Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Е.А. ДЕЙ

ИЗУЧЕНИЕ И ПРИМЕНЕНИЕ ГРАФИЧЕСКИХ СРЕДСТВ ЯЗЫКА ПРОГРАММИРОВАНИЯ ТУРБО-ПАСКАЛЬ

ПРАКТИЧЕСКОЕ ПОСОБИЕ
для студентов
специальности 1-31 04 01 02 – «Физика»

Гомель 2005

УДК 681.3.06(075)
ББК 32.973я7
Д27

Рецензенты:
В.В.Свиридова, доцент, кандидат физико-математических наук; кафедра теоретической физики учреждения образования «Гомельский государственный университет имени Франциска Скорины»

Рекомендовано к изданию научно-методическим советом учреждения образования «Гомельский государственный университет имени Франциска Скорины» 27.10.2004 (№ 134).

Д27 Изучение и применение графических средств языка программирования Турбо-Паскаль: Практическое пособие для студентов вузов специальности «Физика». / Е.А.Дей; Учреждение образования «Гомельский государственный университет имени Франциска Скорины». – Гомель: ГГУ им. Ф.Скорины, 2005. – 85 с.

Практическое пособие содержит изложение основных программных элементов графического режима языка Турбо-Паскаль. По каждому разделу приведены теоретические сведения, примеры применения, Задачи и задачи.

Предназначено студентам физического факультета университета.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Умение разрабатывать собственные программы для решения физических задач с использованием современных средств программирования является необходимым элементом подготовки студента-физика.

Язык программирования Турбо-Паскаль обладает развитой системой графических команд, которые позволяют создавать достаточно сложные и полезные программы.

Данное практическое пособие содержит изложение основных программных элементов графического режима языка Турбо-Паскаль. Отбор учебного материала продиктован основными направлениями использования компьютерной графики в деятельности физика-инженера и физика-педагога: иллюстративная графика, научная графика, печать изображений и файловые форматы графических данных. Излагаемые программные элементы и методы построения изображений поясняются примерами программ и результатами их выполнения.

С учетом того, что практически все современные компьютеры обеспечивают работу в графическом режиме VGA с разрешением 640*480, 16 цветов, в тексте все примеры относятся к этому графическому режиму.

Предполагается, что студентами уже изучены основы программирования на языке Турбо-Паскаль (типы данных, операторы ввода-вывода, операторы выбора, операторы цикла, массивы, подпрограммы).

По каждой теме приводятся теоретические сведения, описания алгоритмов, примеры программ, а также вопросы для самоконтроля и задания для самостоятельного выполнения.

Практическое пособие предназначено для студентов физического факультета при изучении второй части курса «Программирование и математическое моделирование».

ТЕМА 1. ГРАФИЧЕСКИЙ РЕЖИМ И ГРАФИЧЕСКИЕ ПРИМИТИВЫ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ТУРБО-ПАСКАЛЬ

1.1 Установка графического режима в языке Турбо-Паскаль

Дисплей персонального компьютера (ПК) может функционировать в двух совершенно различных и несовместных режимах: текстовом и графическом. При включении компьютер автоматически переводится в текстовый режим, в котором на экране изображаются символы, причем экран состоит из 25 строк по 80 символов в строке.

В графическом режиме экран дисплея состоит из регулярно расположенных светящихся точек, называемых пикселами (сокращение английских слов “picture elements” - pixel). Высвечивая различные точки разным цветом, можно получить на экране нужное изображение. Координаты точек отсчитываются от левого верхнего угла (0,0) вправо по оси OX и вниз по оси OY

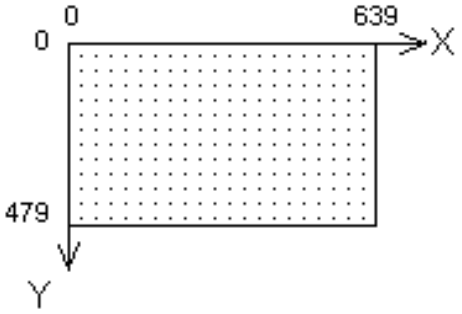


Рис. 1.1 Графические координаты экрана

Координаты X и Y могут принимать только целые значения (по сути, это номера точек экрана) и изменяются в пределах: $0 \le X \le 639$; $0 \le Y \le 479$

Если значения координат X, Y выходят за границы допустимых, то компьютер “рисует” линии как бы за пределами экрана. Естественно, что этого изображения не видно, но графические подпрограммы выполняются.

