А.Н. Маслобоев

Практикум по алгоритмическому программированию в системе Turbo Pascal
Часть II

Учебно-методическое пособие

Санкт-Петербург
2011

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ РАСТИТЕЛЬНЫХ ПОЛИМЕРОВ»

80-ЛЕТИЮ СПбГТУРП ПОСВЯЩАЕТСЯ

А.Н. Маслобоев

Практикум по алгоритмическому программированию в системе Turbo Pascal

Часть II

Учебно-методическое пособие

Санкт-Петербург
2011

ББК 32.97я7
П 286
УДК 681.3 (075)

Маслобоев А.Н. Практикум по алгоритмическому программированию в системе Turbo Pascal: учебно-методическое пособие / СПб ГТУ РП. — СПб., 2011. Ч.II. — 50 с.

В настоящем учебно-методическом пособии рассматриваются вопросы алгоритмического программирования 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 по направлению 151000 «Технологические машины и оборудование», изучающих дисциплину «Информационные технологии» по полной форме обучения: I курс (I и II семестры) и по сокращенной форме обучения: I курс (II семестр) и II курс (III семестр).

Рецензенты:
заведующий кафедрой ПМИ, д-р техн. наук, проф. В.М. Пестриков;
ведущий конструктор СКБ «Турбина», канд. техн. наук С.Г. Рыбаков.

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

Редактор и корректор Н.П. Новикова

Техн. редактор Л.Я. Титова Темплан 2011 г., поз. 103

Подп. к печати 26.12.2011 Формат 60x84/16. Бумага тип. № 1.

Печать офсетная. 3,25 уч.-изд. л. ; 3,25 усл. печ. л. Тираж 50 экз.

Изд. № 103. Цена «С». Заказ

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

Оглавление

Глава 5. Работа с массивами

Массив представляет собой упорядоченную последовательность однородных элементов. Элементами массива могут быть различные величины, как числовые (целые и вещественные), так и относящиеся к другим типам. Но при этом строго должно соблюдаться следующее правило: все элементы каждого отдельно взятого массива должны относиться к одному и тому же типу, что и называется однородностью массива. В массиве каждый элемент имеет свой порядковый номер, который называется индексом. Массивы бывают одномерные и двумерные. В начале рассмотрим более простые одномерные массивы.

5.1. Одномерные массивы

Примером простейшего одномерного массива является список учеников одного школьного класса или студентов одной группы. В этом случае элементами массива будут фамилии учеников или студентов, а индексами – номера учеников или студентов в списке. Важной характеристикой массива является его диапазон, т. е. пределы, в которых может изменяться значение индекса массива.

В случае использовании массива в программе, он предварительно должен быть описан в разделе описания переменных. Но описывается массив иначе, чем обычная переменная. В общем виде описание массива, состоящего из элементов-переменных, выглядит следующим образом:

```
var  имя_массива: array[a..b] of тип_элементов;
```

где **var**, **array** и **of** – служебные слова. **Array** означает массив, предлог **of** в данном случае – из, **a** и **b** – соответственно, нижняя и верхняя границы диапазона массива.

Приведем примеры описания массивов:

```
var a: array[1..20]  of  integer;
```

это – описание массива с именем **a**, который может содержать до 20 элементов, причем все эти элементы целого типа;

```
var st: array[1..10] of real;
```

в данном случае описан массив с именем **st** из 10 элементов вещественного типа.

В программе можно работать не только со всем массивом целиком, но и с отдельными его элементами. Для того чтобы обратиться в программе к какому-либо элементу массива, нужно указать имя массива и индекс содержащегося в нем элемента. Например, если Вы встретите в программе

следующую запись: **a[10]** , то она означает, что мы обращаемся к элементу массива **a** с порядковым номером 10.

Рассмотрим программу определения максимальной и минимальной температуры за месяц. Приведем текст данной программы и разберем эту программу:

```
program month;
 Uses Crt;
 var m:array[1..31] of real;
     i,k:integer;
     min,max,x:real;
begin
 ClrScr;
 writeln('Введите количество дней в месяце');
 readln(k);
 writeln('Введите среднюю температуру за 1-е сутки');
 readln(m[1]);
 min:=m[1]; max:=m[1];
 for i:=2 to k do
     begin
     writeln('Введите среднюю температуру за ',i,'-е
     сутки');
     readln(x); m[i]:=x;
     if m[i]<min then min:=m[i];
     if m[i]>max then max:=m[i];
     end;
 writeln('Максимальная среднесуточная температура
 месяца равна ',max:7:2);
 writeln('Минимальная среднесуточная температура
 месяца равна ',min:7:2);
 readln;
end.
```

Данные о температуре за каждый день в программе вводятся с клавиатуры и хранятся в массиве. Нижней границей диапазона массива будет 1, а верхней 31 (максимальное количество дней, которое может быть в месяце). Количество дней, фактически содержащееся в том месяце, данные по которому обрабатываются программой, вводится с клавиатуры и содержится в переменной **k**. Заполнение массива будет производиться в цикле с заданным числом повторений. Счетчик цикла **i** будет меняться от 2 до **k,** так как первый элемент массива заполняется не в цикле. (Почему так делается, будет объяснено ниже). В ходе работы цикла очередному элементу массива присваивается значение, равное температуре за **i**–й день месяца.

Если количество дней в месяце меньше, чем 31, то массив будет заполнен не до конца. Это вполне допустимо и никак не повлияет на работоспособность программы.

Для того чтобы определить максимальный и минимальный элементы массива (и соответственно, температуры месяца) используем следующий прием. Введем две вспомогательные переменные **min** и **max.** До начала работы цикла присвоим каждой из этих переменных значение, равное первому элементу массива (температура за первый день месяца). Затем каждый вновь вводимый (в цикле) элемент массива будем сравнивать с этими двумя переменными. Если значение элемента меньше, чем **min,** то вспомогательной переменной присваивается новое (минимальное на текущем этапе) значение, равное этому элементу. Если же значение элемента больше или равно **min**, то переменная остается без изменения. Для этого используется сокращенный условный оператор. Аналогично производится работа со вспомогательной переменной **max.**

В итоге, после завершения работы цикла, переменные **min** и **max** действительно будут содержать, соответственно, минимальное и максимальное значения температур, которые останется только вывести на печать оператором **writeln**.

Отметим еще две особенности массивов. Во-первых, элементами массива могут быть не только переменные, но и константы. Такой массив, естественно, описывается в разделе объявлений как константа, причем константы описываются перед переменными. В этом же разделе такому массиву присваиваются значения. Во-вторых, нижней границей диапазона массива необязательно должна быть единица. Важно только, чтобы значение нижней границы было меньше, чем верхней.

Работу с массивом, состоящим из констант, хорошо иллюстрирует программа, которая по введенному пользователем номеру года по европейскому летоисчислению определяет соответствующее ему название по традиционному китайскому календарю (для упрощения программы то обстоятельство, что начало китайского Нового года приходится на конец января или начало февраля, не учитывается). В китайском календаре, как известно, каждый год называется по какому-либо животному, причем эти годы образуют 12-летний цикл, по истечении которого названия годов цикла снова повторяются. Первым годом 12-летнего цикла считается год мыши.

Рассмотрим программу, которая решает задачу для второй половины XX в. и первой половины XXI в. по европейскому летоисчислению. Известно, что начало очередного года мыши приходилось на 1948 г. Для решения данной задачи создадим в программе массив из строковых элементов. Элементами массива являются строковые константы, каждая из которых содержит название одного из годов 12-летнего цикла. Нижней границей диапазона индекса данного массива является единица. Верхней границей – 12, т.е. номер последнего года цикла. Пользователь может ввести год, начиная с 1948 и заканчивая 2055 (последний год очередного цикла), и в ответ будет выведен строковый элемент массива с соответствующим индексом, т. е. название года.

Так как пользователь может ошибиться при вводе исходных данных, то в программе предусмотрена их проверка, которая производится при помощи условного оператора **if.** При этом проверяются сразу два условия. Вводимое число не должно быть меньше минимального или больше максимального. Для того чтобы оба условия проверялись параллельно, они объединены служебным словом **or**, что означает «или». Если нарушена или верхняя, или нижняя граница диапазона, пользователю сообщается о допущенной им при вводе ошибке.

Затем в программе определяется номер года в 12-летнем цикле, обозначенный переменной **k**. Для определения величины **k** воспользуемся следующим приемом. Из введенного пользователем номера года **y** вычтем число 1947 (количество лет, прошедших с начала нашей эры по европейскому летоисчислению до начала очередного цикла восточного календаря). Полученная разность представляет собой количество лет, которое прошло с начала очередного цикла. Далее найдем остаток от деления полученной разности на 12. Это и будет номер года в цикле.

Наконец, останется вывести на экран компьютера элемент массива с номером, равным значению переменной **k**. Таким образом мы получим искомое название года по восточному календарю. Ниже приведен текст данной программы:

```
program kalend;
Uses Crt;
 const vost:array[1..12] of string
=('мыши','коровы','тигра','кролика','дракона','змеи',
'лошади','овцы','обезьяны','петуха','собаки','свиньи');
 var k,y:integer;
begin
 ClrScr;
 writeln('введите год (с 1948 по 2055)');
 readln (y);
 if (y<1948)or(y>2055)
 then writeln('Вы ошиблись при вводе')
 else
   begin
   k:=(y-1947) mod 12;
   writeln (y,'году по европейскому летоисчислению
   соответствует');
   writeln ('год ',vost[k],' по китайскому
   календарю');
   end;
 readln;
end.
```

5.2. Двумерные массивы

Простейшим примером двумерного массива является таблица умножения, в которой результат умножения двух чисел определяется по номеру строки, соответствующей одному из сомножителей, и номеру столбца, соответствующему другому. Так же и в любом двумерном массиве элемент определяется по двум индексам.

В языке Паскаль в общем виде двумерные массивы описываются следующим образом:

```
var имя_массива: array[a..b,c..d] of тип_элементов;
```

где **a** и **b** – соответственно верхняя и нижняя границы диапазона значений для первого индекса; **c** и **d** - верхняя и нижняя границы диапазона значений для второго индекса.

Двумерный массив таким образом можно представить в виде таблицы, имеющей **b-a+1** строк и **c-d+1** столбцов. Пример описания двумерного массива:

```
var tabl: array [1..9,1..9] of integer;
```

таким образом описан целочисленный массив, содержащий 9 строк и 9 столбцов.

