

А.Н. Маслобоев

**Операции с файлами
и нестандартными
типами данных
в Turbo Pascal**

Учебно-методическое пособие



**Санкт-Петербург
2013**

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ
РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ»**

А.Н. Маслобоев

**Операции с файлами
и нестандартными типами данных
в Turbo Pascal**

Учебно-методическое пособие

**Санкт-Петербург
2013**

УДК 681.3 (075)

М 316

ББК 32.9я7

Маслобоев А.Н. Операции с файлами и нестандартными типами данных в Turbo Pascal: учебно-методическое пособие / СПбГТУРП. — СПб., 2013. — 52 с.

В настоящем учебно-методическом пособии рассматриваются операции с файлами, а также использование авторских типов данных при программировании на языке Паскаль в системе Turbo Pascal 7.0. Пособие содержит необходимый теоретический минимум, примеры программ с подробными комментариями к ним и задания для самостоятельной работы студентов.

Пособие предназначено для студентов факультета МАП, обучающихся по направлению 151000.62 «Технологические машины и оборудование», изучающих дисциплину «Информационные технологии».

Рецензенты:

зав. кафедрой прикладной математики и информатики СПбГТУРП,
проф., д-р техн. наук В.М. Пестриков;
доц. кафедры информатики СПбГУСЭ, канд. техн. наук С.В. Тихов.

Рекомендовано к изданию Редакционно-издательским советом университета в качестве учебно-методического пособия.

© Маслобоев А.Н., 2013

© Санкт-Петербургский
государственный технологический
университет растительных
полимеров, 2013

Оглавление

Глава 1. Операции с файлами.....	5
1.1. Текстовые файлы	6
1.2. Типизированные файлы	11
1.3. Нетипизированные файлы	17
Задания для самостоятельной работы к главе 1	22
Глава 2. Работа с данными авторских типов	24
2.1. Перечисляемый тип данных	24
2.2. Ограниченный тип данных	31
Задания для самостоятельной работы к главе 2	35
Глава 3. Записи и множества	36
3.1. Записи	36
3.2. Множества	43
Задания для самостоятельной работы к главе 3	52
Библиографический список	53

Глава 1. Операции с файлами

Еще на начальных этапах работы в системе Turbo Pascal пользователь должен научиться сохранять в виде файлов тексты написанных на языке Паскаль программ (исходный код).

Но сохранение файла исходного текста программы не гарантирует сохранение результатов работы программы. Эти результаты пропадали после выхода из системы программирования. А ведь именно получение конкретных результатов, в конечном счете, и является целью создания программ.

Отсюда можно сделать следующий вывод: желательно предусмотреть возможность сохранения этих результатов в долговременной памяти компьютера. Как известно, именно файл и является основной единицей хранения любой информации на носителях информации вне зависимости от их устройства и физической природы (к таким носителям информации относятся и жесткий магнитный диск – винчестер, и гибкие магнитные диски, называемые также дискетами или флоппи-дисками, и лазерные компакт-диски и флэш-память).

Средства языка Паскаль, рассматриваемые в этой главе позволяют сохранять данные, полученные при выполнении программы, разными методами и в файлах различных типов, а затем анализировать и использовать эти данные в других программах.

Над файлами можно производить различные операции, которые делятся на две большие группы. Это операции записи информации в файл из оперативной памяти компьютера и считывания информации из файла в оперативную память. При записи информации в файл необходимо выполнить следующие операции: открытие файла для записи, собственно запись информации в файл, закрытие файла. При чтении информации из файла выполняются операции: открытие файла для чтения, собственно чтение файла и закрытие файла.

Существует два способа записи и чтения информации – метод последовательного доступа и метод прямого доступа. Метод последовательного доступа можно сравнить с процессом записи на магнитофонную пленку и воспроизведения ее содержимого. Для того, чтобы произвести одну или другую операцию, требуется предварительно перемотать пленку до нужного места. Также и для того, чтобы добраться в файле до интересующей нас информации методом последовательного доступа, нужно предварительно «прокрутить» всю предшествующую ей информацию. При считывании информации методом прямого доступа можно непосредственно получить доступ к нужной нам компоненте (т.е. составной части файла) подобно тому, как можно непосредственно проиграть любую запись на лазерном диске.

Файлы, с которыми можно работать средствами языка Паскаль, делятся на три основные разновидности: текстовые файлы, типизированные файлы и нетипизированные файлы. Используемые файлы, подобно переменным,

константам и массивам должны быть обязательно описаны, причем для каждого из трех видов файлов существует свой способ описания.

1.1. Текстовые файлы

Начнем рассмотрение работы с файлами с текстовых файлов. Перед использованием такого файла в программе в разделе описания переменных должна быть обязательно описана связанная с ним переменная. Общий вид описания следующий:

```
var имя_переменной: text;
```

где **имя_переменной** – имя переменной, которая впоследствии будет связана с файлом; **text** – соответствующий тип переменной.

Пример описания:

```
var k:text;
```

Текстовые файлы состоят из строк символов. Каждая такая строка заканчивается символами возврата каретки **CR** (этот символ имеет в таблице **ASCII** код 13) и перевода строки **LF** (данный символ имеет в таблице код 10). Вышеупомянутые символы относятся к так называемым управляющим символам, которые как правило при просмотре и редактировании файла не выводятся на экран компьютера. В конце файла находится признак конца файла, который сокращенно обозначается **EOF** (сокращение от английской фразы **end of file** – конец файла). Файл такого вида имеет расширение **txt**.

При этом следует иметь в виду, что для того, чтобы с файлом можно было в дальнейшем можно было производить какие-либо действия, необходимо выполнить еще две операции. Во-первых – связать имя файла с именем соответствующей ему переменной. Во-вторых – указать способ работы с файлом – запись в него информации или чтение информации.

Общий вид процедуры, связывающей имя файла с именем переменной следующий:

```
assign (имя_переменной, 'имя_файла');
```

где

assign – служебное слово, означающее в переводе «назначить»;

имя_переменной – имя соответствующей файлу переменной;

имя_файла – полное имя текстового файла.

Обратите внимание, что оно должно быть заключено в апострофы. Полное имя файла включает в себя имя диска, на котором находится данный файл, путь к файлу, который включает в себя имена каталога (каталогов), содержащих данный файл, и собственное имя файла с расширением (каталог – это термин, используемый в операционной системе MS DOS, и аналогичный папке в операционной системе Windows). Если текстовый файл находится на том же диске и в том же каталоге, что и вызывающая его программа, то диск и каталог можно не указывать.

Пример использования процедуры:

```
assign (k, 'D:otchet.txt');
```

Здесь переменной **k** ставится в соответствие файл **otchet.txt**, который находится на дискете, содержащейся на диске **D** того компьютера, с которым работает пользователь.

Данная процедура устанавливает соответствие между вышеописанной переменной **k** и текстовым файлом **otchet.txt**. В дальнейшем в программе имя текстового файла более не будет упоминаться, а при всех операциях с файлом будет указываться имя “назначенной” ему текстовой переменной.

Дальнейший ход работы зависит от того, какие именно действия необходимо произвести с данным файлом. Если содержимое файла необходимо прочитать, то его открытие производится следующей процедурой:

```
Reset (имя_переменной);
```

где **имя_переменной** - имя связанной с файлом переменной.

Что касается непосредственно самого процесса чтения информации из файла в программу, то он происходит с помощью уже знакомых нам операторов **read** или **readln**. При этом в скобках после служебного слова **read** или **readln**, как правило, указываются имена двух переменных: первой пишется имя переменной, связанной с файлом, а за ней указывается имя какой-либо вспомогательной переменной, в которую и считывается информация. После окончания операции чтения (как и рассматриваемой ниже операции записи) файл следует закрыть с помощью процедуры **close**.

Общий вид данной процедуры:

```
close (имя_переменной);
```

где **имя_переменной** – имя связанной с файлом переменной.

Рассмотрим теперь операцию чтения информации из файла на конкретном примере.

Составим программу, которая считывает информацию из файла **comp.txt**, находящегося на логическом диске **D** компьютера в каталоге **obuch**. Данный файл изначально должен содержать следующий текст:

Основные компоненты персонального компьютера:

- 1. Системный блок.**
- 2. Монитор или дисплей.**
- 3. Клавиатура.**
- 4. Координатное устройство.**

Программа называется **obrtext** и в ней используются две переменные **t**, которой ставится в соответствие файл **comp.txt** и **ln** – вспомогательная переменная строкового типа, в которую строка за строкой будет считываться содержимое текстового файла. В начале основной части программы между файлом и файловой переменной устанавливается соответствие с помощью процедуры **assign**. Затем файл открывается для чтения процедурой **reset**.

Сам процесс считывания информации производится в цикле с предусловием типа **while**. Условием завершения работы цикла является обнаружение признака конца файла (**eof**), то есть цикл будет выполняться при условии, что конец файла в ходе считывания информации еще не найден. Это условие записывается следующим образом: **not eof(t)**, то есть здесь используется стандартная функция Паскаля, которая также называется **eof** и относится к логическим функциям, значением которых может быть только **false** или **true**. Пока не обнаружен конец файла, функция имеет значение **false**, а при его обнаружении она меняет значение на **true**. Аргументом же данной функции является имя файловой переменной. При каждом выполнении тела этого цикла из файла оператором **readln** считывается очередная строка и передается в строковую переменную **ln**. Затем содержимое строковой переменной (то есть очередная строка файла) выводится на экран компьютера оператором **writeln**. По окончании работы цикла файл закрывается процедурой **close**.

```
program obrtext;  
Uses Crt;  
  var t:text; ln:string[80];  
begin  
  Clrscr;  
  assign(t, 'D:\obuch\comp.txt');  
  reset(t);  
  while not eof(t) do  
    begin  
      readln(t, ln);  
      writeln(ln)
```

```

    end;
close(t) ;
readln
end.

```

Процесс записи информации в файл также имеет свои особенности. Назначение файлу соответствующей ему файловой переменной производится так же, как и в случае чтения файла, но открытие файла для записи производится либо процедурой **rewrite**, либо процедурой **append**. Общий вид этих процедур следующий:

```

rewrite (имя_переменной) ;

append (имя_переменной) ;

```

т. е. он аналогичен виду процедуры **reset**. Надо помнить, что процедуры **rewrite** и **append** открывают файл для записи по-разному. Процедура **rewrite** при открытии файла очищает его, т. е. предыдущее содержимое файла (если таковое было) удаляется, и вместо него записывается новый текст. Если же содержимое файла нужно сохранить, то используют процедуру **append**, которая дополняет текущее содержимое текстового файла новой информацией, дописывая ее в конец данного файла. Сам же процесс записи информации в файл производится либо с помощью оператора **write**, либо оператором **writeln**. После служебного слова **write** или **writeln** в скобках указывается имя файловой переменной, а затем имя той строковой переменной, из которой осуществляется запись данных в файл.

Рассмотрим использование процедуры **append** на следующем примере. Пусть в конец файла **comp.txt** необходимо дописать следующий текст:

К компьютеру также могут подключаться:

5. Принтер.
6. Сканер.
7. Колонки.
8. Модем.