Для преобразования информации, поступающей от ПК, в видеосигналы, необходимые для работы дисплея, используется специальная электронная схема (графический адаптер, или видеокарта), размещаемая в системном блоке ПК. Типы адаптеров: EGA, VGA, SVGA. Они различаются разрешением, т.е. количеством воспроизводимых по горизонтали и вертикали точек, и числом одновременно воспроизводимых цветов. Адаптеры новейших типов обладают совместимостью с предыдущими типами и могут функционировать в различных режимах. Номер режима указывается при инициализации графики.

Информация о цвете всех точек экрана хранится в специальной области оперативной памяти, называемой видеопамятью. Графический адаптер обеспечивает постоянное сканирование видеопамяти и высвечивание точек на экране заданным цветом.

В языке программирования Турбо-Паскаль взаимодействие с графическим адаптером поддерживает специальная программа (графический драйвер). Загрузочный модуль драйвера для работы с адаптерами EGA и VGA хранится в файле EGAVGA.BGI.

Для формирования графических изображений в языке Турбо-Паскаль используются программные элементы, собранные в модуле GRAPH. В программе модуль подключается стандартным способом с помощью описания Uses Graph. Это позволяет программисту использовать все определенные в модуле константы, типы, переменные, процедуры и функции.

Для установки графического режима служит процедура InitGraph. Она загружает графический драйвер в оперативную память, устанавливает один из возможных для имеющегося адаптера графических режимов и стандартные значения параметров графических процедур . Формат процедуры:

```
InitGraph(var GrDriver, GrMode :integer;
          PathToDriver :string);
```

Назначение параметров:

```
GrDriver - тип графического адаптера
```

```
GrMode - номер графического режима
PathToDriver - полный адрес файла EGAVGA.BGI
```

Первые два параметра являются параметрами-переменными и должны быть описаны в программе как переменные целого типа, причем первому параметру обязательно должно быть присвоено конкретное значение (Таблица 1)

Таблица 1.1 Коды адаптеров и графических режимов

Адаптер	Режим		Разрешение	Цветов	Страниц
EGA (3)	EGALo	(0)	640*200	16	4
	EGAHi	(1)	640*350	16	2
VGA (9)	VGALo	(0)	640*200	16	4
	VGAMed	(1)	640*350	16	2
	VGAHi	(2)	640*480	16	1
Detect(0)			Максимальное разрешение		

Если номер графического режима не задан перед вызовом процедуры InitGraph, то автоматически будет выбран режим с максимально возможным разрешением.

Третий параметр - строкового типа - указывает каталог, где находится файл графического драйвера. Если файл находится в рабочем каталоге, то значение этого параметра - пустая строка ('). Например:

```
GrDriver:=9;  GrMode:=1;
InitGraph(GrDriver,GrMode,'C:\TP70\BGI');
```

Имеется возможность автоматического определения типа адаптера с помощью режима Detect, выбора нужного файла графического драйвера и установки режима максимального разрешения

```
GrDriver:=Detect;
InitGraph(GrDriver,GrMode,'');
```

Для выбора параметров графических процедур используется набор констант, значения которых уже заданы в модуле Graph (предопределенные константы). При использовании таких констант в качестве параметров процедур можно записывать или имя константы, или ее числовое значение. Например, при работе с адаптером VGA режим низкого разрешения можно задать константой VGALo, а можно числом 3, ибо в модуле Graph уже содержится описание: Const VGALo=3. Далее в тексте пособия в таблицах параметров графических процедур указываются имена

констант и (в скобках) их значения, но при вызове процедур следует указать одно из двух.

Работа в режимах с разрешением 640*480 удобна тем, что отношение ширины экрана к его высоте (4:3) равно отношению числа точек по горизонтали и вертикали. В результате на единицу длины вдоль вертикальной и горизонтальной осей экрана приходится равное количество точек.

Кроме того, при установке графического режима могут быть использованы следующие функции и процедуры модуля Graph:

◊ GraphResult:integer; - функция позволяет получить код ошибки для последней графической операции (0 или grOk - нет ошибок).

◊ GraphErrorMsg(ErrorCode:integer):string; - функция формирует строку с текстом сообщения об ошибке, соответствующим коду ErrorCode.

◊ SetBkColor(N:word); - процедура устанавливает цвет фона экрана, соответствующий номеру N.

◊ GetBkColor; - функция, дающая номер цвета фона (в подпрограммах, устанавливающих цвет, цвет фона всегда имеет номер 0);

◊ GetMaxColor - функция, дающая максимально возможный в данном графическом режиме номер цвета;

◊ ClearDevice; - процедура очищает графический экран и закрашивает его в цвет фона.

◊ CloseGraph; - процедура завершает работу в графическом режиме и выполняет переход в текстовый режим.

После построения изображения удобно остановить выполнение программы для его просмотра. Простейший способ сделать паузу – записать в конце программы пустой оператор Readln; - при выполнении которого программа будет ждать нажатия клавиши Enter.