Рассмотрим основные приемы работы с двумерными массивами на примере программы **arrsum** , составленной для решения следующей задачи. Дан двумерный массив размерностью 4×5. Элементами данного массива являются вещественные числа. Требуется: заполнить массив произвольными числами, затем вывести содержимое массива на экран компьютера, подсчитать сумму элементов для каждого столбца, имеющегося в массиве, и сформировать из этих сумм одномерный массив. Ниже приводится текст программы.

```
program arrsum;
Uses Crt;
 var
   f:array[1..4,1..5]  of  real;  x:array[1..5]  of
real;  i,j:integer; s:real;
begin
 ClrScr;
 for i:=1 to 4 do
   for j:=1 to 5 do
     begin
     writeln('Введите ',j,' элемент ',i,' строки');
     readln(f[i,j])
     end;
 writeln('элементы двумерного массива');
```

```
  for i:=1 to 4 do
    begin
    for j:=1 to 5 do write (f[i,j]:7:2,' ');
    writeln
    end;
  writeln('элементы одномерного массива');
  for j:= 1 to 5 do
    begin
    s:=0;
    for i:=1 to 4 do
      s:=s+f[i,j];
    x[j]:=s;
    write(x[j]:7:2,' ');
    end;
  readln;
end.
```

В программе **arrsum** в разделе описания переменных описываются два массива. Первый из них – это двумерный массив **f,** который в дальнейшем будет заполнен числами вещественного типа. В данном массиве, как видно по описанию, имеется 4 строки и 5 столбцов. Второй описанный в разделе массив **x** является одномерным и в дальнейшем будет заполнен элементами, каждый из которых представляет собой сумму элементов, содержащихся в одном из столбцов двумерного массива. Всего в одномерном массиве **x** должно быть 5 элементов – по количеству столбцов в двумерном массиве **f.**

В начале программы производится заполнение массива **f** произвольными вещественными числами, вводимыми с клавиатуры компьютера. Этот процесс осуществляется с помощью двух вложенных циклов. Во внутреннем цикле производится поэлементное заполнение одной из строк массива числами. Этот цикл повторяется 5 раз в соответствии с количеством элементов в каждой строке (количество элементов двумерного массива, содержащихся в одной строке, равно количеству столбцов, имеющихся в данном массиве). Соответственно, и счетчик данного цикла – переменная **j** изменяет свое значение от 1 до 5.

Перед непосредственным вводом очередного элемента, производимым оператором **readln**, оператор **writeln** выводит приглашение на ввод данного элемента, указывающее номер текущей строки и порядковый номер элемента в строке. Внешний цикл с переменной **i** обеспечивает повторение всех действий внутреннего цикла 4 раза (по количеству строк, имеющихся в массиве).

Далее для того, чтобы наглядно убедиться в том, что двумерный массив **f** целиком заполнен данными, производим построчный вывод содержимого массива на экран компьютера. Эта операция, как и предыдущая, производится с помощью двух циклов, один из которых вложен в другой.

Внутренний цикл с переменной **j** обеспечивает вывод элементов одной строки. При этом, для того чтобы значения элементов массива представлялись на экране в удобном для восприятия человеком виде с фиксированной точкой (а не в нормализованном), элементы выводятся в отформатированном виде. Под каждый элемент отводится по 7 позиций, в том числе 4 под целую часть, 1 – под точку и 2 – под дробную часть. Внешний цикл с переменной **i,** как и в предыдущем случае, обеспечивает четырехкратное повторение действий внутреннего цикла.

Третья группа из двух вложенных циклов используется для подсчета суммы элементов каждого столбца двумерного массива и формирования нового одномерного массива. Внешний цикл в данном случае повторяет всю совокупность действий, производимых во внутреннем цикле, 5 раз – по числу элементов формируемого цикла (и, соответственно, столбцов двумерного массива). Внутренний же цикл с переменной **i** подсчитывает сумму элементов, содержащихся в одном из столбцов.

Эта сумма подсчитывается с помощью вспомогательной переменной **s.** Перед началом работы внутреннего цикла эта переменная обнуляется с помощью оператора присваивания – первого оператора внешнего цикла, а затем во внутреннем цикле к ее текущему значению каждый раз прибавляется значение очередного элемента из обрабатываемого столбца. В итоге работы внутреннего цикла в переменной **s** мы и получаем искомую сумму.

Далее, следующий оператор внешнего цикла присваивает получившееся значение соответствующему элементу одномерного массива. Наконец, последний из операторов внешнего цикла выводит значение этого элемента одномерного массива на экран. Таким образом, после завершения работы данного внешнего цикла массив будет заполнен данными, и поставленная задача будет решена.

5.3. Сортировка массивов

Одной из наиболее распространенных задач, с которыми приходится сталкиваться при обработке массивов, является сортировка массивов, т. е. расположение элементов массивов определенным упорядоченным образом. Можно встретить следующие варианты сортировки массивов:

1. Сортировка по возрастанию. В этом случае элементы массива располагаются таким образом, что в начале его располагается наименьший элемент, затем – следующий по величине элемент и так далее вплоть до наибольшего, то есть каждый следующий элемент должен быть больше предыдущего.

2. Сортировка по убыванию. В данном случае после проведения сортировки в начале массива оказывается наибольший по величине элемент, затем идет следующий по величине и так далее вплоть до наименьшего, т. е. каждый следующий элемент должен быть меньше предыдущего.

3. Сортировка по неубыванию. В массиве, отсортированном таким образом, каждый последующий элемент должен быть больше или равен предыдущему.

4. Сортировка по невозрастанию. В отсортированном по невозрастанию массиве каждый последующий элемент должен быть меньше или равен предыдущему.

Существует несколько различных способов сортировки массивов. Мы разберем один из самых простых способов, название которого – сортировка методом простого выбора. Суть этого способа в том случае, если нужно упорядочить массив по возрастанию, сводится к следующему: в массиве нужно найти максимальный по величине элемент и поставить его на последнее место в массиве. В результате максимальный элемент массива займет подобающее ему место. Тот элемент, который ранее находился в массиве на последнем месте, необходимо переместить на то место, которое ранее занимал в массиве максимальный элемент.

Далее работаем с группой элементов, которая начинается с первого элемента массива, а заканчивается предпоследним элементом. В этой группе также следует найти максимальный элемент и затем поставить его на последнее место в данной группе элементов, т. е. на предпоследнее место во всем массиве. Элемент, стоявший предпоследним, также помещается в массиве на освободившееся место. Затем рассматривается группа элементов без последнего и предпоследнего. В этой группе также находится максимальный элемент, который помещается на положенное ему последнее в группе место. Такие операции будут продолжаться до тех пор, пока каждый элемент массива не займет подобающее ему место. Последняя операция сведется к выбору максимального в группе из двух оставшихся элементов. Описанный метод реализован в программе, текст которой приведен ниже.

```
program vybsort;
Uses Crt;
 var
    m:array[1..10] of integer; i,k,max,n:integer;
 begin
 ClrScr;
 for i:=1 to 10 do
   begin
   writeln('Введите ',i,' элемент массива');
   readln(m[i]);
   end;
 for n:=10 downto 2 do
   begin
   max:=m[1];
   k:=1;
   for i:=2 to n do
     begin
```

```
      if m[i]>max then
        begin
        max:=m[i];
        k:=i
        end;
      end;
    m[k]:=m[n];
    m[n]:=max;
    end;
  writeln;
  for i:=1 to 10 do write(m[i],' ');
  readln
end.
```

Основная часть данной программы начинается с заполнения одномерного массива **m,** состоящего из десяти элементов целого типа. Заполнение массива производится в цикле с заранее заданным числом повторений. В теле данного цикла содержится два оператора. Оператор вывода, содержащий приглашение на ввод очередного элемента массива и указывающий его номер, и оператор ввода, заполняющий данный массив.

После заполнения массива начинается собственно процесс его сортировки. Этот процесс является однотипным, но при этом он производится в соответствии с вышеприведенным алгоритмом сначала для группы из десяти элементов, затем для девяти (без последнего элемента), потом для группы из восьми элементов и далее по убывающей. Поэтому процесс сортировки будет осуществляться в цикле с заранее заданным числом повторений и с убывающим значением переменной цикла **n**. Начальное значение данной переменной будет равно 10, а конечное – двум, так как на последнем этапе сортировки мы будем иметь дело с группой, состоящей только из двух элементов.

В самом теле цикла мы будем производить следующие действия. В начале вспомогательной переменной **max**, которая должна содержать значение максимального элемента в рассматриваемой группе, мы присваиваем значение первого по счету элемента массива. Одновременно мы присваиваем начальное значение и другой вспомогательной переменной **k**. Эта переменная должна содержать индекс максимального по значению элемента в рассматриваемой группе. Так как в начале поиска максимума мы условно считаем максимальным первый элемент, то переменной **k** мы присваиваем значение 1.

Далее для нахождения действительно максимального элемента в группе мы используемый внутренний цикл, вложенный во внешний цикл с переменной **n**. Переменной внутреннего цикла будет переменная **i,** показывающая текущее значение индекса элемента. В процессе поиска максимального элемента значение этой переменной будет изменяться от 2 до **n** – индекса последнего по счету элемента в группе. Для каждого элемента

группы, начиная со второго, мы сравниваем его значение с максимальным. Если значение данного элемента окажется больше значения переменной max, то переменной **max** присваивается новое значение, которое равно значению данного элемента массива **m[i]**, а переменной **k** также присваивается новое значение, которое равно **i** – индексу данного элемента. В итоге, по окончании работы внутреннего цикла, после перебора всех элементов группы переменная **max** будет содержать значение действительно максимального элемента группы, а переменная **k** – индекс этого элемента.

Теперь осталось в соответствии с алгоритмом поставить максимальный элемент на последнее место в группе, а последний элемент на то место, которое ранее занимал максимальный. Для этого мы элементу с номером **k** присваиваем значение последнего элемента массива **m[n],** а последнему элементу – найденное максимальное значение. Таким образом, в результате работы двух вложенных циклов мы получим исходный массив уже в упорядоченном виде. Последний имеющийся в программе цикл с переменной **i** выводит на экран компьютера в одну строку элементы упорядоченного массива **m**.

Задания для самостоятельной работы к главе 5

1. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Подсчитать количество положительных элементов массива и также вывести это значение на экран. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

2. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Подсчитать количество отрицательных элементов массива и также вывести это значение на экран. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

3. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Подсчитать среднее арифметическое положительных элементов массива и также вывести это значение на экран. Фон экрана должен быть черным. Ввод данных должен производиться бирюзовым цветом, вывод результатов – сиреневым.

4. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Подсчитать среднее арифметическое отрицательных элементов массива и также вывести это значение на экран. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

5. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку.

Подсчитать количество элементов массива, которые делятся без остатка на 3, и также вывести это значение на экран. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

6. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Подсчитать количество элементов массива, которые делятся без остатка на 5, и также вывести это значение на экран. Фон экрана должен быть черным. Ввод данных должен производиться бирюзовым цветом, вывод результатов – сиреневым.

7. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Определить индекс последнего положительного элемента в массиве. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

8. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Определить индекс последнего отрицательного элемента в массиве. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

9. Заполнить двумерный целочисленный массив размерностью 4*4 данными, введенными с клавиатуры. Подсчитать среднее арифметическое в каждой строке массива и вывести эти данные на экран в одну строку. Ввод данных должен производиться бирюзовым цветом, вывод результатов – сиреневым.

10. Заполнить двумерный целочисленный массив размерностью 4*4 данными, введенными с клавиатуры. Подсчитать среднее арифметическое в каждом столбце массива и вывести эти данные на экран. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

11. Заполнить двумерный целочисленный массив размерностью 4*4 данными, введенными с клавиатуры. Подсчитать количество положительных элементов в каждой строке массива и вывести эти данные на экран. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

12. Заполнить двумерный целочисленный массив размерностью 4*4 данными, введенными с клавиатуры. Подсчитать количество отрицательных элементов в каждой строке массива и вывести эти данные на экран. Фон экрана должен быть черным. Ввод данных должен производиться бирюзовым цветом, вывод результатов – сиреневым.

13. Заполнить двумерный целочисленный массив размерностью 4*4 данными, введенными с клавиатуры. Подсчитать количество четных элементов в каждой строке массива и вывести эти данные на экран. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

14. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Произвести сортировку элементов массива по убыванию. Отсортированный массив также вывести на экран компьютера. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

15. Заполнить одномерный массив из 10 целых чисел, вводя их значения с клавиатуры. Затем вывести все элементы массива на экран в одну строку. Создать второй массив, элементами которого должны быть только положительные элементы, взятые из первого. Произвести сортировку элементов второго массива по возрастанию. Отсортированный массив также вывести на экран компьютера. Фон экрана должен быть черным. Ввод данных должен производиться бирюзовым цветом, вывод результатов – сиреневым.