Для того чтобы выполнить требуемое действие, составим соответствующую программу. Текст программы приведен ниже.

```

program doptext;
Uses Crt;
    var t:text; ln:string[80];
        i,n:integer;
begin
    Clrscr;

```

```

assign(t, 'D:\obuch\comp.txt');
append(t);
writeln('Укажите количество вводимых строк');
readln(n);
writeln('Вводите текст. Ввод каждой строки заканчивайте
нажатием "Enter"');
for i:=1 to n do
begin
  readln(ln);
  writeln(t,ln)
end;
close(t);
Clrscr;
reset(t);
while not eof(t) do
begin
  readln(t,ln);
  writeln(ln)
end;
close(t);
readln;
end.

```

В этой программе, помимо тех переменных, которые мы использовали в предыдущей, нам понадобятся еще две: **n** – количество строк, добавляемых в конец файла, и **i** – счетчик цикла. Затем, после открытия файла для записи, указываем количество вводимых строк. Сам процесс ввода происходит в цикле с заранее заданным числом повторений (это число равно количеству вводимых строчек; в данном случае оно равно 5). В теле цикла содержится два оператора. В первом из них каждая вводимая строка оператором **readln** считывается во вспомогательную переменную **ln**. Во втором операторе содержимое переменной **ln** (т. е. очередная строка) переписывается в файл с помощью оператора **writeln**. По окончании операции записи файл закрывается.

Для того чтобы можно было проконтролировать правильность ввода в файл дополнительной информации, после записи данных произведем чтение всего содержимого файла с выводом его на экран компьютера аналогично тому, как это было сделано в предыдущей программе.

1.2. Типизированные файлы

Типизированным называется файл, который состоит из однородных элементов, относящихся к одному и тому же типу. Такими элементами могут быть переменные различных типов и массивы. Каждый элемент такого файла имеет свой индекс, подобно элементам массива. Но в отличие от обычного массива, который хранится в оперативной памяти компьютера, и содержимое которого стирается после выхода из системы программирования, типизированный файл может длительное время сохраняться на жестком диске компьютера или на другом носителе. К каждому элементу такого файла возможен как прямой (по индексу элемента), так и последовательный доступ, но недостатком его является то, что в отличие от текстового файла, его нельзя просматривать и редактировать с помощью обычного текстового редактора.

Для того, чтобы в типизированный файл можно было производить запись или считывать из него информацию, его следует связать с помощью процедуры **assign** с файловой переменной подобно тому, как мы это уже делали с текстовыми файлами. Например, так:

```
assign(nab, 'D:\inf\nabor');
```

Здесь мы связываем имеющуюся в программе файловую переменную **nab** с типизированным файлом **nabor**, находящимся на логическом диске **D** в каталоге **inf**. Обратите внимание, что в отличие от текстового файла, типизированный файл необязательно должен иметь расширение.

В общем виде описание файловой переменной для типизированного файла выглядит следующим образом:

```
var имя_переменной: file of тип;
```

где

имя_переменной - имя переменной, связанной с типизированным файлом;

file – служебное слово;

тип – тип элементов, из которых состоит данный файл. Это могут быть уже известные нам типы, используемые для описания обычных переменных и массивов (**integer**, **real**, **character**, **string**, **array**).

Пример описания файловой переменной для типизированного файла:

```
var nab: file of character;
```

Таким образом описан файл, состоящий из элементов вещественного типа, с которым связана в программе файловая переменная **nab**.

Подобно текстовым файлам, типизированные файлы открываются для чтения процедурой **reset** и для записи процедурой **rewrite**, а закрываются процедурой **close**.

Непосредственное чтение данных из файла производится с помощью следующего оператора:

```
read(имя_переменной1, имя _переменной2) ;
```

где

имя_переменной1 – имя файловой переменной;

имя_переменной2 – имя обычной переменной, относящейся к тому же типу, что и элементы типизированного файла.

Пример:

read (nab, r), где **r** – обычная переменная вещественного типа.

Запись в файл производится оператором:

```
write(имя_переменной1, имя _переменной2) ;
```

Операторы **writeln** и **readln** для записи и чтения в типизированном файле использовать нельзя, так как данный файл не содержит строк. Поэтому при попытке их использования компилятор выдаст сообщение об ошибке.

Типизированные файлы, ввиду упорядоченности их структуры, удобно использовать для создания электронных справочников, что мы и рассмотрим на следующем примере.

Известно, что в математике, физике и других точных науках для обозначения различных величин широко используются буквы греческого алфавита. Составим программу, которая облегчит пользователю запоминание названий этих букв и порядка их расположения в алфавите. Эта программа должна по порядковому номеру буквы в алфавите определять ее название, и, наоборот, по названию буквы указывать ее положение в алфавите.

Поставленную задачу мы будем решать в два этапа. Вначале создадим типизированный файл, элементами которого будут строковые переменные – названия букв греческого алфавита. Этот файл, имеющий имя **alphabet** и находящийся в каталоге **obuch** на логическом диске **D** компьютера, мы создадим и заполним данными с помощью программы, текст которой приведен ниже.

```
program greek;  
Uses Crt;  
var  
    a:file of string; b:string; i,j:integer;  
begin
```

```

Clrscr;
assign(a, 'D:\obuch\alphabet');
rewrite(a);
for i:=1 to 24 do
begin
writeln('Введите название очередной буквы');
readln(b);
write(a,b)
end;
close(a);
Clrscr;
reset(a);
for i:=1 to 4 do
begin
writeln;
for j:=1 to 6 do
begin
read(a,b);
write(b, ' ')
end;
end;
close(a);
readln;
end.

```

В начале мы связываем типизированный файл с файловой переменной **a** с помощью процедуры **assign**, затем открываем файл для записи. Если файла с таким именем на диске не существовало, то он будет создан программой. Процесс заполнения созданного файла данными мы производим в цикле с заранее заданным числом повторений. Всего тело цикла будет повторено 24 раза (по числу букв греческого алфавита). В самом теле цикла производятся следующие действия: название очередной буквы, введенное с клавиатуры, считывается оператором **readln** во вспомогательную переменную строкового типа **b**. Затем оператором **read** значение переменной **b** передается файловой переменной **a** (т. е. в конечном счете в типизированный файл **alphabet**).

Для того чтобы проконтролировать правильность заполнения файла **alphabet**, выведем его содержимое на экран компьютера. Элементы файла выводятся на экран в 4 строки, в каждой из которой должны быть видны названия 6 букв. Такой способ вывода осуществляется с помощью двух циклов, один из которых вложен в другой. Элементы цикла (названия букв)

считываются из файла в ту же вспомогательную переменную **b**. После окончания вывода данных эта программа завершает свою работу.

Данные из файла, созданного в программе **greek**, мы используем во второй программе, которая уже непосредственно будет выполнять поставленную задачу. Для того, чтобы разобраться в том, как работает эта вторая программа, необходимо предварительно рассмотреть используемые в ней стандартную процедуру и стандартную функцию, которые применяются для работы с типизированными файлами.

Процедура **seek** осуществляет прямой доступ к элементам типизированного файла. Общий вид данной процедуры:

```
seek (имя_переменной, n) ;
```

где

seek – служебное слово, которое в переводе с английского означает «искать»;

имя_переменной – имя файловой переменной;

n – порядковый номер элемента в файле (необходимо иметь в виду, что нумерация элементов в типизированном файле начинается не с единицы, а с нуля).

Результатом работы данной процедуры является то, что элемент файла с номером **n** становится текущим. При считывании данных оператором **read** будет считан именно этот элемент, при записи данных оператором **write** они также будут записаны в этот элемент.

Если в ходе работы программы необходимо установить, какой элемент файла является текущим, то для этого используется функция **filepos**.

Общий вид описания данной функции:

```
filepos (имя_переменной) : longint ;
```

Аргументом данной функции является имя файловой переменной, а значением – порядковый номер текущего элемента в файле.

Теперь можно приступить к рассмотрению второй программы, текст которой приводится ниже.

```
program grsearch;
```

```
Uses Crt;
```

```
label 10;
```

```
var a:file of string; s:char; b,c:string; l,m,n,x:longint;
```

```
begin
```

```
assign(a, 'D:\obuch\alphabet');
```

```
reset(a) ;
```

```
repeat
```

```
ClrScr;
```

```
10: writeln('Выберите режим работы программы');
```

```

    writeln('1 - определение названия буквы по ее номеру в
алфавите');
    writeln('2 - определение порядкового номера буквы по ее
названию');
    readln(m);
    case m of
        1: begin
            writeln('Введите порядковый номер буквы в
алфавите');
            readln(n);
            seek(a,n-1);
            read(a,b);
            writeln(n, ' буква греческого алфавита - ',b);
            end;
        2: begin
            writeln('Введите название буквы');
            readln(c);
            l:=0;
            repeat
                seek(a,l);
                read(a,b);
                l:=l+1
            until c=b;
            x:=filepos(a);
            writeln('Буква ',c,' является ',x,' буквой
греческого алфавита');
            end;
        else begin
            writeln('Вы ввели неверную цифру');
            goto 10
            end;
    end;
    writeln('Будете продолжать работу с программой(Д/Н) ');
    readln(s);
    until s='Н';
    close(a);
end.

```

В начале работы данной программы для чтения открывается типизированный файл **alphabet**, созданный предыдущей программой. Затем в программе используется цикл с постусловием. В этом цикле после

очистки экрана операторами **writeln** на экран выводится текст меню, позволяющий выбрать один из двух вариантов работы программы – определение названия буквы или определение порядкового номера путем ввода соответствующей цифры – 1 или 2. Эта цифра присваивается вспомогательной переменной **m**.

Затем выбор варианта работы производится с помощью условного оператора **case**, в котором роль переменной-селектора играет **m**. Перед началом работы условного оператора файловая переменная **a** связывается с типизированным файлом **alphabet**, а затем файл открывается для чтения процедурой **reset**.

В случае, если пользователь выбирает первый вариант работы программы, ему необходимо ввести порядковый номер буквы – **n**. Значение **n** используем в качестве одного из параметров процедуры **seek** (другим является имя файловой переменной), но при этом нужно учесть, что в алфавите нумерация букв начинается с единицы, следовательно, первой букве алфавита будет соответствовать нулевой элемент типизированного файла, второй букве – первый элемент, **n**-й букве – элемент с номером **n-1**. Поэтому, для того чтобы правильно определить нужный нам элемент файла, содержащий название буквы, в качестве входного параметра в процедуру вводим значение **n**, уменьшенное на единицу.

Затем найденный процедурой **seek** элемент типизированного файла (название буквы с порядковым номером **n**) передается оператором **read** во вспомогательную переменную **b**. Затем сообщение о номере буквы и ее названии выводится на экран компьютера операторами **write**.

Если же пользователь выбирает второй вариант работы, то вначале он должен ввести название буквы, порядковый номер которой ему надо определить. Это название считывается во вспомогательную строковую переменную **c**. Далее в программе открывается цикл с постусловием **repeat**. В этом цикле с помощью процедуры **seek** последовательно перебираются все элементы типизированного файла, начиная с нулевого. Входными параметрами данной процедуры являются имя файловой переменной и значение вспомогательной переменной целого типа **l**. Перед началом работы цикла переменной **l** присваиваем значение 0. В теле цикла каждый найденный процедурой элемент считывается во вспомогательную строковую переменную **b**, а значение переменной **l** увеличивается на единицу. Значение **b** сравнивается с «эталонным» значением, находящимся в переменной **c**.

Равенство значений этих двух переменных, свидетельствующее о том, что искомый элемент файла, соответствующий введенному названию буквы, найден, и является условием окончания работы цикла. Теперь осталось определить номер найденного элемента и увеличить его на единицу. Это и будет искомый порядковый номер буквы в алфавите. Данную операцию мы выполняем с помощью процедуры **filepos**, значение которой

присваивается целочисленной переменной **x**. Введенное название буквы и соответствующий ей порядковый номер, находящийся в переменной **x**, будут выведены на экран компьютера с помощью оператора **writeln**.

