цвета }
      c:=0
    else
      c:=c+1;
   Circle(x,y,r);        { Рисование окружности }
   Delay(100);           { Задержка на 0.1с }
   r:=r+2;               { Вычисление следующего радиуса }
until r>y-5;

{ Стирание окружностей }
r:=1;
repeat
   SetColor(15);
   Circle(x,y,r);
   Delay(100);
   r:=r+2;
until r>y-5;

ClearDevice;
SetColor(3);
```

```
for i:=1 to 2 do          { Рисование двух прямоугольников }
   Rectangle(i*80,i*80,GetMaxX-i*80,GetMaxY-i*80);
Delay(5000);              { Задержка на 5с }
SetBkColor(0);            { Изменение цвета фона }
Delay(5000);
SetColor(14);             { Изменение цвета изображения }
{ Вывод на экран текста }
OutTextXY(220,200,'Нажмите на любую клавишу...');

repeat until keypressed; { Ожидание }
CloseGraph;

END.
```

Задания для самостоятельной работы

1. Измените программу примера 18 так, чтобы она выводила меню в самой верхней строке экрана.

2. Напишите программу, которая имитирует звездное небо (рисует точки, выбирая их координаты случайным образом).

3. Напишите программу, рисующую линейный график по заданному массиву целых положительных чисел. По оси x – номер числа, по оси y – само число.

7. ФАЙЛЫ

В языке Турбо-Паскаль ввод и вывод информации осуществляется через ***файловые переменные***. Перед осуществлением ввода-вывода, файловая переменная должна быть связана с конкретным файлом на диске с помощью процедуры `Assign`. Затем файл должен быть открыт для чтения и/или записи. Только после этого можно осуществлять ввод-вывод. По окончании работы с файлом его необходимо закрыть процедурой `Close`.

Каждый файл состоит из последовательности некоторых компонент, по которым перемещается ***указатель***. Компонента, на которой стоит указатель, является текущей.

В Турбо-Паскале есть три вида файлов: типизированные (`file of` *`имя_типа`*), нетипизированные (`file`) и текстовые (`text`). Работа с ними осуществляется по разному, но существуют некоторые общие процедуры и функции. Приведем некоторые из них, остальные – можно посмотреть в дополнительной литературе.

`Assign(f,Name)` – осуществляет связь файловой переменной `f` с внешним файлом, имеющим имя `Name` (переменная типа `string`, содержащая путь MS-DOS к файлу).

`Reset(f)` – открытие существующего файла, связанного ранее с переменной `f`; указатель ставится на начало файла (на компоненту с номером 0).

`Rewrite(f)` – открытие нового файла, связанного с переменной `f`; если файл с таким именем уже существует, то он уничтожается; указатель ставится на начало файла (на компоненту с номером 0).

`Close(f)` – закрытие файла с которым связана переменная `f`; в случае необходимости в содержимое файла вносятся все произведенные изменения.

`Eof(f)` – функция, возвращающая значение `true`, если текущий указатель находится за последней компонентой файла, и `false` – в противном случае.

Типизированный файл – это тип, который подразумевает файл, содержащий последовательность значений указанного базового типа. Например:

```
type
   IntFile = file of integer;
var
   f_in, f_out : IntFile;
```

Описанный в данном случае тип `IntFile` можно использовать для связи с файлами на диске, содержащими последовательность целочисленных данных.

Для работы с типизированными файлами используются следующие процедуры и функции.

Процедуры:

`Read(f,<список ввода>)` – считывает из файла, связанного с переменной `f`, значения для одной или нескольких переменных, указанных в <списке ввода> (вспомните стандартные процедуры `read` и `readln`);

`Write(f,<список вывода>)` – записывает в файл, связанный с переменной `f`, значения выражений, указанных в <списке вывода> (аналогично стандартной процедуре `write`);

`Seek(f,n)` – ищет компоненту с номером `n` файла, связанного с переменной `f`, и устанавливает на нее текущий указатель;

`Truncate(f)` – удаляет часть файла, начиная с текущего указателя и до его конца.

Функции:

`FileSize(f)` – возвращает текущий размер файла (количество его компонент);

`FilePos(f)` – возвращает положение указателя – номер текущей компоненты.

Работу с типизированными файлами проиллюстрируем на примере создания файла, содержащего случайные числа в диапазоне от 0 до 1 и вывода его содержимого на экран (пример 20).

Пример 20

```
uses Crt;
type
   RealFile=file of real;
var
  f : RealFile;

PROCEDURE OutFile(f:RealFile);
var
  x:real;
begin
while not eof(f) do {пока не достигнут конец файла}
 begin
   read(f,x);       {чтение из файла очередной компоненты}
   writeln(x:4:2);  {вывод ее на экран}
 end;
end;

PROCEDURE FillFile(f:RealFile);
var
  i:byte;
begin
randomize;
for i:=1 to 20 do
   write(f,random); {запись в файл очередного случайного
                     числа}
end;

BEGIN
ClrScr;
Assign(f,'demo.dat');
Rewrite(f);
```

```
FillFile(f);
OutFile(f);
Close(f);
END.
```

Рассмотрим еще один пример (пример 21). В файле `matrix.dat` содержится квадратная матрица целых чисел. Требуется первый элемент каждой строки заменить на номер строки. Но сначала давайте разберемся, каким образом матрица хранится в файле. Запись матрицы в файл типа `file of integer` происходит построчно (рис. 7.1).

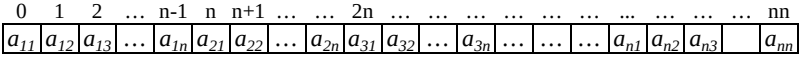

Рис. 7.1.

Из приведенного рисунка видно, что номер компоненты файла, содержащей элемент матрицы a_{ij}, равен `n*(i-1)+j-1`. Так что первый элемент каждой строки можно найти по формуле `n*(i-1)`.

Пример 21

```
type
   IntFile = file of integer;
var
   f : IntFile;
   a, b : integer;
n,i : word;
BEGIN
Assign(f,'matrix.dat');
Reset(f);
n:=round(sqrt(filesize(f)));
for i:=1 to n do
 begin
   seek(f,n*(i-1));
   write(f,i);
 end;
close(f);
END.
```

Текстовый файл – это совокупность символов, разделенных на строки, причем в конце каждой строки стоит символ конца строки. Особенность текстовых файлов заключается в том, что параметры, значения которых вводятся и выводятся с помощью процедур `Read` и `Write`, могут быть не только типа `char` или `string`, но и других простых типов.

Существует две стандартные файловые переменные для текстового файла: `Input` и `Output`. Переменная `Input` по умолчанию связана с клавиатурой, а `Output` – с экраном дисплея. Если при работе с файлами типа `Text` в проце-

дуре или функции не указана файловая переменная, считается, что используется файловая переменная Input или Output.

Файл типа Text может быть открыт либо для чтения (процедурой Reset), либо для записи (процедурой Rewrite или Append).

Некоторые процедуры и функции для работы с текстовыми файлами приведены ниже.

Процедуры:

Append(f) – открывает файл , связанный с переменной f, для добавления информации в его конец (указатель находится в конце файла);

Read(f,<список ввода>) – считывает из файла, связанного с переменной f, значения для одной или нескольких переменных, указанных в <списке ввода>;

Readln(f,<список ввода>) – считывает строку из файла; тоже, что и read, но непрочитанная часть строки опускается;

Write(f,<список вывода>) – записывает в файл, связанный с переменной f, значения выражений, указанных в <списке вывода>;

Writeln(f,<список вывода>) – записывает строку в файл; то же, что и write, но выводимая информация завершается признаком конца строки.

Функции:

Eoln(f) – конец строки файла; принимает значение true, если указатель стоит на признаке конца строки или если функция Eof(f) принимает значение true;

SeekEof(f) – конец файла; отличается от Eof тем, что стоящие в конце файла символы пробела и табуляции пропускаются;

SeekEoln(f) – конец строки файла; отличается от Eoln тем, что стоящие в конце строки символы пробела и табуляции пропускаются.

Пример работы с текстовым файлом (пример 22). В текстовом файле 'data.txt' содержатся вещественные числа (не больше тысячи). Необходимо считать их в массив для дальнейшей обработки.

Пример 22