Глава 6. Обработка текста

До сих пор в настоящем практикуме мы рассматривали программы, в которых производилась обработка только числовых данных (программы из главы 1 ч. I практикума не являются исключением, так как в них использовались текстовые данные, но при этом производился их вывод, а не обработка). Но современный мультимедийный компьютер, как известно, представляет собой устройство, способное работать не только с числовой информацией, но и с другими ее видами (текстовой, графической, аудио- и видеоинформацией). В языке Паскаль также предусмотрена возможность обработки текстовых данных. Для работы с ними используются типы **`char`** и **`string`**, которые, соответственно, применяются для работы с данными символьного и строкового типов.

6.1. Символьные константы и переменные

В программах написанных на Паскале широко используются символьные константы и переменные. Значениями, как тех, так и других являются символы. Символами называются буквы, цифры, пробел, знаки препинания, знаки математических операций и некоторые другие, которые можно вводить с клавиатуры компьютера.

Как известно, центральный процессор компьютера может работать только с информацией, представленной в численном виде. Любая вводимая в компьютер информация другого вида должна быть обязательно преобразована в числовую форму или, как говорят, закодирована с помощью соответствующих чисел. Так обстоит дело и при вводе в компьютер и дальнейшей обработке в нем текстовой информации. Фактически компьютер работает не с буквами, пробелами, точками и запятыми, а с соответствующими им числовыми кодами. Когда пользователь при работе на компьютере использует готовые программы, такие как текстовые редакторы, электронные словари и переводчики, то ему не обязательно знать механизм обработки текстовых данных внутри компьютера. Программисту же необходимо представлять себе взаимосвязь между символом, отображаемым на экране компьютера, и хранящимся в памяти компьютера его числовым кодом.

Понятно, что если каждый программист будет изобретать свои собственные числовые коды для используемых в компьютере символов, то использовать созданные им программы для обработки текстов сможет только он сам. Поэтому возникла необходимость в создании единой, общепринятой системы представления символов числовыми кодами. Такая система была реализована в виде кодовой таблицы **`ASCII`**.

`ASCII` – это аббревиатура от ***`American Standart Code for Information Interchange`***, что в переводе с английского означает

«американский стандартный код для обмена информацией». Это – используемая во всем мире таблица для кодирования вводимой в компьютер текстовой информации. Данная таблица содержит буквы латинского алфавита (прописные и строчные), цифры от 0 до 9, символы арифметических операций и знаки препинания, а также различные специальные символы. Каждому из имеющихся в таблице символов соответствует свой числовой код. Например, прописной букве «N» латинского алфавита соответствует числовой код 78, строчной букве «n» - код 110 (строчные и прописные буквы имеют разные коды). Цифре «6» соответствует код 54, пробелу – код 32, запятой – код 44 и т.д.

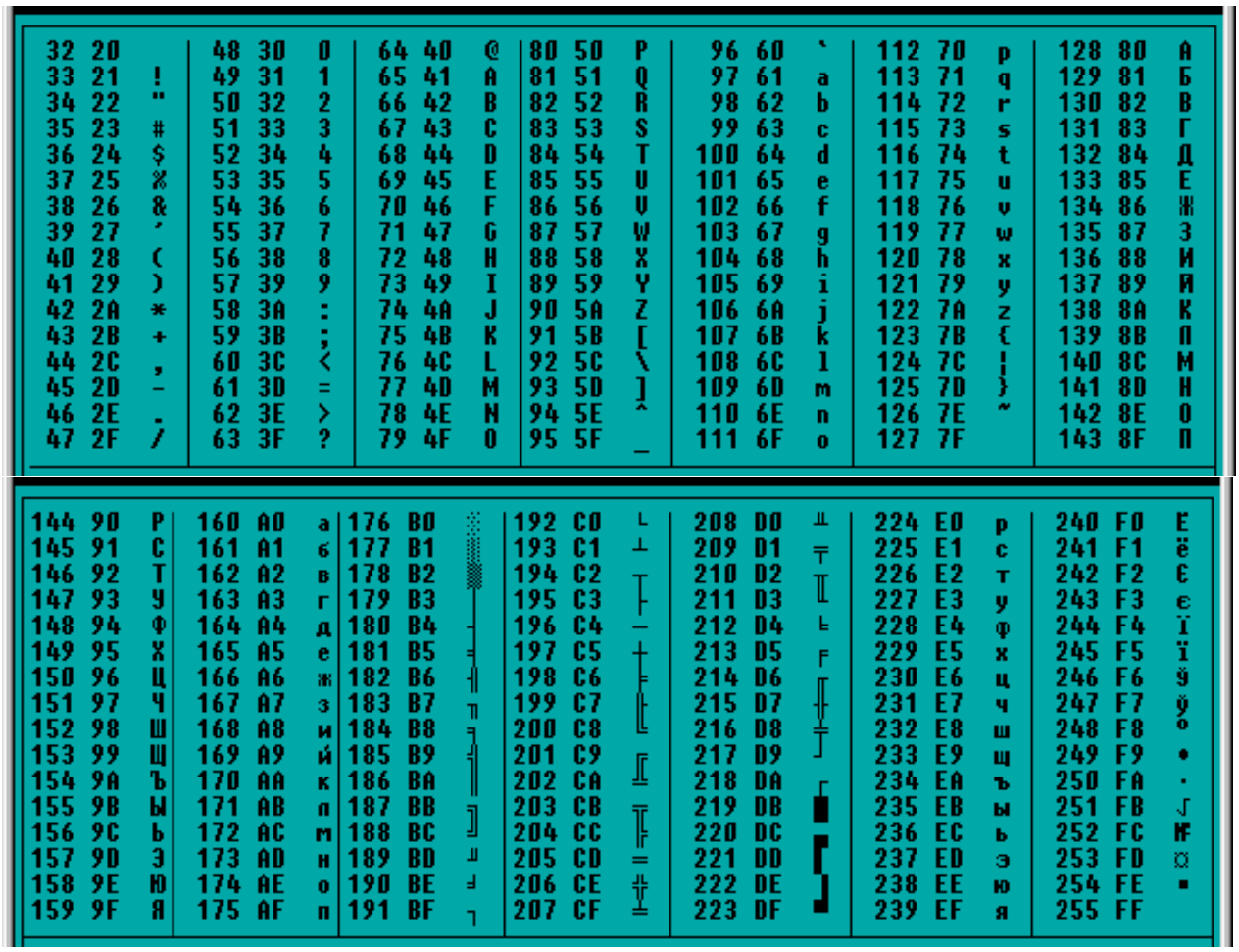
Кроме того, кодовая таблица содержит так называемые управляющие символы, которые при вводе не отображаются непосредственно на экране компьютера, но при этом выполняют определенные действия, например, производят перевод курсора на следующую строку при работе с текстом или выдают звуковой сигнал. Каждому из управляющих символов также соответствует числовой код. Всего эта таблица содержит 128 символов. Данная таблица является единой для всех IBM PC совместимых компьютеров.

К основной таблице ASCII существует дополнение, также содержащее 128 символов. Эта вторая половина таблицы зарезервирована для символов национальных алфавитов и, соответственно, для разных стран выглядит по-разному. В России во второй половине содержатся, естественно, прописные и строчные буквы кириллицы. Помимо символов кириллицы в этой части таблицы содержатся так называемые псевдографические символы, которые не имеют соответствующих им клавиш на клавиатуре компьютера, но могут быть выведены на экран путем набора их числового кода при нажатой клавише **Alt.** Содержит эта часть таблицы и некоторые специальные символы, отсутствующие в первой, международной части таблицы. К их числу относится радикал (знак квадратного корня). В этой части таблицы содержатся, например, коды для прописной буквы «Ж» - 134, строчной буквы «ж» -166, знака √ (радикал) – 251 и другие.

Следовательно, всего расширенная таблица, состоящая из международной и национальной части, содержит 256 символов. Обычно числовой код записывают в привычной всем нам десятичной системе счисления, но в ряде случае для его записи применяют и другую, шестнадцатеричную систему счисления, в которой, помимо цифр от 0 до 9, используют также буквы латинского алфавита от «A» до «F», обозначающие числа от 10 до 15. На рисунке приведены символы, входящие в таблицу ASCII, и соответствующие каждому из этих символов десятичный и шестнадцатеричный числовые коды.

В каждом из столбцов первая запись слева обозначает десятичный код символа, вторая запись – код этого же символа, записанный в

шестнадцатеричной системе счисления, широко применяемой в программировании. Справа в столбце изображен сам символ. *



32 20		48 30	0	64 40	@	80 50	P	96 60	`	112 70	p	128 80	А
33 21	!	49 31	1	65 41	A	81 51	Q	97 61	a	113 71	q	129 81	Б
34 22	"	50 32	2	66 42	B	82 52	R	98 62	b	114 72	r	130 82	В
35 23	#	51 33	3	67 43	C	83 53	S	99 63	c	115 73	s	131 83	Г
36 24	$	52 34	4	68 44	D	84 54	T	100 64	d	116 74	t	132 84	Д
37 25	%	53 35	5	69 45	E	85 55	U	101 65	e	117 75	u	133 85	Е
38 26	&	54 36	6	70 46	F	86 56	V	102 66	f	118 76	v	134 86	Ж
39 27	'	55 37	7	71 47	G	87 57	W	103 67	g	119 77	w	135 87	З
40 28	(	56 38	8	72 48	H	88 58	X	104 68	h	120 78	x	136 88	И
41 29	)	57 39	9	73 49	I	89 59	Y	105 69	i	121 79	y	137 89	Й
42 2A	*	58 3A	:	74 4A	J	90 5A	Z	106 6A	j	122 7A	z	138 8A	К
43 2B	+	59 3B	;	75 4B	K	91 5B	[	107 6B	k	123 7B	{	139 8B	Л
44 2C	,	60 3C	<	76 4C	L	92 5C	\	108 6C	l	124 7C	¦	140 8C	М
45 2D	-	61 3D	=	77 4D	M	93 5D	]	109 6D	m	125 7D	}	141 8D	Н
46 2E	.	62 3E	>	78 4E	N	94 5E	^	110 6E	n	126 7E	~	142 8E	О
47 2F	/	63 3F	?	79 4F	O	95 5F	_	111 6F	o	127 7F		143 8F	П

144 90	Р	160 A0	а	176 B0	▒	192 C0	└	208 D0	╨	224 E0	р	240 F0	Ё
145 91	С	161 A1	б	177 B1	▓	193 C1	┴	209 D1	╤	225 E1	с	241 F1	ё
146 92	Т	162 A2	в	178 B2	│	194 C2	┬	210 D2	╥	226 E2	т	242 F2	Є
147 93	У	163 A3	г	179 B3	┤	195 C3	├	211 D3	╙	227 E3	у	243 F3	є
148 94	Ф	164 A4	д	180 B4	┤	196 C4	─	212 D4	╘	228 E4	ф	244 F4	Ї
149 95	Х	165 A5	е	181 B5	╡	197 C5	┼	213 D5	╒	229 E5	х	245 F5	ї
150 96	Ц	166 A6	ж	182 B6	╢	198 C6	╞	214 D6	╓	230 E6	ц	246 F6	Ў
151 97	Ч	167 A7	з	183 B7	╖	199 C7	╟	215 D7	╫	231 E7	ч	247 F7	ў
152 98	Ш	168 A8	и	184 B8	╕	200 C8	╚	216 D8	╪	232 E8	ш	248 F8	°
153 99	Щ	169 A9	й	185 B9	╣	201 C9	╔	217 D9	┘	233 E9	щ	249 F9	∙
154 9A	Ъ	170 AA	к	186 BA	║	202 CA	╩	218 DA	┌	234 EA	ъ	250 FA	·
155 9B	Ы	171 AB	л	187 BB	╗	203 CB	╦	219 DB	▉	235 EB	ы	251 FB	√
156 9C	Ь	172 AC	м	188 BC	╝	204 CC	╠	220 DC	▄	236 EC	ь	252 FC	№
157 9D	Э	173 AD	н	189 BD	╜	205 CD	═	221 DD	▌	237 ED	э	253 FD	¤
158 9E	Ю	174 AE	о	190 BE	╛	206 CE	╬	222 DE	▐	238 EE	ю	254 FE	▪
159 9F	Я	175 AF	п	191 BF	┐	207 CF	╧	223 DF	▀	239 EF	я	255 FF	

Кодовая таблица ASCII

Знание числовых кодов символов понадобится в дальнейшем при составлении программ по обработке текстов, но большинство символов, которые не являются псевдографическими и управляющими, можно использовать в программе непосредственно в виде символьных констант. Употребляемые в программе символьные константы записываются так:

```
'A'  'y'  '?'  'щ',
```

т. е. запись символьной константы состоит из символа, заключенного в апострофы.