В том случае, если пользователь в ходе выбора вариантов работы ошибочно ввел не 1 или 2, а какую-либо другую цифру или иной символ, то выполняется та ветвь оператора **case**, которая находится после служебного слова **else**. В результате выполнения этой ветви выводится сообщение об ошибке, а затем пользователь возвращается к выбору возможных вариантов работы посредством использования оператора безусловного перехода.

После завершения работы оператора **case..of** пользователю предлагается на выбор возможность продолжить работу с программой или завершить ее. Этот выбор производится путем присваивания соответствующего значения символьной переменной **s**. Проверка значения этой переменной производится в последней строке цикла с постусловием. Если значение переменной **s** равно «Н», то цикл завершает свою работу, а если значение иное, то происходит повторное выполнение тела цикла с возможностью выбора варианта работы программы.

По окончании работы с файлом закрываем его процедурой **close** (это необходимо сделать вне зависимости от того, какой вариант работы программы мы выбрали, чтобы сохранить файл для дальнейшей работы).

1.3. Нетипизированные файлы

Нетипизированными называются такие файлы, в которых в отличие от двух рассмотренных выше видов, не указывается тип входящих в них элементов. Элементами одного и того же нетипизированного файла могут быть данные различных типов: целого, вещественного, символьного, строкового и других. Такие файлы используются для высокоскоростного обмена информацией между внешними файлами и программой, находящейся в оперативной памяти компьютера. При этом нетипизированные файлы, подобно типизированным, обеспечивают прямой доступ к данным.

Описание нетипизированной файловой переменной имеет следующий общий вид:

```
var имя_переменной:file;
```

где

имя_переменной – имя связываемой с нетипизированным файлом файловой переменной.

Пример описания:

```
var unt:file;
```

Связывание файловой переменной с файлом производится так же, как и для других файлов с помощью процедуры **assign**. Открытие файла для чтения и для записи производится, соответственно, процедурами **reset** и **rewrite**. Формат этих процедур также аналогичен формату других видов файлов. Операторы же, используемые для непосредственной записи информации в файл и для чтения информации из файла, отличаются от обычных операторов ввода-вывода, которые мы использовали в предыдущих случаях.

Для непосредственной записи информации в нетипизированный файл используется оператор **blockwrite**.

Общий вид данного оператора следующий:

```
blockwrite(имя_переменной1, имя_переменной2, n) ;
```

где

имя_переменной1 – имя файловой переменной;

имя_переменной2 – имя вспомогательной переменной, из которой данные передаются в файл;

n – **количество** элементов, в которые записывается информация за одно обращение к нетипизированному файлу. Значение **n** должно быть целого типа.

Пример использования данного оператора:

```
blockwrite(unt, y, 1) ;
```

где

unt – **файловая** переменная;

y – **вспомогательная** переменная (может относиться к любому из обычных типов);

1 - данный оператор обрабатывает один элемент.

Чтение информации из файла производится оператором **blockread**.
Общий вид данного оператора:

```
blockread(имя_переменной1, имя_переменной2, n) ;
```

где

имя_переменной1 – имя файловой переменной;

имя_переменной2 – имя вспомогательной переменной, в которую передаются данные из файла;

n – количество элементов, из которых считывается информация за одно обращение к нетипизированному файлу. Значение **n** должно быть целого типа.

Нетипизированные файлы удобно использовать для долговременного хранения результатов работы программ. Эти результаты могут включать в себя как численные величины, так и тексты, комментирующие их содержание. Создадим программу решения квадратного уравнения, в которой вводимые пользователем исходные данные и получающиеся при решении уравнения результаты, а также поясняющие их тексты передаются в нетипизированный файл, находящийся в другом каталоге, откуда в случае необходимости их можно прочитать. Текст модифицированной программы приведен ниже.

```
program qwurrez;
Uses Crt;
var
  r:file; a,b,c,d,x1,x2:real; k,x:string;
begin
  Clrscr;
  assign(r,'d:\TP7\rez');
  writeln('Введите коэффициенты уравнения');
  writeln('После ввода очередного коэффициента нажимайте
Enter');
  writeln('Введите a');
  readln(a);
  writeln('Введите b');
  readln(b);
  writeln('Введите c');
  readln(c);
  k:='Коэффициенты уравнения';
  d:=b*b-4*a*c;
  if d>=0 then
    begin
      x1:=(-b-sqrt(d))/(2*a);
      x2:=(-b+sqrt(d))/(2*a);
      x:='Корни уравнения';
      rewrite(r);
      blockwrite(r,k,1);
      blockwrite(r,a,1);
      blockwrite(r,b,1);
      blockwrite(r,c,1);
      blockwrite(r,x,1);
      blockwrite(r,x1,1);
      blockwrite(r,x2,1);
      writeln('Ввод данных в файл завершен');
```

```

close(r)
end
    else writeln('уравнение не имеет решения');
readln;
end.

```

В начале модифицированной программы, названной **qwurrez**, с помощью процедуры **assign** файловой переменной **r** «назначается» нетипизированный файл **rez**, находящийся на логическом диске **D** в каталоге **obuch**. Затем по уже известному нам алгоритму решается квадратичное уравнение **ax²+bx+c=0**. В том случае, если уравнение имеет решение, файл **rez** открывается для записи процедурой **rewrite**. Если файла с таким именем на диске не существовало, то он будет создан данной процедурой.

Далее в этот файл с помощью операторов **blockwrite** производится передача данных из текущей программы. Вначале в файловую переменную **r** передается текстовая строка, содержащаяся в переменной **k**. Эта строка поясняет, что далее в эту файловую переменную, т.е. в итоге в типизированный файл **rez**, будут передаваться исходные данные для решения задачи – коэффициенты уравнения. Эти коэффициенты передаются в файловую переменную из вещественных переменных **a, b** и **c**.

Затем в переменную **r** пересылается из строковой переменной **x** текстовая строка, в которой сообщается о том, что далее в файл будут записаны результаты решения задачи – найденные в программе корни уравнения. Эти корни находятся в вещественных переменных **x1** и **x2** и также передаются с помощью соответствующих операторов **blockwrite** в файловую переменную. После окончания передачи данных оператором **writeln** на экран компьютера выводится сообщение об этом.

Завершается процесс записи в файл, как и всегда, оператором **close**. В случае же, если уравнение не имеет решения, процесс создания файла с последующей записью в него данных не происходит.

Созданный в результате работы нетипизированный файл обладает тем же недостатком, что и типизированные файлы: его нельзя прочитать обычным текстовым редактором. Поэтому для того, чтобы проконтролировать правильность создания и заполнения информацией текстового файла, необходимо создать еще одну программу, которая позволяла бы прочитать текст созданного предыдущей программой файла **rez** с выводом его содержимого на экран компьютера. Эту контролирующую функцию и выполняет программа **rezread**, текст которой приведен ниже:

```

program rezread;
Uses Crt;
var r:file; x:real; k:string;

```

```

begin
  Clrscr; assign(r,'d:\TP7\rez');
  reset(r);
  blockread(r,k,1);
  write(k,' ');
  blockread(r,x,1);
  write(x:7:2,' ');
  blockread(r,x,1);
  write(x:7:2,' ');
  blockread(r,x,1);
  write(x:7:2,' ');
  writeln;
  blockread(r,k,1);
  write(k,' ');
  blockread(r,x,1);
  write(x:7:2,' ');
  blockread(r,x,1);
  write(x:7:2,' ');}
  close(r);
  readln
end.

```

Уже на первый взгляд заметно, что в данной программе используется меньшее количество переменных, чем в предыдущей. Это обстоятельство объясняется тем, что если запись информации в программе **qwurrez** производилась из 7 различных переменных вещественного и строкового типа, то для чтения информации из созданного предыдущей программой файла нам понадобятся всего две вспомогательные переменные – одна строкового типа и одна – вещественного. Следует обратить внимание на то, что каждая из этих переменных будет использована в программе несколько раз. Если бы все данные, содержащиеся в файле, были одного типа, то можно было бы обойтись вообще одной вспомогательной переменной, но в данном случае в нетипизированном файле содержатся данные двух разных типов, что и требует наличия для каждого типа своей переменной. Как следует из раздела описаний программы, помимо вспомогательных переменных в программе используется еще только одна переменная – файловая переменная **r**.

В начале основной части программы эта файловая переменная связывается с типизированным файлом **rez**. Затем файл открывается для чтения процедурой **reset**. Сами операции чтения информации из файла с передачей ее в соответствующую по типу вспомогательную переменную производятся операторами **blockread**, а последующий вывод значения переменной на экран – операторами **write**. Вывод информации на экран

производится в две строки. В начале первой строки посредством строковой переменной **k** выводится поясняющий текст, а затем с помощью вещественной переменной **x** по одному выводятся три коэффициента уравнения, т. е. данная строка содержит начальные условия задачи. Далее, с помощью пустого оператора **writeln**, производится перевод на следующую строку. Во вторую строку выводятся результаты решения задачи: сначала – текстовый комментарий, а затем два последних элемента файла **rez** – корни квадратного уравнения. Для вывода информации в эту строку используются те же переменные, что и для первой. Закрывать файл нужно в обязательном порядке и после операций чтения.

Задания для самостоятельной работы к главе 1

1. Составьте программу, которая создает на диске текстовый файл, записывает в него построчно информацию (количество строк заранее неизвестно, а признаком окончания ввода является ввод в конце очередной строки символа *) и подсчитывает количество строк, которое содержится в этом файле. Информация об имеющемся в файле количестве строк должна быть выведена на экран компьютера. Вывод должен быть произведен символами синего цвета на светло-сером фоне.

2. Дополните предыдущую программу таким образом, чтобы она в конец текстового файла дописывала информацию об имеющемся в файле количестве строк (включая последнюю строку с данными о количестве строк). Информация об имеющемся в файле количестве строк (включая последнюю) должна быть выведена на экран компьютера. Вывод должен быть произведен символами желтого цвета на синем фоне.

3. Составьте программу, которая создает на диске типизированный файл, содержащий **n** целочисленных элементов (количество элементов пользователь вводит с клавиатуры), записывает в файл **n** чисел, введенных пользователем, и подсчитывает сумму элементов, содержащихся в этом файле, а затем добавляет в файл еще один элемент – подсчитанную сумму. Для контроля правильности ввода данных предусмотреть в программе вывод всех данных из файла на экран компьютера. Вывод данных из файла должен быть произведен символами зеленого цвета на черном фоне.

4. Составьте программу, которая создает на диске типизированный файл, содержащий **n** целочисленных элементов (количество элементов пользователь вводит с клавиатуры), записывает в файл **n** чисел, введенных пользователем, а затем подсчитывает среднее арифметическое элементов файла, среднее геометрическое и среднее гармоническое. Последние три величины также добавляются в качестве элементов в типизированный файл. Для контроля правильности ввода данных предусмотреть в программе вывод всех данных из файла на экран компьютера. Вывод данных из файла должен быть осуществлен символами желтого цвета на зеленом фоне.