В целом установка графического режима осуществляется стандартной последовательностью действий, например:

```
Program Primer01;
{---Установка графического режима }
Uses Graph,Crt;
Var
  ErrorCode : Integer;
  GrDriver, GrMode : Integer;
```

```
Begin
  GrDriver:=Detect;
  InitGraph(GrDriver,GrMode,'');
  If GraphResult <> grOK then Halt(1);      {**}
  ......................
 {...Выполнение операторов графического режима...}
  ......................
  Readln;         {-пауза до нажатия клавиши Enter}
  CloseGraph;     {-Закрытие графического режима}
End.
```

Более подробное оформление позволяет вывести на экран сообщение при возникновении ошибки. Для этого вместо строки {**} следует набрать в программе

```
If GraphResult <> grOK then begin
  WriteLn(' ### Ошибка установки графического
режима:');
  WriteLn(GraphErrorMsg(ErrorCode));
  WriteLn(' ### Выполнение прекращается...');
  Halt(1);
end;
```

Установленные пределы изменения координат пикселов можно уточнить и использовать в самой программе. Для этого служать функции:

◊ GetMaxX :integer; - выдает максимально возможное значение координаты X для установленного графического режима.

◊ GetMaxY :integer; - выдает максимально возможное значение координаты Y для установленного графического режима.

1.2 Установка свойств линии

Основной метод построения изображения любой сложности – «сборка» его из отдельных простейших элементов - графических примитивов. В любом развитом языке программирования имеется достаточно полный набор процедур, изображающих простейшие элементы: линии, прямоугольники, дуги, окружности, эллипсы и т.д. С помощью параметров, указываемых при вызове процедуры,

можно изобразить графический элемент в нужном месте экрана, требуемого размера и цвета.

Использование большинства графических процедур в языке Турбо-Паскаль состоит из двух этапов: а) настройка параметров изображения; б) построение изображения.

Так, перед построением линейных объектов с помощью специальных процедур нужно выбрать характеристики линии (цвет, толщину, шаблон). Линией с этими свойствами и будут затем рисоваться отрезки, дуги, окружности, и т.п.

Шаблон определяет последовательность высвечивания точек вдоль линии. Задавая чередование ярких и темных точек, можно получить пунктирные и штрихпунктирные линии.

Установка цвета линии. SetColor(N:word); - процедура, которая устанавливает цвет линий, соответствующий номеру N; этим цветом будут изображаться также контуры закрашенных фигур и тексты. Имена констант цвета и соответствующие им значения приведены в таблице:

Цифровое значение	Символическая константа	Изображение или фон	Цвет
0	BLACK	общий	черный
1	BLUE	общий	синий
2	GREEN	общий	зеленый
3	CYAN	общий	бирюзовый
4	RED	общий	красный
5	MAGENTA	общий	пурпурный
6	BROWN	общий	коричневый
7	LIGHTGRAY	общий	светло-серый
8	DARKGRAY	изображение	темно-серый
9	LIGHTBLUE	изображение	голубой
10	LIGHTGREEN	изображение	светло-зеленый
11	LIGHTCYAN	изображение	светло-бирюзовый
12	LIGHTRED	изображение	светло-красный
13	LIGHTMAGENTA	изображение	светло-пурпурный
14	YELLOW	изображение	желтый
15	WHITE	изображение	белый

Все вышеприведенные символические константы определены в модуле Graph. Поэтому при желании установить цвет фона, например, зеленым, можно записать SetBkColor(GREEN) вместо SetBkColor(2).

◊ ClearDevice; - процедура очищает графический экран, закрашивает его цветом фона и устанавливает текущий указатель в точку (0,0).

В графическом режиме, по аналогии с курсором в текстовом режиме, используется понятие текущего графического указателя. Практически это означает, что в памяти хранятся координаты пиксела, с которого будет начинаться выполнение некоторых подпрограмм. С этим понятием связаны, в частности, следующие функции и процедуры (все параметры имеют тип Integer):

◊ GetX; - функция, выдающая x-координату текущего указателя;

◊ GetY - функция, выдающая y-координату текущего указателя;

◊ MoveTo(X,Y); - процедура, которая перемещает текущий указатель в точку с координатами (X,Y); точка на экране не высвечивается.

◊ MoveRel(Dx,Dy); - процедура, которая перемещает текущий указатель на заданное количество точек (Dx,Dy) по отношению к его предыдущему положению (точка на экране не высвечивается).

◊ PutPixel(X,Y,C); - процедура закрашивает пиксел с координатами (X,Y) цветом с номером C. (Номер цвета 0-15, координаты X: 0-639, Y: 0-479).

Установка стиля линии. По умолчанию устанавливается рисование сплошными линиями, что в большинстве случаев является достаточным.