* Следует отметить, что в настоящее время кодовая таблица ASCII является не единственной, применяемой в информационных технологиях. Существует ряд других систем кодировки. Так, в современных текстовых процессорах таких, как Microsoft Word, и в других приложениях используется кодовая таблица Unicode. Данная таблица содержит более 65 тысяч символов. Этого количества достаточно, чтобы включить в данную таблицу практически все используемые сегодня на Земле алфавиты, а также множество различных специальных символов.

Если в программе задействована символьная переменная, она должна быть предварительно указана в разделе описаний подобно числовым переменным. Описание символьной переменной выглядит следующим образом:

```
var имя_переменной:char;
```

где **char** – служебное слово, обозначающее тип данной переменной, **char** – это сокращение от английского слова ***character*** , означающего в переводе «символ». Например, если в программе в разделе описания переменных мы видим:

```
var i,n: integer;
      d: real;
      sim,bukv: char;
```

то в данной программе используются две переменных целого типа (**i** и **n**), одна – вещественного типа (**d**), и две – символьного типа (**sim** и **bukv**). Если переменной символьного типа присваивается какое-либо значение, то оно также должно быть заключено в апострофы. Например:

```
sim:='ш';
```

означает, что переменной **sim** было присвоено значение буквы «**ш**». Возможен в программе и такой оператор присваивания:

```
cfr:='7';
```

наличие в программе такой записи говорит о том, что переменной **cfr** присваивается значение символа «7» (а не числа 7, как было бы в случае с числовой переменной).

Подобно обычным числам, символьные величины можно сравнивать. При этом фактически сравниваются не сами эти величины, а соответствующие им числовые коды. Например, можно записать такую операцию сравнения:

```
'л'<'н'
```

Результат этой операции будет истинным (равен **true**), так как код символа «**л**» равен 171, а код символа «**н**» равен 173. А если мы запишем следующее соотношение:

```
'к'>'т'
```

то такое соотношение будет ложным (**false**), так как код символа «**к**» равен 170, а символа «**т**» – 226. Можно проверять символьные величины также на

равенство или неравенство друг другу. Знаки для операций сравнения символьных величин используются те же, что и для числовых величин.

В языке Паскаль для работы с символьными переменными используются специальные стандартные функции **chr** и **ord.** Функции мы подробно рассмотрим в разделе 7.1 данного практикума. Пока же достаточно знать то, что стандартная функция по известной исходной величине, которая называется аргументом и заключается в скобки, находящиеся после имени функции, определяет некоторое значение.

Функция **chr** по известному числовому коду символа определяет сам этот символ. Общий вид функции:

```
chr(k)
```

где **k** – десятичный числовой код символа. Например, значение **chr(33)** будет равно **'!'**, **chr(233)** – **'щ'** и т. д.

Составим программу, которая будет выводить соответствующую группу символов для заданного диапазона числовых кодов. Числовые коды находящихся в таблице ASCII символов могут изменяться от 0 до 255. Правда, символы с кодами от 0 до 31 являются управляющими и корректно отображаться на экране компьютера не будут. Поэтому условимся, что начальный код диапазона должен быть не менее 32, а конечный не более 255. Кроме того, для правильной работы программы необходимо, чтобы начальный код диапазона был меньше, чем конечный.

Для защиты от неправильного ввода данных используем следующий прием. Создадим в программе цикл с постусловием. После операторов, осуществляющих ввод исходных данных, ставим сокращенный условный оператор **if**. В условии, находящемся в заголовке этого оператора, с помощью операций **or** объединены все вышеперечисленные условия. Если хотя бы одно из этих условий будет нарушено, то управление передается на начало программы, цикл выполняется заново, и пользователь снова должен ввести исходные данные, соответствующие указанным в программе условиям. Если пользователь ввел правильные исходные данные, то цикл с постусловием завершает свою работу, и программа переходит к следующим операторам.

Далее в программе осуществляется непосредственно процесс вывода символов. Этот процесс организован в виде цикла с заранее известным числом повторений. Начальным значением переменной цикла **i** является введенный пользователем начальный код диапазона, конечным значением переменной, соответственно, – конечный код диапазона. Тело цикла в данной программе состоит всего из одного простого оператора. Это оператор вывода, который выводит на экран компьютера очередное значение функции **chr**, аргументом которой является текущее значение переменной цикла **i**. При каждом выполнении тела цикла на экран компьютера выводится символ, код которого на единицу больше кода предыдущего выведенного символа.

Чтобы вывести результат работы программы в одной строчке, используем в теле цикла оператор **write**. Приводим текст данной программы:

```
program symbols;
Uses Crt;
var i,a,b:integer;
begin
  repeat
  ClrScr;
  writeln('Введите начало числового диапазона (число от
  32 до 255)');
  readln(a);
  writeln('Введите конец числового диапазона (число от
  32 до 255)');
  readln(b);
  if (a<32) or (b<32) or (a>255) or (b>255) or (a>b)
  then
    writeln('Вы ввели неверные исходные данные');
  writeln('Для продолжения работы нажмите Enter');
  readln;
  until ((a>=32) and (a<=255)) and ((b>=32) and
  (b<=255));
  writeln('Числовым кодам от ',a,' до ',b);
  writeln('соответствуют следующие символы');
  for i:=a to b do
    write(chr(i),' ');
  readln;
end.
```

Действие функции **ord** противоположно функции **chr**. Аргументом данной функции является символ, а значением – десятичный код данного символа. Общий вид данной функции:

```
ord('s')
```

где **s** – символ, код которого мы определяем. Обратите внимание, что обрабатываемый функцией символ должен быть обязательно заключен в апострофы, иначе при компиляции программы с использованием данной функции будет выдано сообщение об ошибке. Поэтому запись функции должна выглядеть в программе следующим образом **ord('ж'),**

ord('ф') и т. д. Для символа «**ж**» функция выдаст код 166, для символа «**ф**» - 228.

Подобно целочисленной переменной, символьную переменную можно использовать в качестве селектора в операторе множественного выбора. Принцип использования символьной переменной тот же, что и для целочисленной переменной, только вместо числовых значений перед описанными в операторе вариантами действий ставятся любые символы.

Однако, при использовании символьных переменных в операторе **case..of** необходимо учитывать один нюанс. Когда в операторе множественного выбора перед одним из вариантов указывается соответствующее ему значение символьной переменной, то это значение должно быть обязательно заключено в апострофы. Как может выглядеть такой оператор **case..of** хорошо видно на следующем примере.

```
case letter of
'п':  writeln('привет!');
'з':  writeln('здравствуйте!');
'д':  writeln('добрый день!');
end;
```

В зависимости от значения, которое присваивается символьной переменной **letter**, компьютер будет выводить на экран различные варианты приветствия. Если символьной переменной присвоено значение «**п**», то на экране мы увидим слово «привет!», если значение будет равно «**з**», то компьютер напишет на экране слово «здравствуйте!», если значение будет равно «**д**», то компьютер напишет пользователю «добрый день!». Естественно, что для правильной работы оператора переменная **letter** должна быть описана в начале программы как переменная типа **char**.

Вышеописанные возможности символьных переменных используются в следующей задаче. Требуется составить программу, которая осуществляет ввод двух целых чисел. Над введенными числами по выбору пользователя может осуществляться одно из следующих действий: определение среднего арифметического, наибольшего общего делителя и среднего гармонического. Выбор варианта действия должен производиться путем нажатия соответствующей буквы: **а** – для среднего арифметического, **б** – для наибольшего общего делителя, **в** – для среднего гармонического.

Известно, что средним гармоническим чисел **m** и **n** называется число **s**, которое удовлетворяет следующему равенству:

$$\frac{1}{s} = \frac{1}{2} \cdot \left(\frac{1}{m} + \frac{1}{n} \right).$$

Путем несложных математических преобразований можно определить, что среднее гармоническое вычисляется по следующей формуле:

$$s = \frac{2}{\frac{1}{m} + \frac{1}{n}} \quad .$$

Ниже приводится текст программы, которая решает поставленную задачу:

```
program charcase;
Uses Crt;
label 10;
var i,k,m,n,d:integer; rez:real; ch,p:char;
begin
10: ClrScr;
    writeln('Введите два числа');
    readln(m);
    readln(n);
    writeln('над введенными числами можно выполнить
    следующие операции');
    writeln('а       -       определение       среднего
    арифметического');
    writeln('б   -   определение   наибольшего   общего
    делителя');
    writeln('в - определение среднего гармонического');
    writeln('для      выбора      операции      нажмите
    соответствующую букву');
    readln(ch);
    case ch of
     'а': begin
          rez:=(m+n)/2;
          writeln('среднее     арифметическое     равно
          ',rez:6:2)
          end;
     'б': begin
          d:=1;
          if m<n
          then k:=m
          else k:=n;
          for i:=1 to k do
           if (m mod i =0) and (n mod i =0)
           then d:=i;
          writeln('наибольший общий делитель двух чисел
          равен ',d)
         end;
      'в': begin
           rez:=2/(1/m+1/n);
```

```
        writeln('среднее гармоническое равно
        ',rez:7:3)
        end;
    end;
    writeln('будете продолжать вычисления (д/н)');
    readln(p);
    if p='д'then goto 10;
end.
```

Разберем текст данной программы. После команды очистки экрана с помощью операторов **readln** производится ввод двух целых чисел **m** и **n**. Затем на экран компьютера с помощью операторов **writeln** выводится текст меню, который сообщает пользователю о том, какие действия можно выполнить над введенными числами и какому действию соответствует нажатие какой буквы. Введенная пользователем буква присваивается символьной переменной **ch** с помощью оператора ввода **readln**.

Далее переменная **ch** используется в качестве переменной-селектора в операторе множественного выбора **case..of**. Оператор множественного выбора включает в себя в данном случае три возможных варианта действия, соответствующих трем возможным значениям переменной-селектора. Вычисление среднего арифметического и среднего гармонического производится по соответствующим формулам, а полученный результат присваивается переменной **rez**, значение которой затем выводится на экран компьютера.