5. Составьте программу, которая создает на дискете нетипизированный файл, содержащий следующую введенную пользователем информацию о прочитанной им книге: номера глав и названия этих глав, причем номера должны записываться в файл как целочисленные элементы, а названия – как строковые. Количество глав в прочитанной книге пользователь вводит с клавиатуры. Для контроля правильности вводимых данных, следует вывести содержимое файла на экран компьютера, причем номер и название каждой главы должны выводиться в отдельной строке. Вывод данных из файла должен производиться символами светло-красного цвета на светло-синем фоне.

Глава 2. Работа с данными авторских типов

В ходе создания различных программ на языке Паскаль мы уже познакомились с самыми различными типами данных, которые используются для описания констант, переменных, значений функций.. В программах можно встретить числовые типы как целые, так и вещественные, символьный и строковый типы, используемые для работы с текстовой информацией, логический тип данных, специальные типы, созданные для описания файловых переменных.

Однако перечисленными типами не исчерпываются возможности Паскаля. Свобода творчества программиста в данном отношении не ограничивается рамками уже существующих в языке типов, так как на их базе можно создавать новые, «авторские» типы данных, которые впоследствии можно с успехом применять для решения многих задач. О том, какими могут быть эти нестандартные типы данных, для чего они применяются, как их создавать и работать с ними и будет изложено в этой главе.

2.1. Перечисляемый тип данных

Если программист хочет создать свой собственный тип данных, то один из способов создания заключается в том, чтобы перечислить все возможные значения, которые могут принимать переменные, относящиеся к данному типу. Такой тип данных и получил соответствующее название - перечисляемый.

Для того, чтобы создать такой тип данных его следует описать в программе в разделе описаний перед описанием переменных, так как среди переменных, которые далее будут описаны, могут быть и переменные нового, «авторского» типа. Описание нового, перечисляемого типа, в общем виде, выглядит следующим образом:

```
type имя_типа = (значение1, значение2, ... значение  
n) ;
```

где

type – служебное слово, в переводе с английского означающее «тип»;

имя_типа – имя вновь создаваемого типа;

значение1, значение2, ... значение n – перечень возможных значений для переменных нового типа (всего таких значений – **n**).

Пример описания нового типа:

```
type week=(Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
```

Данный тип имеет название **week** (что по-английски означает «неделя»). Переменные этого типа могут принимать всего 7 значений. Эти значения являются английскими названиями дней недели. Описание переменной, относящейся к данному типу, будет выглядеть следующим образом:

```
var ned:week;
```

т.е. переменная с именем **ned** относится к типу **week**. Вышеописанная переменная (как и все переменные, относящиеся к перечисляемым типам) может принимать только те значения, которые указаны в описании типа, к которому данная переменная относится. В нашем случае переменная **ned** может принимать значения **Monday**, **Tuesday** или **Wednesday**, но не может принимать, например, значение **Day** (по английски – день).

К недостаткам переменных перечисляемого типа следует отнести то, что их нельзя использовать в операторах ввода-вывода. При этом, однако, их можно использовать в ряде других операторов (в частности, в условных операторах и операторах присваивания). Переменная перечисляемого типа может использоваться в качестве переменной-селектора в условном операторе **case ... of**. Преимущества же использования переменных перечисляемого типа следующие: во-первых, использование таких переменных в ряде случаев делает программу более наглядной; во-вторых, используя переменные такого типа, легче защитить программу от неправильного ввода данных, что мы и продемонстрируем на следующем примере.

Требуется составить программу, которая по названию месяца определяет количество дней в этом месяце. Исходные данные вводятся бирюзовым цветом, результат сообщается сиреневым. В случае, если пользователь ввел в программу название месяца, которого не существует или другую неверную информацию, программа должна выдавать сообщение об ошибке.

В программе описан новый тип данных – перечисляемый тип **month**. Значениями этого типа являются сокращенные английские названия месяцев. В программе использована переменная **m** типа **month** и обычная переменная **s** строкового типа.

После ввода исходного данного в программе с помощью группы из двенадцати условных операторов устанавливается соответствие между русскими названиями месяцев и их сокращенными английскими эквивалентами: январь – **jan**, февраль – **feb** и т.д.

Обратите внимание, что в списке возможных значений, которые может применять переменная типа **month**, первым присутствует значение **error** (в переводе с английского – ошибка), которое не связано ни с каким

определенным названием месяца. Это значение применяется для защиты программы от неверного ввода данных. Как это делается, подробно описано ниже в этом же разделе.

Далее переменная **m** используется в качестве переменной-селектора в операторе множественного выбора, который непосредственно и выдает необходимую пользователю информацию.

Текст программы:

```
program month_1;
uses crt;
type
month=(error, jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec);
var m:month;
    s:string;
begin
clrscr;
textcolor(yellow);
writeln('Определение количества дней в месяце');
writeln('для невисокосного года');
textcolor(cyan);
writeln('Введите название месяца');
readln(s);
if s='январь' then m:=jan;
if s='февраль' then m:=feb;
if s='март' then m:=mar;
if s='апрель' then m:=apr;
if s='май' then m:=may;
if s='июнь' then m:=jun;
if s='июль' then m:=jul;
if s='август' then m:=aug;
if s='сентябрь' then m:=sep;
if s='октябрь' then m:=oct;
if s='ноябрь' then m:=nov;
if s='декабрь' then m:=dec;
textcolor(magenta);
case m of
jan, mar, may, jul, aug, oct, dec:      writeln('Количество
дней равно 31');
feb: writeln('Количество дней в месяце равно ', 28);
apr, jun, sep, nov: writeln('Количество дней в месяце
равно 30');
else writeln ('Ошибка ввода данных')
end;
readln;
end.
```

Обратите внимание, что представленная программа в значительной степени является упрощенной, так как в ней не рассматривается вариант с високосным годом, в котором в феврале на один день больше. Наличие такого варианта потребовало бы введения дополнительной переменной – номера года, а также добавления в программу расчета, который должен был по введенному номеру года определять, является он високосным или нет.

Для решения вышеуказанной задачи необходимо ввести в программу дополнительную функцию, которая по номеру года устанавливает, является ли данный год високосным или нет, и, соответственно, вычисляет количество дней в феврале.

Правило для определения високосного года следующее: год считается високосным, если номер года без остатка делится на 4 и не делится на 100. Согласно этому правилу, были високосными, например, 2008-й и 2012-й годы. Если же номер года делится на 100 без остатка, то этот год считается високосным только в том случае, если номер этого года также делится на 400. Поэтому 1800-й и 1900 годы не были високосными, а 2000-й был таковым.

Это правило в программе реализуется в виде алгоритма, содержащегося в функции **v**. Аргументом данной функции является номер года **y**, а значением – количество дней в феврале соответствующего года – 28 для обычного года и 29 для високосного. Информацию о том, является ли год високосным, содержит локальная переменная **flag** логического типа. Если значение **flag** равно **true**, то год – високосный, если **false** – то обычный.

Сам процесс определения високосности или невисокосности года в функции **v** содержится в условном операторе. Первая ветвь условного оператора, которая находится после слова **then**, содержит еще один вложенный условный оператор. Этот вложенный оператор необходим для анализа той ситуации, когда номер года делится без остатка на 100.

Ниже приводится дополненный вариант программы, который содержит указанную функцию **v**.

```
program month_k;  
uses crt;  
type  
month=(jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec)  
;  
var g:integer;  
    m:month;  
    s:string;  
function v(y:integer):integer;  
var flag:boolean;  
begin  
    if y mod 4 =0 then  
        begin
```

```

        if (y mod 100=0) and (y mod 400<>0)
        then flag:=false
        else flag:=true;
        end
    else flag:=false;
    if flag then v:=29
    else v:=28;
    end;
begin
    clrscr;
    textcolor(yellow);
    writeln('Определение количества дней в месяце');
    textcolor(cyan);
    writeln('Введите номер года');
    readln(g);
    writeln('Введите название месяца');
    readln(s);
    if s='январь' then m:=jan;
    if s='февраль' then m:=feb;
    if s='март' then m:=mar;
    if s='апрель' then m:=apr;
    if s='май' then m:=may;
    if s='июнь' then m:=jun;
    if s='июль' then m:=jul;
    if s='август' then m:=aug;
    if s='сентябрь' then m:=sep;
    if s='октябрь' then m:=oct;
    if s='ноябрь' then m:=nov;
    if s='декабрь' then m:=dec;
    textcolor(magenta);
    case m of
        jan,mar,may,jul,aug,oct,dec:      writeln('Количество
дней равно 31');
        feb:  writeln('Количество дней в месяце равно
',v(g));
        apr,jun,sep,nov:  writeln('Количество дней в месяце
равно 30');
        else writeln ('Ошибка ввода данных')
    end;
    readln;
end.

```

Перечисляемый тип данных можно использовать для различных целей, в том числе и для создания электронных словарей и справочников, что наглядно подтверждается на следующем примере.

Требуется составить небольшой англо-русский словарь по информационным технологиям, в котором содержались бы наиболее употребительные термины. Такой словарь, созданный средствами языка Turbo Pascal, можно хранить в памяти Вашего компьютера в виде программного файла. Достаточно ввести с клавиатуры интересующий Вас термин на английском языке, чтобы получить его русский эквивалент. Программу, решающую данную задачу, назовем **vocab** (сокращение от английского **vocabulary** – «словарь»). Текст программы приведен ниже.

```
program vocab;
Uses Crt;

type
    sl = (error, main, key, mouse, hard, soft, user,
power, cpu, sock, freq);
var
    wrd:sl;
    term:string;
begin
    ClrScr;
    writeln('Введите термин');
    readln(term);
    if term='mainboard' then wrd:=main;
    if term='keyboard' then wrd:=key;
    if term='mouse' then wrd:=mouse;
    if term='hardware' then wrd:=hard;
    if term='software' then wrd:=soft;
    if term='user' then wrd:=user;
    if term='power' then wrd:=power;
    if term='cpu' then wrd:=cpu;
    if term='socket' then wrd:=sock;
    if term='frequency' then wrd:=freq;
    case wrd of
        main: writeln('материнская плата');
        key: writeln('клавиатура');
        mouse: writeln('мышь');
        hard: writeln('аппаратура');
        soft: writeln('программное обеспечение');
        user: writeln('пользователь');
        power: writeln('питание');
        cpu: writeln('центральный процессор');
        sock: writeln('разъем');
        freq:writeln('частота');
```

```

    error: writeln('Такого термина в словаре нет')
end;
readln
end.

```

В данной программе перед разделом описания переменных описан новый тип данных – **sl**. Переменные данного типа могут принимать значения, представляющие собой англоязычные термины из области вычислительной техники (в ряде случаев в сокращенном виде). В случае же, если переменной данного типа в программе не присваивается никакое значение, то она получает по умолчанию первое по порядку из ряда значений, перечисленных в описании типа. В нашем случае, как и в предыдущей программе, таким значением по умолчанию будет значение **error** (по-английски – «ошибка»). В разделе программы, где описываются переменные, имеется переменная **word**, относящаяся к созданному в программе типу **sl**. Кроме того, в том же разделе описывается вспомогательная переменная **term** строкового типа.

В основной части программы после соответствующего приглашения на ввод во вспомогательную переменную **term** с клавиатуры вводится какой-либо термин. Затем в программе следует ряд условных операторов, каждый из которых проверяет, соответствует ли введенное слово одному из терминов, содержащихся в нашем электронном словаре. Если такой термин находится, то переменной-селектору **word** присваивается соответствующее значение из перечисленного ряда. Это значение используется в условном операторе **case...of**. В зависимости от значения переменной-селектора одна из ветвей этого оператора выводит на экран компьютера соответствующий англоязычному русскоязычный термин. Например, при вводе слова **mainboard** будет выдан его перевод – «материнская плата», для слова **frequency** – «частота» и т. д.