```
var
   f : Text;
   m : array [1..1000] of real;
   i : word;
BEGIN
assign(f,'data.txt');
reset(f);
i:=1;
while not SeekEof(f) do
 begin
   read(f,m[i]);
   i:=i+1;
 end;
close(f);
END.
```

Задания для самостоятельной работы

1. Напишите программу, которая значения, содержащиеся в файле вещественных чисел, переписывает в текстовый файл по пять чисел в строку в удобном виде.

2. В файле `matrix.dat` содержится квадратная матрица целых чисел. Требуется заменить все отрицательные числа в этой матрице, находящиеся в предпоследнем столбце, на их абсолютное значение.

8. ЗАПИСИ

Под ***записью*** понимается структура данных, объединяющая под одним именем данные различных типов. Запись состоит из фиксированного числа элементов, называемых ***полями***. Поле – это переменная определенного типа. Различные поля могут быть разных типов. При описании типа-записи после зарезервированного слова `record` следует перечислить все поля с указанием их типов. Заканчивается описание записи словом `end`. Например:

```
type
  Person=record
    FIO : string[40];
    Year : word;
    Adress: string[40];
  end;
```

В данном случае тип-запись `Person` предназначен для хранения сведений о некотором человеке – его фамилии, имени, отчества, года рождения и адреса.

Доступ к конкретному полю записи осуществляется по имени переменной и имени поля, записанноым через точку:

```
var
   person1, person2 : Person;
. . .
person1.Year:=1983;
person2.Year:=person1.Year;
. . .
```

Для того, чтобы не писать каждый раз имя записи при обращении к ее полям, можно использовать оператор над записями `WITH`. Его структура выглядит следующим образом:

```
WITH имя_записи DO
   оператор
```

где `WITH`, `DO` – зарезервированные слова;

`оператор` - любой оператор языка Турбо-Паскаль.

В этом случае внутри оператора можно указывать только поле записи. Например:

```
with person1 do
 begin
   FIO:='Петров Алексей Леонидович';
   Year:=1984;
  Adress:='г.Томск, пр. Ленина, 1-43';
 end;
```

Рассмотрим пример работы с записями. В некотором файле `tovar.dat` в виде записей содержится информация о товарах: наименование, цена, единица измерения, количество проданного товара за текущий день. Нужно написать программу, определяющую общую сумму выручки за день (пример 23).

Пример 23

```
uses Crt;
type
   Tovar = record
     Name : string[20];
     Price : real;
     Measure : string[5];
     Amount : word;
   end;
   TovarFile = file of Tovar;
var
   t:Tovar;
   f:TovarFile;
   S:real;
BEGIN
S:=0;
Assign(f,'tovar.dat');
Reset(f);
while not Eof(f) do
 begin
   read(f,t);            {чтение из файла очередной записи}
   with t do
      S:=S+Price*Amount; {накопление суммы}
 end;
Close(f);
ClrScr;
Writeln('Сумма продаж за день составила ',S:8:2,' руб');
Readln;
END.
```

Задания для самостоятельной работы

1. Напишите программу для ввода значений в файл `tovar.dat` и программу, которая исправляет этот файл, уменьшая все цены на 25 процентов.
2. Напишите программу для вывода на экран из файла, содержащего записи типа `Person`, фамилий всех людей, родившихся позже 1980 года.

9. ДИНАМИЧЕСКОЕ РАЗМЕЩЕНИЕ ДАННЫХ

9.1. Динамическая память. Указатели

Во время работы программы все переменные, объявленные в ней, размещаются в статической области памяти, которая называется ***сегментом данных***. Еще на этапе компиляции в сегменте данных под каждую переменную отводится определенное место. Например:

```
type RealMas=array [1..20] of real;
var
   a, b : RealMas;
```

В этом случае под каждую из переменных `a` и `b` в памяти отводится по 120 байт (20 элементов по 6 байт).

Размер сегмента данных составляет всего 64 килобайта. Для программы, выполняющей какие-нибудь сложные действия, например, различные преобразования матриц большой размерности, этого может оказаться недостаточно. Но ведь современные компьютеры обладают гораздо большей оперативной памятью. Неужели в ней никак нельзя отвести место для хранения данных? Конечно, можно. Для этого нужно использовать динамическую память.

Динамическая память – это часть оперативной памяти, предоставляемая программе для работы, за вычетом сегмента данных (64К), стека (обычно 16К) и собственно тела программы. Размер динамической памяти можно варьировать в широких пределах. И всю ее можно использовать для размещения данных.

Динамическое размещение данных означает выделение и освобождение динамической памяти непосредственно во время работы программы. Оперативная память при этом используется наиболее эффективным образом. Динамическое размещение данных удобно и в том случае, когда размерность массива заранее неизвестна, а статическое объявление массива максимально возможного размера приводит к неэффективному использованию памяти.

Работа с динамической памятью осуществляется с помощью указателей. ***Указатель*** – это переменная, в качестве своего значения содержащая адрес ячейки памяти. Указатель занимает 4 байта памяти (2 слова – адрес сегмента и смещение).

С помощью указателей можно размещать в динамической памяти любой из известных в Турбо-Паскале типов данных. Некоторые типы занимают в памяти более одного байта, поэтому на самом деле указатель адресует лишь первый байт данных.

Чаще всего в Турбо-Паскале указатель связывается с определенным типом данных. В этом случае тип данных называется ***базовым***, а указатель – ***типизированным***. Для объявления типизированного указателя используется значок ^, который помещается перед соответствующим типом. Например:

```
var
   p_int : ^integer;  {указатель на значение типа integer}
   p_real : ^real;    {указатель на значение типа real}
```

Выделение динамической памяти под переменную P происходит при вызове процедуры New(P). При этом размер выделяемого блока соответствует размеру базового типа. Для освобождения памяти, занятой динамической переменной P, используется процедура Dispose(P), при этом значение указателя P остается прежним. Чтобы обнулить освободившийся указатель, ему можно присвоить значение Nil.

```
. . .
New(p1);
. . .
Dispose (p1)
p1:=Nil;
. . .
```

Обращение к переменной базового типа, находящейся в динамической памяти осуществляется по имени указателя со значком ^.