Несколько сложнее производится определение наибольшего общего делителя двух чисел **d**. Как известно, наибольшим общим делителем двух чисел **m** и **n** называется такое наибольшее число **d**, на которое без остатка делятся оба эти числа. Понятно, что наибольший общий делитель не может быть больше, чем меньшее из двух исходных чисел. Поэтому в программе с помощью условного оператора **if** определяется меньшее из двух исходных чисел. Затем этот минимум присваивается переменной **k**. Таким образом, значение переменной **k** – это максимально возможное значение наибольшего общего делителя.

Затем для нахождения наибольшего общего делителя используется цикл с заранее известным числом повторений. В этом цикле перебираются все возможные значения наибольшего общего делителя, начиная с единицы и заканчивая **k**. Каждое из этих значений поочередно присваивается переменной цикла **i**. Затем проверяется, является ли текущее значение переменной **i** общим делителем для **m** и **n**, т.е., делятся ли одновременно оба этих числа без остатка на **i**. Для обеспечения совместной проверки двух условий используется логическая операция **and**. Если оба условия делимости выполняются одновременно, то текущее значение **i** присваивается переменной **d**. Затем переменной **i** присваивается новое значение, которое

на единицу больше предыдущего, и для него осуществляется аналогичная проверка. Таким образом, в результате работы цикла в переменной **d** будет содержаться максимальное значение общего делителя двух чисел, т.е. искомая величина, которая затем выводится на экран компьютера.

В заключительной части программы компьютер запрашивает пользователя о том, хочет ли он продолжать вычисления. Для ответа пользователь должен ввести букву «**д**» или «**н**». Если пользователь дал положительный ответ («**д**»), то управление программой передается на ее начало, обозначенное меткой 10. В результате производится очистка экрана и затем пользователю вновь предлагается ввести два числа, над которыми можно произвести одну из трех вышеуказанных операций. Если пользователь дал отрицательный ответ, то программа завершает свою работу.

6.2. Строковые константы и переменные

Естественно, что для эффективной работы с текстом необходимо уметь обрабатывать не только отдельные символы, но и слова, фразы, строки, т. е. группы символов. Для обработки таких групп символов в Паскале используется строковый тип данных. Для описания символьных переменных используется служебное слово **string**. Описание символьной переменной в общем виде выглядит следующим образом:

```
var имя_переменной: string[длина];
```

т.е. после слова **string** указывается длина описываемой переменной, то есть максимальное количество символов, которое может содержать данная переменная. Например, если в разделе описания переменных в программе мы видим:

```
var i:integer;
    x:real;
    stroka:string[25];
```

то это означает, что в данной программе наряду, с переменными целого типа **i** и вещественного типа **x,** используется и переменная строкового типа **stroka,** которая может содержать до 25 символов. Параметр, заключенный в квадратные скобки, не является обязательным и используется в том случае, если существуют причины, по которым необходимо жестко ограничить максимальную длину строковой переменной. Если же после имени переменной ее длина не указывается, то по умолчанию она считается равной 80 символам. В этом случае описание символьной переменной будет выглядеть так:

```
var stroka:string;
```

Значения символьных переменных так же, как и уже знакомые нам символьные константы (к текстовым константам относятся, в частности, сообщения, выводимые оператором **writeln**), должны обязательно заключаться в апострофы. Таким образом, оператор присваивания строковой переменной **stroka** значения «набор_символов» будет выглядеть следующим образом:

```
stroka:= 'набор_символов';
```

Подобно числовым переменным, строковые переменные можно складывать. Результат сложения также можно присваивать какой-либо символьной переменной. Например, если значение переменной **slovo1** равно «Турбо », а переменной **slovo2** – «Паскаль», то в результате операции:

```
slovo:=slovo1+slovo2;
```

значение переменной **slovo** станет равным словосочетанию «Турбо Паскаль». Можно прибавлять к строковой переменной текстовую константу. Например, если в программе имеется следующий оператор:

```
fraza:=slovo + 'версия 7.0';
```

то в результате значение строковой переменной **fraza** станет равным «Турбо Паскаль версия 7.0».

Рассмотрим работу со строковыми величинами на примере программы, которая спрашивает у пользователя, как его зовут, а затем здоровается с ним и желает ему успехов в изучении программирования. Ниже приводится текст данной программы:

```
program privet;
Uses Crt;
 var x:string[40];
begin
 ClrScr;
 writeln('Как Вас зовут?');
 readln(x);
 writeln;
 writeln('Здравствуйте, ',x,'!');
 writeln('Желаю Вам успехов в изучении
программирования!');
 writeln('С уважением. Ваш ПК');
 readln;
end.
```

В данной программе в разделе описания переменных присутствует только одна переменная **x** строкового типа. В основной части программы с помощью оператора **writeln** пользователю задается вопрос о его имени.

Введенное с клавиатуры имя с помощью оператора **readln** присваивается строковой переменной **x**. Максимальная длина строковой переменной принята равной 40, так как даже если пользователь в ответ на вопрос компьютера введет полностью свое имя, отчество и фамилию, то суммарная длина их не превысит 40 символов. Далее в программе ставится «пустой» оператор **writeln** для того, чтобы между ответом пользователя и последующим сообщением компьютера оставалась чистая строка.

Затем на экран выводится приветствие компьютера. В операторе **writeln** список вывода состоит из трех элементов. Первый – это заключенное в апострофы слово «Здравствуйте», то есть текстовая константа. Второй – это значение строковой переменной **x**, то есть введенное пользователем имя. Третий – это заключенный в апострофы восклицательный знак. В следующих строках программы операторами **writeln** завершается вывод текста приветствия.

Для работы со строковыми переменными в языке Паскаль используются различные функции, среди которых отметим функцию **length.** Аргументом данной функции является имя какой-либо символьной переменной, а значением – фактическое количество символов, которое содержится в данной переменной (эта величина может быть меньше максимальной длины, заданной при описании переменной). Так, значение функции **length,** аргументом которой является переменная **stroka** после выполнения вышеуказанного оператора присваивания, будет равно 14, а не 25, как указано в разделе описаний, так как фраза «набор_символов» содержит 14 символов.

Если есть необходимость работать не со всей переменной целиком, а с каким-либо отдельным содержащимся в ней символом, то к этому символу можно обратиться в программе непосредственно, указав имя содержащей его строковой переменной и его порядковый номер в этой переменной, заключенный в квадратные скобки. Например, если мы напишем в основной части программы (а не в разделе описаний) **stroka[5],** то работать мы будем с буквой «р», которая является пятым по счету символом в данной переменной.

При обращении к элементу строковой переменной в скобках после ее имени может указываться не только числовая константа, но и имя переменной целого типа. В этом случае номер «вызываемого» символа будет равен значению данной целочисленной переменной. Например, если мы присвоим целочисленной переменной **i** значение 10, а затем укажем в программе **stroka[i]**, то в результате будет «вызвана» буква «в», которая является в строковой переменной десятой по счету.

В качестве еще одного примера использования строковых переменных рассмотрим программу, которая для введенной с клавиатуры строки символов выводит на экран компьютера последовательность соответствующих им числовых кодов таблицы **ASCII.** Ниже приведен текст данной программы, которую мы далее будем разбирать.

```
program naborkod;
Uses Crt;
 var i,dl,n:integer;
     sl:string[50];
begin
 ClrScr;
 writeln ('Введите строку (не более 50 символов)');
 readln(sl);
 writeln('Данная строка содержит символы со следующими
 числовыми кодами:');
 dl:=length(sl);
 for i:=1 to dl do
  begin
  n:=ord(sl[i]);
  write(n,' ')
  end;
 readln;
end.
```

В начале данной программы пользователю предлагается ввести некоторую строку символов, длина которой не должна превышать 50. Данная строка символов, введенная с помощью оператора **readln**, присваивается строковой переменной **sl**. Далее с помощью известной нам функции **length** определяется фактическая длина введенной строки. Полученное числовое значение присваивается целочисленной переменной **dl**.

Вывод числовых кодов содержащихся в строке символов мы производим с помощью цикла с заданным числом повторений. Начальным значением переменной цикла **i** является единица, а конечным значением переменной цикла является значение переменной **dl**. Таким образом, в ходе работы цикла будут выведены значения десятичных кодов для всех содержащихся в строке символов, начиная с первого и заканчивая последним, порядковый номер которого равен значению переменной **dl**.

В теле цикла, производящего вывод десятичных значений, содержатся следующие операторы. Это, во-первых, оператор присваивания. В этом операторе целочисленной переменной **n** присваивается значение, которое равно числовому коду текущего символа, т.е. символа, который имеет в строке порядковый номер, равный текущему значению переменной цикла **i**. Числовой код определяется посредством известной нам стандартной функции **ord**, аргументом которой является текущий символ. Второй оператор тела цикла осуществляет вывод полученного числового кода на экран компьютера. Коды выводятся в одну строку. Для того чтобы выводимые коды не сливались друг с другом, в операторе вывода предусмотрено, чтобы после каждого выводимого числа ставился пробел. По завершении работы цикла завершается и работа всей программы.

Задания для самостоятельной работы к главе 6

1. Определить, является ли введенный с клавиатуры символ буквой латинского алфавита. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

2. Определить, является ли введенный с клавиатуры символ буквой кириллицы. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

3. Определить, является ли введенный с клавиатуры символ цифрой. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

4. Вывести на экран компьютера все псевдографические символы, содержащиеся в кодовой таблице ASCII. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

5. Преобразовать слово, введенное с клавиатуры строчными буквами кириллицы, в это же слово, написанное прописными буквами. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

6. Преобразовать слово, введенное с клавиатуры прописными буквами кириллицы, в это же слово, написанное строчными буквами. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

7. Преобразовать слово, введенное с клавиатуры прописными буквами латиницы в это же слово, написанное строчными буквами. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

8. Составить программу, которая подсчитывает число слов во введенной в компьютер с клавиатуры строке. Словом считается любая последовательность символов, которая отделена от других таких же последовательностей пробелом. Например, в предложении «персональный компьютер фирмы Apple» имеются 4 слова, отделенные друг от друга 3 пробелами. Как видно из приведенного примера, количество слов в строке можно определить как сумму имеющихся в строке пробелов плюс единица. Для упрощения задачи будем считать, что исходные данные введены корректно, т.е. лишних пробелов в строке нет. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

9. Составить программу, которая проверяет, является ли введенная с клавиатуры последовательность символов целым десятичным числом. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

10. Составить программу, которая проверяет, является ли введенная с клавиатуры последовательность символов целым числом, записанным в

двоичной системе счисления, если известно, что в данной системе счисления для записи числа используются только две цифры - нуль и единица. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

11. Составить программу, которая подсчитывает среднюю длину слова в строке, введенной пользователем с клавиатуры. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

12. Составить программу, которая подсчитывает количество русских и латинских букв во введенной пользователем строке. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

13. Составить программу, которая подсчитывает количество строчных и прописных букв в строке, введенной пользователем. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

14. Составить программу, которая удаляет лишние пробелы в строке, введенной пользователем, если таковые обнаружатся. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

15. Составить программу, которая подсчитывает количество гласных и согласных букв в строке, введенной пользователем. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

Глава 7. Подпрограммы

При разработке ряда сложных программ достаточно часто приходится сталкиваться с ситуацией, когда в различных частях программы выполняется одна и та же последовательность действий. В этом случае гораздо рациональней бывает оформить эту последовательность в виде отдельной части программы, которая называется подпрограммой, а затем обращаться к ней из основной части программы по мере необходимости.