Если же пользователь сделал ошибку при вводе термина или ввел тот термин, который отсутствует в словаре, то переменная-селектор сохранит свое первоначальное значения - **error**. В этом случае условный оператор **case...of** осуществит выдачу сообщения:

Такого термина в словаре нет.

Конечно, в качестве значений переменной-селектора можно было бы указать и целые числа (1, 2, 3 и т.д.) и использовать их в условном операторе, но тогда программа лишилась бы, в значительной степени, своей наглядности.

Данную программу несложно дополнить, добавив в нее переводы каких-либо новых терминов. Эту операцию можно провести в три этапа:

1. Вставить английский термин, перевод которого Вы хотите добавить, в перечень значений типа **sl** (можно в сокращенном виде).

2. В основную часть программы добавить новый условный оператор **if**, в котором проверяется равенство вспомогательной переменной **term** данному термину (обязательно приведенному в полном виде), а переменной-селектору **word** в случае равенства присваивается значение, вставленное в список значений **sl**.

3. В условный оператор **case** вставить новую ветвь, в которой оператором **writeln** выводится русский перевод данного термина.

Теперь в Вашем электронном словаре появится перевод еще одного слова.

2.2. Ограниченный тип данных

При решении ряда задач из области программирования бывает удобно на основе уже существующего типа данных создать новый, который может принимать только строго определенный ряд значений. Например, оценка, выставленная учащемуся за контрольную или за экзамен, является целым числом, но переменная, которой будет присвоено значение-оценка может принимать далеко не все значения из ряда целых чисел. Во-первых, оценка не может быть отрицательным числом или нулем. Во вторых, она не может быть больше 5. Поэтому было бы удобно отнести переменную для оценки к особому типу данных, который может принимать только значения 1,2,3,4,5. Такой тип данных называется ограниченным.

Основой для создания ограниченного типа данных может быть не любой стандартный тип данных, а только порядковый. Порядковыми называются такие типы данных, в которых для каждого данного можно указать предыдущее или последующее (либо то и другое вместе). К порядковым типам данных относятся, в частности, типы **integer**, **longint**, **char**, **boolean**, а также перечисляемый тип данных. В качестве ряда значений для ограниченного типа данных берется какой-либо отрезок одного из порядковых типов. Такой ряд должен иметь начальное и конечное значения. Порядковый тип, на основе которого создается перечисляемый, является для вновь создаваемого типа базовым.

Общий вид описания ограниченного типа:

```
type имя_типа = диапазон;
```

где

type – служебное слово;

диапазон – диапазон возможных значений, которые могут принимать переменные и константы этого типа.

Примеры описания ограниченных типов данных:

```
type mark=1..5;
```

Базовым здесь является тип **integer**. Значениями переменных типа **mark** могут быть только целые числа от 1 до 5.

```
type latin='a'..'z' ;
```

Для данного типа базовым является тип **char**. Значениями переменных такого типа могут быть только строчные буквы латинского алфавита.

```
type weekend=Friday..Sunday ;
```

Для этого типа базовым является перечисляемый тип **week**, описанный в разделе 2.1 настоящего практикума. Значениями переменных типа **weekend** могут быть только переменные **Friday**, **Saturday** и **Sunday**.

Соответственно, используемые в программе переменные данных типов будут описаны таким образом **otmetka:mark; bukva:latin; day:weekend**. Над переменными ограниченного типа можно выполнять все те же арифметические и логические операции, что и над переменными соответствующего базового типа.

Ограниченные типы данных удобно использовать в программе для контроля вводимых данных. Например, в программе переменная **bukva** должна принимать только значения букв латинского алфавита. Пользователь же ошибочно присвоил данной переменной значение – русскую букву «с» вместо латинской «с» (эта ошибка часто встречается, так как обе буквы расположены на одной клавише компьютерной клавиатуры). В этом случае при компиляции сразу будет выдано сообщение об ошибке.

Разберем использование данных ограниченного типа на следующем примере. Нам уже известна задача нахождения факториала целого числа. Факториал представляет собой геометрическую прогрессию, значение которой с увеличением исходного числа возрастает очень быстро. Но если мы возьмем программу вычисления факториала и введем в качестве исходного значения число 40, то получим результат, равный нулю, т. е. результат явно ошибочный. В чем заключается причина того, что правильно составленная программа выдает неправильный результат? Оказывается, дело в том, что при вводе достаточно больших исходных значений происходит переполнение ячеек памяти, отведенных для результата, что и приводит к ошибке. Причем, это будет происходить не только, если мы опишем переменную, содержащую результат как переменную типа **integer**, но и для переменной типа **longint**.

Для того, чтобы избежать этой ошибки, следует сделать переменную, которой присваивается исходное значение, переменной ограниченного типа. Создадим в программе специальный ограниченный тип данных **ogr**, который может принимать только значения из диапазона от 1 до 12. К этому типу отнесем переменную **n**, в которую вводится исходное значение. Тогда для указанного диапазона программа будет выдавать правильные значения, а

при попытке ввести слишком большое значение компилятор прямо выдаст сообщение об ошибке. Ниже приведен текст программы:

```
program ftrl;  
Uses Crt;  
  type ogr=1..12;  
  var f,i:longint;  
      n:ogr;  
begin  
  ClrScr;  
  f:=1;  
  writeln('Введите число');  
  readln(n);  
  for i:=1 to n do  
    f:=f*i;  
  writeln('Факториал числа ',n,' равен ',f);  
  readln  
end.
```

К вышеописанной программе необходимо сделать следующее примечание. Возможна ситуация, когда Вы использовали в программе ограниченный тип данных, а программа все-таки «проглатывает» значения, которые выходят за границы его диапазона. Например, в программе вычисления факториала вводим исходное значение 20, а программа производит расчеты с выдачей, естественно, неправильного результата. Тогда нужно настроить компилятор среды Turbo Pascal таким образом, чтобы он автоматически контролировал значения переменных ограниченного типа.

Для этого в главном меню системы Turbo Pascal, расположенном в верхней части экрана, нужно войти в раздел **Options (Настройки)**, в нем выбрать пункт **Compiler... (Компилятор)**, в открывшемся диалоговом окне найти группу флажков **Runtime errors (Ошибки времени выполнения программы)** и в этой группе установить флажок **Range checking (Проверка диапазона)**, а затем щелкнуть экранную кнопку **Ok** для подтверждения сделанных настроек. В результате такой последовательности действий включится контроль значений переменных ограниченного типа.

Если режим работы компилятора сразу не изменится, то следует выйти из системы программирования и запустить ее снова, после чего новые настройки компилятора вступят в силу. Перед выходом из системы новые настройки рекомендуется сохранить с помощью находящейся в том же разделе **Options** команды **Save TURBO.TP (TURBO.TP** - это файл, хранящий информацию о настройках системы программирования Turbo Pascal).

Ниже приводится текст программы, которая вычисляет сумму цифр трехзначного числа. В программе на основе целочисленного типа создан новый, ограниченный тип данных **z3**, значениями которого могут быть только трехзначные целые числа, т. е. числа в диапазоне от 100 до 999. Именно к этому ограниченному типу в программе принадлежит переменная **x**, которой присваивается исходное значение:

```
program znak3_z;  
uses crt;  
type z3=100..999;  
var x:z3;  
    sum,r1,r2,r3,ostat:integer;  
begin  
  clrscr;  
  textcolor(yellow);  
  writeln('Введите трехзначное число');  
  readln(x);  
  r1:=x div 100;  
  ostat:=x mod 100;  
  r2:=ostat div 10;  
  r3:=ostat mod 10;  
  sum:=r1+r2+r3;  
  textcolor(green);  
  writeln('Сумма цифр данного числа составляет'  
,sum);  
  readln;  
end.
```

Если пользователь (при наличии вышеуказанных настроек в меню системы Turbo Pascal) попытается ввести число, меньшее чем 100 или большее, чем 999, то при попытке запустить программу на выполнение компилятор выдаст следующее сообщение:

Range check error (Ошибка проверки диапазона).

Наличие подобного сообщения говорит о том, что посредством использования ограниченного типа программа защищена от неправильного ввода данных.

Задания для самостоятельной работы к главе 2

1. Составить программу, которая по названию месяца определяет текущее время года. В программе следует использовать оператор множественного выбора. Для описания переменной-селектора в этом операторе использовать перечисляемый тип. Ввод данных должен производиться желтым цветом на черном фоне, вывод результата зеленым.

2. Составить программу, которая по названию месяца определяет количество дней в этом месяце. В программе следует использовать оператор множественного выбора. Для описания переменной-селектора в этом операторе использовать перечисляемый тип. Ввод данных должен производиться бирюзовым цветом на черном фоне, вывод результата фиолетовым.

3. Составить программу, которая по названию части света, определяет, какого цвета кольцо соответствует ей на эмблеме Олимпийских игр (Европа – синее кольцо, Азия – желтое, Африка – черное, Америка – красное, Австралия – зеленое). В программе следует использовать оператор множественного выбора. Для описания переменной-селектора в этом операторе использовать перечисляемый тип. Сообщение должно выводиться на светло-сером фоне цветом, соответствующим цвету кольца.

4. Составить программу вычисления суммы цифр положительного двузначного числа. Для защиты от ошибок при вводе использовать для исходного числа ограниченный тип данных (диапазон от 1 до 99). Ввод данных должен производиться желтым цветом на черном фоне, вывод результата – зеленым цветом.

5. Составить программу для вычисления суммы факториалов трех натуральных чисел, введенных с клавиатуры. Для всех исходных величин следует использовать ограниченный тип данных (диапазон от 1 до 12). Вычисление факториала необходимо оформить в виде отдельной функции. Ввод данных должен производиться красным цветом на черном фоне, вывод результата – светло-синим цветом.

Глава 3. Записи и множества

В предыдущей главе мы рассматривали создание авторских типов переменных. Но средства языка Паскаль позволяют программисту использовать и более сложные авторские наборы данных. Эти наборы могут представлять собой совокупность однородных элементов, а могут иметь и внутреннюю структуру, содержащие элементы разных типов.

Особенностям создания и описания таких групп данных, методам работы с ними и посвящена эта глава практикума.

3.1. Записи

В языке Паскаль широко используются массивы, которые позволяют работать с большими объемами информации. Но информация, содержащаяся в массиве, должна состоять из однородных элементов. На практике же часто приходится сталкиваться с задачами, которые сводятся к обработке разнородной информации. В качестве примера возьмем работу отдела кадров какого-либо предприятия, в котором нужно обрабатывать анкеты сотрудников данного предприятия. Каждая из таких анкет содержит различные данные о сотруднике: фамилию, имя и отчество сотрудника, год и место рождения, пол, образование, общий стаж работы, стаж работы на данном предприятии, занимаемая должность и т. д.