```
x:=p1^; {переменной x присваивается значение элемента,
         на который указывает p1}
p2^:=x; {элементу, на который указывает P2, присваивается
         значение переменной x}
```

Итак, запомните, что сами данные, размещенные в динамической памяти, обозначаются значком ^, который ставится сразу за указателем. Если же такого значка нет, то имеется в виду только адрес, по которому размещены данные.

Рассмотрим следующий пример.

```
var
   p_int1, p_int2  : ^integer;
   p_real : ^real;
   i : integer;
BEGIN
i:=7;
New(p_int1);    {выделение памяти под переменные}
New(p_int2);
New(p_real);
p_int1^:=48;   {переменным типа integer присвоить значение}
p_int2^:=3;
p_real^:=2.732;   {переменной типа real присвоить значение}
p_int2^:=i;
i:=p_int1^;       {i=48}
Dispose(p_int1);  {освобождение памяти, занимаемой}
Dispose(p_int2);  {переменными}
Dispose(p_real);
END.
```

В разделе описаний этой программы объявляются три указателя: p_int1 и p_int2, которые будут указывать на ячейки памяти, содержащие значения типа integer, и p_real, указывающий на значение типа real.

Прежде всего, нужно выделить в динамической памяти место под переменные, на которые ссылаются указатели. Поэтому сначала для каждого из указателей вызывается процедура New.

Теперь с выделенными участками памяти можно работать как с обычными переменными, имена которых состоят из имени соответствующего указателя и знака ^.

После выполнения всех необходимых действий с динамическими переменными, нужно освободить выделенные участки памяти: в конце программы для каждого указателя вызывается процедура `Dispose`.

Для выделения и освобождения динамической памяти используются также процедуры:

`GetMem(P,Size)` – выделяет память размером `Size` под переменную `P`;

`FreeMem(P,Size)` – освобождает память размером `Size`, занятую переменной `P`.

Эти процедуры используются аналогично процедурам `New` и `Dispose`. Но запомните, что освобождать нужно ровно столько памяти, сколько ранее было выделено.

Давайте посмотрим, как можно использовать динамическую память для хранения больших массивов данных (пример 24).

Пример 24

```
type
  DynArray = [1..2] of real;
var
  A : ^DinArray;
  n, i : word;
BEGIN
write('Укажите размер массива:');
readln(n);
GetMem(A,n*6);
. . .
{$R-}
for i:=1 to n do
  A[i]:=random;
. . .                {любые действия с элементами масссива}
{$R+}
FreeMem(A,n*6)
END.
```

Обратите внимание на директивы компилятора. Они используются для временного отключения проверки значения индекса массива. Количество выделяемой памяти должно быть равно произведению количества элементов массива на количество байт, отводимое под один элемент (в данном случае – под тип `real` отводится 6 байт).

Вся динамическая память в Турбо-Паскале рассматривается как сплошной массив байтов, называемый ***кучей*** (heap). Адрес начала кучи хранится в стандартной переменной `HeapOrg`, конец – в переменной `HeapEnd`. На нача-

ло незанятого участка динамической памяти указывает переменная-указатель `HeapPtr`.

Задания для самостоятельной работы

1. Напишите программу сортировки динамического массива размерностью *N*.

9.2. Динамические структуры данных

Наибольшее распространение среди динамических структур данных получили списки. ***Списком*** называется структура данных, каждый элемен которой содержит ссылку, указывающую на следующий элемент списка. Последний элемент указывает на `Nil`.

Для организации списков используются записи, состоящие из двух частей: ***информационной***, которая и содержит всю информацию, подлежащую обраюотке, и ***адресной*** – указателя на следующий элемент списка. (рис 9.1).

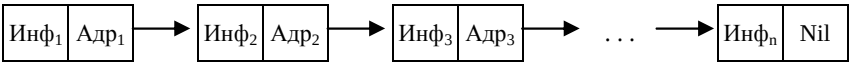

Рис. 9.1.

Описание отдельного элемента списка целых чисел имеет вид:

```
type
  ListPointer = ^List;
  List = record
           x : integer;
           adr : ListPointer;
         end;
var
  el : ListPointer;
```

Обратите внимание на то, что в данном случае допустимо использование типа `List` до его непосредственного описания. Такое исключение в Турбо-Паскале сделано только для описания динамических структур данных.

Рассмотрим общую схему формирования списка. Допустим, в программе с использованием приведенного выше типа были объявлены два указателя `u1` и `u2`.

```
var
  u1, u2 : ListPointer;
```

Размещаем в памяти динамическую переменную `u1^` типа запись, являющуюся первым элементом списка.

```
New(u1);       {размещение 1-го элемента списка}
Read(u1^.x);   {ввод значения 1-го элемента списка}
u1^.adr:=Nil; {следующий элемент пока отсутствует}
```

```
u2:=u1;        {адрес 1-го элемента списка используется
                для формирования ссылки на 2-й элемент}
```

Размещаем второй элемент списка, все следующие элементы размещаются аналогично второму.

```
New(u2^.adr); {размещение 2-го (следующего) элемента
               списка и помещение его адреса в адресную
               часть 1-го (предыдущего) элемента}
u2:=u2^.adr;  {переход ко 2-му (следующему) элементу
               списка}
Read(u2^.x);  {ввод значения 2-го элемента списка}
u2^.adr:=Nil; {следующий элемент пока отсутствует}
```

Теперь каждый элемент списка содержит ссылку на следующий элемент. Последний элемент не содержит никакой ссылки – на нем список заканчивается.

Ниже приведена процедура вставки в описанный ранее список нового элемента с информационной частью, равной `Inf`, после элемента `u0` (пример 25).

Пример 25

```
PROCEDURE InsertElem(u0:ListPointer; Inf:integer);
var
  u : ListPointer;
begin
New(u);
u^.x:=Inf;
u^.adr:=u0^.adr;
u0^.adr:=u;
end;
```

Процедура удаления элемента, следующего за элементом `u0`, из списка (пример 26).

Пример 26

```
PROCEDURE DeleteElem(u0:ListPointer);
var
  u : ListPointer;
begin
if u0^.adr<>Nil then
 begin
   u:=u0^.adr;
   u0^.adr:=u0^.adr^.adr;
   dispose(u);
 end;
end;
```

Задания для самостоятельной работы

1. Напишите программу, формирующую список записей на основе файла `tovar.dat`.
2. Дополнить предыдущую программу процедурой поиска товара по наименованию.

10. МЕТОДИЧЕСКИЕ УКАЗАНИЯ

10.1. Методические указания по выполнению контрольных работ

10.1.1. Контрольная работа №1. «Обработка строк»

Варианты заданий на контрольную работу №1:

Вариант 1

Задана строка символов. Исключить из этой строки группы символов, расположенные между фигурными скобками *{* и *}*. Сами скобки тоже должны быть исключены. Предполагается, что внутри каждой пары скобок нет других скобок.

Например, задана строка :

```
'begin {начало программы} end. {конец программы}'
```

Результат работы программы:

```
'begin  end. '
```

Вариант 2

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Вывести самое длинное слово в строке и его длину.

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
программировать  Длина этого слова равна 15
```

Вариант 3

Задана строка символов. Вывести в алфавитном порядке все латинские буквы, встречающиеся в строке. Все выводимые буквы преобразовать в прописные.

Например, задана строка :

```
'Я учусь программировать на языке Turbo-Pascal'
```

Результат работы программы:

```
A A B C L O P R S T U
```

Вариант 4

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Найти количество слов в строке, у которых первый и последний символ совпадают.

