Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра «Вычислительные методы и программирование»

А.А. Бурцев, В.П. Шестакович

ПРОГРАММИРОВАНИЕ

МЕТОДИЧЕСКОЕ ПОСОБИЕ
К ВЫПОЛНЕНИЮ КОНТРОЛЬНЫХ ЗАДАНИЙ
ДЛЯ СТУДЕНТОВ ВСЕХ СПЕЦИАЛЬНОСТЕЙ БГУИР
ЗАОЧНОЙ ФОРМЫ ОБУЧЕНИЯ

В 2-х частях

Часть 2

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО ПАСКАЛЬ

Минск 2005

УДК 004.43 (075.8)
ББК 32.973.26-018.1
Б 91

Бурцев А.А.

Б 91 Программирование: Метод. пособие к выполнению контрольных заданий для студ. всех спец. БГУИР заочной формы обуч.: В 2 ч. Ч.2: Программирование на языке Турбо Паскаль / А.А. Бурцев, В.П. Шестакович. – Мн.: БГУИР, 2005. – 70 с.

ISBN 985-444-784-7 (ч. 2)

В работе приводятся краткие теоретические сведения и методические указания по программированию на языке Турбо Паскаль. Содержатся контрольные задания (пять задач по 30 вариантов) по следующим темам: множества, файлы, модули, текстовый и графический режимы работы дисплея. Включены тексты программ-примеров.

УДК 004.43 (075.8)
ББК 32.973.26-018.1

Часть 1 издана в БГУИР в 2004 г.

ISBN 985-444-784-7 (ч.2)
ISBN 985-444-462-7

Содержание

ЗАДАНИЕ 8. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МНОЖЕСТВ

Цель работы – изучить правила описания данных типа «множества», допустимые в Паскале операции над множествами, а также способы составления программ для обработки данных типа «множества».

Краткие теоретические сведения

Множество – это некоторый набор или совокупность однотипных элементов. Примерами множеств являются совокупности целых чисел и букв латинского алфавита.

Над множествами в математике допустимы следующие операции:

объединение множеств: C = A È B, где множество **C** содержит элементы, принадлежащие или A, или B, или A и B одновременно;

пересечение множеств: C = A Ç B, где множество **C** содержит элементы, принадлежащие и **A,** и **B** одновременно;

разность двух множеств: C = A \ B, где множество **C** содержит элементы, принадлежащие A и не принадлежащие B.

Например, {1,2,3} ∪ {3,2,4}={1,2,3,4};
{1,2,3} ∩ {3,2,4}={2,3};
{1,2,3} \ {3,2,4}={1}.

В отличие от элементов массива элементы множества неупорядочены, поэтому следующие множества одинаковы: {1,2,3,4}, {4,3,2,1}, {4,2,3,1} и т.д.

В языке Паскаль под *множеством* понимают ограниченный, неупорядоченный набор различных элементов одного типа. При этом можно работать как с множествами-константами, так и с переменными типа множества.

Запись [] обозначает *пустое множество*, т.е. это множество, не содержащее ни одного элемента. При этом пустое множество является единственным и принадлежит всем множествам. Для обозначения множеств-констант в языке Паскаль используются квадратные скобки, в которые заключаются элементы множества: [1,2,3,4], ['a','b','c','d'].

Множество-константу можно задать по следующей общей форме:

Const имя=[список констант];

Например, const s=[1, 3, 5, 7, 9] или использовать типизированные константы типа set.

Например:

```
Type s1=set of 0..9;
     s2=set of char;
Const
      ml:sl=[0, 2, 4, 6, 8];
      m2:s2=[‘a’, ’b’, ’c’];
```

Под *мощностью множества* понимается общее число его элементов. Так, мощность пустого множества равна нулю.

Формат описания переменных типа множества следующий:

type имя типа=set of базовый тип элементов;
var имя множества: имя типа;
или
var имя множества: set of базовый тип;

Здесь базовый тип – это тип элементов, входящих во множество. В качестве базового типа можно использовать любой простой тип, за исключением вещественного типа, так как мощность множества в этом случае равна бесконечности. В качестве целого типа можно использовать byte, shortint, но нельзя word, integer, longint.

Пример описания множеств:

```
type set1=set of 1..3;
var a:set1;
```

Описанное таким образом множество **a** может содержать следующие элементы:

[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3].

Еще одно ограничение касается количества элементов множества – от 0 до 256.

В качестве элементов множества можно пользоваться ограниченным типом. Так, множество-константу целых чисел от 1 до 5 можно или записать как [1, 2, 3, 4, 5], или ввести в виде сокращенной формы [1..5]. Однако, если значение первой константы диапазона строго больше значения второй константы, то задается пустое множество. Так, [5..1] обозначает пустое множество.

Новые множества можно формировать, используя операции над множествами. В Паскале имеются следующие операции над множествами:

операция объединения множеств: + ;
операция пересечения множеств: * ;
разность двух множеств: - ;

а также операции:

сравнения: =, <>, >=, <=;
проверки принадлежности: IN;
присваивания: := .

Поясним каждую из этих операций. Пусть имеются два однотипных множества **A** и **B**, а также переменная **X**.

Равенство и неравенство множеств. Множества равны между собой (т.е. выражение **A=B** – «истинно») тогда и только тогда, когда **A** и **B** содержат одни и те же элементы. Если **A** и **B** отличаются хотя бы одним элементом, то **A** и **B** не равны между собой (выражение **A<>B**- «истинно»).

Включение множества. Выражение **A<=B** (или **B>=A**) принимает значение «истинно», когда все элементы **A** являются также элементами **B**, и значение «ложно» – в противном случае.

Проверка принадлежности. Операция «проверка принадлежности» заключается в следующем: выражение **X IN A** принимает значение «истинно», если значение переменной **X** принадлежит множеству **A**, и значение «ложно» –

в противном случае. Тип переменной **X** должен быть таким же, как и базовый тип элементов множества **A**.

Присваивание значения. Оператор **A:=B**; означает, что переменной типа «множество» **A** присваивается текущее значение множества **B**.

Объединение множеств. Операция объединения множеств заключается в том, что множество **A+B** будет содержать элементы, которые принадлежат или **A**, или **B**, или и тому и другому одновременно.

Пересечение множеств. Операция пересечения множеств заключается в том, что множество **A*B** будет содержать элементы, которые принадлежат и **A,** и **B** одновременно.

Разность множеств. Разность множеств заключается в том, что множество **A-B** будет содержать элементы из **A**, которые не входят в **B**.

В отличие от массивов к элементам множества нет прямого доступа по номерам-индексам этих элементов. Поэтому ввод множеств осуществляется с использованием операций объединения, а при выводе используется операция принадлежности **IN** .

Пример 8.1. Сформировать множество W, состоящее из произвольных символов персонального компьютера и символа «точка». Для заполнения множества **W** используем символьную переменную **K**. Формирование множества прекращается после ввода символа «точка». Фрагмент программы формирования множества выглядит следующим образом:

```
……..
   var w:set of char;
       k:char;
……..
   begin
……..
{ Ввод символов в множество W }
            w:=[];     { Задаем пустое множество }
          repeat
              read(k);
              w:=w+[k];   { Добавляем новый символ в множество }
          until k='.';
……..
```

Вывод различных символов, входящих в сформированное множество, выглядит так:

```
……..
        for k:=#0 to #255 do
             if  k in  w  then write ('  ',k);
……..
```

Контрольные вопросы

1. Что такое множество, как описать множество в программе?
2. Как задать множество-константу?
3. Как осуществляется ввод / вывод множеств-переменных?
4. Какие операции допустимы над данными типа множества?

Задача 8. Варианты

Написать программу, которая формирует множество

8.1 – из латинских букв с точкой в конце – и которая определяет число различных букв, входящих в это множество.

8.2 – **Y=(X1ÇX2)** – и вывести на печать элементы **Y**, которые делятся на число 3 без остатка. Проверить выполнение условия **X3 Í Y.**

Дано: **X1={1,2,3,4,5,6}; X2={1,3,4,5,6}; X3={1,5}.**

8.3 – **Y=(X1\X2)** – и вывести на печать элементы **Y**, делящиеся на 3 без остатка.

Дано: **X1={1,2,3,4,5,6}; X2={1,3,4,5,6}.**

8.4 – **Y=(X1È X2)** – и вывести на печать его мощность.

Дано: **X1={T1,T2,T4,T5}; X2={T1,T3,T4,T5}.**

8.5 – **Y=(X1\X3)** – и вывести его мощность.

Дано**: X1={T1,T2,T4,T5}; X3={T1,T5}.**

8.6 – **Y=(X1È X2)** – и выделить из него подмножество **Y1** чисел, которые являются четными числами.

Дано: **X1={1,2,3,4,5,6}; X2={1,3,4,5,6}.**

8.7 – **Y=(X1\X2)** – и вывести на печать его мощность.

Дано: **X1={2,4,6,8,10}; X2={1,3,5,6,7}.**

8.8 – **Y=(X1È X2È X3)** – и выделить из него подмножество **Y1,** которое представляет все цифры, входящие в Y.

Дано: **X1={‘s’, ’v’, ’e’, ’t’, ’a’, ’2’}; X2={‘*’, ’-‘, ’*', ’-‘}; X3={‘1’, ’4’, ’7’, ’a’, ’b’, ’c’}.**

Написать программу, которая формирует множество символов **Y** и выделяет из него подмножество **Y1**, представляющее собой:

8.9 – все буквы, входящие в **Y**;

8.10 – звонкие согласные буквы, входящие в **Y**;

8.11 – глухие согласные буквы, входящие в **Y**;

8.12 – гласные буквы, входящие в **Y**;

8.13 – буквы русского алфавита, входящие в **Y**;

8.14 – буквы английского алфавита, входящие в **Y**;

8.15 – цифры и буквы, входящие в **Y**.

В вариантах 8.16 – 8.30 с клавиатуры вводится произвольный набор символов. Требуется построить и распечатать множество с элементами, символами которого являются:

8.16 – цифры от 0 до 5 и от 7 до 9;
8.17 – буквы от A до F и от X до Z;
8.18 – буквы от g до n и цифры от 1 до 6;
8.19 – знаки препинания и скобки;
8.20 – буквы от A до Z и цифры от 0 до 5 ;
8.21 – буквы от t до x и знаки препинания;
8.22 – цифры от 1 до 5 и знаки арифметических действий;
8.23 – знаки препинания и логических операций отношения;
8.24 – цифры от 5 до 9 и знаки арифметических действий;
8.25 – знаки препинания и буквы от E до N ;
8.26 – знаки операций отношения и скобки;
8.27 – буквы от A до F и цифры от 3 до 9 ;
8.28 – знаки арифметических операций и операций отношения;
8.29 – буквы от s до w и знаки арифметических операций;
8.30 – знаки препинания и операций отношения.

Пример 8.2. Даны три множества: **X1={1,2,4}; X2={3,4,5,6}; X3={1,6};** Сформировать новое множество **Y=(X1ÈX2)ÇX1** и вывести на печать его элементы. Проверить выполнение условия **X3 Í Y**.

```
Program zad6;
var x1,x2,x3,y: set of 1..6;
    k:integer;
begin x1:=[1,2,4]; x2:=[3,4,5,6]; x3:=[1,6];
      y:=(x1+x2)*x1;
      write (' МНОЖЕСТВО Y:');
      for k:=1 to 6 do
       if (k in y)  then  write(' ',k); writeln;
       if x3<=y then write (' X3 принадлежит Y')
                else write (' X3 не принадлежит Y');
end.
```

ЗАДАНИЕ 9. ФАЙЛЫ В ПАСКАЛЕ

Цель работы – изучить приемы составления программ с использованием данных типа файл, получить практические навыки работы с файлами различных типов.

Краткие теоретические сведения

Под *файлом* понимается любой набор элементов одного и того же типа. Число элементов, называемое *длиной файла*, не фиксировано.

По отношению к программе файлы могут быть внешними и внутренними. *Внутренние* – это файлы, которые создаются, используются и существуют только во время работы данной программы. К внутренним относятся: файл ввода данных Input и файл вывода результата выполнения программы Output.

Файлы, которые существуют вне программы, называются *внешними*. В качестве носителей внешних файлов обычно используют магнитные диски. Внешние файлы должны быть описаны в разделе описаний программы одним из способов:

```
Type  имя типа = File Of  базовый тип;
Var   имя файла  :  имя типа;
```

или

```
Var  имя файла  :  File Of  базовый тип;
```

В качестве базового типа элементов файла можно использовать любой тип данных (как простой, так и сложный), за исключением типа File.

Например:

```
Type Mas = Array[1..6,1..9] Of Real;
     Zap = Record
             K : Word;
             C : Char;
           End;
Var  Fmas  : File Of  Mas;
     Fzap  : File Of  Zap;
     Fint  : File Of  Integer;
     Fr    : File Of  Real;
     Fc    : File Of  Char;
```

Доступ к элементам осуществляется через указатель файла (буферную переменную). При считывании или записи этот указатель перемещается к следующему элементу и делает его доступным для обработки. В каждый момент доступен для записи (чтения) только тот элемент файла, на который установлен указатель.

Стандартные процедуры и функции для работы с файлами

Процедура **Assign (Var F : file; FileName : String)** устанавливает связь между логическим файлом, описываемым файловой переменной F, и конкретным файлом на диске, название которого содержится в строковой переменной FileName. Строка FileName может содержать имя файла на диске (в том числе полное имя файла), имя стандартного устройства MS DOS (‘CON’,”PRN’ и т.п.) или пустую строку.

Например:

Assign (F, ’File.Dat’); – связь с файлом текущего каталога.

Assign (F, ’A:\Proba.Txt’); – связь с файлом Proba на диске A:.

После выполнения Assign все действия над переменной F будут эквивалентны действиям над файлом, определенным строкой FileName. Процедуру Assign необходимо использовать до начала работы с файлом.

Процедура **Reset (Var F : File)** открывает логический файл F для чтения данных, при этом указатель устанавливается на первый элемент файла. Процедура Reset может применяться многократно к одному и тому же файлу. Если файл был до этого открыт, то он автоматически предварительно закрывается. Повторный вызов Reset переустановит последовательность чтения вновь на самый первый элемент файла, при этом потеря данных исключается.

Процедура **ReWrite (Var F : File)** открывает логический файл F для записи данных. Указатель файла устанавливается в первую позицию (с номером 0). Фактически файл не содержит ни одного компонента, он только подготовлен для загрузки. Повторное обращение к ReWrite сотрет текущее содержимое файла и снова подготовит файл к заполнению с первого элемента.

Процедура **Read (Var F : File; Var x1, x2,…, xN)** считывает элементы файла F в переменные x1,x2,…xN, начиная с элемента, на который указывает указатель файла.

Процедура **Write (Var F : File; Var x1, x2,…, xN)** записывает значения переменных x1, x2,…, xN в файл F, начиная с той позиции, на которую установлен указатель файла.

Процедура **Close (Var F : File)** записывает открытый до этого логический файл F. Попытка закрыть уже закрытый или еще не открытый файл вызывает сбой. Особенно важно закрывать файлы, открытые на запись, это гарантирует сохранность и полноту их заполнения. Также необходимо закрывать файлы перед их удалением (Erase) или переименованием (Rename).

Процедура **Rename (Var F : File; NewName : String)** позволяет переименовать физический файл на диске, связанный с логическим файлом F. Процедура выполнима только с закрытым файлом. Так, с помощью конструкции

```
Assign(F, ’A:\Real.Dat’);
Rename(F, ’A:\Float.Dat’);
```

будет заменено имя файла Real.Dat на Float.Dat. После открытия файла процедурой Reset(F) или ReWrite(F) переменная F будет связана с новым именем. Конструкция

```
Assign(F, 'A:\Progr0.1.Dat');
Rename(F, 'C:\Progr0.2.Txt');
```

будет ошибочной, поскольку кроме имени файла изменяется содержащий его диск.

Процедура **Erase (Var F : File)** уничтожает (стирает) физический файл на носителе (диске). Файловая переменная F должна быть предварительно связана с существующим физическим файлом. Сам файл к моменту вызова Erase должен быть закрыт. Чтобы уничтожить файл, достаточно конструкции

```
Assign(F, Name);
Erase(F);
```

Функция **Eof (Var F : File) : Boolean** анализирует состояние файла. Она возвращает значение True, когда при чтении достигнут конец файла F. Это означает, что уже прочитан последний элемент в файле или файл F после открытия оказался пуст. Во всех остальных случаях функция возвращает значение False. Наиболее часто Eof используется в цикле While, читающем файл до конца:

```
While Not Eof(F) Do Read(F, x1,…),
```

т.е. пока не достигнут конец файла F осуществляется считывание данных из этого файла. Следует обратить внимание, что используется именно цикл While…Do, а не Repeat…Until. Функция Eof постоянно следит за положением указателя и позволяет опознать конец файла до того, как мы его непосредственно прочитаем.

Функция **FileSize (Var F : File) : Longint** возвращает реальное число записей в открытом файле F. Для пустого файла возвращает значение, равное 0.