Если перед программистом стоит задача автоматизировать процесс обработки этих анкет, то он столкнется с тем, что ему нужно будет создать некую структуру, состоящую из ряда элементов, каждый из которых, в свою очередь, содержит данные различных типов. Рассмотрим один такой элемент – данные о сотруднике предприятия Иванове Сергее Петровиче, 1960 года рождения, уроженце Санкт-Петербурга, имеющем высшее образование, общий стаж работы которого составляет 20 лет, в том числе 10 лет на данном предприятии, который работает в настоящее время инженером. Для фамилии, имени, отчества, места рождения, образования понадобятся переменные строкового типа:

```
fam,im,otch,mesto,obr:string;
```

где в данном случае

```
fam:='Иванов'; im:='Петр'; otch:='Сергеевич';  
mesto:='Санкт-Петербург'; obr:='высшее';
```

Для пункта анкеты «пол» можно использовать переменную символьного типа, так как для указания пола хватит одной буквы:

```
pol:char;
```

в нашем случае **pol:= 'м'**;

Для работы с данными о годе рождения, общем стаже работы и стаже на данном предприятии нужны будут переменные целого типа:

```
god, stob, stpr: integer;
```

где в нашем случае **god:=1960; stob:=20; stpr:=10.**

Если же в анкете для специалистов с высшим образованием имеется, например, дополнительный пункт о среднем балле в дипломе, то придется ввести еще одну переменную вещественного типа (так как средний балл является, как правило, дробным числом).

```
ball: real;
```

Для инженера П.С. Иванова, имеющего в дипломе средний балл 4,78, **ball:=4.78.**

Естественно, что аналогичный набор данных разного типа будет в анкете у каждого сотрудника, работающего на предприятии. Для описания такого набора будет нужен специальный тип данных. Такой тип существует в языке Паскаль и называется записью. Запись состоит из ряда полей. Каждое поле представляет собой элемент записи, имеющий свой конкретный тип. Описание поля в общем виде выглядит следующим образом:

```
type имя_зап = record  
  имя_поля1: тип_поля1;  
  имя_поля2: тип_поля2;  
  .....  
  имя_поля n: тип_поля_n;  
end;
```

где

type, **record** и **end** – служебные слова (**record** в переводе с английского и означает «запись»);

имя_зап – общее имя записи;

имя_поля – имя одного из полей, составляющих запись.

Всего запись содержит **n** полей.

Возвращаясь к нашему примеру, можно сказать, что описание записи с анкетными данными по какому-либо сотруднику будет выглядеть следующим образом:

```
type anketa=record  
  god, stob, str: integer;  
  ball: real;  
  pol: char;
```

```
fam,im,otch,mesto,obr:string;
end;
```

т. е. здесь приведена запись, имеющая имя **anketa**, содержащая 11 полей, относящихся к 4 различным типам. Из вышеприведенного примера видно, что, как и в случае с описанием обычных переменных, однотипные поля можно перечислять через запятую, а после двоеточия указывать их общий тип.

Некоторые из полей, входящих в состав записи, могут, в свою очередь, сами быть записями, содержащими ряд полей. В частности, если в нашем примере в анкете требуется указать не просто год рождения, а точную дату рождения, включающую день, месяц и год, то эту информацию можно представить в виде записи **date**, содержащей поля **den**, **mes** и **gr**. В данном случае для описания даты и года употребим целый тип, а для месяца – строковый.

Таким образом, описание записи **date** будет выглядеть следующим образом:

```
date:record
den, gr:integer;
mes:string;
end;
```

причем вся эта структура является составной частью более крупной структуры **anketa**. Естественно, что в этом случае поле **god**, содержавшее в записи **anketa** год рождения, становится ненужным. Теперь измененная запись будет выглядеть следующим образом:

```
type anketa=record
stob,stpr:integer;
ball:real;
pol:char;
fam,im,otch,mesto,obr:string;
date:record
den, gr:integer;
mes:string;
end;
end;
```

В данном варианте записи присутствуют два служебных слова **end**, причем первый из них закрывает запись **date**, а второй всю запись **anketa** в целом. В свою очередь, запись **anketa** может быть элементом массива либо типизированного файла. Например, массив **otdel**, содержащий оформленные в виде записей анкеты сотрудников одного из отделов организации, будет описываться в программе в разделе переменных следующим образом:

```
var otdel:array[1..n] of anketa;
```

Данный массив содержит информацию о **n** сотрудниках отдела. Каждая запись будет содержать, в общей сложности, 10 полей, причем 9 полей являются простыми, а одно также является записью, содержащей 3 поля. Таким образом, с помощью типа «запись» можно создавать весьма сложные информационные структуры.

Информацию обо всех сотрудниках организации можно поместить в случае необходимости в специальный типизированный файл **kadry**. Описание файловой переменной **kad**, связанной в программе с файлом **kadry** будет выглядеть так:

```
var kad: file of anketa;
```

К такому типизированному файлу применимы все те же операции, что и к типизированному файлу, содержащему простые элементы. Естественно, что в разделе описаний описание самой записи должно предшествовать описанию того массива или файловой переменной, которые включают в себя данную запись.

В программах на языке Паскаль можно работать не только со всей записью в целом, но и производить определенные действия над отдельными полями (присваивать им значения, выводить эти значения на экран компьютера, сравнивать однотипные поля и т.д.). Обращение к полю в общем виде производится следующим образом:

```
имя_записи.имя_поля
```

Например, если инженер П.С. Иванов отработал в данной организации еще год после занесения его данных в компьютер и нужно прибавить этот год к его стажу, то это можно сделать следующими операторами:

```
anketa.stob:=21;            для общего стажа
```

```
anketa.stob:=11;           для стажа на данном предприятии.
```

Если в запись требуется поместить данные о точной дате рождения сотрудника (например, 22 августа 1960 года), то соответствующие операторы присваивания будут выглядеть так:

```
anketa.date.den:=22;        для дня рождения
```

```
anketa.date.mes:='август';   для месяца рождения
```

```
anketa.date.gr:=1960;       для года рождения.
```

Рассмотрим способы работы с записями на примере следующей практической задачи. Создадим электронный телефонный справочник, в который пользователь сможет вносить следующие данные о своих

родственников, друзьях, знакомых, коллегах по работе: их фамилии, имена, номер телефона, домашний адрес. Далее, по введенной фамилии, пользователь сможет получать из справочника необходимую ему информацию – телефон и адрес. Первая часть этой задачи решается с помощью программы, текст которой приведен ниже.

```
program telefon;
Uses Crt;
  type punkt=record
    fam: string; name: string; nomer:string; adres: string;
  end;
  var abon:file of punkt; v:punkt; k:char;
begin
  Clrscr;
  assign(abon, 'D:\obuch\sprav');
  reset(abon);
  if ioresult<>0 then rewrite(abon);
  repeat
    writeln('Введите фамилию абонента');
    readln(v.fam);
    writeln('Введите имя абонента');
    readln(v.name);
    writeln('Введите номер телефона абонента');
    readln(v.nomer);
    writeln('Введите адрес абонента');
    readln(v.adres);
    write(abon,v);
    writeln('Будете продолжать ввод данных? (Д/Н)');
    readln(k)
  until k='Н';
  readln
end.
```

В разделе описаний программы вначале описывается вновь создаваемый тип **punkt**, представляющий собой запись, содержащую 4 поля – для фамилии, имени, телефона и адреса, заносимых в справочник абонента. Каждое из вышеперечисленных полей относится к строковому типу. Затем в разделе переменных описывается файловая переменная **abon**. В связанном с ней типизированном файле **sprav** будут содержаться элементы типа **punkt**. Вспомогательная переменная **v** также относится к типу **punkt**. Еще одна вспомогательная переменная **k** является переменной строкового типа.

Далее в основной части программы после процедуры связывания файл **sprav** открывается процедурой **reset** для операций записи и чтения. В

случае же, если такого файла на диске не существует, он создается заново. Для проверки факта существования файла используется встроенная функция языка Паскаль **ioresult**. Данная функция используется для контроля за осуществлением операций ввода-вывода при работе с файлами (в том числе открытия и закрытия файла). Значением данной функции является целое число. В случае, если операция ввода-вывода была проведена успешно, функция **ioresult** принимает нулевое значение. В нашей программе после операции открытия файла и производится проверка на равенство или неравенство значения этой функции нулю. Если значение **ioresult** равно нулю, то операция открытия файла прошла успешно, и, следовательно, такой файл уже существует. Если же значение функции не равно нулю, то вероятно, такого файла на диске нет и его нужно создать командой **reset**.

После осуществления проверки можно переходить непосредственно к операциям ввода данных в файл. Процесс ввода представляет собой тело цикла **repeat..until**. В теле цикла после соответствующей подсказки, выводимой оператором **writeln**, пользователь вводит фамилию, имя, телефон и адрес добавляемого в справочник абонента. Информация по каждому из вышеуказанных пунктов оператором **readln** передается в соответствующее поле вспомогательной переменной **v**. Затем, после того как всем полям переменной-записи **v** присвоены значения, вся эта запись целиком передается в типизированный файл **sprav** посредством записи в связанную с ним файловую переменную **abon**. Затем программа запрашивает пользователя, будет ли он продолжать ввод данных. Полученный ответ, которым должна быть буква Д или Н, присваивается символьной переменной **k**. В завершающей строке цикла производится проверка условия его завершения. Этим условием является равенство значения переменной **k** букве Н. Таким образом, можно за один сеанс работы с программой ввести данные на нескольких человек.

Программа **telsprav** производит считывание информации из типизированного файла **sprav**, который был создан и заполнен информацией с помощью предыдущей программы. В разделе описаний программы описывается уже знакомый нам тип **punkt**, а также файловая переменная **abon** типа **punkt**, переменная целого типа **i** и переменная строкового типа **fff**. Текст программы приведен ниже.

```
program telsprav;  
Uses Crt;  
  type punkt=record  
    fam: string; name: string; nomer:string; adres:string;  
  end;  
  var  
    abon:file of punkt; v:punkt; i:integer; fff:string;  
begin
```

```

Clrscr;
assign(abon, 'D:\obuch\sprav');
reset(abon);
i:=0;
writeln('Укажите фамилию абонента');
readln(fff);
repeat
    seek(abon,i);
    read(abon,v);
    i:=i+1;
    if v.fam=fff then
        begin
            writeln('абонент ', v.fam, ' ', v.name);
            writeln('проживает по адресу ',v.adres);
            writeln('телефон абонента ', v.nomer)
        end;
until eof(abon);
readln
end.

```

Основная часть программы начинается со связывания файловой переменной с внешним файлом **sprav** и открытия этого файла для чтения. Далее, после вывода на экран соответствующей подсказки, пользователь должен ввести фамилию абонента, по которому нужна справка. Эта фамилия считывается в переменную **fff**. Затем по введенной фамилии производится поиск информации, представляющей собой запись – элемент типизированного файла.

Как и в предыдущей программе, для организации процесса используется цикл **repeat..until**, но содержание тела цикла здесь иное. Сам поиск осуществляется с помощью уже знакомой нам процедуры **seek**. Поиск ведется, начиная с нулевого элемента (поэтому параметру процедуры **i**, соответствующему номеру элемента, еще до начала работы цикла было присвоено нулевое значение). Значение каждого **i**-го элемента типизированного файла по очереди присваивается вспомогательной переменной-записи **v**. Затем в условном операторе **if** производится сравнение того из полей переменной **v**, которое соответствует фамилии (**v.fam**) с искомым значением содержащимся в переменной **fff**. Если эти величины совпадают, то на экран компьютера операторами **writeln** выводится содержимое всех полей, имеющих в данной записи – фамилия, имя, телефон и адрес абонента, т. е. интересующая пользователя информация. Работа программы, однако, на этом не заканчивается, так как чтение и сравнение информации, содержащейся в элементах файла, будет продолжаться до тех пор, пока в ходе этой процедуры не будет обнаружен

конец файла (**eof**). После обнаружения конца файла завершается работа цикла, а затем и работа программы. Таким образом, если в Вашем справочнике имеется несколько однофамильцев, то по всем этим людям будут найдены и выведены их данные.