Кроме того, сам процесс программирования значительно облегчается, если разбить одну сложную задачу на ряд более простых, а затем уже объединить эти части между собой. Этот метод разработки программ также требует использования подпрограмм, которые подразделяются на подпрограммы-функции и подпрограммы-процедуры.

7.1. Функции

Такие операции, как возведение числа в квадрат, извлечение квадратного корня, определение модуля (абсолютной величины) числа, вычисление тригонометрических функций, округление дробного числа до ближайшего целого, определение длины строковой переменной и многие другие, используются при решении самых разнообразных задач из области программирования.

Для того чтобы облегчить процесс составления программ, в языке Паскаль предусмотрены специальные функции, которые применяются для автоматического выполнения этих действий. Такие функции называются стандартными и являются неотъемлемой частью языка Паскаль.

Каждая из этих функций по вводимым в нее исходным данным определяет некоторый результат, который далее используется в программе. Исходные данные называются аргументом функции, а вычисленный результат – ее значением. При обращении к функции аргумент указывается в скобках.

К широко употребляемым стандартным функциям (кроме уже известных нам функций **`length`**, **`ord`** и **`char`**, используемых для работы со строковыми переменными) относятся также следующие:

- **`abs(x)`** – определяет абсолютное значение аргумента, которым может быть число или выражение целого или вещественного типа;
- **`arctan(x)`** – вычисляет арктангенс угла, значение которого выражено в радианах;
- **`copy(x,n,l)`** – выделяет в строковой переменной **`x`** группу символов, начиная с позиции с номером **`n`**. Длина группы равна **`l`**. Параметры **`n`** и **`l`** должны быть величинами целого типа;
- **`cos(x)`** – вычисляет косинус угла, значение которого выражено в радианах;

- **delete(x,n,l)** – удаляет из строковой переменной **x** группу символов, начиная с позиции с номером **n**. Количество символов равно **l**;
- **exp(x)** – вычисляет экспоненту аргумента, т. е. **e** в степени **x**;
- **int(x)** – определяет целую часть аргумента. Значением функции является величина вещественного типа;
- **ln(x)** – вычисляет натуральный логарифм аргумента (т. е. логарифм по основанию **e**);
- **pi** – данная функция не имеет параметров, а значением ее является число «Пи»;
- **round(x)** – округляет значение аргумента до ближайшего целого числа;
- **sin(x)** - вычисляет синус угла, значение которого выражено в радианах;
- **sqr(x)** – вычисляет квадрат аргумента, которым может быть число или выражение целого или вещественного типа;
- **sqrt(x)** – вычисляет квадратный корень из аргумента;
- **trunc(x)** – определяет целую часть аргумента. Значением функции является величина целого типа.

В качестве примера программы с использованием стандартных функций рассмотрим программу решения квадратичного уравнения $ax^2+bx+c=0$. Текст программы приведен ниже:

```
program qwur;
Uses Crt;
 var a,b,c,d,x1,x2:real;
begin
 ClrScr;
 writeln('Введите коэффициенты уравнения - a,b,c');
 readln(a);
 readln(b);
 readln(c);
 d:=sqr(b)-4*a*c;
 if d<0 then writeln('корней нет')
        else
          begin
          x1:=(-b+sqrt(d))/(2*a);
          x2:=(-b-sqrt(d))/(2*a);
          writeln('Корни квадратного уравнения');
          writeln(x1:6:2,' ', x2:6:2)
          end;
```

```
 readln
end.
```

В данной задаче в начале с помощью стандартной функции **sqr** вычисляется дискриминант уравнения. В зависимости от значения дискриминанта выясняется, имеет ли данное уравнение решение. Если решения нет, то соответствующее сообщение выводится на экран компьютера. В случае же, если решение имеется, оно определяется с помощью стандартной функции **sqrt.**

Набор стандартных функций языка Паскаль достаточно обширен (он не ограничивается вышеприведенным списком), однако в ряде случаев программисту может потребоваться для решения поставленной задачи создать свою собственную функцию. Такая функция должна быть описана в тексте программы после раздела описания констант и переменных и до начала ее основной части (т. е. до слова **begin**). Такую функцию часто называют функцией-подпрограммой, так как она представляет собой отдельный блок внутри основной программы, который по своей структуре напоминает основную программу. Структура описания создаваемой программистом функции выглядит следующим образом:

```
заголовок функции;
раздел описания  констант и переменных, используемых внутри функции;
begin
операторы функции
end;
```

Теперь разберем более подробно элементы этой структуры. Общий вид заголовка функции следующий:

```
function имя_функции (параметры функции): тип значения;
```

где **function** – служебное слово, означающее «функция». Имя функции дается по тем же правилам, что и имена переменных. В скобках указываются аргументы функции, называемые ее ***параметрами*****,** причем для каждого параметра обязательно должен быть указан его тип. Если параметры относятся к одному типу, то они перечисляются через запятую, а после двоеточия указывается их общий тип, если же параметры относятся к разным типам, то они отделяются друг от друга точкой с запятой. После скобок обязательно указывается ***тип значения*** самой функции.

Пример заголовка функции:

```
function beta (x,y:integer; z:real):real;
```

данная функция имеет имя **beta,** в ней используются 3 параметра: **x** и **y** – целого типа**,** **z** – вещественного, а значение самой функции является вещественным.

Таким образом, заголовок функции в целом напоминает описание переменной, но следом за заголовком в описании функции указываются оператор или группа операторов, по которым вычисляется ее значение. Эти действия записываются в виде составного оператора, который начинается со служебного слова **begin** и заканчивается словом **end**. В составном операторе могут использоваться свои, локальные переменные. Раздел описания этих переменных (для каждой из локальных переменных также должен быть указан ее тип) помещается между заголовком и составным оператором. Не забывайте о том, что в составном операторе обязательно должен быть оператор присваивания, который присваивает получившийся результат функции.

В основной части программы действия, указанные в разделе описания функции, выполняются тогда, когда необходимо найти ее значение. Для этого необходимо осуществить ***обращение*** к функции. Обращение к функции в основной части программы включает в себя имя функции и следующий за ним список параметров, заключенный в скобки. Параметры, указанные при обращении к функции, называются ***фактическими*** параметрами, а параметры указанные в описании функции, – ***формальными***. Фактические параметры должны быть того же типа, что и формальные.

Например, правая часть оператора присваивания

```
d:=beta(3,4,7.5);
```

представляет собой обращение к функции **beta**, а 3, 4 и 7.5 – фактические параметры данной функции, в отличие от формальных параметров **x,y** и **z**. В качестве фактических параметров функции могут выступать и константы (как в вышеприведенном примере), и переменные. Обратите внимание, что все фактические параметры перечисляются через запятую, даже если они принадлежат к разным типам.

Для того чтобы на практике закрепить все вышеизложенное, рассмотрим программу, в которой описывается и применяется создаваемая пользователем функция. Это программа определения числа сочетаний из **n** по **m** . Данное число определяется по следующей формуле:

$$C_n^m = \frac{n!}{m!(n-m)!} \quad ,$$

где *n!*, *m!* и *(n-m)!* – соответственно, факториалы **n**, **m** и **(n-m).** Факториалом числа **n** называется произведение всех натуральных чисел от 1 до **n**. Разберем текст данной программы, приведенный ниже.

```
program sochet;
Uses Crt;
 var n,m:integer;
    a,b,c,d:longint;
 function faktor(k:integer):longint;
   var i:integer;  r:longint;
   begin
   r:=1;
   for i:=1 to k do
      r:=r*i;
   faktor:=r;
   end;
begin
 ClrScr;
 writeln ('Введите 2 натуральных числа для которых');
 writeln ('определяется число возможных сочетаний');
 writeln ('из первого по второму');
 readln(n,m);
 a:=faktor(n);
 b:=faktor(m);
 c:=faktor(n-m);
 d:=a div (b*c);
 writeln('Число возможных сочетаний из ',n,' по ',m,'
 равно ',d);
 readln;
end.
```

Так как в программе нам предстоит три раза вычислять факториал различных чисел, то в целях рационализации программы вычисление факториала оформим в виде отдельной функции **faktor**.

Для того чтобы было понятно, как работает данная функция, рассмотрим вначале алгоритм вычисления факториала. Проще всего начать вычисления с нахождения факториала единицы. Согласно определению факториала, он будет равен произведению единицы на единицу, т.е. единице. Далее определяем факториал двух. Для этого факториал единицы умножаем на два и получаем два. Факториал трех находим путем умножения факториала двух, т.е. двойки, на число три и получаем шесть. Таким образом, алгоритм нахождения факториала некоторого числа сводится к тому, что мы последовательно умножаем каждое следующее число на

факториал предыдущего. Понятно, что для многократного повторения аналогичных действий удобнее всего использовать циклическую структуру.

Следовательно, в функции вычисления факториала должны использоваться следующие переменные. Это переменная **k**, которой присваивается значение того натурального числа, факториал которого вычисляется. Далее будет использоваться переменная **r**, которой будет присвоено начальное значение, равное единице, а затем будут поочередно присваиваться значения факториалов всех чисел от единицы до **k** включительно. Наконец, мы используем переменную **i** – переменную цикла, в котором будет происходить это последовательное присваивание. Переменная **k** является параметром функции, а переменные **r** и **i** являются вспомогательными.

Тело функции вычисления факториала состоит из следущих операторов. Это оператор присваивания, в котором вспомогательной переменной **r** присваивается начальное значение, равное 1. Далее в теле функции содержится оператор цикла с заданным числом повторений. При каждом очередном выполнении тела цикла переменная **r** получает новое значение, равное произведению предыдущего значения этой же переменной на текущее значение переменной цикла **i** (в ходе работы цикла **i** будет изменять значение от 1 до **k**). Тело функции завершается оператором присваивания, в котором функция **faktor** получает значение, равное конечному значению **r**. Так как факториалы даже небольших натуральных чисел представляют собой достаточно большие величины (например, факториал числа 10 равен 3 628 800), то для описания значения функции используется тип **longint**.

В основной части программы осуществляется ввод исходных данных, затем производятся вычисления по формуле с использованием значений функции **faktor.** С помощью этой функции мы находим **a** – факториал числа **n**, **b** – факториал числа **m** и **c** – факториал разности этих чисел. Для вычисления этих величин в программе трижды производится обращение к функции **faktor** с различными фактическими параметрами. После нахождения **a, b** и **c** мы вычисляем **d** – искомое число сочетаний. Полученный результат оператором **writeln** выводится на экран.

7.2. Процедуры

Процедуры по своему внешнему виду напоминают функции, но если при обращении к функции мы можем получить только один результат – значение данной функции, то процедура может вырабатывать на выходе несколько значений. Кроме того, процедуры используются в программах для выполнения ряда стандартных действий. Такие процедуры, подобно стандартным функциям, являются частью языка Паскаль. К таким стандартным процедурам относятся уже известные нам команды **ClrScr**,

TextColor и **TextBackground**. Процедурами являются и операторы ввода и вывода **read, readln, write** и **writeln**. Наряду с этими стандартными процедурами в языке Паскаль широко используются также следующие:

- **Delay(i) -** осуществляет задержку выполнения программы на **i** миллисекунд (тысячных долей секунды);
- **Exit** - эта процедура не имеет параметров. Она осуществляет выход из процедуры или функции в основную часть программы (не путать с командой **Exit** меню системы программирования Турбо Паскаль, которая закрывает систему программирования и производит выход в среду операционной системы);
- **GotoXY(a,b)** - переводит курсор в точку экрана с координатами **a,b;**
- **Halt** - эта процедура не имеет параметров. Она завершает выполнение программы и передает управление операционной системе;
- **Str(x,s) -** производит преобразование числовой величины **x** в строковую **s**;
- **Val(s,x,n) -** производит преобразование строки **s**, изображающей число, в числовую величину **x**. В процедуре также должен присутствовать параметр **n**, относящийся к целочисленному типу. Если преобразование было выполнено успешно, то значение **n** будет равно нулю. Если преобразование не может быть произведено, то в переменную **n** записывается номер символа, который явился причиной ошибки при преобразовании.