Функция **FilePos (Var F : File) : Longint** возвращает текущую позицию указателя (номер элемента) в файле F. Так, если файл только что открылся, то значение текущей позиции равно 0. В случае, когда FilePos возвратило значение, равное FileSize, то указатель находится в конце файла за последней записью. В остальных случаях FilePos возвращает значение на единицу меньше реального номера записи, которая готова к прочтению или созданию.

Процедура **Seek (Var F : File; N : Longint)** позволяет осуществлять прямой доступ к элементам файла F. Здесь N – целая положительная константа, соответствующая порядковому номеру элемента в файле. Процедура Seek не выполняет операцию чтения или записи элемента файла, она лишь перемещает указатель к элементу с номером N. При этом первый элемент имеет номер N=0, второй – N=1 и т.д. Например:

```
Seek(F, 0); – установит указатель на первом элементе файла F;
Seek(F, FileSize(F)); – за последним элементом файла F;
Seek(F, FileSize(F) - 1); – на последний элемент файла F.
```

Процедура **Truncate (Var F : File)** позволяет отсекать часть открытого файла F, начиная с положения текущего указателя файла. Так, конструкция Seek(F, 0); Truncate(F); сделает файл F совершенно пустым, а конструкция Seek(F, 2); Truncate(F); удалит все элементы до конца файла F, начиная с третьего.

Использование приведенных процедур и функций покажем на следующих примерах.

Пример 9.1. Сформировать файл, состоящий из целых чисел 5, 6, 7, 8, 9:

```
Program Number_1;
  Var Fb : File Of Byte;
      I  : Byte;
   Begin
     Assign(Fb, 'FileByte.Dat');
     Rewrite(Fb);
     For I := 5 To 9 Do Write(Fb, I);
     Close(Fb);
    End.
```

Пример 9.2. К файлу, созданному в предыдущем примере, добавить числа 20, 30, 40:

```
Program Number_2;
   Var Fb  : File Of Byte;
       I, A : Byte;
    Begin
      Assign(Fb, 'FileByte.Dat');
      Reset(Fb);
      Seek(Fb, FileSize(Fb));
      For I := 2 To 4 Do
        Begin
          A := I*10;
         Write(Fb, A);
        End;
     Close(Fb);
   End.
```

Пример 9.3. Видоизменить содержимое файла, созданного в первом примере, заменив последние три элемента на числа 30, 40, 50. Затем вывести на печать содержимое файла:

```
Program Number_3;
   Var Fb  : File Of Byte;
```

```
    I, A : Byte;
Begin
  Assign(Fb, 'FileByte.Dat');
   Reset(Fb);
     {Корректировка элементов}
  For I := 3 TO 5 DO
    Begin
      Seek(Fb, I-1);
      Read(Fb, A);
      A := I*10;
      Seek(Fb, I-1);
      Write(Fb, A);
    End;
          {Печать элементов}
  Reset(Fb);
  While  Not  Eof(Fb)  Do
     Begin
       Read(Fb, A);
        Write(A,' ');
     End;
   Close(Fb);
End.
```

К особому типу файлов в Паскале относится *текстовый файл*. Его содержимым является текст, состоящий из символов, например букв латинского алфавита. Сам текст хранится в файле в виде строк. Строки файла могут иметь различную длину и заканчиваются специальным символом конца строки.
Формат описания текстового файла:

Var имя файла : Text;

Здесь Text – стандартный идентификатор, точно такой же, как Real, Byte, Char и т.д.

Определить конец строки позволяет функция

Eofn(Var F : Text) : Boolean;

Эта функция принимает значение True, если достигнут конец строки, и значение False – в противном случае.

Для работы с текстовыми файлами используются приведенные выше процедуры (кроме Seek, Filepos и Filesize), а также следующие стандартные процедуры:

Append(Var F : Text) – открытие уже существующего текстового файла и установка указателя в конец файла;

Writeln(Var F : Text) – завершение текущей строки текстового файла;

Writeln(Var F : Text, Var x1, x2,…, xN) – запись в текстовый файл F значений переменных x1, x2,…, xN с завершением текущей строки;

Readln(Var F : Text) – переход к началу следующей строки текстового файла F (при чтении);

Readln(F : Text, Var x1, x2,…, xN) – чтение N символов файла F с переходом к новой строке.

Контрольные вопросы

1. Что называют файлом и файловой переменной?
2. Как описывают файловую переменную, текстовый файл?
3. Какими стандартными процедурами для работы с файлами располагает Паскаль?
4. Каковы особенности текстового файла?

Задача 9. Варианты

9.1. Создать файл, содержащий сведения о месячной заработной плате сотрудников отдела. Каждая запись содержит поля: фамилия сотрудника, наименование отдела, размер заработной платы за месяц. Вычислить общую сумму выплат за месяц по отделу A, а также среднемесячный заработок сотрудника этого отдела. Напечатать для бухгалтерии ведомость для начисления заработной платы сотрудникам этого отдела.

9.2. Создать файл, содержащий сведения о количестве изделий, собранных сборщиками цеха за неделю. Каждая запись содержит поля: фамилия сборщика, количество изделий, собранных им ежедневно в течение шестидневной недели, т.е. раздельно – в понедельник, вторник и т.д. На печать вывести: фамилию сборщика и общее количество деталей, собранное им за неделю; фамилию сборщика, собравшего наибольшее число изделий, и день, когда он достиг наивысшей производительности труда.

9.3. Создать файл, содержащий сведения о количестве изделий категорий A, B, C, собранных рабочим за месяц. Структура записи имеет поля: фамилия сборщика, наименование цеха, количество изделий по категории, собранных рабочим за месяц. Считая заданными значения расценок Sa, Sb, Sc за выполненную работу по сборке единицы изделия категорий соответственно A, B, C, выдать на печать следующую информацию: общее количество изделий категорий A, B, C, собранных рабочим цеха X; ведомость заработной платы рабочих цеха X; средний размер заработной платы работников этого цеха.

9.4. Создать файл, содержащий сведения о телефонах абонентов. Каждая запись имеет поля: фамилия абонентов, год установки телефона, номер телефона. На печать вывести: по фамилии абонента выводится номер телефона; определяется количество установленных телефонов с XXXX года. Номер года вводится с клавиатуры.

9.5. Создать файл, содержащий сведения об ассортименте игрушек в магазине. Структура записи: название игрушки, цена, количество, возрастные границы, например 2–5, т.е. от 2 до 5 лет. На печать вывести: названия игрушек, которые подходят детям от 1 до 3 лет; стоимость самой дорогой игрушки и ее наименование; название игрушки, которая по стоимости не превышает «x» руб. и подходит ребенку в возрасте от «a» до «б» лет. Значения «x», «a», «б» вводятся с клавиатуры.

9.6. Создать файл, содержащий сведения о сдаче студентами сессии. Структура записи: номер группы, фамилия студента, оценки по пяти экзаменам, признак участия в общественной работе: «1» – активное участие, «0» – неучастие. Написать программу формирования списка студентов группы X на стипендию. Студент, получивший все оценки 5 и активно участвующий в общественной работе, зачисляется на повышенную стипендию (доплата 50 %), а не активно участвует – доплата 25 %. Студенты, получившие 4 и 5, зачисляются на обычную стипендию. Студент, получивший одну оценку 3, но активно занимающийся общественной работой, также зачисляется на стипендию, в противном случае зачисление не производится. Номер группы вводится с клавиатуры.

9.7. Создать файл, содержащий сведения о сдаче студентами сессии. Структура записи: номер группы, фамилия студента, оценки по пяти экзаменам и пяти зачетам (зачет – незачет). На печать вывести: фамилии неуспевающих студентов с указанием номера группы и количества задолженностей; средний балл, полученный каждым студентом группы X и всей группой в целом.

9.8. Создать файл, содержащий сведения о личной коллекции книголюба. Структура записи: шифр книги, автор, название, год издания, местоположение (номер стеллажа, номер полки в шкафу и т.п.). На печать вывести: местонахождение книги автора X, название Y (значения X и Y вводятся с клавиатуры); список книг автора Z, находящихся в коллекции; число книг издания XXXX года.

9.9. Создать файл, содержащий сведения о наличии билетов и рейсах Аэрофлота. Структура записи: номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест в салоне. На печать вывести: время отправки самолетов в город X; наличие свободных мест на рейс в город X с временем отправления Y. Значения X, Y вводятся по запросу с клавиатуры.

9.10. Создать файл, содержащий сведения об ассортименте обуви в магазине. Структура записи: артикул, наименование, количество, стоимость одной пары. Артикул начинается с буквы Д для женской обуви, М – для мужской, П – для детской. На печать вывести информацию: о наличии и стоимости обуви артикула X; ассортиментном списке женской обуви с указанием наименования и имеющегося в наличии числа пар каждой модели.

9.11. Создать два файла, содержащие сведения о десяти нападающих хоккейных команд соответственно «Динамо» и «Спартак»: имена нападающих, число заброшенных ими шайб, количество сделанных голевых передач, заработанное штрафное время. По данным, извлеченным из этих файлов, создать новый, третий, файл, содержащий имя, название команды, сумму очков (голы +

передачи) для шести лучших игроков обеих команд. Имена и показатели результативности хоккеистов вывести на экран.

9.12. Создать файл, содержащий сведения о том, какие из пяти предлагаемых дисциплин по выбору желает слушать студент. Структура записи: фамилия студента, номер группы, пять дисциплин, средний балл успеваемости. Выбираемая дисциплина отмечается символом l, иначе – пробел. Напечатать списки студентов, желающих прослушать дисциплину X. Если число желающих превысит восемь человек, то необходимо отобрать студентов, имеющих более высокий балл успеваемости.

9.13. Создать файл, содержащий сведения об отправлении поездов дальнего следования. Структура записи: номер поезда, станция назначения, время отправления, время в пути, наличие билетов. На печать вывести: время отправления поездов в город X во временном интервале от A до Б часов; наличие билетов на поезд с номером XXX.

9.14. Создать файл, содержащий сведения о сотрудниках института. Структура записи: фамилия работающего, название отдела, год рождения, стаж работы, должность, оклад. На печать вывести: список сотрудников пенсионного возраста на сегодняшний день с указанием стажа работы; средний стаж работающих в отделе X.

9.15. Создать файл, содержащий сведения о пациентах глазной клиники. Структура записи: фамилия пациента, пол, возраст, место проживания (город), диагноз. На печать вывести: количество иногородних пациентов, прибывших в клинику; список пациентов старше X лет с диагнозом Y. Значения X и Y ввести с клавиатуры.

9.16. Создать файл, содержащий сведения о телефонах абонентов. Каждая запись имеет поля: фамилия абонента, номер телефона, год установки. Вывести на экран следующую информацию: по фамилии абонента выводится номер его телефона; по номеру телефона выводится фамилия абонента; выводится количество установленных в XXXX году телефонов.

9.17. Создать файл, содержащий сведения о студентах в общежитии. Каждая запись имеет поля: фамилия студента, номер группы, номер комнаты. Вывести: фамилии и номер группы студентов, проживающих в комнате X; по фамилии студента – номер комнаты.

9.18. Составить программу, которая подсчитывает количество элементов непустого файла f типа real, меньших среднего арифметического всех элементов этого файла.

9.19. Дан текстовый файл «a», который содержит данные о студентах группы (фамилия, имя, отчество и возраст):

```
type student = record
                 name: string[20];
                 vozr: 1..99
               end;
```

Создать файл, который будет содержать данные о студентах, имеющих наименьший возраст.

9.20. Создать файл учета ГАИ. Запись имеет поля: ФИО автолюбителя, марка автомобиля, год выпуска, цвет кузова, номер двигателя. Вывести список автолюбителей, проходящих техосмотр в текущем году, сгруппированный по маркам автомобилей. Учесть, что если текущий год четный, то техосмотр проходят автомобили с четными номерами, иначе – с нечетными.

9.21. Для участия в конкурсе исполнителей сформировать файл: ФИО, год рождения, название страны, класс музыкального инструмента (гитара, фортепиано, скрипка, виолончель). Вывести список исполнителей по классу X.

9.22. Написать программу, которая удваивает в текстовом файле t каждую цифру.

9.23. Написать программу, которая удаляет из файла t все символы «+» и «-».

9.24. Написать программу, которая упорядочивает элементы файла t типа integer по убыванию.

9.25. Написать программу, которая обрезает строки текстового файла, если в строке больше, чем 72 символа.

9.26. Написать программу, которая подсчитывает количество пустых строк текстового файла «a».

9.27. Написать программу, переписывающую содержимое текстового файла t2 в текстовый файл t1 с сохранением деления на строки.

9.28. Написать программу, которая переписывает в текстовый файл t1 содержимое текстового файла t2, но без пустых строк.

9.29. Написать программу, которая печатает самую короткую строку из текстового файла.

9.30. Написать программу, которая вставляет в начало каждой строки текстового файла t1 ее порядковый номер.

Пример 9.4. Необходимо написать программу, которая формирует и выводит на экран дисплея текстовый файл T, состоящий из девяти строк, в первой из которых содержится один символ «1», во второй – два символа «2»,..., в девятой – девять символов «9». Файл будет создан и хранится на диске a: под именем tt.txt:

```
Program Zadanie_9;
 Var T     : Text;
     C,D : Char;
   Begin
      Assign(T, 'a:tt.txt');
      Rewrite(T);
      For D := '1'  To  '9'  Do
        Begin
          For C := '1'  To  D  Do    Write (T, D);
          Writeln(T);
```

```
  End;
 Reset(T);
 While Not Eof(T) Do
    Begin
      Read(T, C);
       Write(C);
     End;
 Close(T);
End.
```

ЗАДАНИЕ 10. МОДУЛИ

Цель работы – изучить приемы создания библиотек и работу с модулями.

Модуль – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части. Основное назначение модулей – создание библиотек прикладных программ пользователя.

Структура модуля имеет вид

```
unit <имя>;
interface
<интерфейсная часть>

implementation
<исполняемая часть>

begin
<инициирующая часть>
end.
```

и состоит из заголовка и трех составных частей, любая из которых может быть пустой.

Заголовок состоит из ключевого слова unit и следующего за ним имени модуля, которое должно совпадать с именем дискового файла, содержащего текст модуля (например, если заголовок – unit bibliot; то текст модуля должен находиться в файле с именем bibliot.pas).

Интерфейсная часть расположена между ключевыми словами interface и implementation и содержит разделы описаний типов, констант, переменных и подпрограмм (процедур и функции), которые должны быть доступными в той программе или другом модуле, к которым будет подключен данный модуль. При объявлении подпрограмм в интерфейсной части указываются только их заголовки.

Исполняемая часть начинается после ключевого слова implementation и содержит описание процедур и функций, объявленных в интерфейсной части. Она может содержать разделы описаний типов, констант, переменных, процедур и функций, которые будут использоваться только в исполняемой части и будут недоступны внешним программам.

Инициирующая часть содержит операторы, которые выполняются до передачи управления основной программе, к которой будет подключен модуль, и обычно используются для подготовки ее к работе. Инициирующая часть вместе с начинающим ее ключевым словом begin может отсутствовать.

Компиляция модулей

При компиляции модуля результаты компиляции будут помещены в одноименный файл с расширением TPU (в нашем примере bibliot.tpu) и занесены на диск в каталог, объявленный опцией UNIT DIRECTORIES.

Стандартные модули

В Турбо Паскале имеется восемь стандартных модулей, содержащих большое число разнообразных типов, констант, процедур и функций. Этими модулями являются SYSTEM, DOS, CRT, PRINTER, GRAPH, OVERLAY, TURBO3 и GRAPH3. Модули GRAPH, TURBO3 и GRAPH3 выделены в отдельные TPU-файлы, а остальные входят в состав библиотечного файла TURBO.TP. Лишь один модуль – SYSTEM– подключается к любой программе автоматически, остальные становятся доступными только после указания их имени в разделе USES. Рассмотрим данные модули подробнее:

SYSTEM – в него вошли все процедуры и функции стандартного Паскаля.

PRINTER – делает доступным вывод текстов на матричный принтер. В нем определяется файловая переменная LST типа TEXT, которая связывается с логическим устройством PRN. После подключения модуля может быть выполнена, например, такая программа:

```
Uses Printer;
Begin
Writeln (LST, ‘Турбо Паскаль’)
End.
```

CRT– в нем сосредоточены процедуры и функции, обеспечивающие управление текстовым режимом работы экрана. С помощью входящих в модуль подпрограмм можно перемещать курсор в произвольную позицию экрана, менять цвет выводимых символов и окружающего фона, создавать окна. Кроме того, в модуль включены также процедуры «слепого» чтения клавиатуры и управления звуком.

GRAPH – содержит обширный набор типов, констант, процедур и функций для управления графическим режимом работы экрана. С помощью подпрограмм, входящих в модуль GRAPH, можно создавать разнообразные графические изображения и выводить на экран текстовые надписи стандартными или разработанными программистом шрифтами.