Возвращаясь к тому примеру, с которого мы начали этот раздел, можно сказать, что аналогичным образом можно составить и программу, которая бы создавала файл с данными по каждому из сотрудников организации и по введенному с клавиатуры запросу выдавала бы о нем всю необходимую информацию. Но в связи с тем, что в организации может быть достаточно большое количество сотрудников, среди которых может встретиться немало однофамильцев, поиск информации лучше производить по какому-либо другому параметру, который для каждого работающего в данной организации человека является уникальным. Таким параметром может служить, например, табельный номер сотрудника. В таком случае в запись с данными о сотруднике следует добавить еще одно поле целого типа, в котором и будет содержаться табельный номер.

Описание этого поля будет выглядеть так:

```
tabel:integer;
```

а присваивание ему значения так:

```
anketa.tabel:=4567;
```

Поле **tabel** можно будет использовать в качестве ключевого, т. е. такого, по которому однозначно идентифицируется содержащая его запись.

3.2. Множества

При работе над различными задачами из области программирования часто возникает необходимость использования структур, которые представляют собой некий набор, состоящий из однородных элементов. При этом элементы данного набора не должны быть жестко закреплены в этой структуре в определенном порядке, как в массиве или в типизированном файле, а могут компоноваться различным образом. Примером такой структуры, с которой человек часто сталкивается в повседневной жизни, является набор букв того или иного языка. Для удобства изучения и использования в определенных целях (например, при создании справочников, энциклопедий, разного рода списков, ведомостей) буквы упорядочивают в виде алфавита, в котором за каждой буквой раз и навсегда закреплено строго определенное место. Алфавит, таким образом, представляет собой аналог массива. Но чаще буквы компонуют различным образом так, что из 33 букв кириллицы или 26 букв латинского алфавита, сочетая их по определенным правилам, можно создать деловое письмо и научный труд, техническую инструкцию и газетную статью, прозаический роман и поэму в стихах, и многое другое.

В программировании такая структура, состоящая из элементов одного типа, которые можно располагать по-разному, называется множеством. Общий вид описания переменной множественного типа в языке Паскаль следующий:

```
var имя_множества: set of тип;
```

где

var и **set** – служебные слова. Слово **set** в переводе с английского означает «набор, множество»;

имя_множества – имя создаваемого типа-множества, которое может дать по своему усмотрению программист;

тип – один из уже существующих типов, к которому относятся элементы вновь создаваемого множества.

В качестве базового типа может выступать любой из порядковых типов (см. раздел 2.2 настоящего практикума). Если набор элементов множества представляет собой какой-либо отрезок значений одного из порядковых типов, то можно указать только начальное и конечное значения, а между ними поставить символ «..**»**. Если же элементы, составляющие множество, не идут подряд в базовом типе, то эти элементы нужно перечислить в скобках через запятую.

Приведем примеры описания множеств:

```
var triznak: set of 100..999;
```

Это множество натуральных трехзначных чисел. Базовый тип для данного множества – **integer**.

```
var prop: set of 'А'..'Я' ;
```

Это множество прописных букв русского алфавита. Базовый тип – **char**.

Описание множества строчных букв русского алфавита будет выглядеть сложнее, так как строчные буквы от «а» до «п» располагаются в одном месте расширенной кодовой таблицы, а буквы от «р» до «я» - в другом.

```
var compon: set of (keyboard, motherboard, mouse, display)
```

Это множество устройств, входящих в состав персонального компьютера.

При описании множества необходимо учитывать два немаловажных момента. Во-первых в описании элементы множества не должны повторяться. Во-вторых, если множество создается на базе числового типа

данных, то значения элементов множества не должны быть, с одной стороны, отрицательными, а с другой - не превышать число 255.

Наряду с вышеописанными переменными типа «множества» в программах могут встречаться и постоянные множества. Элементы такого множества перечисляются в квадратных скобках: **[1,2,3,4,5,6,7,8,9]** – так будет выглядеть описание постоянного множества, состоящего из однозначных целых чисел. Множество может содержать несколько элементов, один элемент – **[1]** или вообще ни одного – **[]**. В последнем случае множество называется пустым.

Над множествами, так же как и над обычными переменными, можно выполнять ряд операций. При этом нельзя забывать о том, что хотя некоторые из этих операций обозначаются так же, как арифметические, это – особые операции, отличающиеся от обычных.

Операция, обозначаемая знаком *****, хотя и описывается тем же значком, что и операция умножения, называется и осуществляется по-другому – пересечение двух множеств. Результатом выполнения этой операции над двумя множествами является нахождение их общих элементов, принадлежащих как одному, так и другому множеству. Например, если мы присвоим переменной типа «множество» **A** значение **[20,30,40]**, что записывается в программе следующим образом:

A:=[20,30,40];

а переменной того же типа **B** присвоим значение **[30,40,50]**, что записывается как

B:=[30,40,50];

то в результате операции **C:=A*B** множество **C** получит следующее значение: **[30,40]**.

Если при операции над двумя множествами используется знак **+**, то эта операция называется объединением двух множеств. Результатом такой операции будет получение множества, в которое будут входить все элементы, содержащиеся хотя бы в одном из исходных множеств. В качестве примера возьмем два множества: **lat**, значением которого будет **[q,w,e,r,t,y]** и **rus**, значением которого будет **[й,ц,у,к,е,н,г]** (в качестве элементов данных множеств взяты буквы, составляющие название традиционной английской и русской раскладок клавиатуры компьютера). В результате операции **latrus:=lat+rus** множество **latrus** получит следующее значение: **[q,w,e,r,t,y, й,ц,у,к,е,н,г]**.

Операция, обозначаемая знаком **-** называется разностью двух множеств. В результате выполнения данной операции над двумя множествами мы получаем новое множество, содержащее те элементы первого, которые не входят во второе. Рассмотрим эту операцию на примере двух множеств **language** и **low**. Первому из этих множеств присвоим

значение, представляющее собой перечень различных языков программирования, а второе представляет собой список только тех языков, которые не являются языками высокого уровня. К последним относятся язык ассемблера и машинные коды. Операции присваивания будут выглядеть следующим образом:

```
language:=[basic,pascal,c,fortran,algol,assembler_1  
language,machine_code];  
low:=[assembler_language,machine_code];
```

Далее выполним операцию нахождения разности и результат присвоим множеству **high**:

```
high:=language-low;
```

В результате данной операции множество **high** получит следующее значение: **[basic,pascal,c,fortran,algol]**, т. е. данное множество будет содержать в себе перечень языков высокого уровня.

Наряду с вышеописанными операциями над множествами можно также производить сравнение множеств друг с другом. В сравнении также участвуют два множества. Для сравнения двух множеств используются операции отношения, которые обозначаются теми же знаками, что и операции сравнения обычных переменных, но имеют иной смысл. Результатом операции отношения является логическая величина, которая может принимать значения **true** или **false**.

Рассмотрим более подробно данные операции:

= - равенство двух множеств (все элементы первого множества совпадают с элементами второго);

<> - неравенство двух множеств (одно из множеств содержит хотя бы один элемент, отсутствующий в другом).

Пример использования данных операций: даны два множества **G** и **H**. Значение множества **G** – **[4,5,6]**; **H** также имеет значение **[4,5,6]**. Для данных множеств результат операции сравнения **G = H** будет равен **true**, а операции **G <> H** будет равен **false**. В случае же, если мы множеству **H** присвоим другое значение

```
H:=[6,7,8];
```

а **G** оставим без изменения, то результат операции **G=H** будет равен **false**, а операции **G <> H** будет равен **true**;

< – проверка на строгое вхождение первого множества во второе (т.е. все элементы первого множества должны одновременно входить во второе. Кроме того, второе множество должно содержать какие-либо дополнительные элементы.);

<= – проверка на нестрогое вхождение первого множества во второе (все элементы первого множества должны одновременно входить во второе. Помимо элементов первого множества второе множество может как содержать, так и не содержать какие-либо дополнительные элементы.);

> – проверка на строгое вхождение второго множества в первое;

>= – проверка на нестрогое вхождение второго множества в первое.

Пример использования операций вхождения: даны два множества **K** и **L**. Значение множества **K** равно **[10,15,20]**; множества **L** – **[10,15,20,25,30]**. Результаты операций **K<L** и **K<=L** будут равны **true**. Результаты операций **K>L** и **K>=L** будут равны **false**. Присвоим множеству **L** новое значение:

L:=[10,15,20];

Значение **K** оставим без изменения. После выполнения операции присваивания результаты операций **K<=L** и **K>=L** будут равны **true**, а результаты операций **K>L** и **K<L** будут равны **false**.

При работе с множествами часто используется еще одна операция, в которой участвуют не два множества, а множество и какой-либо элемент. Такая операция называется операцией определения принадлежности элемента множеству. Общий вид данной операции следующий:

элемент in [значение_множества];

где

элемент – элемент, относящийся к базовому для данного множества типу;

in – служебное слово, обозначающее в переводе с английского «в»;

значение_множества – список всех элементов, входящих в множество.

Операция имеет значение **true**, если элемент входит в данное множество (т. е. совпадает с одним из элементов множества), и **false**, если он в данное множество не входит.

Например, выражения **D in [A..E]** и **5 in [3,5,7,9]** имеют значение **true**, а выражения **D in [E..N]** и **5 in [7,9,11]** имеют значение **false**.

При осуществлении операций над множествами, так же как и при выполнении арифметических и логических операций над обычными переменными, соблюдается определенный приоритет.

Наибольшим приоритетом обладает операция *****.

Следующими по приоритету являются операции **+** и **-**.

Наименьший приоритет имеют следующие операции:

in, =, < >, <, <=, >, >=.

При необходимости изменить порядок выполнения операций над множествами, как и для обычных переменных, используют круглые скобки. Значение выражения, заключенного в скобки, вычисляется в первую очередь.

Принадлежность каких-либо элементов к определенному множеству широко используется для различных статистических подсчетов. Рассмотрим работу с множествами на примере программы, которая для любого введенного с клавиатуры текста, написанного на русском языке, определяет количество имеющихся в нем гласных и согласных букв. Такая и аналогичные ей программы могут, в частности, использоваться для исследований в области филологии. Текст программы приведен ниже.

```
program filolog;  
Uses Crt;  
var  
    gl,sog:integer; a:char;  
begin  
    ClrScr;  
    gl:=0;  
    sog:=0;  
    writeln('Вводите текст построчно. ');  
    writeln('Ввод каждой строки заканчивайте нажатием клавиши  
Enter');  
    writeln('Для завершения ввода всего текста введите 0 и  
Enter');  
    repeat  
        repeat  
            read(a);  
            if a in ['a','e','и','o','y','ы','э','ю','я',  
                'А','Е','И','О','У','Ы','Э','Ю','Я'] then gl:=gl+1;  
            if a in ['б'..'д','ж','з','й'..'н','п','р'..'т','ф'..'щ',  
                'Б'..'Д','Ж','З','Й'..'Н','П'..'Т','Ф'..'Щ'] then  
sog:=sog+1  
            until eoln;  
        until a='0';  
    readln;  
    writeln('Число гласных букв в тексте равно ',gl);  
    writeln('Число согласных букв в тексте равно ',sog);  
    readln  
end.
```

В программе, пользователь должен вводить текст построчно, а считывается текст оператором **read** посимвольно. Процесс считывания символов, содержащихся в строке, производится во внутреннем цикле **repeat..until**. Каждый считываемый символ проверяется условным оператором **if** на принадлежность к множеству гласных букв. Если вводимый символ принадлежит данному множеству, то значение переменной-счетчика **gl** увеличивается на единицу (в начале программы переменной-счетчику было присвоено нулевое значение). Второй условный оператор аналогичным образом проверяет принадлежность символа к множеству согласных букв. Переменной-счетчиком здесь является **sog**. Условием окончания внутреннего цикла является обнаружение признака конца строки. Нахождение конца строки осуществляется с помощью соответствующей стандартной функции **eoln**. Эта функция по умолчанию имеет значение **false**, а при обнаружении ею двух управляющих символов возврата каретки **CR** и перевода строки **LF** (см. раздел 1.1) получает значение **true**, что приводит к завершению работы внутреннего цикла.