Например, задана строка :

```
'В этой строке 33 символа и 8 слов'
```

Результат работы программы:

```
3 слова
```

Вариант 5

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Удалить из каждого слова строки все последующие вхождения его первой буквы.

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
'Я учсь программировать на языке Турбо-Паскаль'
```

Вариант 6

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Вывести все буквы, которые входят в наибольшее количество слов строки. Все выводимые буквы преобразовать в прописные.

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
Ь
```

Вариант 7

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Определить количество слов, которые содержат ровно две буквы *а*.

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
2 слова
```

Вариант 8

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Вывести все слова строки в алфавитном порядке.

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
Турбо-Паскаль Я на программировать учусь языке
```

Вариант 9

Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Каждое слово заменить на число, соответствующее количеству символов в слове. Вывести полученную строку.

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
1 5 15 2 5 12
```

Вариант 10

Задана строка символов, содержащая только буквы. Составить программу "сжатия" исходной строки символов: каждая подстрока, состоящая из нескольких вхождений одного и того же символа, заменяется на текст *x(k)*, где *x* - символ, а *k* – число вхождений символа в исходную строку .

Например, задана строка :

```
'Я учусь программировать на языке Турбо-Паскаль'
```

Результат работы программы:

```
'Я учусь програм(2)ировать на языке Турбо-Паскаль'
```

10.1.2. Контрольная работа №2. «Обработка матриц»

Варианты заданий на контрольную работу №2:

Вариант 1

Задана матрица вещественных чисел размерности $m \times n$. Не используя вспомогательного массива, транспонировать данную матрицу. Размерность матрицы и значения ее элементов ввести с клавиатуры.

Вариант 2

Задана матрица вещественных чисел размерности $m \times n$. Упорядочить строки матрицы по возрастанию элементов первого столбца (считать, что в столбце нет одинаковых элементов). Размерность матрицы и значения ее элементов ввести с клавиатуры.

Вариант 3

Задана матрица вещественных чисел A размерности $m \times n$. Получить матрицу AA^T (ее размерность - $m \times m$), где A^T – транспонированная матрица. Размерность матрицы и значения ее элементов ввести с клавиатуры.

Вариант 4

Начиная от центра, обойти по спирали все элементы квадратной матрицы размером 13×13, распечатывая их в порядке обхода. Значения элементов матрицы ввести с клавиатуры.

Вариант 5

Задана матрица вещественных чисел A размерности $n \times n$. Поменять местами элементы, расположенные над главной и побочной диагоналями с элементами, расположенными под ними (первую строку с последней, вторую с предпоследней и т.д.). Размерность матрицы и значения ее элементов ввести с клавиатуры.

Вариант 6

Задана целочисленная матрица $A(15 \times 20)$. Получить массив $B(15)$ типа *boolean*, присвоив *k*-му элементу значение *true*, если *k*-я строка массива сим-

метрична и значение *false* в противном случае. Значения элементов матрицы ввести с клавиатуры.

Вариант 7

Задана вещественная матрица размером 7×7, все элементы которой различны. Найти скалярное произведение строки, в которой находится наибольший элемент матрицы, на столбец с наименьшим элементом. Значения элементов матрицы ввести с клавиатуры.

Вариант 8

Задана действительная квадратная матрица порядка *n*. Найти наибольший по модулю элемент матрицы. Не используя вспомогательного массива, получить квадратную матрицу порядка *n-1* путем выбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением. Размерность матрицы и значения ее элементов ввести с клавиатуры.

Вариант 9

Задана действительная квадратная матрица порядка 10. Построить вектор длиной 19, элементами которого являются максимумы элементов диагоналей, параллельных главной диагонали. Значения элементов матрицы ввести с клавиатуры.

Вариант 10

Задана матрица вещественных чисел *A* размерности *m*×*n*. Переставляя ее строки и столбцы добиться того, чтобы наибольший элемент (один из них) оказался в левом верхнем углу. Размерность матрицы и значения ее элементов ввести с клавиатуры.

10.2. Методические указания по выполнению курсовой работы

Целью настоящей курсовой работы является закрепление полученных знаний по дисциплине «Информационные Технологии Обработки Данных». В результате работы должна быть создана программа на языке Паскаль в соответствии с заданием и оформлена пояснительная записка.

10.2.1. Задания на курсовую работу

Вариант 1.

Разработать программу создания и корректировки файла, содержащего сведения о студентах. Каждый элемент этого файла должен содержать следующие данные: номер группы, номер в группе по списку, фамилию, имя, отчество, год рождения, оценки за последнюю сессию. Программа должна предусматривать создание текстового файла, содержащего информацию о студентах заданной группы.

Вариант 2.

Разработать программу создания и корректировки файла, содержащего сведения о книгах, находящихся в библиотеке. Каждый элемент этого файла должен содержать следующие данные: фамилию и инициалы автора, название книги, наименование издательства, год издания, количество страниц. Программа должна предусматривать создание текстового файла, содержащего список книг указанного автора, имеющихся в библиотеке.

Вариант 3.

Разработать программу создания и корректировки файла, содержащего сведения об абитуриентах, сдавших вступительные экзамены в институт. Каждый элемент этого файла должен содержать следующие данные: фамилию и инициалы абитуриента, специальность, на которую он поступает, полученные оценки по физике, математикеи литературе. Предусмотреть создание текстового файла, содержащего информацию об абитуриентах, поступающих на указанную специальность.

Вариант 4.

В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны номер рейса, тип самолета, пункт назначения, время вылета. Имеются справочники: по расстояниям до каждого из пунктов назначения и по расходу горючего (в тоннах на тысячу километров) для каждого типа самолета. Разработать программу создания и корректировки файлов, содержащих указанную информацию. Программа должна также формировать в виде текстового файла заявку на горючее на следующие сутки (количество в тоннах).

Вариант 5.

Разработать программу создания и корректировки файла, содержащего номера частных телефонов. Каждый элемент этого файла должен содержать следующие данные: номер телефона, фамилию его владельца и адрес. Программа должна формировать в виде текстового файла список телефонов, владельцы которых проживают на указанной улице.

Вариант 6.

Разработать программу создания и корректировки файла, содержащего сведения о датах рождения сотрудников. Каждый элемент этого файла должен содержать следующие данные: имя, фамилию и отчество сотрудника, день, месяц и год его рождения. Предусмотреть создание текстового файла, содержащего список сотрудников, которые отмечают свой день рождения в указанном месяце.

Вариант 7.

Разработать программу создания и корректировки файла, содержащего сведения о товарах, проданных в магазине за последний рабочий день. Каждый элемент этого файла должен содержать следующие данные: наименование

товара, единицы измерения, количество проданных единиц товара, цена товара за единицу. Программа должна формировать в виде текстового файла список всех проданных товаров с указанием вырученной суммы за каждый товар, в конце файла должна быть указана общая выручка за день.

Вариант 8.