DOS – в нем собраны процедуры и функции, отрывающие доступ к программам и средствам дисковой операционной системы MS-DOS.

OVERLAY – необходим для разработки программ с перекрытиями.

Два библиотечных *модуля TURBO3* и *GRAPH3* введены для совместимости с ранней версией 3.0 системы Турбо Паскаль.

Пример. Создать модуль, содержащий две подпрограммы – функцию, вычисляющую выражение $x^2+\cos(x)-1$, и процедуру, находящую в числовой по-

следовательности a_1, a_2, a_3, …, a_n сумму положительных элементов и количество отрицательных элементов. Написать программу, использующую процедуру и функцию, помещенные в модуль для вывода таблицы значений функции в интервале [a,b] с шагом h и нахождения суммы положительных и количества отрицательных элементов числовой последовательности b_1, b_2, b_3, …, b_m, содержащей m элементов:

```
Unit bibliot;    {начало модуля}
        Interface
  Const nmax=100;
  Type vek=array [1..nmax] of real;
  Function f(x:real):real;
  Procedure fox (a:vek;  n:byte;  var s:real;  var kol:byte);

        Implementation
  Function f;
  Begin
  f:=x*x+cos(x)-1;
  end;

  procedure fox;
        var k:byte;
  begin
  s:=0;   kol:=0;
  for k:=1 to n do
        begin
        if a[k]>0 then s:=s+a[k];
        if a[k]<0 then kol:=kol+1;
        end;
  end; {конец процедуры}

end. {конец модуля}

program luxol;
  uses crt, bibliot;
  var i, m, k: byte;
      b:vek;
      a, b, h, x, sum: real;
  begin
    clrscr; {очистка экрана}
    write (‘Введите интервал изменения x’);
    readln (a, b);
  writeln (‘Введите шаг изменения x’);
  readln (h);
```

```
    clrscr;
    writeln('          x                    f(x)');  {Вывод заголовка таблицы}
    x:=a;
    repeat {Вывод таблицы}
      writeln(x:9:3, f(x):12:6);
      x:=x+h;
    until x> b+0.000001;
    readln; {остановка выполнения программы до нажатия любой клавиши
                для просмотра выведенной таблицы}
    clrscr;
    write ('Введите длину числовой последовательности'); readln (m);
    writeln ('Введена последовательность');
    for i:=1 to m do
          read (a[i]);
    readln;
    clrscr;
    writeln ('Введите последовательность');
    for i:=1 to m do
          write (a[i], '  ');
    writeln;
    fox (b, m, sum, k);
    writeln ('Сумма положительных элементов ', sum:9:6);
    writeln('Количество отрицательных элементов  ', k);
    readln;
end.
```

В программе luxol в разделе uses подключен стандартный модуль CRT для того, чтобы очищать экран монитора (процедура очистки экрана clrscr описана в модуле CRT) на определенных этапах работы программы и пользовательский модуль bibliot, созданный для формирования библиотеки процедур и функций отсутствующих в Паскале.

Контрольные вопросы

1. Что называется модулем?
2. Как оформляется модуль?
3. Для чего служит модуль?
4. Назначение интерфейсной части модуля.
5. Назначение раздела реализации.

Задача 10. Варианты

Составить программу, оформив вычисления в виде подпрограммы (процедуры или функции) помещенной в созданный модуль. В головной программе произвести ввод исходных данных, вызов подпрограммы из модуля и вывод результатов.

10.1. Дана квадратная целочисленная матрица N-го порядка. Упорядочить элементы в строках по убыванию.

10.2. Дана квадратная матрица N-го порядка, содержащая целые числа, каждое из которых занимает три позиции. Найти сумму чисел, в среднем разряде которых содержится число 5 и которые расположены на побочной диагонали и ниже.

10.3. Дана квадратная целочисленная матрица N-го порядка. Упорядочить элементы в столбцах по возрастанию.

10.4. Дана квадратная целочисленная матрица N-го порядка. Найти минимальный элемент среди положительных и максимальный среди отрицательных и их координаты (номера строк и столбцов на пересечении которых они находятся).

10.5.Задана матрица размером N x M. Получить массив B, присвоив его K-му элементу значение 0, если все элементы K-го столбца матрицы нулевые, и значение 1 в противном случае.

10.6. Дана квадратная целочисленная матрица N-го порядка. Найти суммы элементов, расположенных на линиях, параллельных главной диагонали матрицы и находящихся выше нее.

10.7. Задана матрица размером N x M. Получить массив B, присвоив его K-му элементу значение 1, если k-я строка матрицы симметрична, и значение 0 в противном случае.

10.8. Задана матрица размером N x M. Определить K – количество «особых» элементов матрицы, считая, что элемент «особый», если он больше суммы остальных элементов своего столбца.

10.9. Задана матрица размером N x M. Определить K – количество «особых» элементов матрицы, считая, что элемент «особый», если в его строке слева от него находятся элементы, меньшие его, а справа – большие.

10.10. Дана квадратная целочисленная матрица N-го порядка. Найти сумму элементов тех строк матрицы, у которых на главной диагонали расположены отрицательные элементы.

10.11. Задана символьная матрица размером N x M. Определить K – количество различных элементов матрицы (т.е. повторяющиеся элементы считать один раз).

10.12. Определить, является ли заданная квадратная матрица n-го порядка симметричной относительно побочной диагонали.

10.13. Определить, является ли заданная квадратная матрица n-го порядка симметричной относительно главной диагонали.

10.14. В матрице n-го порядка найти максимальный среди элементов, лежащих ниже побочной диагонали.

10.15. В матрице n-го порядка найти минимальный среди элементов, лежащих выше главной диагонали.

10.16. В вещественной квадратной матрице N-го порядка найти максимальный и минимальный элементы. Переставить строки, в которых они находятся. Если они находятся в одной строке, выдать об этом сообщение.

10.17. Задана одномерная матрица N-го порядка, содержащая целые числа, каждое из которых занимает четыре позиции. Подсчитать количество единиц в числах и их порядковые номера.

10.18. Дана вещественная матрица размером N x M. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (один из них) оказался в верхнем левом углу.

10.19. Дана вещественная матрица размером N x M. Упорядочить ее строки по возрастанию наибольших элементов в строках матрицы.

10.20. Задана квадратная матрица N-го порядка, состоящая из 0 и 1. Повернуть элементы матрицы на 90^0 по часовой стрелке.

10.21. Задана одномерная матрица N-го порядка, содержащая нули и целые числа. Заменить нули полусуммой последующего и предыдущего чисел. Если нуль является первым или последним числом матрицы, то его соответственно заменить последующим или предыдущим числом.

10.22. Дана вещественная квадратная матрица N-го порядка, все элементы которой различны. Найти скалярное произведение строки, в которой находится наибольший элемент матрицы, и столбца с наименьшим элементом.

10.23. Определить, является ли заданная целочисленная квадратная матрица N-го порядка ортонормированной, т.е. такой, в которой скалярное произведение каждой пары различных строк равно 0, а скалярное произведение каждой строки на себя равно 1.

10.24. Определить, является ли заданная матрица N-го порядка магическим квадратом, т.е. такой, в которой сумма элементов во всех строках и столбцах одинакова.

10.25. Дана целочисленная матрица размером N x M. Найти сумму наименьших элементов ее нечетных строк и наибольших элементов ее четных строк.

10.26. Дана действительная квадратная матрица N-го порядка. Рассмотреть элементы, расположенные в строках, начинающихся с отрицательного элемента. Найти сумму тех из них, которые расположены ниже главной диагонали матрицы.

10.27. Дана вещественная квадратная матрица N-го порядка. Получить целочисленную квадратную матрицу, в которой элемент равен 1, если соответствующий ему элемент исходной матрицы больше элемента, расположенного на главной диагонали, и равен 0 в противном случае.

10.28. Дана квадратная целочисленная матрица N-го порядка. Упорядочить элементы каждой строки по возрастанию.

10.29. Дана действительная квадратная матрица N-го порядка. Рассмотреть элементы, расположенные в строках, начинающихся с отрицательного элемента. Найти сумму тех из них, которые расположены выше главной диагонали матрицы.

10.30. Дана действительная квадратная матрица N-го порядка. Рассмотреть элементы, расположенные в строках, начинающихся с отрицательного элемента. Найти сумму тех из них, которые расположены на главной диагонали матрицы.

ЗАДАНИЕ 11. ТЕКСТОВЫЙ РЕЖИМ РАБОТЫ ДИСПЛЕЯ

Цель работы– изучить приемы составления программ с использованием процедур и функций, реализованных в модуле CRT и поддерживающих текстовый режим работы дисплея.

Теоретические сведения

Аббревиатура CRT – это английское сокращение от слова «электронно-лучевая трубка», говорит о том, что большинство процедур и функций этого модуля относятся к выводу на экран текстовой информации. Подключение модуля:

```
Uses CRT;
```

Имеет смысл всегда подключать модуль Crt, даже если его процедуры или функции не используются в программе.

Вывод специальных символов

При подключении модуля CRT можно выводить на дисплей строки и символы, содержащие в себе управляющие коды (0..31). При этом они не будут оказывать управляющего воздействия, а будут изображаться на дисплее согласно таблице их ASCII кодов. Исключение составляют лишь четыре кода:

#7 – вызывает один короткий звук динамика;

#8 – сдвигает текущую позицию курсора влево на один символ, если есть куда сдвинуться в пределах строки; в противном случае не имеет эффекта;

#10 – переводит текущее положение курсора на строку ниже, не меняя текущего столбца;

#13 – переводит текущее положение курсора в начало строки.

Пример. Составить демонстрационную программу использования четырех управляющих кодов экрана:

```
Program Demo_4_Cods;
  Uses Crt;
  Var I:Byte;
  Begin
    ClrScr;
    Writeln('Нажмите клавишу ввода для продолжения');
    Readln;
    Writeln('Эффект от кода 7 – короткий звук'#7);
      Writeln;
        Writeln;
          Readln;
    Writeln('Демонстрация кода возврата –#8 (BS)');
    For I:=1 To 40 Do Write('/');        { 40 правых косых скобок }
```

```
  For I:=1 To 40 Do
    Begin
      Delay(100);                  { задержка в 100 мс }
      Write(#8,'\',#8);    { передвижение на символ влево, за-   }
    End;                 { мена на \ и снова один сдвиг влево }
  Writeln;
    Writeln;
      Readln;
  Writeln('Демонстрация кода разрыва строки - #10 (LF)');
  Writeln;
  Write('Эта '#10'строка '#10'разорвана '#10'кодом 10');
  Writeln(#10#10);
  Readln;
  Writeln('Работа с кодом "возврата каретки" - #13'#10);
  For I:=1 To 40 Do Write('/');      { 40 правых косых скобок   }
  Write(#13);                        { перевод курсора в начало }
  For I:=1 To 40 Do Begin
    Delay(100);          { задержка в 100 мс}
    Write('\');          { поочередная печать '\'   }
      End;
  Writeln;
    Readln
End.
```

Системные переменные модуля CRT

Всего их определено восемь.

CheckSnow : Boolean – на цветных дисплеях с видеоадаптером CGA могут наблюдаться белые штрихи при смене изображения. Это явление, вызванное рассогласованием между обновлением видеопамяти и сменой изображения, называют снегом (Snow). Для его устранения переменной CheckSnow следует присвоить значение True.

CheckBreak: **Boolean –** если ее значение равно True (стартовое значение), то нажатие клавиш Ctrl + Break во время выполнения операций ввода-вывода будет прерывать работу программы. Нажатие Ctrl + Break не во время ввода-вывода информации не имеет эффекта. Если установить CheckBreak := False, то механизм прерывания работы программы с помощью Ctrl + Break отключается;

TextAttr : Byte – может принимать значение от 0 до 255. В ней хранятся текущие цветовые атрибуты для фона, символов и атрибут мерцания символов. Каждый из восьми битов переменной TextAttr содержит определенную информацию (табл. 11.1).

Таблица 11.1

Номер бита	7	6 5 4	3 2 1 0
Что определяет	Мерцание: 1 – да 0 – нет	Цвет фона – 8 значений 0*16..7*16	Цвет символов – 16 значений от 0 до 15
компоновка Цвета (RGB)		красный зеленый голубой	Ярко-красный зеленый голубой

Переменную TextAttr можно использовать для управления цветовым режимом вывода символов на экран с помощью формулы

TextAttr := Цвет символов (0..15) + 16 * Цвет фона (0..7) [+128];

Запись [+128] означает необязательный атрибут мерцания. Когда 128 добавлено, надпись будет мерцать. При такой установке цвета становятся почти «бесполезными» процедуры TextColor и TextBackGround, которые всего лишь изменяют соответствующие биты системной переменной TextAttr.

В приведенной формуле вместо цифр можно указывать цветовые константы (они рассмотрены вместе с TextColor и TextBackGround). Например:

TextAttr := White + 16 * Red + Blink;

что задает мерцающий белый цвет на красном фоне.

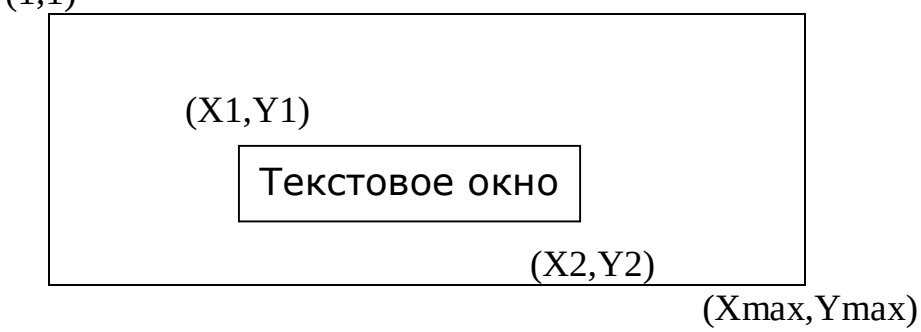
Монохроматические мониторы способны отображать лишь три градации яркости: черный, белый и ярко-белый. Текст может быть инверсным или подчеркнутым. Все эти атрибуты задаются по-прежнему в TextAttr, но их кодировка уже иная (табл. 11.2).

Таблица 11.2

Номер бита	7	6 5 4	3	2 1	0
Что определяет	Мерцание: 1 – да 0 – нет	Цвет фона/ Инверсия	Яркость: 1 – да 0 – нет	Цвет символа	Подчеркивание: 1 – да 0 - нет

Процедуры и функции модуля Crt

Процедура **Window (X1, Y1, X2, Y2 : Byte)** устанавливает текущее текстовое окно на экране согласно схеме:



Координаты диагонали окна X1, Y1, X2, Y2 всегда отсчитываются от левого верхнего угла экрана в абсолютных координатах и должны удовлетворять следующим условиям:

$$1 <= X1 < X2 <= Xmax; \qquad 1 <= Y1 < Y2 <= Ymax;$$

Параметр Xmax может принимать два значения – 40 и 80 (при видеоадаптере MDA/Hercules только 80), а Ymax – два или одно в зависимости от типа видеоадаптера: 25 (все типы) и 43 или 50 (EGA или VGA соответственно).

После выполнения процедуры Window все операции с экраном относятся к той его части, которая определена координатами X1, Y1, X2, Y2. Отсчет строк и столбцов теперь будет выполняться в координатах текущего окна, и позиция (1,1) – это верхний левый угол окна. Часть экрана вне окна практически изымается из обращения и становится невидимой.

При создании очередного окна его координаты всегда даются в «абсолютных» экранных координатах, а не в относительных координатах последнего текстового окна.

Пример использования окон:

```
Program Windows;
  Uses Crt;                          { используется модуль CRT }
  Var I:Integer;
  Begin
    TextAttr:=White + 16*Black;      { цвет белый на черном}
    ClrScr;
    For I:=1 To 153 Do
      Write('*Полный экран');        { вывод на основной экран }
    Repeat
      TextAttr:=White + 16*Red;      { цвет белый на красном   }
      Window(5,5,20,15);             { задание одного окна     }
      For I:=1 To 150 Do
        Write('*Окно 1');            { вывод текста в это окно }
      ClrScr;                        { очистка первого окна    }
      TextAttr:=White + 16*Blue;     { цвет белый на синем     }
      Window(40,10,55,20);           { задание другого окна    }
      For I:=1 To 150 Do
        Write('*Окно 2');            { вывод текста в это окно }
      ClrScr;                        { очистка второго окна    }
    Until KeyPressed;                { цикл до нажатия клавиши }
  End.
```

Переменные **WindMax и WindMin** типа Word хранят информацию о размерах текущего окна на дисплее. Каждая из переменных хранит одновременно координаты X и Y в младшем и старшем байтах соответственно. Извлечь значения X и Y можно с помощью встроенных функций Lo и Hi:

X1:=Lo (WindMin) + 1; X2:=Lo (WindMax) + 1;
Y1:=Hi (WindMin) + 1; Y2:=Hi (WindMax) + 1;

где X1, Y1, X2, Y2 – координаты диагонали окна.