При необходимости можно изменить стиль линии. Для этого служит процедура

◊ SetLineStyle(Style,Pattern,Thickness:word); - процедура, устанавливающая шаблон и толщину линии.

Параметр Style задает тип линии:

0 - сплошной;

1 - пунктирный;

2 - штрихпунктирный;

3 - штриховой;

4 - заданный программистом.

При значении Style = 1, 2 или 3 параметр Pattern не играет роли и может быть любым числом, например, 0.

Параметр Thickness устанавливает толщину линии: 1 или 3 (толщина в 1 или 3 пиксела).

Для задания собственного шаблона пользователя следует задать значение параметраStyle=4 и задать шаблон-комбинацию 16 светящихся (1) или погашенных (0) пикселов (16 бит – для типа word) и присвоить это значение параметру Pattern.

Каждый бит соответствует одному пикселу в шаблоне линии, если пиксел светится, то его бит равен 1, если не светится, то 0.

Сформированный двоичный шаблон нужно перевести в 16-ричную систему счисления и полученное число присвоить параметру Pattern.

Двоичное число разбивается на тетрады справа налево (тетрада содержит 4 бита). Значение каждой тетрады переводится в 16-ричное число по таблице:

Двоичное представление	16-ричное представление
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Например, SetLineStyle(4, $CCCC,1) реализует стиль линии вида

1100 1100 1100 1100 = 52428 = $CCCC
$C $C $C $C - мелкий равномерный пуктир

Пример изображения линий:

```
        ... {инициализация графики}
        X:=GetMaxX;
        SetLineStyle(CenterLn,0,3);
    Line(0,10,X,10);{штрихпунктирная толстая
линия}
    SetLineStyle(0,0,1);{стандартный режим}
    Line(0,20,X,20);{тонкая сплошная линия}
    Pattern1:=$CCCC;{или Pattern1:=52428}
    SetLineStyle(4,Pattern1,1);
    Line(0,30,X,30);{линия,заданная шаблоном}
```

Примечание: Процедура установки стиля SetLineStyle при выводе дуг влияет только на толщину линий.

Работа с отдельными точками. Для определения или изменения цвета отдельного пиксела в языке Турбо-Паскаль имеются следующие процедуры и функции:

◊ PutPixel(X,Y:integer; Color:word); - процедура окрашивает на экране точку с координатами X,Y в цвет с номером Color.

◊ GetPixel(X,Y:integer):word; - функция выдает номер цвета точки с координатами X,Y.

Пример. Множество случайных точек случайного цвета, равномерно распределенных по экрану («Звездное небо»):

```
N:=100
For i:=1 to N do begin
 Cv:=random(16);
 X:=random(640);  y:=random(480);
 PutPixel(x,y,cv);
End;
```

При построении изображений необходимо рассчитывать положение точек изображения относительно центра экрана или его крайних точек. Для этого удобно использовать вспомогательные переменные:

```
Xc:=GetMaxX div 2;      Yc:=GetMaxY div 2;
Xmax:=GetMaxX;          Ymax:=GetmaxY;
```

Рисование линейных элементов изображения. При рисовании линейных графических примитивов используется цвет, установленный процедурой SetColor, и стиль линии, установленный процедурой SetLineStyle.

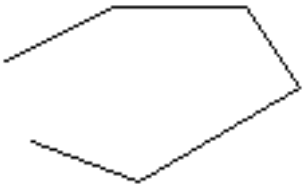
◊ LineTo(X,Y); - процедура, которая проводит отрезок прямой линии из точки, где находится текущий указатель, в точку с координатами (X,Y). Текущий указатель перемещается в точку (X,Y).

◊ LineRel(Dx,Dy); - процедура, которая проводит отрезок прямой из точки, где находится текущий указатель, в точку с приращением координат на Dx (по оси X) и на Dy (по оси Y). Текущий указатель перемещается в конец линии.

◊ Line(X1,Y1,X2,Y2); - процедура, которая проводит отрезок прямой линии установленного цвета и шаблона из точки с координатами (X1, Y1) в точку (X2,Y2). Положение текущего указателя не изменяется.

◊ Rectangle(X1,Y1,X2,Y2); - процедура, которая рисует прямоугольник со сторонами, параллельными осям координат. Координаты (X1,Y1) определяют верхний левый угол, (X2,Y2) - нижний правый угол прямоугольника.