Каждая строка текстового файла представляет собой запись комплексного числа. Разработать программу создания типизированного файла, содержащего комплексные числа, из заданного. Программа должна также позволять добавлять в файл новые числа и сортировать его.

Вариант 9.

Разработать программу создания и корректировки файла, содержащего сведения о геометрических фигурах на плоскости (смотрите дополнительную литературу). Каждый элемент этого файла должен содержать следующие данные: координаты базовой точки, форму (треугольник, круг или прямоугольник), для треугольника – координаты еще двух точек, для круга – радиус, для прямоугольника – координаты противоположной точки. Программа должна также позволять формировать текстовый файл, содержащий список фигур заданной формы с указанием их площади и периметра.

Вариант 10.

Разработать программу создания и корректировки файла, содержащего сведения о маршрутах поездов на ближайшую неделю. Каждый элемент этого файла должен содержать следующие данные: номер поезда, конечный пункт назначения, день недели, время отправления и перечень остановок. Программа должна также позволять формировать текстовый файл, содержащий список поездов, которые останавливаются в заданном городе.

10.2.2. Оформление пояснительной записки (отчета)

Отчет должен содержать:

- титульный лист;
- содержание;
- введение;
- основную часть;
- заключение;
- список использованных источников.

Титульный лист оформляется согласно стандарту.

Введение содержит основную цель работы и область применения разрабатываемой темы.

Основная часть работы должна отражать процесс и результаты разработки программы. Примерное содержание основной части:

- назначение разрабатываемой программы;
- описание структуры программы;
- описание процедур и функций;

- руководство пользователю.

Заключение содержит краткие выводы по результатам выполненной работы.

Список использованных источников включает в себя только те источники, на которые имеются ссылки в пояснительной записке, иоформляется согласно стандарту.

Пояснительная записка оформляется в редакторе MS Word шрифтом Times New Roman размером 12.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Офицеров Д.В. и др. Программирование на персональных ЭВМ. Практикум – Минск: Вышэйшая школа, 1993.

2. Епанешников А.М., Епанешников В.А. Программирование в среде Турбо-Паскаль 7.0 – М.: Диалог-МИФИ, 1995.

3. Перминов О.Н. Программирование на языке Паскаль – М.: Радио и связь, 1988.

ПРИЛОЖЕНИЕ

ОСНОВЫ РАБОТЫ В MS-DOS

А.1. Работа в системе MS-DOS

А.1.1. Основные понятия MS-DOS

Для того чтобы пользователь мог управлять компьютером и давать ему какие-либо указания (команды), нужен посредник, осуществляющий взаимодействие человека и компьютера. В роли такого посредника выступает операционная система. В первых двух семестрах курса «Информатика» будет рассматриваться операционная система MS-DOS и ее основные команды.

Основными понятиями операционной системы MS-DOS являются понятия диска, файла и каталога.

Диск – это устройство, предназначенное для хранения данных. К дискам относятся как винчестер (жесткий диск), находящийся внутри компьютера, так и дискеты (гибкий диски), используемые для переноса данных.

Каждый диск имеет собственное имя, состоящее из латинской буквы, за которой следует двоеточие, например: `A:`, `B:`, `C:` и т.д. При этом для гибких дисков приняты имена `A:` и `B:`, а для жесткого диска – `C:`.

Файл – это некоторая область данных на диске, имеющая собственное уникальное имя. Файл может содержать текст, какие-либо данные, готовую программу и т.д. В соответствии с содержимым файла ему присваивается определенный ***тип***. Для указания конкретного файла используются его имя и тип, разделенные точкой. Имя файла состоит не более чем из восьми символов, тип – не более чем из трех символов. Тип файла также называют ***расширением***. Расширение файла можно не указывать, при этом точка не обязательна.

Как правило, файлы, относящиеся к одной задаче, имеют одинаковые имена, но разные расширения (типы). Например:

`program.pas` – текст программы на языке Паскаль;

`program.exe` – готовая программа в машинных кодах.

В MS-DOS существуют стандартные обозначения типов файлов, например:

`EXE`, `COM` – исполняемые файлы (готовые программы);

`TXT` – текстовый файл;

`PAS` – текстовый файл, содержащий исходный текст программы на языке Паскаль.

При выполнении однотипных операций над группой файлов (копирование, перенос, удаление) используется так называемый ***шаблон*** имени файла. Шаблон предусматривает использование символов `*` и `?`. Символ `*` обозначает любое число любых символов в имени или в расширении файла. Символ `?` обозначает один любой символ либо отсутствие символа в имени или в расширении файла. Например:

`*.bak` – все файлы с расширением `bak` из ткущего каталога;

`f*.d*` – все файлы из текущего каталога, имена которых начинаются на `f`, а расширения – на `d`;

`a??.*` – все файлы из текущего каталога, имена которых начинаются на `a` и состоят не более, чем из трех символов.

Со временем количество файлов на компьютере неизбежно растет, и найти нужный файл на диске становится все сложнее. Для того, чтобы как-то упорядочить файлы, используются каталоги.

Каталог – это группа логически связанных по какому-то признаку файлов на одном диске, имеющая собственное имя. Каталог в MS-DOS можно сравнить в каталогом в библиотеке. Так, например, в библиотеке все книги по истории объединены в каталог с названием «История», все книги по программированию в каталог «Программирование» и т.д. Так и на компьютере: программы на Паскале удобно хранить в каталоге PASCAL, различные текстовые файлы (письма, документы и прочее) – в каталоге TEXTS, - то есть все файлы можно распределить по каталогам так, как удобно пользователю.

В каталоге могут находиться не только файлы, но и другие каталоги, называемые ***подкаталогами***. На каждом диске всегда имеется один ***главный*** или ***корневой каталог***. В корневом каталоге находятся файлы и подкаталоги первого уровня, в подкаталогах первого уровня находятся файлы и подкаталоги второго уровня и т.д. Так образуется ***иерархическая (древовидная) файловая система***. Пример иерархической файловой системы приведен на рис. 1.1.

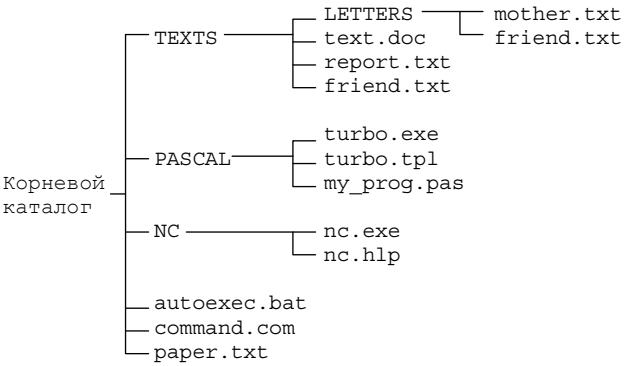

Рис. А.1. Пример файловой системы

Древовидная файловая система позволяет хранить несколько файлов с одинаковыми именами и расширениями, но, в таком случае, они должны содержаться в различных каталогах. Например, на рисунке 1.1 файл с. именем friend.txt содержится и в каталоге TEXTS, и в каталоге LETTERS, причем содержимое этих файлов может различаться. Если указать имя файла, не указывая каталог, в котором он находится, то MS-DOS будет искать его в текущем каталоге. ***Текущим*** называется каталог, с которым в настоящий момент рабо-

тает пользователь. Для точного указания файла уже недостаточно задать только его имя и тип, необходимо, кроме того, задать и ***путь*** к нему – цепочку из имен каталогов, разделенных символом «\» (этот символ называют «бэк слэш»), в которой каждый последующий каталог является подкаталогом предыдущего. Путь задает маршрут от текущего или корневого каталога диска к тому каталогу, в котором находится нужный файл.

Если путь начинается с символа «\», то маршрут задан от корневого каталога, иначе – от текущего.

Например, предположим, что в данный момент текущим является каталог TEXTS (рис. А.1). Тогда указать файл mother.txt можно следующими способами:

`\texts\letters\mother.txt` – из корневого каталога;

`letters\mother.txt` – из текущего каталога.

При указании пути к файлу от корневого каталога можно также указать диск, например (рис. 1.1):