Здесь ко всем значениям добавлена единица. Это сделано для совмещения с экранными координатами, у которых начало расположено в верхнем левом углу дисплея и имеет координаты (1,1). В переменных WindMax и WindMin все значения отсчитываются от нуля (начало координат – точка (0,0)).

Процедура **ClrScr** – очищает текущее текстовое окно или весь экран. При этом окно как бы закрашивается текущим цветом фона. Так, например, чтобы сделать поле экрана голубым, необходимо вставить в программу две строки:

```
TextBackGround (Blue);
ClrScr;
```

Кроме того, процедура ClrScr всегда устанавливает курсор в позицию с координатами (1,1) в текущем текстовом окне.

Процедура **ClrEOL –** может применяться как для стирания «хвостов» строк, так и для раскраски чистого экрана в полоску максимально быстрым способом. Смысл процедуры содержится в ее названии – «очистка до конца строки». Она стирает все символы в строке, начиная с текущей позиции курсора и до правого края текущего окна. Вместо стираемых символов она ставит пробелы, при этом цвет строки определяется цветовым атрибутом фона. Процедура не имеет других эффектов – даже курсор при выполнении процедуры не меняет своих координат.

Процедура **TextMode (M:Word)** – переключает текстовый режим вывода информации на дисплей. Для нее в Crt определено восемь констант (табл. 11.3).

Таблица 11.3

Имя = числовое значение	Разрешение экрана	Цветовой режим
BW40 = 0	40 x 25	Ч-б. при цветном адаптере
CO40=1 или C40=1	40 x 25	Цветной
BW80 = 2	80 x 25	Ч-б. при цветном адаптере
CO80=3 или C80=3	80 x 25	Цветной
Mono = 7	80 x 25	Монохроматический на дисплеях MDA/Hercules
Font8x8 = 256	80/40 x 43	Цветной – адаптер EGA
	80/40 x 50	Цветной – адаптер VGA
Все остальные числовые значения до 65 535, не являющиеся суммой одной из указанных констант (0, 1, 2, 3, 7), и Font8x8 включают в процедуре TextMode режим C80.		

Эти константы используются как параметры процедуры, например:

TextMode (CO80); или TextMode (3);

одинаковы по своему действию и включают цветовой режим.

В качестве параметра процедуры TextMode кроме констант и просто чисел может быть использована и системная переменная LastMode.

Из перечня констант выделяется Font8 x 8. В переводе она означает «шрифт 8 x 8» и включает режим большей текстовой вместимости дисплеев с видеоадаптерами EGA и VGA, в которых вместо стандартных шрифтов из матриц 8 x 14 и 8 x 16 подключаются более мелкие шрифты с матрицей 8 x 8, что дает увеличение числа строк до 43 (EGA) или 50 (VGA). Эта константа является дополнительной, т.е. прибавляется к выбранному коду режима, например:

```
TextMode (CO80 + Font8 x 8);
```

В этом случае включается цветовой режим CO80 одновременно со сменой разрешающей способности экрана с 80 x 25 на 80 x 43 (50).

Просто вызов TextMode (Font8 x 8) равносилен вызову TextMode (BW80+ + Font8 x 8).

Кроме смены режима текстового изображения команда TextMode обновляет значения системных переменных и производит переустановку цветовых атрибутов. Можно считать, что происходит выполнение следующих действий:

```
Window(1,1,Xmax,Ymax);   { окно делается равным всему экрану  }
DirectVideo <-- True;    { включается режим прямого вывода    }
CheckSnow   <-- False;   { отключается снятие «снега» (CGA)   }
NormVideo;               { устанавливается  стартовый атрибут }
ClrScr;                  { экран очищается }
LastMode <--Параметр;    { параметр в TextMode запоминается   }
```

Легко заметить, что «букет» эффектов от команды TextMode не располагает к слишком частому ее использованию в программе. Нормальное место процедуры TextMode – в начале программы и (или) в самом ее конце для включения и выключения режимов работы программы. С ней тесно связана системная переменная LastMode.

Переменная **LastMode**. В ней сохраняется текстовый режим работы, который установлен последним выполнением процедуры TextMode. Ее значение (типа Word) соответствует разрешенным значениям параметров TextMode. Самое первое значение LastMode соответствует режиму, из которого запускалась программа.

С помощью переменной LastMode можно реализовать «хороший тон» работы программы, когда после ее выполнения восстанавливается исходный текстовый режим:

```
Uses Crt;
Var
  StartMode : Word;
Begin
  StartMode := LastMode;     { сохранение стартового режима }
    . . .                    { работа в разных режимах }
```

```
  TextMode (StartMode)        { восстановление режима }
 End.
```

Процедура **GoToXY (X,Y : Byte)** устанавливает курсор в столбец X и строку Y текущего окна. GoToXY использует систему координат текущего текстового окна, т.е. координата (1,1) соответствует левому верхнему углу окна.

Пример. С помощью процедуры GoToXY вывести строку на экран вертикально:

```
Program Demo_GoToXY;
  Uses Crt;
  Procedure VertStr(X,Y : Byte; S : String);
    Var
      Len : Byte Absolute S;                { длина строки S }
       I : Byte;
    Begin
     For I:=1 To Len Do   { или можно:  For I:=1 To Length(s) Do }
       Begin
         GoToXY(x, y+Pred(i));       { назначение позиции вывода }
         Write(S[i])                 { вывод очередного символа  }
       End;
    End;
  Begin
    ClrScr;
    VertStr(40,7,'Вертикально !');
    Readln
  End.
```

Функции **WhereX** и **WhereY** нужны для определения текущего местоположения курсора в текстовом окне. Так, номер столбца можно получить функцией WhereX, а номер строки – WhereY. Эта пара функций обратна процедуре GoToXY.

Процедуры **InsLine** и **DelLine** работают со строками. Они позволяют «прокручивать» часть текстового окна или весь экран вверх или вниз. Так InsLine вставляет пустую строку на место той, где находится в текущий момент курсор. Все нижние строки, начиная с нее, смещаются вниз на одну строку. Самая нижняя строка при этом уходит за нижнее поле окна и исчезает.

Пример. Организовать выдачу информации на экран, при которой изображение смещается сверху вниз:

```
Program Demo_InsLine;
  Uses Crt;
  Var I : Byte;
  Begin
    ClrScr;
    For I:=1 To 25 Do
```

```
    Begin
      GoToXY(1,1); InsLine;        { расчистка места }
      Writeln('Строка N', i);
      Delay(200);
    End;
  Readln
End.
```

Процедура DelLine удаляет строку, в которой находится курсор, подтягивая на ее место все нижестоящие строки. При этом освобождается самая нижняя строка окна.

Пример. С помощью процедуры DelLine организовать «традиционную» прокрутку:

```
Program Demo_DelLine;
  Uses Crt;
  Var I,J : Byte;
  Begin
    ClrScr;
    For I:=1 To 7 Do
      Begin
        GoToXY(25,5+i);              { или: (25,6+Pred(i)); }
        Writeln('С т р о к а :  ',i);
        GoToXY(25,6+i);
        Delay(500);
      End;
    For J:=1 To 15 Do
      Begin
        GoToXY(25,6);
         Delay(500);
          DelLine;
          GoToXY(25,5+i);
         Delay(500);
        Writeln('С т р о к а :  ',7+j);
      End;
    Repeat Until KeyPressed
  End.
```

Процедуры **TextColor (C : Byte)** и **TextBackGround(C : Byte)** записывают в системную переменную TextAttr определенные значения. Процедура TextColor устанавливает цвет символа, а TextBackGround – цвет фона. Для этих процедур определены константы, соответствующие различным цветам (табл. 11.4).

Таблица 11.4

Константа	Число	Цвет	Процедуры
Black	0	Черный	TextColor, TextBackGround
Blue	1	Синий	То же
Green	2	Зеленый	– // –
Cyan	3	Циан (морской волны)	– // –
Red	4	Красный	– // –
Magenta	5	Фиолетовый	– // –
Brown	6	Коричневый	– // –
LightGray	7	Ярко-серый	– // –
DarkGray	8	Темно-серый	TextColor
LightBlue	9	Голубой	То же
LightGreen	10	Зеленый	– // –
LightCyan	11	Светлый циан	– // –
LightRed	12	Ярко-красный	– // –
LightMage-	13	Светло-фиолетовый	– // –
nYellow	14	Желтый	– // –
White	15	Белый	– // –
Blink	128	Мерцание	TextColor (как слагаемое)

В процедурах достаточно указать нужный цвет, подставив соответствующую константу, например:

```
TextColor (Red + Blink);
TextBackGround (LightGray);
```

В результате будет установлен мигающий красный цвет символов на ярко-сером фоне.

Процедуры установки яркости **HighVideo** и **LowVideo** устанавливают бит яркости в значение «да» (1) или «нет» (0). Процедура LowVideo изменяет цвет выводимых символов относительно указанного в процедуре TextColor, делая их темными. Формально это действие означает, что если установленный символ был от 8 до 15, из него вычитается 8. Процедура HighVideo имеет обратное действие – она делает все символы яркими, увеличивая на 8 цвета, заданные в диапазоне 0…7. Обычно это темно-серые символы на черном фоне. Процедура NormVideo используется для восстановления тех цветовых атрибутов (цвет фона, символов и мерцания), которые были на момент начала работы программы. Следует отметить, что процедуры LowVideo, HighVideo и NormVideo действуют на выводимый текст до очередного объявления цвета процедурой TextColor, которая отменяет действие любой из этих процедур.

Пример

```
Program High_Low_Video;
  Uses Crt;
Begin
  TextColor(LightGray);                { неяркий белый цвет   }
    TextBackGround(Black);             { черный цвет фона     }
  ClrScr;
  Write('Легко использовать');
    HighVideo;                         { включение яркости    }
  Write(' яркость ');
    LowVideo;                          { выбор низкой яркости }
  Write('для выделения слов');
    Readln;
  ClrScr
End.
```

Обязательным системным средством PC является генератор, к которому подключен встроенный динамик. Подобный звуковой канал способен генерировать только однотональный звук заданной частоты. В модуле CRT есть две процедуры управления встроенным динамиком: процедура **Sound(Hz : Word)** включает звук с заданной частотой в герцах, который будет продолжаться до тех пор, пока не будет выполнена процедура **NoSound**. Очень часто процедуры Sound и NoSound используются вместе с процедурой задержки времени **Delay (ms : Word)**, программирующей паузу в миллисекундах. Например, строка программы

```
Sound (300);  Delay (1000);  Nosound;
```

издает ровный звук на частоте 300 Гц продолжительностью 1 с. Но при этом во время звучания программа будет «стоять».

Пример.

```
Program Demo_Sounds;
  Uses Crt;
  Procedure Phone;                     { телефонный сигнал }
    Var I: Word;
  Begin
    Repeat
      For I:=1 To 100 Do Begin         { собственно сигнал }
        Sound(1200);  Delay(10); NoSound;
      End;
      Delay(1500)                      { задержка 1.5 с }
    Until keyPressed
  End; {Phone}
  Procedure Bell;                      { резкий сигнал     }
    Begin
```

```
    Repeat
      Sound(1800);  Delay(2);
      Sound(2000);  Delay(2);
      Sound(2200);  Delay(2);
      Sound(2400);  Delay(2)
    Until KeyPressed;
    NoSound
  End; {Bell}
 Procedure Sirena;                    { имитация сигнала тревоги }
  Var I : Word;
  Begin
    Repeat
      For I:=400 To 800 Do Begin
        Sound(i);  Delay(3)             { восходящий тон }
      End;
      For I:=800 DownTo 400 Do Begin
        Sound(i); Delay(3)              { нисходящий тон }
      End;
      NoSound
    Until KeyPressed
  End; {Sirena}
 Procedure Pause;
  Var Ch : Char;
 Begin
  While KeyPressed Do Ch:=ReadKey;          { очистка буфера }
  Delay(200)
 End; {Pause}
Begin
 ClrScr;
 Write('Нажите любую клавишу'#10#10#13);
     { Вызовы процедур - "исполнителей" }
 Write('  Звук телефона'#13);  Phone;
 Pause;
 Write('  Звук зуммера '#13);  Bell;
 Pause;
 Write('  Звук сирены  '#13);  Sirena;
 Pause;
 ClrScr
End.
```

Для написания музыкальных мотивов можно составить таблицу нот по рабочей формуле (в терминах Турбо Паскаля):

```
Var    Hz  : Word;
       Okt  : Integer;
      Nota  :  Byte;
```

```
Hz:=Round(440 * Exp(Ln(2) * (Okt - (10 - Nota)/12)));
```

Здесь Okt – номер одной из восьми октав, покрывающих диапазон от 32 Гц до почти 8 кГц. Самая низкотональная октава в таком диапазоне имеет отрицательный номер (-3), и дальнейшая нумерация соответственно будет -2, -1, 0, 1, ..., +4. Параметр Nota – это номер ноты в октаве:

"ДО" --> 1, "ДО-диез" --> 2, "РЕ" --> 3, ...,"СИ" --> 12.

Опрос клавиатуры

Работа клавиатуры происходит следующим образом. При нажатии любой из клавиш возникает аппаратное прерывание, подключающее DOS к обработке вызываемых клавиатурой сигналов, так называемых скан-кодов. Скан-коды выдаются как при нажатии, так и при отпускании клавиш, что, в частности, позволяет обнаруживать одновременное нажатие клавиш. DOS, обработав скан-коды, помещает в специальный буфер клавиатуры результаты этой обработки в виде кодов нажатия клавиш. В большинстве случаев в программах не требуется обработка скан-кодов клавиатуры и можно ограничиться обработкой символов из буфера клавиатуры. Коды, соответствующие клавишам буквенно-численного набора, с учетом переключения регистров клавишами Shift и CapsLock соответствуют ASCII кодировке этих символов. Другой способ получить в буфере клавиатуры требуемый код – при нажатой клавише Alt набрать числовой код символа. В частности, таким образом, вводятся символы псевдографики и другие специальные символы, отсутствующие на клавиатуре. При нажатии функциональных клавиш F1-F10, клавиш управления курсором и некоторых других, а также при совместном нажатии клавиш, например Alt+..., Control+... , в буфер клавиатуры помещается не один, а два символа, первый из которых #0. Такая пара называется «расширенный код клавиатуры».

Буфер клавиатуры организован по принципу очереди: поступающие символы попадают в конец очереди, а выбираются символы из начала очереди. Длина ограничена 15 символами.

Функция **KeyPressed**, не имеющая аргумента, дает значение True, если нажата какая-то клавиша. Она проверяет наличие символов в буфере клавиатуры, не изменяя состояние этой очереди.

Функция **ReadKey** имеет тип Char и возвращает первый символ из буфера клавиатуры, уменьшая размер очереди. Если очередь пуста, происходит задержка выполнения программы до нажатия клавиши.

Функция KeyPressed применяется в программах для проверки факта нажатия какой-либо клавиши, например, ее можно использовать для остановки циклов Repeat или While:

```
Uses CRT;
Begin
  Repeat
   Write('Этому нет конца');
```

```
Uses CRT;
  Begin
     While Not KeyPressed Do
      Write('Этому нет конца')
```

```
Until KeyPressed   End.                    End.
```

Эти программы будут выводить на экран текст до тех пор, пока не будет нажата одна из клавиш. Возможно сочетание функций KeyPressed и ReadKey:

```
If KeyPressed Then Key:=ReadKey;
```

Переменной Key присваивается значение символа из буфера только при нажатии какой-либо клавиши. Если клавиша не нажата и буфер пуст, выполняется следующий оператор программы.

Иногда возникает потребность прервать выполнение программы, сохранив при этом результат работы. Обычно в таких программах имеется цикл вычислений, который и может быть прерван пользователем. Последующий фрагмент программы показывает решение данной задачи:

```
For I:= 1 To N Do
  Begin
     {здесь следует содержательная часть цикла}
   If KeyPressed Then            { если нажата клавиша, то }
     Begin
        Key:=ReadKey;                { чтение символа }
        Case Key Of                 { анализ символа }
           's','S': Begin
                    Saving;   { обращение к процедуре сохранения }
                      Halt  { выход          }
                    End;
           'e','E': Halt       { выход без сохранения }
        End                     { конец оператора Case }
     End
  End;                          { конец цикла         }
```

Функции KeyPressed и ReadKey могут также использоваться для задержки работы программы до нажатия какой-либо клавиши: например циклы

```
Repeat Until KeyPressed;      или      While Not KeyPressed Do;
```

Необходимо иметь в виду, что функция KeyPressed оставляет код клавиши в буфере памяти, что может повлиять на дальнейшее чтение из буфера. Функция ReadKey извлекает символ из буфера, но только один, а в случае нажатия клавиш, дающих расширенные коды, в буфере будет два символа. Задержка программы и очистка буфера обеспечиваются оператором

```
If ReadKey = #0 Then ReadKey;
```

В начале выполнения этого оператора происходит обращение к буферу памяти. Когда буфер пуст, программа ожидает нажатия клавиши, а когда в нем есть информация, она считывается и происходит проверка логического условия. Если символ имеет отличный от 0 код, символ не является расширенным. Считывание из буфера заканчивается, и выполняются следующие операторы: если была нажата клавиша, дающая расширенные коды, логическое условие оператора If истинно, и из буфера выбирается вторая часть расширенного кода.