Пример. Для изображения ломаной линии можно воспользоваться серией команд Line. В этом случае началом следующего отрезка будет являться конец предыдущего.



```
Line(20,40,60,20);
Line(60,20,110,20);
Line(110,20,130,50);
Line(130,50,70,85);
Line(70,85,30,70);
```

Более удобным (координаты каждой точки указываются только один раз) для вывода последовательности отрезков оказывается применение процедур, использующих графический указатель:

```
MoveTo(20,40);  LineTo(60,20); LineTo(110,20);
LineTo(130,50); LineTo(70,85); LineTo(30,70);
```

При большом количестве точек их координаты удобно хранить в массивах и в цикле осуществлять обход всех точек.

1.4 Рисование окружностей, дуг и эллипсов

При изображении дуг угловые параметры задаются в градусах и отсчитываются от положительного направления оси OX против часовой стрелки.

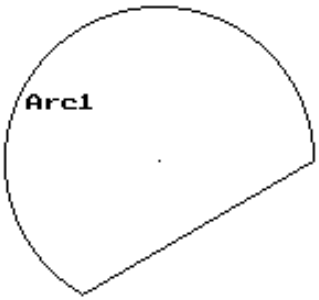
◊ Circle(X,Y:integer; R:word); - процедура, которая рисует окружность с центром в точке (X,Y) и радиусом R.

◊ Arc(X,Y:integer; A1,A2,R:word); - процедура рисует дугу окружности с центром в точке (X,Y) и радиусом R. Параметры A1 и A2 – целые положительные. Посредством параметра A1 задается угол (в градусах, против часовой стрелки) от оси OX до начальной точки дуги, параметра A2 – угол (в градусах) до конечной точки дуги. Если A2 ≥ 360, используется разность A2-360.

◊ Ellipse(X,Y:integer; A1,A2,XR,YR:word); - процедура, которая рисует дугу эллипса с центром (X,Y) и полуосями XR (по оси X), YR (по оси Y) от начального угла A1 до конечного угла A2. Значения A1=0 и A2=360 приведут к вычерчиванию полного эллипса.

◊ Procedure GetArcCoords(Var ArcCoords : ArcCoordsType); - процедура, возвращает координаты последней команды Arc в переменной типа ArcCoordsType: координаты центра (X, Y), координаты начальной точки (XStart, YStart) и координаты конечной точки (XEnd, YEnd), полученные при последнем обращении к процедурам Ellipse и Arc.

Эти значения могут быть полезны, если нужно соединить линию с одной из крайних точек дуги окружности или эллипса.

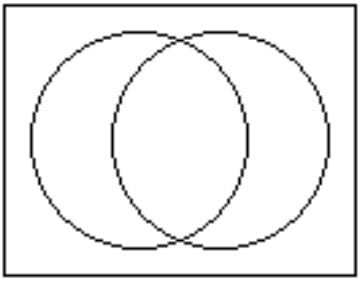



```
Arc(80,80,0,240,70);
 GetArcCoords(ArcCoords);
 With ArcCoords Do
 Line(Xstart,Ystart,Xend,Yend);
OutTextXY(20,50,'Arc1');
```


```
Arc(80,40,240,0,90);
 GetArcCoords(ArcCoords);
 With ArcCoords Do begin
   Line(Xstart,Ystart,80,40);
   Line(80,40, Xend, Yend);
 end;
OutTextXY(110,130,'Arc2');
```

```
Program Primer02;
{ применение графических функций и процедур }
Uses Graph,Crt;
Var
  ErrorCode : Integer;
  Gd, Gm : Integer;
Begin
 Gd := Detect;     {установка графического режима}
 InitGraph(Gd,Gm,'');
 if GraphResult <> grOK then Halt(1);
 SetBkColor(1);              {установка цвета фона}
 ClearDevice;                {очистка экрана}
 SetColor(11);              {установка цвета линий}
 Xc:=GetmaxX div 2;     {координаты центра экрана}
 Yc:=GetMaxY div 2;
 Circle(Xc,Yc,45);{окружность радиуса 45 в центре}
 MoveTo(400,50);  {перемещение текущего указателя}
 SetColor(1);                {цвет линий - синий}
 LineTo(300,220);{отрезок прямой в заданную точку}
 PutPixel(40,40,15);     {желтая точка x=40, y=40}
 Rectangle(100,100,300,180);{прямоугольник}
 SetLineStyle(3,0,3);     {тип линии - штриховая
                            утолщенная}
 SetColor(14);               {цвет линий - желтый}
 Line(0,0,GetMaxX,GetMaxY); {линия по диагонали
                             экрана}
  Readln;          {пауза до нажатия клавиши Enter}
  CloseGraph;       {закрытие графического режима}
End.
```

Пример. Нарисовать прямоугольник и две находящиеся внутри окружности.



```
SetColor(13);
Rectangle(20,20,150,120);
SetColor(14);
Circle(70,70,40);
Circle(100,70,40);
```

При необходимости изображения системы однотипных элементов, различающихся некоторым параметром, в программе организуется цикл, в котором параметру элемента присваивается значение, и элемент изображается на экране.