```
a:\texts\letters\friend.txt
```

При этом первый символ «\» может быть опущен:

```
a:texts\letters\friend.txt
```

Имя файла вместе с путем принято называть ***полным именем файла***. Полное имя файла однозначно определяет файл. Например, файлы с одинаковыми именами, находящиеся в разных каталогах, можно различать по их полным именам:

```
a:\texts\letters\friend.txt
a:\texts\friend.txt
```

А.1.2. Диалог пользователя с MS-DOS

Диалог пользователя с MS-DOS осуществляется в форме команд. Команда пользователя означает, что MS-DOS должна выполнить указанное действие. Каждая команда состоит из имени команды и, возможно, параметров, разделенных пробелами.

Когда MS-DOS готова к диалогу с пользователем (после загрузки компьютера, а также по окончании выполнения последней команды), она выдает на экран приглашение, указывая текущий диск:

```
C>_
```

Приглашение может также содержать и информацию о текущем каталоге

```
C:\TEXTS>_
```

и/или время

```
12:10 C:\TEXTS>_
```

В ответ на приглашение пользователь может ввести команду. Для этого необходимо набрать имя команды, затем – ее параметры и нажать клавишу `Enter`.

Рассмотрим основные команды MS-DOS. При описании формата команд будем использовать следующие соглашения:

- в квадратных скобках указываются параметры, не являющиеся необходимыми для выполнения команды, все остальные параметры должны быть набраны обязательно (например, `[/p]` означает, что ключ `/p` может отсутствовать, при этом команда не будет ошибочной);

- курсивом (наклонным шрифтом) выделены элементы команды, вместо которых должно быть указано нужное конкретное значение (например, *диск* означает, что вместо этого слова должен быть указан какой-либо конкретный диск: A:, C: и т.п.).

Изменение текущего диска

Для изменения текущего диска достаточно указать его имя (рис. А.2):

диск

```
C:\>a:

A:\>_
```

Рис. А.2. Изменение текущего диска

Пример:

a: - переход на диск a:.

Изменение текущего каталога

Для изменения текущего каталога в MS-DOS предназначена команда cd (рис. А.3).

cd [*диск*]*путь*

Если *диск* задан, то текущий каталог изменяется на указанном диске, в противном случае – на текущем.

```
C:\>cd texts

C:\texts>_
```

Рис. А.3. Изменение текущего каталога

Примеры:

cd \ - переход в корневой каталог текущего диска;

cd a:\doc - переход в каталог doc диска a:.

Просмотр каталога

Для вывода ***оглавления каталога*** – списка всех содержащихся в нем файлов и подкаталогов – используется команда dir (рис. А.4).

dir [*диск*][*путь*\][*имя-файла*] [/p][/w]

В имени файла можно употреблять символы * и ?. Если *имя-файла* не задано, то будет выведено все оглавление каталога, в противном случае – только сведения об указанном файле или группе файлов.

Если в команде не указаны *диск* и *путь*, то подразумеваются текущий диск и текущий каталог.

Для каждого файла команда `dir` сообщает его имя, расширение, размер файла в байтах, дату и время его создания. Подкаталоги отмечаются `<КАТАЛОГ>`. В конце сообщается о размере свободного пространства на диске.

```
C:\>dir \texts

 Том в устройстве C не имеет метки
 Серийный номер тома: 0F55-16F9
 Содержимое каталога C:\texts

.              <КАТАЛОГ>     11.10.99  11:24   .
..             <КАТАЛОГ>     11.10.99  11:42   ..
LETTERS        <КАТАЛОГ>     15.10.99  17:25   letters
TEXT     DOC          6 162   18.10.99  14:10   text.doc
REPORT   TXT             83   19.10.99  10:21   report.txt
FRIEND   TXT            187   21.10.99  15:07   friend.txt
         3 файл(а,ов)             6 432 байт
         3 каталог(а,ов)    300 122 112 байт свободно

C:\>_
```

Рис. А.4. Просмотр каталога

Если оглавление каталога не помещается на экране полностью, то указание параметра `/p` позволит просмотреть каталог постранично: после заполнения экрана MS-DOS будет ждать до тех пор, пока пользователь не нажмет любую клавишу, после чего – выведет очередную страницу.

В большинстве случаев, пользователя интересует только информация об именах файлов. В таком случае для вывода оглавления в более компактной форме можно использовать параметр `/w`. При этом имена выводятся по пять в каждой строке (рис. А.5).

```
C:\texts>dir /w

 Том в устройстве C не имеет метки
 Серийный номер тома: 0F55-16F9
 Содержимое каталога C:\texts

[.]          [..]         [LETTERS]    TEXT.DOC     REPORT.TXT
FRIEND.TXT
         3 файл(а,ов)             6 432 байт
         3 каталог(а,ов)    300 122 112 байт свободно

C:\texts>_
```

Рис. А.5. Просмотр каталога с параметром `/w`

Примеры:

`dir` - вывести оглавление текущего каталога;

`dir *.exe` - вывести сведения обо всех файлах с расширением `.exe` из текущего каталога ;

`dir a:\doc` - вывести оглавление каталога `doc` на диске `a:`.

Создание нового каталога

Для создания нового каталога используется команда `md`:

`md [`*диск*`]`*путь*

Примеры:

`md news` - создание подкаталога news в текущем каталоге;

`md a:\docs` - создание подкаталога `docs` в корневом каталоге диска `a:`.

Удаление каталога

Для удаления (пустого) каталога используется команда `rd`:

`rd [`*диск*`]`*путь*

Отметим, что удалить можно только пустой каталог, т.е. каталог, не содержащий файлов и подкаталогов. Если необходимо удалить непустой каталог, нужно сначала удалить все файлы и подкаталоги, в нем содержащиеся, а затем – применить команду `rd`.

Примеры:

`rd news` - удаление подкаталога `news` в текущем каталоге;

`rd a:\docs` - удаление подкаталога `docs` в корневом каталоге диска `a:`.

Удаление файлов

Для удаления файлов применяется команда `del`:

`del [`*диск*`][`*путь*`\]`*имя-файла*

В имени файла возможно употребление символов `*` и `?`.

Если Вы захотите удалить все файлы из каталога, например, с помощью команды `del *.*`, то MS-DOS попросит подтверждения, при этом нужно ответить `y` или `n` (рис. А.6):

```
C:\texts\letters>del *.*
Будут удалены все файлы в каталоге!
Продолжить [Y(да)/N(нет)]?y

C:\texts\letters>_
```

Рис. А.6. Удаление всех файлов в текущем каталоге

Примеры:

`del \texts *.txt` - удаление из подкаталога `texts` всех файлов с расширением `.txt`;

`del report.doc` - удаление файла `report.doc` в текущем каталоге.

Копирование файлов

Копирование файлов в другой каталог осуществляется командой `copy`.

`copy [`*диск*`][`*путь*`\]`*имя-файла* `[`*диск*`][`*путь*`\][`*имя-файла*`]`

В именах файлов возможно употребление символов `*` и `?`.

Файлы, указанные в первом параметре команды, копируются в каталог, указанный во втором параметре. Если *имя-файла* во втором параметре отсутствует, то при копировании имена файлов не меняются, если же *имя-файла* задано, то оно указывает новое имя копируемого файла. Символы `*` и `?` в имени файла во втором параметре команды указывают, что соответствующие символы в именах копируемых файлов при копировании не меняются (рис. А.7).