Оператор ReadKey, необходимый, чтобы приостановить программу и рассмотреть экран, может сработать, не ожидая нажатия клавиш, если в буфере экрана остался невыбранный «мусор» от нажатий. Такая ситуация часто возникает, если при работающей программе случайно нажали на клавиатуру. Результаты такого нажатия хранятся в буфере, и при первом же обращении к ReadKey будут использованы этой функцией. Поэтому надежное ожидание нажатия клавиши для продолжения работы должны содержать два оператора:

```
While KeyPressed Do ReadKey;       { очистка буфера клавиатуры }
If ReadKey = #0 Then ReadKey;      { ожидание нажатия клавиши  }
```

Первый оператор выбирает весь «мусор», оставшийся в буфере клавиатуры к моменту обращения, второй – ожидает нажатия очередного символа.

Функция ReadKey удобна, когда необходимо ввести с клавиатуры один символ, определяющий, продолжить ли работу программы. Выбор между продолжением и завершением работы может обеспечить конструкция

```
Write('Продолжать? (y/n)');
Repeat
  Key:=ReadKey
Until Key in['y','Y','n','N'];
Case Key Of
  'y','Y': . . . (продолжение работы);
  'n','N': . . . (завершение работы);
```

Здесь при нажатии любых других клавиш, кроме 'y' и 'n', цикл Repeat повторяется, и только нажатие на одну из этих клавиш приводит к выходу из цикла и продолжению или завершению работы. Отличие этого способа ввода от ввода оператором Read(Key) состоит в том, что не надо нажимать клавишу Enter и вводимый символ не отображается на экране.

Задача 11. Варианты

Во всех заданиях в последней строке экрана сформировать подсказку о «горячих» клавишах и реализовать их в программе. Предусмотреть завершение выполнения программы нажатием клавиши «ключ» (ESC).

11.1. Составить программу вывода на экран двух окон, обрамленных рамкой. В первое окно ввести строку текста. При нажатии клавиши «Ввод» во втором окне появляется ее копия в виде непрерывно «бегущей» справа налево строки.

11.2. То же, что в задании 11.1, но строка «бежит» сверху вниз.

11.3. Составить программу вывода на экран трех вертикальных окон. В первое окно ввести вертикальную строку текста. Нажатием клавиши «Ввод» организовать непрерывное движение копии строки по «кольцу» последовательно через второе и третье окна. Направление «бегущей строки» сверху вниз во втором окне.

11.4. То же, что в задании 11.3, но направление «бегущей строки» снизу вверх во втором окне.

11.5. Составить программу вывода на экран горизонтального меню, включающего пункты 1, 2, 3 и «Выход». Сформировать курсор. Предусмотреть активизацию пункта меню перемещением в него курсора и нажатием клавиши «Ввод». При активизации пункта 1 формируется окно, в котором находится информация о названии вуза; пункта 2 – окно с информацией о номере группы студента; пункта 3 – окно с ФИО студента. При активизации «Выход» программа завершает работу.

11.6. То же, что в задании 11.5, но при активизации любого пункта меню на экране формируется и сохраняется только одно окно, соответствующее этому пункту.

11.7. То же, что в задании 11.5, но меню расположено по вертикали.

11.8. То же, что в задании 11.5, но меню организуется в виде прямоугольников, расположенных последовательно по диагонали.

11.9. Составить программу вывода на экран горизонтального меню, состоящего из четырех пунктов и «Выхода». Названия пунктов считывать из файла при последовательной активизации каждого пункта меню курсором. При этом учесть, что строка файла – это один пункт меню. Если в файле меньше четырех строк, то выводится сообщение об ошибке. При активизации «Выход» программа завершает работу.

11.10. То же, что в задании 11.9, но меню организуется в виде прямоугольников, расположенных последовательно по диагонали.

11.11. Составить программу вывода на экран двух окон. Первое – обрамлено рамкой и в него занесен текст, второе – сформировано в виде ячеек разного оттенка цвета. Перемещение по ячейкам осуществляется стрелками управления движения курсора, при этом цвет фона окна с текстом меняется в соответствии с цветом ячейки.

11.12. То же, что в задании 11.11, но изменяется цвет символов.

11.13. То же, что в задании 11.11, но изменяется цвет рамки первого окна.

11.14. Составить программу вывода на экран окна, обрамленного рамкой, заполнить его текстом с помощью ReadKey. При нажатии клавиши «Ввод» символы текста должны изменить свой цвет (включая мерцание) по случайному закону. В программе учесть, что первый генератор случайных чисел используется при выборе цвета символа текста, а второй – при выборе цвета фона. Предусмотреть возможность дискретного изменения длительностей свечения символов.

11.15. Составить программу вывода на экран окна, обрамленного рамкой. Окно состоит из ячеек. При нажатии клавиши «Ввод» первый генератор случайным образом меняет цвет ячейки, второй – цвет фона рамки окна. Изменение цвета ячейки сопровождается изменением тональности звука.

11.16. Составить программу вывода на экран окна и заполнить его текстом с помощью ReadKey. Окно разбить на вертикальные столбцы шириной в один символ. Столбцы закрасить периодически повторяющимися цветами фона

с номерами от единицы до восьми. При нажатии клавиши «Ввод» цвет фона i-го столбца меняется на цвет фона i+1 столбца по замкнутому кольцу.

11.17. То же, что в задании 11.16, но для строк.

11.18. То же, что в задании 11.16, но изменение цвета фона происходит только во время нажатия клавиши управления движением курсора «Влево» или «Вправо».

11.19. Составить программу, реализующую работу одноактавного музыкального инструмента, с использованием цифровых клавиш от 0 до 9 и клавиш с символами <-> и <=>. Нажатие любой из этих клавиш должно сопровождаться соответствующим тоном звука и появлением на экране в таблице клавиш значения частоты звука.

11.20. Составить программу вывода на экран окна и заполнить его текстом с помощью ReadKey. При нажатии клавиши «Ввод» в границах этого окна формируется второе «непрозрачное» окно меньших размеров. С помощью клавиш управления движением курсора оно может перемещаться в границах первого окна, при этом должен восстанавливаться ранее закрываемый текст первого окна.

11.21. То же, что в задании 11.20, но цвет фона второго окна выбран таким, что оно «прозрачно» для текста первого окна.

11.22. Составить программу вывода на экран окна, которое условно «разбито» на вертикальные столбцы. Верхние части столбцов окрашены одним фоном, нижние – другим. Граница раздела фонов столбцов формирует синусоиду произвольной амплитуды с периодом, равным ширине окна.

При нажатии клавиши «Ввод» начинают изменяться соотношения заполнения столбцов разными фонами, т.е. создается эффект «бегущей» слева направо волны.

11.23. То же, что в задании 11.22, но волна бежит справа налево.

11.24. То же, что в задании 11.22, но для строк и волна «бежит» сверху вниз.

11.25. То же, что в задании 11.24, но волна бежит снизу вверх.

11.26. Составить программу вывода на экран основного (внешнего) окна и заполнить его текстом с помощью ReadKey. При нажатии клавиши «Ввод» внутри этого окна появляется новое окно меньших размеров с другим цветом фона. При этом текст основного окна должен быть сохранен. Повторное нажатие «Ввод» формирует следующее окно еще меньших размеров со своим фоном. Текст основного окна должен все время сохраняться.

11.27. То же, что в задании 11.26, но сначала формируется само внутреннее меньшее окно и заполняется текстом, а затем – очередное внешнее.

11.28. Составить программу вывода на экран окна, заполнить его текстом с помощью ReadKey. В пределах окна сформировать курсор. При его перемещении с помощью клавиши управления движением курсора за ним должен оставаться «след» с другим фоном. При пересечении «следа» со «следом» в точке пересечения фон должен измениться на фон окна.

11.29. Составить программу вывода на экран двух окон. В первом окне набрать целые или действительные числа. Если число набрано нужного типа, то оно отображается во втором окне. В противном случае – сообщение об ошибке. Предусмотреть коррекцию числа в первом окне.

11.30. Разработать текстовый редактор содержимого одной строки. Предусмотреть режим вставки, удаления и замены символов.

ЗАДАНИЕ 12. ГРАФИЧЕСКИЙ РЕЖИМ РАБОТЫ ДИСПЛЕЯ

Цель работы– изучить приемы составления программ с использованием процедур и функций, реализованных в модуле GRAPH и поддерживающих графический режим работы дисплея.

Теоретические сведения

Модуль Graph представляет собой библиотеку подпрограмм, обеспечивающую полное управление графическими режимами различных адаптеров дисплеев: CGA, EGA, VGA, MCGA, Hercules, PC3270, AT&T6300 и IBM8514. Чтобы запустить программу, использующую процедуры модуля Graph, необходимо, чтобы в рабочем каталоге находились соответствующие графические драйверы (файлы с расширением .BGI). Файл BGI – это графический интерфейс (Borland Graphic Interface) фирмы Borland. Он обеспечивает взаимодействие программ с графическими устройствами. Перед работой программы в графических режимах дисплея процедура InitGraph определяет тип адаптера, представленного в PC, и загружает в память соответствующий BGI-драйвер, в котором определены возможные режимы работы. Таким образом, в рабочем каталоге могут находиться следующие файлы:

CGA.BGI – драйвер для IBM CGA, MCGA;
EGAVGA.BGI – драйвер для IBM EGA, VGA;
HERC.BGI – драйвер для Hercules;
ATT.BGI – драйвер для AT&T6300;
PC3270.BGI – драйвер для IBM 3270PC;
IBM8514.BGI – драйвер для IBM 8514.

Такой набор файлов необходим при составлении программ, которые будут работать практически на всех PC, совместимых с PC, фирмы IBM. Если такая задача не стоит, то достаточно иметь один файл, соответствующий представленному в используемом PC графическому адаптеру.

Инициализация и закрытие графического режима

Любая программа, использующая графику, обязательно должна содержать блок вызова процедур инициализации графического режима и обращение к процедуре его закрытия. Такой блок инициализирует графический режим, проверяет правильность переключения и, если все операции прошли успешно,

допускает дальнейшую работу программы. Процедура инициализации объявляется следующим образом:

```
InitGraph( Var GrDriver : Integer;    { тип адаптера    }
           Var GrMode   : Integer;    { режим графики   }
           DrPath   : String);        { путь к драйверу }
```

В модуле Graph определены константы для задания вида графического адаптера параметром GrDriver перед вызовом InitGraph. Приведем наиболее часто используемые:

```
Const
  Detect   = 0;  { автоопределение }
  CGA      = 1;  { адаптер CGA     }
  MCGA     = 2;  { адаптер MCGA    }
  EGA      = 3;  { адаптер EGA 256K }
  VGA      = 9;  { адаптер VGA }
```

Если параметру GrDriver присвоить значение константы Detect, то система включится в режим автоопределения. При этом инициализируется соответствующий BGI-драйвер и включается режим с максимальным разрешением. В параметрах GrDriver и GrMode при этом будут возвращены автоматически выбранные значения или код ошибки. Если же параметр GrDriver содержит номер конкретного адаптера, то и второй параметр, GrMode, должен иметь значение (номер) режима, допустимого при этом адаптере.

Параметр DrPath указывает путь в каталог, содержащий файлы с необходимыми драйверами. Если в него передается значение (пустая строка), то драйверы должны находиться в текущем каталоге. Это же значение должно передаваться DrPath, если необходимые BGI-файлы преобразованы при помощи утилиты BINOBJ в файлы типа .OBJ, а затем скомпонованы с программой в EXE-файл.

Для завершения работы в графическом режиме необходимо производить вызов процедуры CloseGraph без параметров. Она очищает экран и переводит адаптер в текстовый режим. Последующий возврат в графический режим возможен только через повторную инициализацию.

Обработка ошибок инициализации

В модуле Graph реализован способ проверки результата проведения графической операции. Он осуществляется с помощью функции

GraphResult : Integer;

которая возвращает код ошибки, лежащий в диапазоне от 0 до -15.

Отметим, что после одного вызова GraphResult следующий ее вызов даст нулевое значение, поэтому для дальнейшего использования результатов тестирования рекомендуется сохранить значение этой функции в какой-либо переменной.

Для быстрой выдачи простого сообщения о типе ошибки графической системы используется функция, преобразующая результат вызова функции GraphResult в сообщение, которое можно вывести на экран процедурой Write. Эта функция объявлена как

GraphErrorMsg(ErrorCode : Integer) : String;

В таблице, приведенной ниже, показаны наиболее часто встречаемые константы кодов ошибок и соответствующие им сообщения (табл. 12.1).

Таблица 12.1

Константа	Код	Сообщение об ошибке	Перевод и пояснения
GrOk	0	No error	Ошибки нет
GrNoInitGraph	-1	Graphics not install led (use InitGraph)	Графика не инициализирована
GrNotDetected	-2	Graphics hardware ¦ not detected	Графический адаптер не найден
GrFileNotFound	-3	Device driver file not detected	BGI-файла нет в указанном каталоге
GrNoLoadMem	-5	Not enough memory to load driver	Нет места в ОЗУ для загрузки драйвера
GrInvalidMode	-10	Invalid Graphics mode for selected driver	Невозможный режим для выбранного драйвера
GrIOError	-12	Graphics I/O error	Ошибка ввода-вывода графики
GrInvalidFontNum	-14	Invalid font number	Несущ. номер шрифта

Пример программы инициализации графического режима:

```
Uses Graph;                    {подключение модуля Graph }
   Procedure GrInit;           {инициализация режима графики}
    Var
     GrDriver : Integer;       {для графического адаптера}
     GrMode   : Integer;       {для графического режима }
     ErCode   : Integer;       {для кода ошибки }
    Begin
     GrDriver := Detect;       {режим автоопределения }
     InitGraph(GrDriver, GrMode, '');  {инициализация}
     ErCode := GraphResult;    {результат инициализации }
     If  ErCode <> grOk Then   {если не успешно, то...}
       Begin
        Writeln('Ошибка графики:', GraphErrorMsg(ErCode));
        Writeln('Программа остановлена.');
        Halt(1);
```

```
    End;
End;
      { Основная  программа }
Begin
  GrInit;                    {вызов инициализации}
  Line(0, 0, GetMaxX, GetMaxY);   {работа с графикой}
  Readln;                    {пауза, в ожидании "Ввода" }
  CloseGraph                 {закрытие режима графики}
End.
```

В дальнейшем процедуру GrInit лучше записать в отдельный файл (например, InitGraf.Pas) и использовать директиву включения этого файла при компиляции.

Классификация и анализ графических режимов

Возможные графические режимы (выборочно) приведены в табл.12.2. Во втором столбце даны имена предопределенных констант, которые можно передавать в процедуры, управляющие графическими режимами. Последний столбец показывает количество полноэкранных изображений, которые могут храниться в памяти видеоадаптера одновременно.

Таблица 12.2

Драйвер	Имя константы режима и ее значение	Разрешение экрана (в точках)	Палитра	Число видеостраниц
CGA	CGAC0 = 0 CGAC1 = 1 CGAC2 = 2 CGAC3 = 3 CGAHi = 4	320 x 200 320 x 200 320 x 200 320 x 200 640 x 200	4 цвета 4 цвета 4 цвета 4 цвета 2 цвета	1 1 1 1 1
MCGA	MCGAC0 = 0 MCGAC1 = 1 MCGAC2 = 2 MCGAC3 = 3 MCGAMed = 4 MCGAHi = 5	320 x 200 320 x 200 320 x 200 320 x 200 640 x 200 640 x 480	4 цвета 4 цвета 4 цвета 4 цвета 2 цвета 2 цвета	1 1 1 1 1 1
EGA	EGALo = 0 EGAHi = 1	640 x 200 640 x 350	16 цветов 16 цветов	4 2
VGA	VGALo = 0 VGAMed = 1	640 x 200 640 x 350	16 цветов 16 цветов	4 2

	VGAHi = 2	640 x 480	16 цветов	1

Для тестирования графического адаптера в модуле Graph объявлена процедура:

DetectGraph(Var GrDriver, GrMode : Integer);

Эта процедура может быть вызвана до инициализации графики. Через формальный параметр GrDriver возвращается значение из первого столбца таблицы, а через параметр GrMode – обычно последнее значение из соответствующего раздела второго столбца. Эти значения и рекомендуется подставлять в качестве фактических параметров в процедуру InitGraph.

Номер текущего графического режима для установленного драйвера определяется функцией

GetGraphMode : Integer;

а функция

GetMaxMode : Word;

возвращает номер максимального режима для графического адаптера; таким образом каждый драйвер поддерживает диапазон режимов 0…GetMaxMode.