Пример. Система окружностей различного цвета с увеличивающимся радиусом в центре экрана.

```
For i:=1 to 10 do begin
  cv:=i+5;
  r:=40+i*20
  circle(0,0,r);
end;
```

Вопросы для самопроверки

1. Каково назначение графического драйвера и графического адаптера.
2. Как производится установка графического режима.
3. Каковы параметры экрана в различных графических режимах.
4. Понятие графического указателя.
5. Процедуры работы с отдельными точками.
6. Процедуры выбора свойств линии.
7. Процедуры изображения отрезков и прямоугольников.
8. Процедуры изображения окружностей, дуг, эллипсов.

Задачи

1. Составить программу, изображающую на экране прямоугольник с вершинами (80,80), (370,80), (370,250), (80,250), и окружности радиуса 65 в его вершинах.

2. Составить программу, изображающую на экране пятиугольник с вершинами (100,100), (350,100), (470,220), (380,240), (150,140) и все его диагонали.

3. Составить программу, изображающую на экране окружность радиуса 60, квадрат со стороной 90 и эллипс с полуосями 130,80 с центром в точке (300,180).

4. Составить программу, изображающую на экране четыре равных сектора окружности радиуса 90 в центре экрана с углом 60 градусов.

5. Создайте свой модуль и включите в его состав свою процедуру, реализующую установку графического режима.

ТЕМА 2. ВЫВОД ТЕКСТА В ГРАФИЧЕСКОМ РЕЖИМЕ

2.1 Выбор шрифта и параметров вывода текста

2.2 Вывод текста в графическом режиме

2.3. Изображение числовой информации в графическом режиме

2.1 Выбор шрифта и параметров вывода текста

Нередко требуется сопровождать графическое изображение буквенными (символьными) обозначениями и некоторым пояснительным текстом. При этом требуется правильно расположить символы по отношению к рисунку.

Для вывода текстовой информации в графическом режиме используются специально разработанные шрифты, хранящиеся в файлах с расширением *.CHR. Последовательность работы с текстом в графическом режиме соответствует общему принципу: вначале устанавливаются нужные свойства, а только затем строится изображение.

◊ SetTextStyle(Font,Direction,Size); - процедура, которая устанавливает тип шрифта, ориентацию текста и размер символов. Все параметры процедуры целочисленные (типа word).

Параметр Direction задает направление вывода текста:

```
HorizDir   (0)     - по горизонтали;
VertDir    (1)     - по вертикали.
```

Параметр Size задает коэффициент увеличения размеров букв при выводе на экран. Допустимые значения 1..10.

Цвет символов текста заранее устанавливается процедурой SetColor.

Параметр Font позволяет выбрать тип шрифта. В качестве значения параметра следует указать одну из констант модуля Graph:

```
    DefaultFont     (0)     - 8*8 битовый шрифт –
стандартный матричный шрифт (растровый),
используется по умолчанию.
    TriplexFont     (1)     - штриховой шрифт
(TRIP.CHR)
    SmallFont       (2)     - уменьшенный шрифт
(LITT.CHR)
    SansSerifFont   (3)     - упрощенный шрифт
(SANS.CHR)
    GothicFont      (4)     - готический (GOTH.CHR)
```

Матричный (растровый) шрифт представляет собой матрицу пикселов.

Векторные шрифты рисуют каждый символ большим количеством линий, причем при изменении размера шрифта линии масштабируются. В ТП начиная с версии 7.0 имеются шрифты с кодами 5-10.

Если строка представленная векторным шрифтом слишком длинная, то отображается только та ее часть, которая помещается на экране. Если не помещается на экране матричная строка, то она полностью не отображается.

При использовании шрифтов 1-4 соответствующий файл должен быть скопирован в рабочий каталог.

Для выравнивания текста при выводе предназначена процедура

◊ SetTextJustify(Horiz,Vert:word); - процедура устанавливает способ выравнивания текста относительно текущего положения графического указателя.

Значения параметра Horiz: 0-слева; 1-по центру; 2-справа.

Параметр Vert: 0-над строкой; 1-по центру; 2-под строкой.

Процедура SetUserCharSize(x1,x2,y1,y2); – изменяет ширину и высоту символов. Отношение x1/x2 задает коэффициент изменения ширины, y1/y2 – задает коэффициент изменения

высоты символов. Например, чтобы увеличить высоту и ширину, нужно записать:

```
SetUserCharSize(2,1,2,1);
```

Для правильного размещения строки символов на экране необходимо знать количество точек экрана, занимаемых строкой. Для получения высоты и ширины строки в пикселах при заданных размерах шрифта и ориентации строки используются функции:

- ◊ `TextHeight(Text:string) :word`
- ◊ `TextWidth(Text:string) :word.`

Эти функции учитывают текущий размер шрифта и коэффициенты изменения высоты и ширины символов.