```
C:\texts>copy friend.txt f.txt
        1 файл(а,ов) скопировано

C:\texts>copy *.txt *.doc
report.txt
friend.txt
f.txt
        3 файл(а,ов) скопировано

C:\texts>_
```

Рис. А.7. Примеры копирования файлов

Примеры:

`copy \texts *.txt` - копирование всех файлов с расширением `.txt` из подкаталога `texts` в текущий каталог;

`copy friend.txt a:\` - копирование файла `friend.txt` из текущего каталога на диск `a:`.

Переименование файла

Для переименования файлов используется команда `ren`.

`ren [`*диск*`][`*путь*`\]`*имя-файла имя-файла*

Первое имя файла в команде задает имя (имена) переименовываемого файла, второе – новое имя (имена) файла.

Диск и путь показывают, в каком каталоге переименовываются файлы. Если эти параметры опущены, то подразумевается текущий каталог.

В именах файлов возможно употребление символов * и ?. В этом случае переименовываются все файлы из заданного каталога, подходящие под шаблон, заданный в первом имени файла. Если эти символы используются во втором имени файла, то символы имен файлов на соответствующих позициях не изменяются.

Примеры:

`ren *.doc *.txt` - переименование всех файлов с расширением `.doc` в текущем каталоге в файлы с расширением `.txt`;

`ren \texts\friend.txt myfriend.doc` - переименование файла `friend.txt` в каталоге `texts` в файл `myfriend.doc`.

Запуск программ

Как уже отмечалось, программы, готовые к выполнению, находятся в файлах с расширениями `.exe` и `.com`. Чтобы запустить на выполнение любую из этих программ, достаточно просто набрать имя соответствующего файла. При этом расширение файла можно не указывать – MS-DOS автоматически будет искать файл с нужным расширением. Если путь не задан, поиск файла ограничится текущим каталогом. Для запуска программы из другого каталога задание пути обязательно.

Пример:

`\nc\nc` - запуск программы, находящейся в файле `nc.exe` в каталоге `\nc`.

A.2. Работа в Norton Commander

A.2.1. Программа-оболочка Norton Commander

Работа пользователей в системе MS-DOS построена по принципу диалога: пользователь набирает на клавиатуре необходимую команду и нажимает клавишу `Enter`, затем – MS-DOS выполняет указанную команду. Такой способ взаимодействия не нагляден и недостаточно удобен, поэтому при работе с MS-DOS часто используются программы-оболочки. ***Программы-оболочки*** позволяют выполнять большинство команд MS-DOS: просматривать содержимое каталогов, переходить из одного каталога в другой, запускать программы, копировать и удалять файлы и т.д. В настоящем разделе мы кратко рассмотрим работу в одной из таких программ – в программе ***Norton Commander***.

A.2.2. Запуск программы Norton Commander

Для того, чтобы запустить Norton Commander, наберите в командной строке `NC`. После запуска Norton Commander на экране появятся два окна, ограниченные двойной рамкой. Эти окна будем называть ***панелями***. Под левой панелью располагается приглашение MS-DOS. В самом низу экрана – строка, напоминающая о назначении функциональных клавиш в программе (рис. А.8).

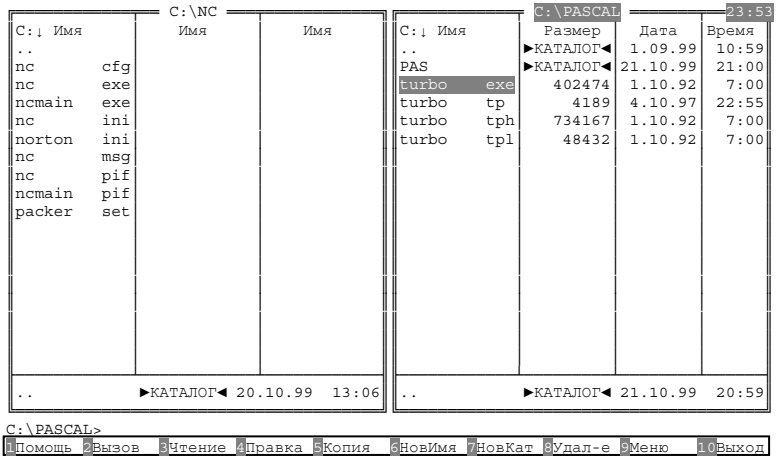

Рис. А.8. Вид экрана при работе с программой Norton Commander

А.2.3. Выход из Norton Commander

Для выхода из программы Norton Commander нужно нажать клавишу `F10`. При этом в центре экрана появится запрос на подтверждение того, что вы действительно хотите выйти из Norton Commander (рис. А.9). Чтобы выйти, нажмите `Enter` или щелкните мышью `Да`. Чтобы отменить выход из программы, нажмите `Esc` или щелкните мышью `Нет`.

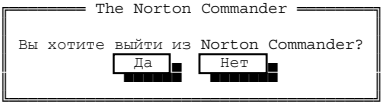

Рис. А.9. Запрос на подтверждение выхода

А.2.4. Панели Norton Commander

Каждая из панелей Norton Commander содержит оглавление каталога на диске (кроме того, она может содержать дерево каталогов или информацию о диске, подробнее об этом можно прочитать в дополнительной литературе).

Вверху панели выводится полное имя каталога (например, `C:\PASCAL`). Имена файлов в оглавлении каталога выводятся строчными буквами, а имена подкаталогов – заглавными.

Оглавление каталога может быть представлено в кратком формате (левая панель на рис. А.8) и в полном формате (правая панель на рис А.8). Как видно из рисунка, краткий формат оглавления представляет собой просто список

имен всех файлов и подкаталогов, содержащихся в текущем каталоге (колонка Имя). Полный формат содержит сведения об имени файла (Имя), его размере (Размер), дате его создания (Дата) и времени создания (Время). В колонке Размер для подкаталогов выводится ►КАТАЛОГ◄. Если Вы находитесь в каком-либо подкаталоге, то в самой верхней строке оглавления выводятся две точки «. .», а справа – в колонке Размер – ►КАТАЛОГ◄. Эта строка является ссылкой на родительский каталог.

А.2.5. Работа с файлами и каталогами в Norton Commander

Один из файлов или каталогов на экране выделен цветом – этот файл или каталог будем называть ***выделенным*** или текущим. С помощью клавиш перемещения курсора можно передвигаться по панели, выделяя другие файлы или каталоги. Для перехода на другую панель Norton Commander нужно нажать на клавишу Tab.

Если выделить какой-либо каталог и нажать Enter, то Norton Commander «войдет» в этот каталог и отобразит его оглавление на панели. Для возврата назад, в родительский каталог, как Вы уже догадались, необходимо выделить первую строку «. .» и нажать Enter. Так можно перемещаться по всему дереву файловой системы. Попробуйте выполнить эти действия, и Вы сможете оценить простоту работы в Norton Commander (вспомните команды MS-DOS).

Если выделен файл с расширением .com, .exe или .bat, то нажатие клавиши Enter приведет к запуску программы, содержащейся в этом файле.

Выбор группы файлов