Очистка экрана и переключение режимов

Для очистки графического экрана используются две процедуры, выполняющие почти одинаковые действия. Первая из них – **ClearDevice** – очищает экран и устанавливает указатель позиции в положение (0,0), а вторая – **GraphDefaults** – кроме очистки экрана устанавливает ряд параметров графической системы: графическое окно становится равным размеру окна, восстанавливается системная цветовая палитра, переназначаются цвета основных линий и фона экрана и т.д. Процедура GraphDefaults неявно вызывается при инициализации графики и выполняет, по сути, все стартовые установки графических параметров.

Переключение режимов осуществляется процедурой

SetGraphMode(GrMode : Integer);

которая переключает систему в указанный параметром GrMode графический режим и очищает экран монитора. При этом все дополнительные характеристики устанавливаются по умолчанию. Однако такие переключения возможны только в рамках текущего драйвера.

Иногда возникает ситуация, когда необходимо возвратить систему в текстовый режим, работавший до инициализации графики. Для разрешения ее можно воспользоваться процедурой CloseGraph, однако после нее возврат в графический режим должен проводиться через процедуру InitGraph, что довольно сложно. Если же воспользоваться процедурой **RestoreCrtMode**, то возвращение в графику будет достаточно простым. При работе RestoreCrtMode выгрузки графического драйвера не происходит, т.е. он остается в памяти активным. Обратное переключение осуществляется при помощи функции **GetGraphMode**, которая возвращает номер текущего графического режима, ус-

танавливает в исходное состояние все графические параметры модуля Graph и сбрасывает изображение с экрана.

Пример переключения текстового и графического режимов работы:

```
Uses Graph;                {подключен модуль Graph  }
 {$I InitGraf.Pas}         {процедура инициализации }
Const                      {константы-сообщения     }
 Gr_str  = 'Это графический режим';
 T_str   = 'А это текстовый режим';
 Gr_back = 'А это снова графический режим';
Begin
 GrInit;                   {инициализация графики   }
 Line(0,0,GetMaxX,GetMaxY);      {диагональ экрана       }
 OutTextXY(0,100,Gr_str);        {вывод первого сообщения }
 Readln;                   {пауза до нажатия ввода  }
 RestoreCrtMode;           {восстановление текстового режима }
 Write(T_str);             {вывод второго сообщения     }
 Readln;                   { пауза до нажатия ввода     }
 SetGraphMode(GetGraphMode);   {восстановление граф. режима  }
 Line(0,0,GetMaxX,GetMaxY);    {диагональ экрана            }
 OutTextXY(0,100,Gr_back);     {вывод третьего сообщения    }
 Readln;                   {пауза до нажатия ввода      }
 CloseGraph                {закрытие графики            }
End.
```

Управление режимом вывода отрезков на экран

При рисовании отрезков на экране можно назначать режим поразрядного совмещения изображений. От него зависит, будет ли стерта при наложении двух точек «нижняя» и каким способом можно снять «верхнее» изображение с экрана. Управляя этим режимом, можно получать эффекты мультипликации. Сам режим задается процедурой

SetWriteMode(WriteMode : Integer);

Для задания параметра WriteMode в модуле Graph описаны пять констант, каждой из которых соответствует поразрядная операция (табл. 12.3).

Таблица 12.3

Имя константы	Значение	Логическая операция	Ее действие
CopyPut	0	MOV	Замещение
XORPut	1	XOR	Исключающее «ИЛИ»
ORPut	2	OR	«ИЛИ»
ANDPut	3	AND	«И»
NOTPut	4	NOT	«НЕ»

Поскольку рисование на экране, по сути, является действием с битами, при прорисовке точек производятся логические операции между битами памяти монитора и битами изображения. Для описываемой процедуры разрешены только первые две операции: первая – замещение (очистка перед прорисовкой) и вторая – XOR. Так, две последовательно проведенные логические операции XOR приведут биты памяти монитора в исходное состояние. Это означает, что если есть какое-нибудь изображение на экране, то использовав его в качестве фона и нарисовав на нем картинку, можно восстановить его, прорисовав картинку еще раз. При инициализации и после смены режимов устанавливается режим CopyPut.

Пример.

Программа, демонстрирующая использование операции XOR:

```
Uses Graph, Crt;                {подключены Graph и Crt    }
 {$I InitGraf.Pas}              {процедура инициализации   }
Var
  Dx,Dy,X,Y : Integer;          {рабочие переменные        }
  Maxx,Maxy : Integer;          {разрешение монитора       }
Begin
  GrInit;                   {инициализация графики     }
  Maxx := GetMaxX;              {присваивание максимальных }
  Maxy := GetMaxY;              {координат экрана          }
  Dx := Maxx Div 4;             {вычисление стороны        }
  Dy := Maxy Div 4;             {прямоугольника            }
  Bar3D(Dx,Dy,Maxx-Dx,
        Maxy-Dy,30,True);       {рисование параллелепипеда }
  SetWrireMode(XORPut);         {установка режима XOR      }
  Repeat                        {пока не нажата клавиша:   }
    X := Random(Maxx - Dx);     {случайная точка экрана    }
    Y := Random(Maxy - Dy);
    Rectangle(X,Y,X+Dx,Y+Dy);   {рисование прямоугольника  }
    Delay(200);                 {пауза в 200 мс            }
    Rectangle(X,Y,X+Dx,Y+Dy);   {стирание прямоугольника   }
  Until KeyPressed;             {условие окончания цикла   }
  Readln;                   {пауза до нажатия ввода    }
  CloseGraph                    {закрытие графики          }
End.
```

Режим, заданный процедурой SetWriteMode, распространяется только на рисование отрезками, т.е. на процедуры Line, LineTo, LineRel, Rectangle и DrawPoly.

Управление палитрой

Максимальный набор цветов, поддерживаемых одновременно BGI-драйвером, называется палитрой и может состоять из 16 цветов, пронумерованных от 0 до 15. Эти 16 цветов используются по умолчанию в режимах 640 x 480 для VGA, 640 x 350, 640 x 200 и 320 x 200 для EGA как в текстовом, так и в графическом режимах. Числа от 0 до 15 определяют цветовые атрибуты, или, как их еще называют, «программные» цвета. Каждому «программному» цвету присваивается «аппаратный» цвет из так называемой полной палитры. Например, для адаптера EGA, выводящего одновременно до 16 цветов, «программные» цвета выбираются из полной палитры в 64 цвета, имеющейся в этом адаптере. В адаптере VGA аппаратная палитра содержит 256 цветов. Для адаптеров CGA полная палитра составляет 16 «аппаратных» цветов, но одновременно появится лишь максимум четыре цвета одной из четырех программных палитр (C0..C3).

Для управления соответствием между «программными» и «аппаратными» цветами в модуле Graph предусмотрен ряд процедур. Для получения системной информации о палитре используют следующие процедуры. Так, в модуле Graph определен тип для описания палитры:

```
Const
  MaxColors = 15;      { максимальный "программный" номер цвета }
Type
  PaletteType = Record
                  Size : Byte;   { размер "программной" палитры }
                  Colors : Array[0..MaxColors] Of ShortInt;
                End;
```

Поле Size содержит количество цветов в палитре, а поле Colors – действующие цвета в первых Size элементах массива. Процедуры GetPalette и GetDefaultPalette возвращают в фактических параметрах значение типа PaletteType:

```
GetDefaultPalette(Var Palette : PaletteType);
  GetPalette(Var Palette : PaletteType);
```

Они отличаются друг от друга тем, что первая процедура возвращает набор цветов, который устанавливается при инициализации графического режима, т.е. по умолчанию, а вторая – возвращает текущий набор цветов. Функция GetPaletteSize : Word; возвращает результат Word, который показывает, какое количество цветов входит в текущую программную палитру. В принципе, эта функция возвращает значение, равное GetMaxColor+1.

Для установки палитры в модуле Graph представлены три процедуры разной сложности:

1. Процедура

   SetPalette(ColorNum : Word; Color : ShortInt);

управляет только одним цветом в палитре. Здесь ColorNum – это номер «программного» цвета, Color – номер «аппаратного» цвета, который будет под

ним пониматься. Например, вызов SetPalette(0,Red) делает красный цвет первым цветом палитры. Параметр Color может превышать максимальный программный номер цвета, но только на адаптерах EGA(0..63) и VGA. При некорректном вызове процедуры функция GraphResult вернет значение grError.

2. Процедура

SetAllPalette(Var Palette);

позволяет самостоятельно производить «перетасовку» всей палитры сразу и назначать соответствие между «программными» и «аппаратными» цветами. Параметр Palette является бестиповым, переменной длины. Первый его байт должен содержать количество цветов N в устанавливаемой палитре, следующие N байтов должны содержать цвета из аппаратной палитры, которые будут использоваться в дальнейшем. Каждый из этих байтов может принимать значения от -1 до максимального «аппаратного», причем диапазон чисел от 0 и выше представляет устанавливаемые цвета, а число -1 задается в том случае, если соответствующий цвет остается без изменений. Параметр Palette удобно представлять типом PaletteType, так как он имеет именно такую структуру.

Пример смены палитры приведен ниже:

```
        { Только для адаптеров EGA или VGA}
Uses Crt, Graph;              { подключение Graph и Crt     }
{$I InitGraf.Pas}             { процедура инициализации     }
Var
  Palette : PaletteType;          { переменная для палитры      }
  I,J,MaxC : Integer;             { счетчики; максимальный цвет }
Begin
  GrInit;                         { инициализация графики     }
  Palette.size := GetPaletteSize; { размер текущей палитры    }
  MaxC := Pred(Palette.size);     { макс. программный цвет    }
  For I := 0 To MaxC Do           { рисование вложенных разно-}
    Begin                          { цветных прямоугольников   }
      SetFillStyle(SlidFill, I);
      Bar(I*10, I*10, GetMaxX - I*10, GetMaxY - I*10)
    End;
  For I :=0 To 63-MaxC Do         { цикл по аппаратным цветам }
    Begin
{Сдвиг программных цветоа относительно аппаратных:}
      For J := 0 To MaxC Do Palette.colors[j] := J + 1;
      SetAllPalette(Palette);     { назначение новой палитры  }
      Delay(100)                  { пауза в 100 мс            }
    End;
  Readln;
  GetDefaultPalette(Palette);     { берется исходная палитра  }
  SetAllPalette(Palette);         { и восстанавливается       }
  CloseGraph                      { закрытие графики          }
```

End.

Заменяя палитры, можно «оживлять» изображения на экране, при условии, что все его «кадры» находятся в поле экрана и не пересекают друг друга. Допустим, есть 15 кадров, тогда назначается палитра со всеми цветами, равными фоновому. Далее рисуются все 15 кадров, причем первый кадр выводится первым цветом программной палитры, второй – вторым и т.д. После вывода 15-го кадра цветом номер 15 все кадры на экране будут «невидимы». Теперь достаточно с нужной задержкой устанавливать один из цветов палитры видимым, и мгновенно будет «проявляться» очередной кадр.

3. Более сложная процедура

SetRGBPalette(ColorNum,RedValue,GreenValue,BlueValue : Integer);

позволяет манипулировать цветовыми сочетаниями развитых графических адаптеров VGA и IBM8514. Параметр программного цвета ColorNum должен быть в диапазоне 0…15 для VGA и 0…255 для IBM8514. Последние три параметра показывают смешение красного, зеленого и синего цветов. Хотя они объявлены как Integer, BGI–драйвером воспринимается в них только младший байт, в котором, в свою очередь, значащими являются только 6 битов, с 0-го по 5-й (т.е. значения в диапазоне 0..63). Это сделано для совместимости по стандартам EGA.

В графическом режиме нет проблемы поменять палитру, но в текстовом режиме это возможно, используя функцию 10H прерывания 10H. Надо только правильно закодировать аппаратный цвет. Структура байта, представляющего собой один аппаратный цвет (составляемый смешением компонентов), показана ниже.

Бит	Компонент цвета
0 (B)	синий (интенсивность 2/3)
1 (G)	зеленый (интенсивность2/3)
2 (R)	красный (интенсивность 2/3)
3 (b)	синий (интенсивность 1/3)
4 (g)	зеленый (интенсивность 1/3)
5 (r)	красный (интенсивность 1/3)
6	не используется
7	не используется

Как видно, для каждого чистого цвета R, G и B имеется возможность получить три градации яркости: 1/3 интенсивности, 2/3 интенсивности и 1/3+2/3=1 интенсивности. Кроме того, возможны различные комбинации битов в регистрах. Возможное количество цветов равно двум в степени шесть (это от 0 до 63).

Как изменить программный цвет ColorNum на аппаратный RGBrgb, построенный по схеме, показано в ниже приведенной программе.

```
Uses DOS;                { необходим модуль DOS       }
Procedure SetPalColor(ColorNum, RGBrgb : ShortInt);
  Var
    Regs : Registers;      { нужна для вызова DOS.Intr() }
```

```
Begin
  With Regs Do Begin      { действия с полями Regs      }
    AX := $1000;         { Al=$00,AH=$10               }
    BL := ColorNum;      { номер изменяемого цвета     }
    BH := RGBrgb         { присваемое ему значение     }
   End   {with}
   Intr($10, Regs)       { прерывание 10H - сервис EGA }
End;
```

Работа с фрагментами изображений

Следующие две процедуры и одна функция используются для запоминания в буфере и восстановления из него прямоугольных фрагментов графического изображения. Это бывает удобно, так как дает возможность оперировать уже готовыми элементами изображений. При работе с фрагментом всегда важно знать его объем в байтах. Функция

ImageSize(X1,Y1, X2,Y2 : Integer) : Word;

возвращает размер памяти в байтах, необходимый для сохранения прямоугольной области экрана. Прямоугольник определяется координатами левого верхнего (X1,Y1) и правого нижнего (X2,Y2) углов. Эта функция обычно используется совместно с процедурой GetMem.

Записать изображение в буфер можно, используя процедуру

GetImage(X1,Y1, X2,Y2 : Integer; Var BitMap);

в которой параметры X1, Y1, X2, Y2 имеют то же значение, что и в ImageSize, а вместо бестипового параметра BitMap должна подставляться переменная-буфер, занимающая область памяти размером, необходимым для полного сохранения изображения (т.е. равного значению ImageSize). При этом максимальный размер сохраняемого изображения не должен превышать 64 К.

Процедура

PutImage(X1,Y1 : Integer; Var BitMap; Mode : Word);

восстанавливает изображение из буфера BitMap в прямоугольник, левый угол которого определен координатами (X1,Y1). В отличие от процедуры GetImage здесь нужна всего одна точка. Объясняется это тем, что в структуре BitMap первые два слова (четыре байта) содержат ширину и высоту в пикселах запомненного изображения. Наиболее интересным в этой процедуре является возможность определять режим вывода изображения: можно суммировать изображение на экране и изображение в буфере, уничтожать изображение, находящееся в определенной области, инвертировать изображение, содержащееся в буфере. Эти операции задаются параметром Mode, для которого в модуле Graph определены константы, уже названные при описании процедуры SetWriteMode. Например, если в режиме ORPut на малиновый цвет изображения (номер 5,

двоичная запись 0101) вывести бирюзовый (номер 3, 0011) из буфера, то результирующая картинка будет светло-серого цвета (номер 7, 0111). Из этих пяти режимов самым интересным является XOR, поскольку проведенные последовательно две операции XOR с одинаковым вторым аргументом оставляют первый из них без изменений. Это свойство операции XOR используется в тех случаях, когда необходимо изобразить некий подвижный объект на сложном фоне: два раза выведя один и тот же фрагмент в одном и том же месте в режиме XOR, оставим фон неизменным. Фактически так экономится память PC – не нужен буфер для запоминания участка фона, находящегося под новой картинкой.