Например, для вывода строки текста в центре экрана можно использовать команды

```
xt:=(GetMaxX-TextWidth(s)) div 2;
yt:=(GetMaxY-TextHeight (s)) div 2;
```

2.2 Вывод текста в графическом режиме

Для вывода текста в графическом режиме используются следующие процедуры:

◊ OutText(Text: string); - процедура, которая изображает на экране строку символов Text, начиная с текущей позиции.

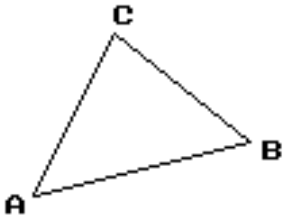
◊ OutTextXY(X,Y:integer; Text:string); - процедура, которая изображает на экране строку Text, начиная с координат (X,Y). Положение текущего указателя не изменяется.

Пример. Вывести на экран текст с применением различных стилей:

```
Program Primer3;
{ Вывод текста в графическом режиме }
Uses Graph,Crt;
var
 d,r,e,k:integer;
begin
 d:=Detect; InitGraph(d,r,'');
 If GraphResult<>grOk then Halt(1);
  Setcolor(14);  SetBkcolor(3);
  For k:=1 to 10 do begin
   Settextstyle(0,0,k);
   OuttextXY(200,20+40*k,'ABCDEFG abcdefg');
```

```
   Settextstyle(0,1,k);
   OuttextXY(10+40*k,400,'АБВГДЕЖ абвгдеж');
 Readln;
 Closegraph;
End.
```

Пример. Нарисовать треугольник АВС с вершинами в точках А(20,80), В(100,70), С(50,20). Вершины обозначить соответствующими буквами.




```
Line(20,80,100,60);
 Line(100,60,50,20);
 Line(50,20,20,80);
 SetTextStyle(DefaultFont,
              HorizDir, 1);
 OutTextXY(10,80,'A');
 OutTextXY(105,60,'B');
 OutTextXY(50,10,'C');
```

2.3. Изображение числовой информации в графическом режиме

Для вывода численных значений в графическом режиме сначала необходимо преобразовать число в строку с помощью процедуры Str, а затем посредством операции конкатенации "+" подключить к выводимой строке.

```
      X:=15.2;
      Str(x,S);
      OutText(S);     {на экране
1.52000000000+E0001}
      Str(x:6:2,S);
      OutText(S);               {на экране  15.20}
```

Пример. Фрагмент программы

```
Var x:real;  xStr:string;
.....
x:=x0+v0*t+a*t*t/2;
Str(x:6:2,xStr);
OutText('Пройденный путь = '+xStr+'метров');
.....
```

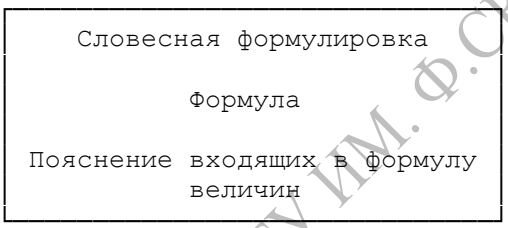
изобразит на экране, начиная с текущей позиции указателя: [Пройденный путь = 243.87 метров].

Вопросы для самопроверки

1. Как осуществляется установка параметров для вывода текста.
2. В чем разница между процедурами OutText и OutTextXY ?
3. Какие особенности вывода числовой информации в графическом режиме ?

Задачи

1. Составьте программу, изображающую на экране плакат, поясняющий содержание физического закона в форме:



- Второй закон Ньютона.
- Закон Эйнштейна для фотоэффекта.
- Формула плоской линзы.
- Закон Ома для участка цепи.
- Закон всемирного тяготения.
- Закон Ома для полной цепи.
- Закон Джоуля-Ленца.
- Закон Бойля-Мариотта.
- Первый закон термодинамики.
- Закон Архимеда.

ТЕМА 3. ЗАКРАШИВАНИЕ ЗАМКНУТЫХ ОБЛАСТЕЙ

3.1 Выбор шаблона закрашивания
3.2 Создание собственного шаблона заполнения
3.3 Изображение стандартных закрашенных фигур

3.1 Выбор шаблона закрашивания

В модуле GRAPH предусмотрены процедуры, с помощью которых можно закрасить ("заполнить", "залить") определенным цветом и орнаментом любую замкнутую (ограниченную замкнутой линией заданного цвета) область изображения на экране.