В качестве примера использования стандартных процедур Паскаля приведем программу «обратный отсчет». Эта программа создает на экране компьютера «электронное табло», на котором в одном и том же месте последовательно выводятся числа от 10 до 1, т.е. производится обратный отсчет времени, как перед стартом космического корабля, а затем выводится слово «Старт». Разберем программу, текст которой приведен ниже.

```
program obrcount;
Uses Crt;
 var i:integer;
begin
 ClrScr;
 for i:=10 downto 1 do
   begin
   GotoXY(1,1);
   write(i);
   Delay(20000);
```

```
    ClrScr
    end;
 GotoXY(1,1);
 writeln('Старт');
 Delay(20000);
 ClrScr;
 readln
end.
```

В данной программе цифры выводятся с помощью цикла с уменьшающимся значением счетчика. В теле цикла используется ряд стандартных процедур. Так, перед выводом очередного числа курсор посредством стандартной процедуры **GotoXY** каждый раз перемещается в левый верхний угол экрана для того, чтобы все числа выводились в одной и той же позиции. Далее, для того чтобы очередное число сразу не исчезало с экрана, а оставалось на некоторое время, используется процедура **Delay**. Затем экран очищается с помощью процедуры **ClrScr**, для того чтобы освободить место для вывода следующего числа. После завершения работы цикла с помощью тех же стандартных процедур на экран выводится слово «Старт».

Конечно, возможности языка Паскаль не ограничиваются использованием только стандартных процедур. Программист, использующий этот язык, может создавать и свои собственные процедуры. Но для того чтобы такую процедуру можно было использовать в программе, ее, подобно вновь создаваемым функциям, следует предварительно описать. Описание процедуры, так же как и описание функции, должно содержаться в программе в разделе описаний после описания констант и переменных. Структура описания функции сходна со структурой основной программы, поэтому создаваемую программистом процедуру часто называют процедурой-подпрограммой. Описание включает в себя заголовок процедуры, раздел описаний и раздел операторов. К процедуре можно обращаться из основной части программы, из другой процедуры или из функции. Такое обращение называют также ***вызовом*** процедуры.

Общий вид описания процедуры следующий:

```
заголовок процедуры;
раздел описаний процедуры;
begin
раздел операторов процедуры
end;
```

Теперь разберем более подробно составные части данного описания.

Общий вид заголовка процедуры следующий:

```
procedure имя_процедуры (параметры процедуры);
```

где **procedure** – служебное слово, имя процедуры дается по тем же правилам, что и имена переменных в Паскале, параметры перечисляются в скобках через запятую с указанием их типа. Эти параметры являются формальными. При вызове же процедуры в обращении к ней указываются ее фактические параметры. Тип каждого фактического параметра должен быть таким же, как тип соответствующего ему формального параметра. Количество формальных параметров, имеющихся в описании процедуры, и фактических параметров, используемых при обращении к ней, должно совпадать. При этом первому по счету формальному параметру в описании ставится в соответствие первый по счету фактический параметр в обращении, второму формальному – второй фактический и так далее.

Все формальные параметры делятся на два вида. Если перед именем параметра в заголовке процедуры стоит служебное слово **var**, то это – ***параметр-переменная.*** Если служебное слово **var** перед именем переменной в заголовке отсутствует, то данный параметр является ***параметром-значением.*** При обращении к процедуре формальному параметру-значению присваивается значение соответствующего ему фактического параметра, причем в качестве такого значения может выступать константа, переменная или выражение. Во время работы процедуры параметр-значение не может изменяться даже в том случае, если он является переменной. При обращении же к процедуре, в которой имеются формальные параметры-переменные, соответствующие им фактические параметры могут быть только переменными (не константами и не выражениями). Вызываемая процедура получает доступ к ячейкам памяти, в которых хранятся эти фактические параметры, и может изменять значения этих параметров в ходе своей работы.

Пример заголовка процедуры:

```
procedure vspomog(a,b:integer; var c,d:real);
```

где **vspomog** – имя процедуры, **a,b,c,d** – имена формальных параметров, причем **a** и **b** являются параметрами-значениями, а **c** и **d** – параметрами-переменными.

В разделе описаний процедуры константы и переменные описываются по тем же правилам, что и в основной программе. При этом следует иметь в виду, что эти переменные могут использоваться только внутри данной процедуры. Такие переменные называются ***локальными***. В процедуре могут применяться и переменные, которые описаны в основной части программы. Такие переменные называются ***глобальными.***

Раздел операторов процедуры принципиально не отличается от такого же раздела в основной программе, он может содержать обращения к другим

функциям и процедурам, но после служебного слова **end** ставится не точка, а точка с запятой, так как конец описания процедуры – это не конец программы.

Рассмотрим программу определения наибольшего и наименьшего числа в группе из 4 чисел, текст которой приведен ниже.

```
program four;
Uses Crt;
 var a,b,c,d,l,m,big,lit,min1,max1,min2,max2:integer;
 procedure minmax(x1,x2:integer; var min,max:integer);
  begin
  if x1>x2
  then
    begin
    max:=x1; min:=x2
    end
  else
    begin
    max:=x2; min:=x1
    end;
  end;
begin
 ClrScr;
 writeln ('введите 1 число');
 readln(a);
 writeln ('введите 2 число');
 readln(b);
 writeln ('введите 3 число');
 readln(c);
 writeln ('введите 4 число');
 readln(d);
 minmax(a,b,min1,max1);
 minmax(c,d,min2,max2);
 minmax(min1,min2,lit,l);
 minmax(max1,max2,m,big);
 writeln('Наибольшее число - ',big,'; наименьшее число -
',lit);
 readln;
```

```
end.
```

Исходные значения введем с клавиатуры. Для определения наибольшего и наименьшего из них воспользуемся следующим алгоритмом. Напишем несложную процедуру **minmax**, которая определяет наибольшее и наименьшее из двух чисел. Затем разобьем введенные числа на пары и в каждой паре с помощью этой процедуры определим максимум и минимум. Далее с помощью той же процедуры **minmax** определим наибольший из двух максимумов и наименьший из двух минимумов. Это и будет искомый результат.

В заголовке процедуры **minmax** описаны 2 формальных параметра-значения **x1** и **x2**, которые используются для ввода в процедуру исходных данных, и 2 параметра-переменные **min** и **max,** используемые для вывода полученных результатов. Внутри процедуры параметры-значения сравниваются между собой, и значение меньшего из них присваивается переменной **min**, а большего - переменной **max.**

В основной части программы обращение к процедуре **minmax** встречается 4 раза. В первых двух случаях в качестве фактических параметров выступают введенные с клавиатуры значения переменных **a** и **b** при первом вызове процедуры и **c** и **d** – при втором. Результатами являются, соответственно, фактические параметры **min1** и **max1** (минимум и максимум в первой паре чисел) и **min2** и **max2** (минимум и максимум во второй паре).

При третьем обращении к процедуре в качестве исходных данных используются 2 найденных минимума, а результатом ее работы является наименьшее из этих 2 значений, передаваемое в переменную **lit.** Вспомогательная переменная **l** в принципе для решения данной задачи не нужна, но вводится, так как число фактических параметров в обращении должно соответствовать числу формальных параметров в описании. Аналогично при четвертом обращении к процедуре находится наибольший из двух максимумов **big**, а переменная **m** играет, подобно переменной **l** только вспомогательную роль.

Искомые наибольшее и наименьшее значения выводятся на экран с помощью оператора **writeln**.

Выше уже отмечалось, что в процедуру могут передаваться не только значения переменных, но и числовые константы. Приведем пример программы, где реализована как раз такая возможность. Это – программа, которая по введенному пользователем текущему числу и номеру месяца определяет дату следующего дня. Данная задача не представляет сложности (нужно только прибавить единицу к текущему числу), но лишь для всех чисел месяца, кроме последнего. В этом случае следующий день будет первым числом следующего месяца, а если последний день месяца – 31 декабря, то следующим днем будет первый день первого месяца нового года. Необходимо также предусмотреть защиту от неверного ввода данных (чтобы пользователь случайно не ввел число, большее, чем количество дней в

текущем месяце). Ниже приводится текст программы, разработанной для решения данной задачи.

```
program tomorrow;
Uses Crt;
 label 100;
 var
      d,m:integer; otv:string[3];
procedure next(kol:integer;var mes,n:integer);
begin
 if n>kol then
     begin
     writeln('Вы ошиблись. Введите верное число');
     Delay(60000);
     Halt
     end;
 if (n=31) and (mes=12) then
      begin
      n:=1;
      mes:=1;
      Exit;
      end;
 if n=kol then
       begin
       n:=1;
       mes:=mes+1
       end
              else
       n:=n+1;
end;
begin
 ClrScr;
 writeln('Введите сегодняшнее число');
 readln(d);
 writeln('Введите номер текущего месяца');
 readln(m);
 case m of
```

```
      1,3,5,7,8,10,12:next(31,m,d);
      4,6,9,11:next(30,m,d);
      2: begin
          100:   writeln('Является ли текущий год
          високосным?(ДА/НЕТ)');
          readln(otv);
          if (otv<>'ДА') and (otv<>'НЕТ') then
              begin
              writeln('Вы дали неверный ответ. Введите ДА
              или НЕТ');
              goto 100;
              end
          else
             if otv='ДА' then next(29,m,d)
                         else next(28,m,d);
           end;
         end;
 writeln('Завтра будет ',d,'.',m);
 readln
end.
```

Разберем текст данной программы. Непосредственное определение числа и номера месяца следующего дня производится в процедуре **next**. В заголовке процедуры **next** описываются один параметр-значение **kol** – количество дней в текущем месяце и два параметра-переменные – **mes** и **n**. При обращении к процедуре **next** параметрам **mes** и **n** присваиваются значения, соответственно, текущего месяца и номера текущего дня в месяце. На выходе процедуры эти параметры должны получить значения соответствующие завтрашнему дню. При этом возможны следующие варианты:

а) день не является последним днем месяца. В этом случае номер следующего дня (**n)** увеличивается на единицу. Номер месяца (**mes**) при этом остается без изменений;

б) день является последним днем месяца. В таком случае номер следующего дня становится равным единице, а номер месяца увеличивается на единицу;

в) день является не только последним днем месяца, но и последним днем года. Тогда номер следующего дня становится равным единице. Первым становится и номер месяца, так как следующий день будет 1 января;

г) при вводе данных была допущена ошибка (например, пользователь ввел номер дня в месяце равный 32). В такой ситуации пользователю нужно

сообщить о допущенной им ошибке, после чего программа в аварийном порядке должна прекратить работу.

Все эти варианты и рассмотрены в процедуре **next.** В начале работы данной процедуры с помощью сокращенного условного оператора **if** проверяется правильность введенных данных (т. е. не больше ли переданное в процедуру текущее число **n**, чем количество дней в текущем месяце). Если при вводе была допущена ошибка, то процедура выводит сообщение об этом и прекращает работу программы используя стандартную процедуру **Halt**. Чтобы пользователь успел прочитать сообщение о допущенной им ошибке, сообщение задерживается на некоторое время на экране компьютера с помощью стандартной процедуры **Delay**.