Пример

```
Uses Graph, CRT;                { используются Graph и CRT }
{$I InitGraf.Pas}               { процедура инициализации  }
Const
  R = 10;                       { радиус подвижного шарика }
Var
  X1,Y1,X2,Y2,sx,sy : Integer;          { переменные для ожив- }
  maxx,maxy,sxmove,symove : Integer;    { ления фрагмента      }
  Size : Word;                  { размер фрагмента     }
  P : Pointer;                  { указатель на буфер   }
Begin
  GrInit;                       { инициализация графики      }
  maxx:=GetMaxX;                { максимальное поле экрана}
   maxy:=GetMaxY;
  X1:=maxx Div 2 - R;           { координаты области экрана, }
   Y1:=maxy Div 2 - R;          { в которой будет нарисован  }
  X2:=X1 + 2 * R;               { шарик и которая будет со-  }
   Y2:=Y1 + 2 * R;              { храненным фрагментом       }
  sx:=X1; sxmove:=3;            { начальная точка движения и }
   sy:=Y1; symove:=-1;                {шаг перемещения шарика}
  SetFillStyle(SolidFill,Red);          {выбор типа заливки}
  PieSlice(X1+R,Y1+R,0,360,R);          {рисование шарика}
  Size:=ImageSize(X1,Y1,X2,Y2);         {фрагмент в байтах}
  GetMem(P,Size);                {размешение буфера}
  GetImage(X1,Y1,X2,Y2,P^);      { фрагмент помещается в буфер}
  SetFillStyle(CloseDotFill,Blue);      { тип заливки фона }
  Bar(50,50,maxx-50,maxy-50);           { фоновая картинка }
  Repeat                        { начинается движение шарика }
    PutImage(sx,sy,P^,XORPut);          { вывод шарика       }
    Delay(12);                  {пауза (подбирается)}
    PutImage(sx,sy,P^,XORPut);          { стирание шарика    }
     {ограничения на движение шарика в пределах поля фона:}
    If (sx<50)Or(sx>maxx-50-2*R) Then sxmove:=-sxmove;
    If (sy<50)Or(sy>maxy-50-2*R) Then symove:=-symove;
```

```
    Inc(sx,sxmove);          { следующая точка появления  }
    Inc(sy,symove);          { шарика на экране}
  Until KeyPressed;          {... пока не нажата клавиша  }
  FreeMem(P,Size);           { освобождение памяти буфера }
  CloseGraph                 { закрытие режима графики    }
End.
```

В приведенном примере продемонстрирован алгоритм мультипликации, применяющий «битовые» методы (используется пересылка битовых массивов процедурами GetImage и PuImage). Скорость движения картинки сильно зависит от разрешения экрана и количества цветов. Следует обратить внимание на стандартную связку:

```
Size:=ImageSize(X1,Y1, X2,Y2);      { фрагмент в байтах      }
GetMem(P, Size);                    { размещение буфера      }
GetImage(X1,Y1, X2,Y2,P^);          { фрагмент занести в буфер }
 .   .   .
PutImage(X, Y, P^, xxxPut);         { вывод фрагмента        }
FreeMem(P, Size);                   { освобождение буфера    }
```

организующую хранение динамического фрагмента через переменную P типа Pointer. В вызовах PutImage/GetImage указатель P должен быть разыменован. Динамический буфер для фрагментов всегда предпочтительнее, поскольку сам размер фрагмента зависит от многих условий, которые трудно удовлетворить, используя для буфера статическую структуру.

Управление видеостраницами

Память видеоадаптеров разделена на так называемые видеостраницы. По умолчанию в графическом режиме действия производятся с нулевой страницей, поэтому в предыдущих примерах было видно, как рисуются на экране фигуры. Однако если направить вывод изображений на ненулевую страницу (при условии, что такая доступна в текущем режиме видеоадаптера), то на экране ничего не отобразится, поскольку по умолчанию видимой является нулевая страница. Если же после этого дать команду: считать видимой «скрытую страницу», то она появится на экране буквально мгновенно (конкретно – за один прямой проход луча в кинескопе). Проделать все это позволяют две процедуры:

SetVisualPage(Page : Word);

которая устанавливает «видимой» на экране видеостраницу номер Page, и процедура

SetActivePage(Page : Word);

устанавливающая перенаправление всех графических операций на страницу номер Page (т.е делающую активной). Ниже приведен типичный пример использования этих процедур применительно для адаптеров EGA и VGA.

Пример

```
Uses Graph, CRT;                { используются Graph и CRT }
```

```
{$I InitGraf.Pas}                { процедура инициализации  }
 Procedure Forms(Kadr : Byte);     { рисование кадров 0..3    }
  Const
   Radius : Array[0..3] Of Integer = (20, 40, 60, 80);
  Var
   R, Rr : Integer;            { радиусы эллипсов в кадрах }
  Begin
   R := Radius[kadr];          { максимальный радиус       }
   Rr := 0;                  { радиус вложенного эллипса }
   Repeat
    Ellipse(GetMaxX Div 2, GetMaxY Div 2, 0, 360, R, Rr);
    Inc(Rr, 5)
   Until Rr >=R
  End;
 Procedure AnimEGAVGA;          { процедура смены кадров    }
  Const
   Ms = 60;                  { задержка между кадрами, мс }
  Var I := Byte;              { параметр циклов смены }
  Begin
   Repeat                   { цикл до нажатия клавиши ...}
    For I := 0 To 3 Do        { смена видеостраниц: прямо  }
     Begin
      SetVisualPage(i);
      Delay(ms)
     End;
    For I := 3 DownTo 0 Do           {... и обратно   }
     Begin
      SetVisualPage(i);
      Delay(ms)
     End;
   Until KeyPressed;           { условие окончания показа }
  End;
 Var I : Byte;                { параметр (номер кадра)   }
    (******    Основная часть примера    ******)
 BEGIN
  GrInit;                   { инициализация графики     }
  SetGraphmode(EGALo);          { режим EGA, 640x200, 4 стр.}
  For I := 3 DownTo 0 Do        { цикл заполнения страниц:  }
     Begin
      SetVisualPage(Succ(i) Mod 4);      { видим "пустоту" }
      SetActivePage(i);               { и готовим кадр  }
      Forms(i);                    { рисунок кадра   }
     End;
  AnimEGAVGA;                { начало "оживления" кадров }
```

```
  CloseGraph                { закрытие режима графики    }
End.
```

В примере показано использование процедур SetActivePage и SetVisualPage для алгоритмов «кадровой» мультипликации. Особенность ее заключается в том, что все кадры (здесь их четыре) сначала записываются на соответствующие страницы, а затем производится последовательное переключение отображения страниц на дисплей процедурой SetVisualPage.

Графические окна

В модуле Graph для описания графического окна объявлены следующий тип и две константы:

```
Type
  ViewPortType = Record
                   X1,Y1,X2,Y2 : Ineger;     { граница окна    }
                   Clip : Boolean;           { режим отсечения }
                 End;
Const
  ClipOn = True;        { отсечение по границе окна включено  }
  ClipOff = False;      { отсечение по границе окна выключено }
```

Здесь первые элементы записи – это координаты прямоугольной области (графического окна), а Clip – это параметр, указывающий графической системе, что делать с изображением, попавшим за пределы этой области. Clip может принимать два значения: ClipOn указывает на то, что все элементы изображения обрезаются по границам графического окна, ClipOff указывает на то, что все рисуется без изменений, как бы «не глядя» на границы окна.

Объявление графического окна производится процедурой

SetViewPort(X1,Y1,X2,Y2 : Integer; ClipMode : Boolean);

где входные параметры соответствуют полям записи типа ViewPortType. После выполнения этой процедуры все текущие установки станут относиться к окну. Текущий указатель установится в его левый угол, и туда же перенесется начало системы координат дисплея, т.е. мы получим локальную систему координат. Если параметры процедуры заданы неправильно, то функция GraphResult возвратит ошибку grError (-11).

Назначение графического окна можно использовать для перемещения начала системы координат. Так, если задать окно вызовом

```
SetViewPort(GetMaxX Div 2, GetMaxY Div 2, GetMaxX, GetMaxY, ClipOff);
```

то получим систему координат с началом в центре экрана. При этом станет «видимой» адресация отрицательных координат. Графическое окно не меняет масштаба системы координат, а лишь выбирает систему отсчета адресуемых пикселов.

Для опроса текущих параметров графического окна служит процедура

GetViewSettings(Var ViewSettings : ViewPortType);

Если воспользоваться ею сразу же после инициализации графического режима, то обнаружится, что графическим окном является весь экран.

Для очистки рабочего пространства графического окна в модуле Graph существует специальная процедура **ClearViewPort**, которая работает следующим образом:

1) устанавливает цвет заполнения равным текущему цвету фона;

2) вызывает процедуру Bar с теми же значениями координат, что и у процедуры SetViewPort, вызванной перед этим;

3) перемещает текущий указатель в точку (0,0).

Несмотря на то, что понятие графического окна является общим для всех процедур и функций, одна процедура все же работает не по правилам: процедура PuImage в силу особенностей программной реализации работает одинаково как для значения параметра Clip, равного ClipOn, так и для значения ClipOff.

Вывод текста

Вывод текста в графическом режиме имеет ряд отличий от подобных, действий в данном режиме. Основное отличие состоит в том, что все действия производятся только со строковыми константами и переменными. Вся числовая информация должна предварительно преобразовываться в строковую (процедурой Str). Другое отличие – в том, что можно использовать различные штриховые шрифты, которые хранятся в файлах с расширением .CHR. Поскольку они построены не матричным способом (как сделаны стандартные шрифты для текстового режима), а векторным, становится возможной манипуляция соразмерами шрифтов без потери качества их изображения. Всего поставляется пять шрифтов. Для их обозначения введены константы:

```
Const
  DefaultFont   = 0;    { матричный шрифт 8 x 8 (по умолчанию) }
  TriplexFont   = 1;    { полужирный шрифт }
  SmallFont     = 2;    { светлый шрифт (тонкое начертание)  }
  SansSerifFont = 3;     { книжная гарнитура (рубленный шрифт)}
  GothicFont    = 4;    { готический шрифт }
```

Следует особо отметить DefaultFont– это специальный, системный матричный шрифт 8 x 8 для графических режимов, и если не принимать никаких действий по смене шрифта, то по умолчанию будет принят именно он.

Активизация любого из названных шрифтов осуществляется процедурой

SetTextStyle(Font, Direction : Word; Charsize : Word);

Здесь параметр Font – номер шрифта, Direction – расположение текста (по умолчанию принимается горизонтальное). Возможны лишь две ориентации текста, обозначенные константами:

```
Const
  HorizDir = 0;          { горизонтальная – слева направо }
  VertDir  = 1;          { вертикальная – снизу вверх     }
```

Размер каждого символа устанавливается параметром CharSize, диапазон изменения которого составляет от 1 до 10. Стандартное значение CharSize для матричного шрифта 8 x 8 равно единице, а для штриховых шрифтов – четырем.

При каждом вызове процедурой SetTextStyle какого-либо шрифта он читается с диска и загружается в память. При этом необходимо, чтобы файлы соответствующих шрифтов находились в известном каталоге совместно с BGI-файлами. В противном случае, не найдя их, система будет использовать DefaultFont. Кроме того, при быстром переключении между несколькими штриховыми шрифтами будет происходить задержка программы на время, необходимое для считывания соответствующего шрифта с диска. Это случается потому, что в рабочей памяти может храниться только один штриховой шрифт. Для хранения в памяти более одного шрифта одновременно необходимо предварительно разместить их в памяти и вызвать функцию

RegisterBGIFont(FontPtr : Pointer) : Integer;

которая регистрирует шрифт в системе и возвращает его номер или отрицательный код ошибки. После этого шрифт становится доступным в любой момент работы программы. Последовательность действий при этом должна быть следующей:

1. В динамической памяти («куче») отводится область размером с файл шрифта.
2. Файл шрифта считывается с диска и помещается в эту область.
3. Указатель на шрифт в памяти регистрируется при помощи функции RegisterBGIDriver.

Пример процедуры загрузки шрифта в память приведен ниже. В процедуру надо передать полное имя одного из CHR-файлов и сохранить возвращаемый ею указатель на место шрифта в памяти FPtr и размер Size (это может понадобиться при удалении шрифтов из «кучи» процедурой FreeMem(FPtr,Size)):

```
Procedure LoadFont(BGIFileName : String; Var FPtr : Pointer;
                                  Var Size : Word);

Var
  FontFile : File;
Begin
  Assign(FontFile, BGIFileName);   { связь файла с диском      }
  Reset(FontFile, 1);              { чтение сначала, побайтно }
  Size:=FileSize(FontFile);        { размер файла со шрифтом   }
  GetMem(FPtr, Size);              { выделение участка памяти  }
  BlockRead(FontFile, FPtr^, Size);     { загрузка туда шрифта }
  Close(FontFile);                      { закрытие файла }
  If RegisterBGIFont(FPtr) < grOK Then
    Begin                       { если возникла ошибка, то }
      Writeln('Ошибка загрузки шрифта: ',        { выдать    }
            GraphErrorMsg(GraphResult));        { сообщение }
      ReadLn
    End;
```

```
End;
```

Регистрация может производиться до инициализации графики. Так, в приведенном примере считается, что текущий режим – текстовый и используется оператор Writeln для выдачи аварийного сообщения. Функция RegisterBGIFont может возвращать значения ошибки.

Модуль Graph может поддерживать таблицу из 10 шрифтов. И если есть необходимость использовать новые шрифты (например, созданные самостоятельно), не имеющие идентификаторов в системе, то пользуясь функцией

InstallUserFont(FontFileName : String) : Integer;

можно установить соответствие между именем шрифтового файла FontFileName и его регистрационным номером в системе. После этого необходимо выбрать этот шрифт процедурой SetTexstStyle, указав первым параметром номер нового шрифта. Если таблица шрифтов уже заполнена, то функция возвращает значение 0 (DefaultFont). В общем случае ошибка установки нового шрифта диагностируется функцией GraphResult. После установки его можно обращаться с новым шрифтом как со стандартным, и в том числе загружать в память.

Непосредственный вывод строк

Для непосредственного вывода строк есть две процедуры. Первая –

OutText(TextString : String);

выводит на графический экран строку TextString, ориентированную относительно позиции текущего указателя, а вторая –

OutTextXY(X,Y : Integer; TextString : String);

выводит строку, ориентированную относительно координат (X,Y). Шрифт предварительно может быть установлен вызовом SetTexstStyle (по умолчанию принимается DefaultFont).

Относительно стартовой точки строка может быть ориентирована различными вариантами, которые задаются процедурой

SetTextJustify(Horizontal, Vertical : Word);

параметры, которой могут принимать одно из трех объявленных в модуле Graph значений:

```
Const
    {Для горизонтального ориентирования}
  LeftText   = 0;    { координата X задает левый край строки  }
  CenterText = 1;    { координата X задает середину строки    }
  RightText  = 2;    { координата X задает правый край строки }
     {Для вертикального ориентирования}
  BottomText = 0;    { координата Y задает нижний край строки  }
  CenterText = 1;    { координата Y задает середину строки     }
  TopText    = 2;    { координата Y задает верхний край строки }
```

По умолчанию параметры ориентировки соответствуют LeftText и TopText.

Отметим, что текстовые процедуры GoToXY, Write/Writeln и установка цвета текста в графическом режиме работают только, если переменная Crt.DirectVideo равна False (или модуль Crt не подключен). Ввод текста через Read/Readln действует всегда, при этом текст стирает фоновое изображение.

Размер букв и строк

Всегда важно знать вертикальный и горизонтальный размер выводимой строки в пикселах. Функции

TextHeight(TextString : String) : Word;

и

TextWidth(TextString : String) : Word;

возвращают высоту и ширину строк TextString в пикселах при условии, что они будут выведены текущим шрифтом и размером (т.е. заданными, последним вызовом SetTextStyle или по умолчанию). Для штриховых шрифтов размеры букв различаются (их начертание неравномерное), и длина и высота строки в пикселах зависит не только от количества букв в ней, но и от их начертания. В программе, приведенной ниже, проводится анализ расположения строк.

Пример:

```
Uses Graph;                    { подключен модуль Graph    }
 {$I Initgraf.Pas}             { процедура инициализации   }
Const
 My_str = 'Turbo Pascal';      { выводимая строка текста   }
Var
 Maxx, Maxy : Integer;         { текущее разрешение экрана }
 Tx, Ty, I, Y : Word;          { временные переменные      }
Begin
 GrInit;
 Maxx:=GetMaxX;  MaxY:=GetMaxY;          { разрешение    }
 SetTexstJustify(CenterText, CenterText);   { ориентация    }
 SetTextStyle(SmallFont, HorizDir, 6);      { стиль шрифта  }
 Tx:=TextWidth(My_str);                  { ширина строки }
 Ty:=TextHeight(My_str);                 { высота строки }
 For J:=1 To (Maxy Div Ty) Do               { цикл по оси Y }
  For I:=1 To (Maxx Div Tx) Do              { цикл по оси X }
    OutTextXY(I*Tx, J*Ty, My_str);          { тело цикла    }
 SetTextStyle(DefaultFont, HorizDir, 6);    { смена шрифта  }
 Tx:=TextWidth( 'W' ) Div 6;                { 1/6 ширины    }
 Ty:=TextHeight( 'E' ) Div 6;               { 1/6 высоты    }
 SetColor(LightRed);
 OutTextXY(Maxx Div 2 + Tx, Maxy Div 2 + Ty, My_str);
 SetColor(LightBlue);
```

```
  OutTextXY(Maxx Div 2, Maxy Div 2, My_str);
  Readln;
  CloseGraph                    { закрытие режима графики }
End.
```

У процедуры OutTextXY есть одна особенность: выводимая текстовая строка всегда обрезается по границе графического окна. Более того, если активным является матричный шрифт DefaultFont, то «влезающая» строка вообще не появляется на экране. Решать подобные проблемы можно, точно рассчитывая место выводимых в графике строк.

Размер букв (высота и ширина) штриховых шрифтов (и только их) можно задавать процедурой

SetUserCharSize(MultX, DivX, MultY, DivY : Word);

Она позволяет оперативно менять размер шрифта, установленный процедурой SetTextStyle. Отношение (MultX/DivX) задает масштабирование ширины начертания шрифта, а отношение (MultY/DivY) выражает масштаб изменения высоты шрифта. Например, задание параметров MultX=3 и DivX=1 говорит о том, что буквы выводимого шрифта будут в 3 раза шире нормы.