Внешний цикл **repeat..until** повторяет процесс построчного ввода до тех пор, пока в ходе его работы не будет обнаружен признак конца всего текста. Таким признаком, завершающим работу цикла, является ввод нуля. После окончания работы цикла полученные статистические данные выводятся на экран компьютера операторами **writeln**.

Множества используются при решении математических задач, особенно тех из них, в которых для получения искомого результата требуется фильтрация данных, т. е. отсеивание ненужных данных для успешного решения задачи. В качестве примера рассмотрим задачу нахождения всех простых чисел в диапазоне от 1 до некоторого **n**. Как известно, простым числом называется такое, которое делится только на единицу и само на себя. Способ решения этой задачи был разработан еще в III в. до н. э. выдающимся древнегреческим ученым Эратосфеном и в честь него получил название «решето Эратосфена». В принципе, составить программу решения данной задачи можно и не прибегая к множественному типу, но его использование делает создаваемую программу более простой и наглядной.

Алгоритм решения данной задачи сводится к следующему. Вначале в последовательности натуральных чисел от 1 до **n** вычеркивается (или заменяется на нуль) каждое второе число, начиная с числа 2 (при этом само число 2 в последовательности остается, так как оно является простым). Затем из оставшихся в последовательности чисел берется первое по порядку после числа 2 ненулевое число. Это будет число 3. Далее в последовательности вычеркивается или заменяется на нуль каждое третье число, начиная с 3, кроме самого числа 3. Затем берется следующее по порядку ненулевое число (число 5), и производится аналогичная операция замены на нуль кратных ему чисел. Эти действия выполняются для всех чисел вплоть до числа, равного **n/2** (если число **n** нечетное, то берется целая часть числа **n/2**) В результате в интервале натуральных чисел от 1 до **n** останутся невычеркнутыми

(ненулевыми) только простые числа. Ниже приводится текст программы, реализующей алгоритм «решето Эратосфена».

```
program eratos;  
  Uses Crt;  
  label 1;  
  var  
    dpn:set of 1..255; i,j,k,l,n:integer;  
begin  
  ClrScr;  
  writeln('Программа вычисления простых чисел в интервале  
от 1 до n');  
  1: writeln ('Введите число n (натуральное число не больше  
255) ');  
  readln(n);  
  if n>255 then goto 1;  
  k:= n div 2;  
  dpn:=[2..n];  
  for i:=2 to k do  
    for j:=2 to n do  
      if (j mod i =0) and (j<>i) then dpn:=dpn-[j];  
  writeln;  
  writeln('Печатаем простые числа в интервале от 1 до ',n);  
  writeln;  
  write(1:4);  
      l:=2;  
  for j:=2 to n do  
    if j in dpn then  
      begin  
        write(j:4);  
        if l mod 10 =0 then writeln;  
        l:=l+1  end;  
  readln  
end.
```

В начале работы программы пользователь после соответствующего приглашения должен ввести число **n** - верхний предел диапазона, в котором будет производиться поиск простых чисел. Эта верхняя граница не может быть больше 255, так как в программе используется целочисленное множество. Если все же при вводе была совершена ошибка (было введено число большее, чем 255) то с помощью оператора безусловного перехода

пользователь отсылается к той строке программы, где указана данная граница.

Как видно из приведенного выше алгоритма, одна и та же последовательность действий повторяется для всех чисел от 2 до целой части $n/2$. Последняя величина в программе обозначена через **k** и вычислена перед началом работы основного алгоритма. Кроме того, множеству целых чисел **dpr** присваивается новое значение, равное диапазону, в котором далее будет производиться поиск простых чисел. В процессе дальнейшей работы программы из этого множества будут удалены все непростые числа, и в итоге **dpr** превратится в множество простых чисел из данного диапазона.

Процесс непосредственного определения множества простых чисел производится с помощью двух вложенных друг в друга циклов. Переменной внешнего цикла является **i**, которая меняет свои значения от 1 до **k**, т. е. эта величина принимает все значения, с помощью каждого из которых производится процедура вычеркивания «лишних» чисел, не относящихся к множеству простых. Сам же процесс удаления из множества этих чисел осуществляется во внутреннем цикле, переменная которого **j** принимает все значения из заданного в начале программы диапазона. Во внутреннем цикле производится проверка текущего значения **j** с помощью условного оператора. Если это текущее значение делится без остатка на текущее значение **i** и при этом не равно делителю, то, следовательно, оно не является простым, и его смело можно удалять из множества, простых чисел, что и делает соответствующий оператор. Данный оператор присваивает множеству **dpr** новое значение, которое равно предыдущему за вычетом удаляемого элемента.

В итоге работы двух циклов искомое множество сформировано, и остается только вывести его элементы на экран компьютера. Данный вывод производится в цикле с переменной **j**, значение которой изменяется от 1 до **n**. Но здесь можно столкнуться со следующей проблемой. Вывод элементов осуществляется в строку оператором **write** (если вывод производить оператором **writeln**, то каждый элемент будет выведен в отдельной строке, общее число которых в текстовом режиме составляет 25, и для большого диапазона все значения просто не поместятся на экране компьютера). Но так как мы заранее не знаем общее количество элементов и значение каждого из них, то может оказаться, что какое-либо число – элемент множества будет при выводе «разрезано» – одна или две его цифры окажутся на одной строке, а остальные – на другой, что, естественно затруднит восприятие пользователем результатов работы программы.

Выход из данной ситуации заключается в следующем: каждый из элементов найденного множества будет выводиться в отформатированном виде – под каждое из чисел будет отведено по 4 позиции, что вполне достаточно, так как даже самые большие из выводимых чисел могут быть трехзначными, и, кроме того, одна позиция резервируется под пробел между числами. Числа в программе выводятся по 10 в одну строку и в каждой

строке будет занято, таким образом, по 40 позиций. Всего же в каждой строке содержится 80 позиций для записи различных символов. Разбивка элементов по 10 производится с помощью следующего приема: в теле цикла, осуществляющего вывод элементов множества, используется вспомогательная переменная-счетчик *l*. Значение данной переменной увеличивается на единицу при выводе каждого очередного элемента множества. Следовательно, как только выводятся очередные 10 элементов множества, значение переменной *l* становится кратным 10. Эта кратность проверяется с помощью операции нахождения остатка при делении на 10. Если значение *l* действительно оказывается кратным 10, то курсор переводится на следующую строчку пустым оператором **writeln**. Перед началом работы данного цикла на экран выводится первое простое число - единица, а переменной *l* сразу присваивается значение 2 (соответствующее второму элементу множества).

Задания для самостоятельной работы к главе 3

1. Составить программу, которая создает типизированный файл и вводит в него данные о результатах сдачи экзаменов группой из *n* студентов. Данные по успеваемости каждого из студентов должны быть оформлены в виде записи. Каждая запись должна содержать следующие данные: фамилию и имя студента, номер зачетной книжки, его имя и оценки по следующим предметам: математика, физика, информатика, история. Количество учеников, сдающих экзамен, заранее неизвестно, поэтому после ввода очередной записи программа должна запрашивать пользователя о том, будет ли он вводить следующую запись. После завершения ввода данных на экран должно быть выведено сообщение о количестве записей, имеющих в файле, а затем должна быть выведена экзаменационная ведомость.

2. Составить программу, которая создает типизированный файл и вводит в него анкетные данные сотрудников предприятия. Анкета каждого из сотрудников должна быть оформлена в виде записи. Каждая запись должна содержать следующие данные: фамилию, имя и отчество сотрудника, табельный номер, год его рождения, занимаемая должность, образование, общий стаж работы, стаж работы на данном предприятии. После ввода очередной записи программа должна запрашивать пользователя о том, будет ли он вводить следующую запись. После завершения ввода данных на экран должно быть выведено сообщение о количестве записей, имеющих в файле.

3. Составить программу, которая из типизированного файла, содержащего анкетные данные по сотрудникам предприятия (см. п.2 задания), после ввода табельного номера сотрудника будет извлекать запись, содержащую соответствующую анкету, и выводить ее на экран компьютера.

4. Составить программу, которая должна определять, является ли введенный с клавиатуры символ цифрой, прописной буквой английского

алфавита, прописной буквой русского алфавита, строчной буквой английского алфавита, строчной буквой русского алфавита или каким-либо другим символом. При решении задачи использовать множественный тип данных.

Библиографический список

Бежанова М.М., Москвина Л.А. Практическое программирование. Приемы создания программ на языке Паскаль. – М.: Научный мир, 2000.

Емелина Е.И. Основы программирования на языке Паскаль. – М.: Финансы и статистика, 1997.

Культин Н.Б. Программирование в Turbo Pascal 7.0 и Delphi. – СПб.: БХВ, 1999.

Немнюгин С.А. Turbo Pascal: практикум. – СПб.: Питер, 2002.

Немнюгин С.А. Turbo Pascal: учебник. – СПб.: Питер, 2002.

Пестриков В.М., Маслобоев А.Н., Федоров О.К. Программирование в системе Turbo Pascal 7.0: учебное пособие /СПбГТУРП. – СПб., 2002.

Пестриков В.М., Маслобоев А.Н. Turbo Pascal 7.0. Изучаем на примерах. – 2-е изд., перераб. и доп. – СПб.: Наука и техника, 2004.

Пестриков В.М., Маслобоев А.Н. Основы программирования в системе Borland Delphi: учебное пособие/ СПбГТУРП. – СПб., 2004.

Пестриков В.М., Маслобоев А.Н. Delphi на примерах. – СПб.: БХВ, 2005.

Пестриков В.М., Маслобоев А.Н. Решение математических задач в Turbo Pascal: учебное пособие/ СПбГТУРП. – СПб., 2009.

Ставровский А.Б. Turbo Pascal 7.0: учебник. – Киев: БХВ, 2000.

Учебное издание

Артур Николаевич Маслобоев

Операции с файлами и нестандартными типами данных в Turbo Pascal

Учебно-методическое пособие

Редактор и корректор Н.П. Новикова

Техн. редактор Л.Я. Титова

Темплан 2013 г., поз. 94

Подп. к печати 28.10.13.

Формат 60х84/16.

Бумага тип. № 1.

Печать офсетная. 3,25 уч.-изд. л. ; 3,25 усл. печ. л.

Тираж 50 экз.

Изд. № 94.

Цена «С».

Заказ

Ризограф Санкт-Петербургского государственного
университета растительных полимеров,
198095, СПб., ул. Ивана Черных, 4.

технологического