Следующий сокращенный условный оператор проверяет, не является ли введенное число 31 декабря. В этом случае параметрам **mes** и **n** присваиваются значения 1 и 1, и затем работа процедуры **next** завершается с помощью стандартной процедуры **Exit**, после чего происходит возвращение в основную программу.

С помощью последнего в данной процедуре оператора **if** определяется следующее число для любого из остальных дней года. Если число не является последним в данном месяце (**n < kol**), то значение параметра **n** увеличивается на единицу, а **mes** остается без изменений. В противном случае **n** присваивается значение 1 (следующее число – начало нового месяца), а значение параметра **mes** увеличивается на единицу. На этом работа процедуры завершается, соответствующие формальным параметрам **mes** и **n** фактические параметры получают новые значения, которые передаются в основную программу.

Из указанного выше понятно, что для того чтобы процедура правильно работала, нужно передать в нее правильную величину параметра-значения**,** которая равна действительному количеству дней в текущем месяце. Это количество определяется в основной части программы. Так как количество дней в месяце зависит от номера текущего месяца **m**, то оно определяется с помощью условного оператора **case**, в котором в роли переменной-селектора как раз и выступает **m**. Для **m**, равного 1,3,5,7,8,10,12 т.е. для января, марта, мая, июля, августа, октября, декабря, данное значение равно 31. Для **m,** равного 4,6,9,11, т.е. для апреля, июня, сентября, ноября, значение будет равно 30.

Сложнее решается вопрос с февралем (**m=2**), так как в этом случае значение, передаваемое в процедуру, зависит от того, является ли текущий год високосным или нет. В первом случае значение будет равно 29, во втором – 28. Необходимую информацию программа запрашивает у пользователя, который на вопрос о том, является ли текущий год високосным, должен дать положительный или отрицательный ответ. Для защиты от ошибочного ответа используется сокращенный условный оператор **if**. Если пользователь вместо «ДА» или «НЕТ» введет что-либо

другое, программа выдаст сообщение об ошибке и с помощью оператора безусловного перехода вернет пользователя к тому месту программы, где вновь повторяется заданный вопрос.

При обращении из основной программы к процедуре **next** формальному параметру **kol** ставится в соответствие найденная с помощью оператора **case** величина, а формальным параметрам **mes** и **n** ставятся в соответствие фактические параметры **m** и **d,** содержащие введенные с клавиатуры значения номеров текущего месяца и текущего дня.

После обращения к процедуре и производимых процедурой расчетов полученные новые значения параметров-переменных возвращаются в основную программу. Затем на экран компьютера выводятся эти новые значения (завтрашнее число и соответствующий ему номер месяца), и на этом программа завершает свою работу.

7.3. Рекурсия

В разделе 7.2 настоящего практикума мы уже рассматривали программу, в которой имеется процедура, содержащая обращения к другим процедурам. Обращения к другим функциям и процедурам (как стандартным, так и созданным программистом) могут содержать и функции. При этом возможен и такой вариант, когда какая-либо функция или процедура содержит обращение к самой себе. Такая функция или процедура называется ***рекуррентной*** (от английского слова ***recurrence***, что в переводе означает возвращение или повторение), а процесс вызова функции или процедуры из нее самой – ***рекурсией***.

Процесс рекурсии происходит следующим образом: вначале происходит вызов некоторой функции (процедуры). Затем, еще до завершения своей работы, эта же функция (процедура) вызывает саму себя, и в результате в памяти компьютера появляется второй экземпляр той же функции (процедуры), второй экземпляр в ходе работы создает третий и т.д. Естественно, что такой процесс может продолжаться до бесконечности, поэтому для решения какой-либо конкретной задачи в рекурсивной функции (процедуре) обязательно должен содержаться условный оператор, одна из ветвей которого обеспечивает прерывание этой бесконечной цепочки и завершение работы всех экземпляров функции (процедуры). Внутри этого оператора и должен находиться вызов функции.

В качестве несложного примера, наглядно иллюстрирующего процесс рекурсии, рассмотрим задачу вычисления целой положительной степени вещественного числа (x^y). Текст соответствующей программы приведен ниже:

```
program stepint;
Uses Crt;
 var x,st:real; y:integer;
```

```
 function rek(a:real;k:integer):real;
   begin
   if k=1
   then rek:=a
   else rek:=rek(a,k-1)*a
   end;
begin
 ClrScr;
 writeln('Введите число');
 readln(x);
 writeln('Введите показатель степени ');
 readln(y);
 st:=rek(x,y);
 writeln('Степень числа равна ',st:12:3);
 readln;
end.
```

В данной программе в основной ее части производится только ввод исходных данных (самого числа **x** и показателя степени **y**), вызов рекурсивной функции **rek** и вывод полученного результата, а само вычисление степени производится в функции **rek.** Разберем более подробно механизм действия данной функции.

Для этого нужно ответить на вопрос: чему равно любое число в первой степени. Ответ очевиден – самому этому числу. Это мы и запишем в начале условного оператора: для **k=1** результат **rek** равен исходному числу **a.** Для нахождения же квадрата числа нужно это число умножить само на себя, для нахождения куба – квадрат числа умножить на это число и т.д. Таким образом, существует следующая закономерность: **k**-я степень числа равна этому числу в степени **k-1**, умноженному на само число.

$$a^k = a^{(k-1)} * a.$$

Эту формулу запишем в той ветви условного оператора, которая находится после служебного слова **else.**

Теперь рассмотрим непосредственно работу функции. Данная функция имеет два формальных параметра **a** и **k**, которым соответствуют в основной части программы фактические параметры **x** и **y.** Значением функции является a^k . Для вычисления значения функции **rek** с параметрами **k** и **a** вызывается другой экземпляр той же функции с параметрами **k-1** и **a** (**a** остается неизменным для всех экземпляров функции), затем этот экземпляр вызывает следующий с параметром **k-2** и т. д., пока процесс не доходит до обращения

к экземпляру для **k=1**. Значение функции для **k=1** уже известно, так как в этом случае оно равно **a**. Это значение передается в экземпляр для **k=2** и т. д., пока не определится значение для **k**. Это последнее значение передается в основную часть программы для вывода на экран компьютера.

В заключение раздела приведем еще один пример использования рекурсии для решения классической математической задачи известной как «числа Фибоначчи». Эта задача была решена в XIII в. выдающимся итальянским математиком Леонардо Фибоначчи. Вот ее условия: пара кроликов каждый месяц дает потомство – двух кроликов, которые через два месяца сами способны давать новое потомство. Сколько кроликов будет через n месяцев, если в начале была одна пара кроликов. Текст программы, которая решает поставленную задачу, приведен ниже:

```
program rabbit;
Uses Crt;
 var i,k:integer;
 function rab(n:integer):integer;
  begin
  if n=0
  then rab:=1
  else
   if n=1
   then rab:=2
   else rab:=rab(n-2)+rab(n-1)
  end;
begin
 ClrScr;
 writeln('Введите количество месяцев');
 readln(i);
 k:=rab(i)*2;
 writeln('Количество кроликов через ',i,' месяцев  равно
 ', k);
 readln;
end.
```

Для решения данной задачи используем рекурсивную функцию **rab(n)**, которая определяет количество пар кроликов через **n** месяцев после начала. Вначале существовала только одна пара кроликов, т.е. **rab(0)=1**, через месяц их стало две, следовательно **rab(1)=2**. Через 2 месяца количество пар кроликов увеличивается еще на 1, т.е. на столько, сколько их

было 2 месяца назад. Таким образом **rab(2)=3** или **rab(2)=rab(1)+rab(0)**. Продолжая наши рассуждения, получаем следующую зависимость:

rab(n)=rab(n-1)+rab(n-2).

Эту рекурсивную функцию используем в программе **rabbit**, которая подсчитывает и выводит на экран количество кроликов через **n** месяцев. Количество кроликов через **n** месяцев будет равно **rab(n)×2**.

Задания для самостоятельной работы к главе 7

1. Составьте программу, которая определяет площадь треугольника по формуле Герона. Данная формула выглядит следующим образом:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

где **a,b** и **c** – длины сторон треугольника, а **p** – половина периметра треугольника. Для извлечения квадратного корня используйте стандартную функцию **sqrt**. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

2. Составьте программу, которая по выбору пользователя возводит введенное с клавиатуры целое число в квадрат, определяет его абсолютную величину или извлекает квадратный корень числа. В программе следует предусмотреть вывод сообщения об ошибке, если пользователь попытается извлечь квадратный корень из отрицательного числа. Используйте в программе стандартные функции **abs, sqr и sqrt**. Фон экрана должен быть черным. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

3. Дополните предыдущую программу функцией, которая определяет, действительно ли введенное с клавиатуры число является целым (целое число может содержать только цифры от 0 до 9). Результат функции должен быть логического типа, т.е. либо **true**, либо **false**. В зависимости от значения функции, программа должна либо предложить пользователю произвести на выбор один из вышеуказанных вариантов вычислений, либо вывести сообщение об ошибке и вернуть пользователя к вводу исходных данных. Фон экрана должен быть черным. Ввод данных должен производиться бирюзовым цветом, вывод результатов – сиреневым.

4. Составьте программу, которая вычисляет сумму факториалов 3 целых чисел, введенных с клавиатуры. Вычисление факториала оформить в виде отдельной функции. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

5. Составьте программу, которая выводит в центре экрана электронные часы, отсчитывающие время от 1 до 20 секунд. Фон экрана должен быть светло-серым, цифры на электронном табло – синими.

6. Составьте программу, которая заполняет одномерный массив из 20 элементов целыми числами от 1 до 99, а затем определяет, сколько в массиве имеется простых чисел (простым числом называется такое, которое делится только на единицу и само на себя). Процесс определения того, является ли число простым, оформить в виде отдельной процедуры. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

7. Составьте программу, которая определяет в интервале от 1 до 10000 все совершенные числа. Совершенным числом называется число, которое равно сумме всех своих делителей, включая единицу. Процесс определения того, является ли данное число совершенным, оформите в виде отдельной процедуры. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

8. Составьте программу, которая определяет наибольшее в группе из 8 чисел. Для решения задачи следует использовать процедуру нахождения большего из двух чисел. Фон экрана должен быть черным. Ввод данных должен производиться желтым цветом, вывод результатов – светло-синим.

9. Составьте программу, которая определяет наименьшее в группе из 8 чисел. Для решения задачи следует использовать процедуру нахождения меньшего из двух чисел. Фон экрана должен быть черным. Ввод данных должен производиться бирюзовым цветом, вывод результатов – фиолетовым.

10. Составьте программу вычисления следующего выражения:

$$\mathbf{a^b+c^d},$$

где a и c – вещественные числа, b и d – целые числа. При вычислениях использовать рекурсивную функцию возведения в степень. Ввод данных должен производиться красным цветом, вывод результатов – зеленым.

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 /СПбГТУ РП. — 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/ СПбГТУ РП. — 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/ СПбГТУ РП. — СПб., 2009.

Ставровский А.Б. Turbo Pascal 7.0: учебник. — Киев: Издательская группа БХВ, 2000.

Редактор и корректор Н.П. Новикова
Техн. редактор Л.Я. Титова Темплан 2011 г., поз. 103

Подп. к печати Формат 60x84/16. Бумага тип. № 1.
Печать офсетная. 3,25 уч.-изд. л. ; 3,25 усл. печ. л. Тираж 50 экз.
Изд. № 103. Цена «С». Заказ

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