Полную информацию о текущем режиме вывода текста можно получить, используя процедуру

GetTextSettings(Var Settings : TextSettingsType);

В параметре Settings она возвращает исчерпывающую информацию обо всем, что относится к выводу строк. Тип этого параметра предопределен в модуле Graph:

```
Type
  TextSettingsType = Record
                       Font : Word;       { номер шрифта    }
                       Direction : Word;    { направление     }
                       CharSize : Word;     { размер шрифта   }
                       Horiz    : Word;     { ориентация по X }
                       Vert     : Word;     { ориентация по Y }
                     End;
```

Текущим всегда является один шрифт, и при необходимости быстро переключаться с одного шрифта на другой удобно сохранять и восстанавливать их параметры через переменные описания типа.

Включение шрифтов и драйверов в EXE-файл

Стандартный режим работы графики, при котором помимо основного EXE-файла необходимо присутствие еще одного или нескольких вспомогательных BGI- и CHR-файлов, не очень удобен. В связи с этим возможно включение содержимого этих файлов непосредственно в EXE-файл, получаемый из программы на Паскале. Для этого надо выбрать, какие драйверы и шрифты необходимы при автономной работе программы. Если она рассчитана на работу с одним адаптером, то достаточно одного соответствующего BGI-драйвера и

шрифта; если программа должна переноситься, то придется вставить в нее как минимум три драйвера: для CGA, EGA/VGA и Hercules. Затем нужно запустить утилиту BinObj.exe для получения из BGI- и (или) CHR-файла OBJ-файлов, что лучше сделать BAT-файлами:

1) Drivers.bat
   BINOBJ %1.BGI %1.OBJ %1DriverProc;
2) Fonts.bat
   BINOBJ %1.CHR %1.OBJ %1FontProc;

В этом случае можно обрабатывать драйверы и шрифты следующим образом:

```
C:\TP\BGI>Driver.bat CGA
C:\TP\BGI>Driver.bat EGAVGA
C:\TP\BGI>Driver.bat HERC
  .  .  .
C:\TP\BGI>Fonts.bat TRIP
C:\TP\BGI>Fonts.bat SANS
```

Если желательно включить в файл и свою часть шрифта 8 x 8 (пусть он хранится в файле 8 x 8.Fon), то надо выполнить команду

```
C:\TP\BGI>binobj 8x8.Fon 8x8 Font8x8Proc
```

После этого можно подготовить полученные OBJ-файлы для компоновки.

Пример Bgi.Tpu. Исходный его текст, с учетом предыдущих действий, приведен ниже:

```
UNIT BGI;                    { модуль с BGI-компонентами }
Interface                    { объявление псевдопроцедур }
 Procedure CGADriverProc;    { BGI-драйвер для CGA       }
 Procedure EGAVGADriverProc; { BGI-драйвер для  EGA/VGA  }
 Procedure HERCDriverProc;   { BGI-драйвер для  Hercules }
   {   . . .  }
 Procedure TRIPFontProc; {CHR-шрифт TriplexFont}
 Procedure SANSFontProc; {CHR-шрифт SansSerifFont}
   {   . . .  }
 Procedure Font8x8Proc; {матричный шрифт 8x8}
Implementation    {подстыковка содержимого}
Uses Graph, Dos;
 {$l Cga.Obj}     Procedure CGADriverProc;     External;
 {$l EgaVga.Obj}  Procedure EGAVGADriverProc;  External;
 {$l Herc.Obj}    Procedure HERCDriverProc;    External;
   {   . . .  }
 {$l Trip.Obj}    Procedure TRIPFontProc;      External;
 {$l Sans.Obj}    Procedure SANSFontProc;      External;
Var
 OldFont8x8 : Pointer;          { адрес старого шрифта 8x8 }
 {$l 8x8.Obj}     Procedure Font8x8Proc;       External;
Begin
 If RegisterBGIDriver(@CGADriverProc) < 0
```

```
    Then Halt(101);
  If RegisterBGIDriver(@EGAVGADriverProc) < 0
    Then Halt(102);
  If RegisterBGIDriver(@HERCDriverProc) < 0
    Then Halt(103);
     {   . . .  }
  If RegisterBGIFont(@TRIPDriverProc) < 0 Then Halt(201);
  If RegisterBGIFont(@SANSDriverProc) < 0 Then Halt(202);
     {   . . .  }
  GetIntVec($1F, OldFont8x8);       { старый адрес шрифта 8x8 }
  SetIntVec($1F, @Font8x8Proc);     { новый адрес шрифта 8x8  }
End.
```

Этот модуль должен быть оттранслирован на диск. После этих действий можно, указав в своей программе

```
Uses Graph, Bgi, ...;
```

использовать процедуру InitGraph с третьим параметром – пустой строкой, не заботясь о наличии конкретного драйвера или шрифта на диске совместно с EXE-файлом.

Простые анимационные алгоритмы

Программы, которые строят, перемещают, изменяют форму различных изображений на экране в соответствии с заранее разработанным сценарием, называют анимационными. Эти программы работают с двумя классами объектов: спрайтами и средой. Спрайт – это логически законченный движущийся элемент изображения на экране, среда – это фон, по которому происходит движение спрайта.

Самый простой метод анимации заключается в следующем: выводится рисунок любым цветом, через некоторое время тот же рисунок формируется цветом, совпадающим с фоном, что вызывает исчезновение изображения. Затем рисунок выводится в другом месте тем же цветом, что и первая картинка, и т.д.

Пример: составить программу, которая создает изображение машущего руками «человечка».

Сначала построим элементы, которые будут постоянно присутствовать на экране: голова, туловище, ноги; затем в цикле через задержку в 1 с выведем три возможных положения рук. Координаты рук заданы в массиве Ver и Hor.

```
Program DemoAnim;
  Uses Crt, Graph;
  Const
    Ver : Array[1..3] Of Integer = (150,112,76);
    Hor : Array[1..3] Of Integer = (230,200,191);
Var
    Pat : Word;
```

```
  DrVar, MdVar, I : Integer;
Begin
  DrVar := Detect;
  InitGraph(DrVar, MdVar, '');
  While Not KeyPressed Do
    Begin
      SetColor(15);
      Circle(250,100,16);      { голова      }
      Line(250,112,250,170);   { туловище    }
      Line(250,170,238,210);   { левая нога  }
      Line(250,170,270,210);   { правая нога }
    { последовательный вывод трех возможных положений рук }
      For I:=1 To 3 Do
        Begin
          SetColor(15);
          Line(250,112,Hor[i],Ver[i]);
          Line(250,112,250+(250-Hor[i],Ver[i]);
          Delay(1000); {}
    { повторный вывод для "стирания" 1-го положения рук }
          SetColor(0);
          Line(250,112,Hor[i],Ver[i]);
          Line(250,112,250+(250-Hor[i],Ver[i]);
        End
    End
End.
```

Алгоритм другого простого метода может быть следующим:

- вывод изображения;
- стирание экрана с помощью ClearDevice или ClearViewPort;
- вывод нового изображения;
- стирание экрана с помощью ClearDevice или ClearViewPort и т.д.

Для указанных двух методов характерно мелькание экрана при движении спрайта. Избавиться от этого можно, организовав вывод на разные страницы видеопамяти по следующему алгоритму:

- вывести изображение на страницу 0 (она видима по умолчанию);
- сформировать новое изображение на невидимой странице 1;
- сделать видимой страницу 1;
- сформировать новое изображение на невидимой странице 0 и т.д.;

В большинстве случаев вывод спрайта осуществляется с помощью процедуры PutImage, затем осуществляется вывод того же спрайта процедурой PutImage, но с параметром Mode=XoPut, что убирает изображение спрайта, но сохраняет изображение фона. Затем спрайт переносится на новое место и т.д.

Задача 12. Варианты

12.1. Составить программу вывода на экран изображения циферблата механических часов с секундной, минутной и часовой стрелками. Запуск часов осуществляется клавишей «Enter», при этом перемещение секундной стрелки сопровождается «характерным» для часов звуком.

12.2. То же, что в задании 12.1, но предусмотреть режим «будильника».

12.3. То же, что в задании 12.1, но в 3, 6, 9 и 12 часов на экране появляется «кукушка», подает соответствующее число сигналов и исчезает.

12.4. То же, что в задании 12.1, но предусмотреть коррекцию времени путем ускоренного перемещения стрелок при нажатии клавиши «курсор влево» и «курсор вправо».

12.5. Составить программу вывода на экран настольных, электронных часов и изображение метронома. При нажатии клавиши «Enter» стрелка метронома начинает колебательные движения, синхронно с которыми начинают изменяться показания электронных часов.

12.6. Составить программу вывода на экран песочных часов. При нажатии клавиши «Enter» моделируется процесс падения песчинок, уменьшение уровня песка в верхней части колбы и увеличение в нижней части колбы.

12.7. Составить программу вывода на экран спирали. При нажатии клавиши «курсор вправо» начинается вращение спирали относительно центра по часовой стрелке. При нажатии клавиши «курсор влево» – против часовой стрелки.

12.8. Составить программу вывода на экран треугольника. При нажатии клавиши «курсор вправо» треугольник вращается по часовой стрелке вокруг одной из вершин, при нажатии клавиши «курсор влево» – против часовой стрелки.

12.9. Составить программу вывода на экран стилизованной «бабочки». При нажатии клавиши «Enter» она начинает полет, взмахивая крыльями.

12.10. Составить программу вывода на экран стилизованного «человека» в положении готовности начать бег. При нажатии клавиши «Enter» спортсмен начинает бежать, периодически перепрыгивая через барьеры (имитация бега с барьерами).

12.11. Составить программу вывода на экран стилизованного «человека» в положении готовности метнуть копье. При нажатии клавиши «Enter» спортсмен осуществляет разбег и метает копье.

12.12. Составить программу вывода на экран окружности. При нажатии клавиши «курсор вправо» она вращается вокруг своего диаметра слева направо, при нажатии клавиши «курсор влево» – справа налево.

12.13. Составить программу вывода на экран трех горизонтальных беговых дорожек с линиями старта и финиша. На линии старта находятся три участника забега (в виде произвольных фигур). При нажатии клавиши «Enter» участники стартуют с одинаковой начальной скоростью, затем скорость каждого участника в процессе гонки начинает изменяться по случайному закону (т.е. изменяется пройденный путь за дискрет времени). При достижении финиша указать место, занятое каждым участником гонки.

12.14. Составить программу вывода на экран стилизованного «человека» в положении готовности осуществить прыжок в высоту. При нажатии клавиши «Enter» спортсмен осуществляет разбег и перепрыгивает планку.

12.15. Составить программу вывода на экран трех вложенных друг в друга окружностей, представляющих собой беговые дорожки. На линии старта находятся три участника забега (произвольной фигуры). При нажатии клавиши «Enter» участники стартуют с одинаковой угловой скоростью, а после старта угловая скорость каждого участника в процессе гонки изменяется по случайному закону. На финише указать место, занятое каждым участником.

12.16. То же, что в задании 12.15, но беговые дорожки представляют собой три одинаковых, расположенных рядом окружностей.

12.17. Составить программу вывода на экран трех вложенных друг в друга треугольников с пропорционально равными сторонами, которые являются беговыми дорожками. На линии старта находятся три участника забега. При нажатии клавиши «Enter» каждый участник стартует со скоростью, пропорциональной длине беговой дорожки. После старта скорость каждого участника начинает изменяться пропорционально длине пути по случайному закону. На финише указать место, занятое каждым участником.

12.18. Составить программу вывода на экран стилизованного «лыжника». При нажатии клавиши «Enter» он начинает бег на лыжах классическим стилем.

12.19. Составить программу вывода на экран стилизованного «человека» в положении готовности осуществить прыжок в длину. При нажатии клавиши «Enter» спортсмен осуществляет разбег и прыгает.

12.20. Составить программу вывода в верхней части экрана, движущегося с постоянной скоростью слева направо «корабля». Ее значение каждый раз задается генератором случайных чисел. В нижней части экрана расположена «пушка». При нажатии клавиши «Enter» происходит выстрел «торпедой» с постоянной заранее заданной скоростью клавишами от 0 до 9. При попадании смоделировать «взрыв» и исчезновение «корабля». При промахе «корабль» достигает правой границы и начинает движение сначала с новой постоянной скоростью.

12.21. То же, что в задании 12.20, но летящий «самолет» осуществляет бомбометание по неподвижной мишени.

12.22. То же, что в задании 12.20, но предусмотреть поворот ствола «пушки» влево и вправо.

12.23. Составить программу вывода в левой части экрана изображения «пушки». В правой части экрана случайным образом появляется и исчезает «мишень». Нажатием клавиши «Enter» производится выстрел из «пушки». Момент попадания фиксируется в виде «взрыва». Предусмотреть возможность перед выстрелом изменять скорость полета «снаряда» с индикацией на экране ее значения.

12.24. То же, что в задании 12.23, но между «пушкой» и «мишенью» имеется препятствие в виде «стены» с высотой от 0,25 до 0,5 экрана. Высота «стены» каждый раз после попадания в «мишень» изменяется.

12.25. Составить программу вывода на экран летящего слева направо «самолета» с постоянной скоростью. Ее значение может задаваться цифровыми клавишами от 0 до 9. В нижней части экрана среди «гор» расположена посадочная площадка. Используя клавиши «курсор вверх» и «курсор вниз», осуществить успешную посадку «самолета» (вертолета). При аварии моделируется «взрыв». Предусмотреть, что клавиши «курсор вверх», «курсор вниз» соответственно увеличивают и уменьшают вертикальную составляющую скорости.

12.26. Составить программу вывода в верхнюю часть экрана изображения «тучи», в нижнюю «емкость» для воды. При нажатии клавиши «Ввод» начинает идти дождь. Размер тучи уменьшается, а «емкость» заполняется водой.

12.27. То же, что в задании 12.26, но из «тучи» идет снег и внизу растут сугробы.

12.28. Составить программу вывода в верхнюю часть экрана изображения двух «туч». При нажатии клавиши «Enter» «тучи» начинают двигаться навстречу друг другу. В момент их касания начинает идти «дождь» только из перекрывающейся области, которая сначала увеличивается, а затем уменьшается по мере движения «туч».

12.29. Составить программу вывода на экран стилизованного «человека». При нажатии клавиши «Enter» человек начинает идти, размахивая в такт движения руками.

12.30. Составить программу вывода на экран стилизованного изображения «велосипедиста». При нажатии клавиши «Enter» он начинает движение, вращая ногами педали.

ЛИТЕРАТУРА

Абрамов В.Г., Трифонов Н.П., Трифонов Г.П. Введение в язык Паскаль. – М.: Наука, 1988.

Богумирский Б. MS-DOS 6.2 – новые возможности для пользователя. – СПб.: Питер, 1994.

Богумирский Б. Эффективная работа на IBM PC. – СПб : Питер, 1995.

Бородич Ю.С. и др. Паскаль для персональных компьютеров: Справ. пособие.– Мн.: Выш. шк., 1991. – 365 с.

Вальвачев А.Н., Крисевич В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС: Справ. пособие.– Мн.:Выш. шк., 1989.– 223 с.

Епанешников А.М., Епанешников В.А. Программирование в среде TURBO PASCAL 7.0. – М.: Диалог-МИФИ, 1996.

Зуев Е.А. Язык программирования Турбо Паскаль 6.0. – М.: Унитех, 1992.

Офицеров Д.В., Старых В.А. Программирование в интегрированной среде Турбо Паскаль: Справ. пособие. – Мн.: Беларусь,1992.– 240 с.

Офицеров Д.В., Долгий А.Б., Старых В.А. Программирование на персональных ЭВМ : Практикум. – Мн.: Выш.шк., 1993. – 256 с.

Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль. – М. : МАИ, 1993.

Фаронов В.В. Паскаль и Windows. – М.: МВТУ-ФЕСТО Дидактик, 1995.

Фаронов В.В. TURBO PASCAL 7.0. – М.: Нолидж, 1997.

Св.план 2005, поз. 39

Учебное издание

Бурцев Анатолий Александрович,
Шестакович Вячеслав Павлович

ПРОГРАММИРОВАНИЕ

МЕТОДИЧЕСКОЕ ПОСОБИЕ
К ВЫПОЛНЕНИЮ КОНТРОЛЬНЫХ ЗАДАНИЙ
ДЛЯ СТУДЕНТОВ ВСЕХ СПЕЦИАЛЬНОСТЕЙ БГУИР
ЗАОЧНОЙ ФОРМЫ ОБУЧЕНИЯ

В 2-х частях

Часть 2

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ТУРБО ПАСКАЛЬ

Редактор Н.А. Бебель
Корректор Н.В. Гриневич

Подписано в печать 13.04. 2005. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 4,3.
Уч.-изд. л. 3,6. Тираж 350 экз. Заказ 11.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0133108 от 30.04.2004.
220013, Минск, П. Бровки, **6**