Norton Commander позволяет выбрать группу файлов или каталогов, над которой можно выполнять некоторые действия: скопировать, переместить в другой каталог, удалить и т.д. Выбранные файлы отображаются другим цветом. Внизу панели появляются сведения об общем числе выделенных файлов и их общем размере.

Выбор отдельного файла или каталога осуществляется нажатием клавиши Ins. Повторное нажатие этой клавиши отменяет выбор файла или каталога.

Чтобы выбрать группу файлов по шаблону, нужно нажать «+» в правой части клавиатуры и задать шаблон, использующий символы * и ? . Отменить выбор группы файлов по шаблону можно нажав «-» в правой части клавиатуры и указав шаблон файлов, выбор которых нужно отменить.

Чтобы сделать невыделенные файлы выделенными, а выделенные – невыделенными, нужно нажать «*» в правой части клавиатуры. При этом подкаталоги окажутся невыделенными.

Выделенную группу файлов можно удалять, копировать и перемещать также как и отдельные файлы.

Просмотр файлов

При нажатии клавиши F3 Norton Commander переходит в режим просмотра выделенного файла.

Для перемещения по просматриваемому файлу можно использовать клавиши перемещения курсора. Клавиши Home и End позволяют переместиться на начало и на конец файла, соответственно.

Редактирование файлов

Для редактирования (изменения) выделенного файла используется клавиша F4. При этом происходит переход в режим редактирования.

Перемещение по файлу осуществляется клавишами перемещения курсора. Клавиши Home и End используются для перемещения в начало и в конец текущей строки, соответственно; Ctrl+Home и Ctrl+End – в начало и в конец файла.

Для того чтобы сохранить внесенные изменения нужно нажать F2.

Выйти из режима редактирования можно по клавише F10.

Копирование файлов и каталогов

Для копирования файлов в программе Norton Commander надо выделить нужный файл или выбрать группу файлов и нажать клавишу F5. Если на панели выбраны какие-либо файлы, то копироваться будет выбранная группа файлов, в противном случае – копируется текущий файл.

После нажатия F5 в центре экрана появляется запрос о том, куда надо копировать файл или группу файлов (рис. А.10). При этом предлагается имя каталога, который в данный момент отображен на другой панели. Это имя может быть изменено пользователем.

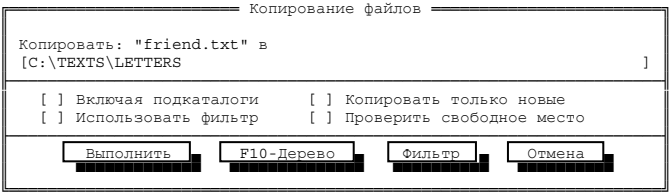

Рис. А.10. Копирование файлов

Если имя каталога в запросе указано верно, то для продолжения копирования нужно нажать Enter, а для отмены – Esc или щелкнуть мышью на Выполнить и Отмена, соответственно.

Во избежание ошибок в имени каталога, рекомендуется перед копированием настроить другую панель Norton Commander так, чтобы в ней отображался тот каталог, куда нужно скопировать файл.

При копировании каталога будут скопированы все содержащиеся в нем файлы. Чтобы при этом копировались еще и все подкаталоги, необходимо включить флажок [ ] Включая подкаталоги . Для этого необходимо поместить курсор в рамку слева от надписи и нажать клавишу Пробел (при этом в рамке должен появиться крестик или щелкнуть на рамке мышью.

Об использовании остальных флажков можно прочитать в справочной системе Norton Commander (ее вызов осуществляется по нажатию клавиши F1).

Переименование и перемещение файлов и каталогов

Чтобы переименовать файл, группу файлов или каталог, нужно выделить файл (группу, каталог) и нажать клавишу F6.

В ответ на запрос Norton Commander нужно ввести новое имя файлов или каталогов и нажать Enter.

С помощью той же клавиши F6 можно перенести в другой каталог файл (группу файлов) или подкаталог. Перемещение (перенос) отличается от копирования тем, что после успешного завершения переноса исходные файлы удаляются.

Удаление файлов и каталогов

Для удаления файлов и каталогов с помощью Norton Commander необходимо выделить нужный файл (группу файлов) или каталог и нажать клавишу F8 (рис. А.11, А.12). В ответ на нажатие F8 будет выдан запрос на подтверждение удаления. Если удаляются только файлы , то запрос имеет вид, представленный на рис. А.11. В этом случае для удаления файлов нужно нажать Enter, а для отмены – Esc (или щелкнуть мышью Удаление и Отмена, соответственно).

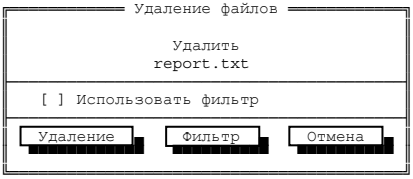

Рис. А.11. Удаление файла

При удалении каталогов запрос имеет вид, изображенный на рис. 1.12. Его отличие от предыдущего заключается в наличии двух дополнительных флажков: [x] Включая подкаталоги и [x] Удалить пустые каталоги.

Если требуется удалить не только файлы из удаляемого каталога, но и из его подкаталогов, нужно поставить крестик в первом из них. Второй флажок отмечается в том случае, если при удалении каталога нужно удалить не только все файлы, но и сам каталог, и все поддерево каталогов. Для удаления файлов и каталогов нужно нажать `Enter`, а для отмены – `Esc` (или щелкнуть мышью `Удаление` и `Отмена`, соответственно).

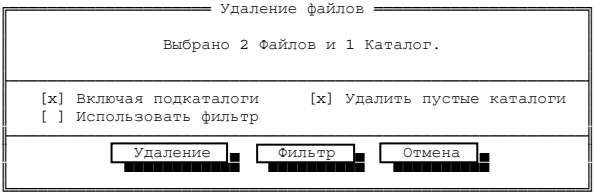

Рис. А.12. Удаление нескольких файлов и каталогов

Создание нового каталога

Чтобы создать новый каталог в текущем каталоге нужно нажать клавишу `F7`. При этом будет выдан запрос, вид которого представлен на рис. А.13. Для создания каталога нужно указать его имя и нажать Enter, а для отмены – нажать Esc.

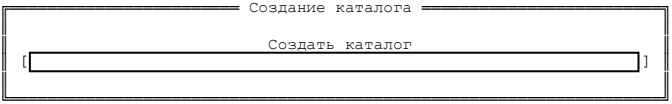

Рис. А.13. Создание нового каталога

Изменение текущего диска

Для того, чтобы в панели Norton Commander вывести оглавление другого диска, нужно нажать `Alt+F1` (для левой панели) или `Alt+F2` (для правой панели). На экран будет выведен список доступных дисков (рис. А.14).

Рис. А.14. Выбор диска

Клавишами перемещения курсора нужно выделить требуемый диск и нажать Enter.

На первый взгляд запоминание всех команд Norton Commander кажется весьма сложным. Поэтому во время работы рекомендуется пользоваться строкой подсказки, расположенной в самом низу экрана.