Предварительно необходимо установить параметры закрашивания: цвет и орнамент (шаблон). Шаблон закрашивания определяет чередование высвечиваемых точек на участке размером 8*8, которое повторяется на всей площади закрашивания. Выбор цвета и шаблона заполнения производится процедурой:

◊ SetFillStyle(Pattern:word; Color:word), где значение параметра Pattern определяет номер стандартного орнамента, а значение параметра Color - номер цвет заполнения.

Возможные значения параметра Patten задаются константами модуля Graph:

```
EmptyFill   (0)   - сплошное заполнение цветом фона
SolidFill   (1)   - сплошное заполнение текущим
цветом,
                    установленным процедурой SetColor
LineFill        (2)   - заполнение типа -----------
LtSlashFill     (3)   - заполнение типа ///////////
SlashFill       (4)   - заполнение ///// утолщен.
линиями
BkSlashFill     (5)   - заполнение \\\\\ утолщен.
линиями
LtBkSlashFill   (6)   - заполнение типа \\\\\\\\\\\
HatchFill       (7)   - заполнение клеткой
XHatchFill      (8)   - заполнение косой клеткой
InterLeaveFill  (9)   - заполнение частой сеткой
WideDotFill     (10)  - заполнение редкими точками
```

```
CloseDotFill    (11)  - заполнение частыми точками
UserFill        (12)  - заполнение, заданное
                        программистом
```

3.2 Создание собственного шаблона заполнения

Для закрашивания области собственным орнаментом необходимо установить его с помощью процедуры

◊ SetFillPattern(MyPattern:FillPaternType; Color:word);

Тип FillPaternType уже описан в модуле Graph как array [1..8] of byte. Каждый элемент этого массива описывает строку матрицы заполнения.

Матрица заполнения представляет собой участок 8*8 пикселов.

В программе необходимо определить типизированную константу этого типа, в которой будет закодировано состояние точек для области экрана 8*8 пиксел.

Каждая строка из 8 пиксел кодируется двоичным числом (или его десятичным или 16-ричным эквивалентом), в котором состояние пиксела описывается одним битом. Если бит содержит 1, то соответствующий пиксел закрашивается в цвет заполнения, если 0 - то пиксел закрашивается в цвет фона.

Пример. Шаблон заполнения "знак суммы":

1		◊	◊	◊	◊	◊	◊	01111110 =126
2			◊					00100000 = 32
3				◊				00010000 = 16
4					◊			00001000 = 8
5				◊				00010000 = 16
6			◊					00100000 = 32
7		◊	◊	◊	◊	◊	◊	01111110 =126
								00000000 = 0

Для использования этого шаблона в программе должна быть определена типизированная константа:

```
Const MyPattern:FillPatternType = (126,32,16,8,16,32,126);
```

При вызове процедуры SetFillPattern необходимо указать эту константу в качестве шаблона

```
SetFillStyle(12,15);
SetFillPattern(VyPattern,4);
```

3.3 Изображение стандартных закрашенных фигур

В Turbo-Pascale имеется ряд процедур, рисующих закрашенные фигуры. Орнамент и цвет заполнения задаются процедурой SetFillStyle или SetFillPattern. Текущий цвет контура фигуры задается процедурой SetColor, текущие параметры линии - процедурой SetLineStyle.

◊ Bar(X1,Y1,X2,Y2:integer); - процедура, которая рисует прямоугольник с левым верхним углом (X1,Y1) и правым нижним углом (X2,Y2), закрашенный с использованием текущего шаблона и цвета.

◊ Bar3D(x1,y1,x2,y2:integer; Depth:word; Top:Boolean); - процедура изображает закрашенный параллелепипед, лицевая грань которого строится по координатам (x1,y1)-(x2,y2). Параметром Depth задается ширина боковой грани в пикселах (отсчитывается по горизонтали). Параметр Top служит для указания режима изображения верхней грани: если его значение TopOn (True) - то верхняя грань закрашивается, если TopOff (False) - не изображается.

◊ Sector(x,y:integer; A1,A2,XRad,YRad:word) - процедура строит закрашенный сектор эллипса с центром в точке (x,y), начальным углом A1, конечным углом A2, горизонтальной и вертикальной полуосями XRad, YRad.

◊ PieSlice(x,y:integer; A1,A2,R:word) - процедура строит закрашенный сектор круга с центром в точке (x,y), начальным углом A1, конечным углом A2, радиусом R.

◊ FillEllipse(x,y:integer;XR,YR:word) - процедура строит закрашенный эллипс с центром в точке (x,y), горизонтальной и вертикальной полуосями XR, YR.

Закрашивание сложных замкнутых фигур, в том числе неправильной формы, ограниченных сплошной линией цвета GrColor производится, начиная с заданной внутренней точки (x,y) процедурой

◊ FloodFill(x,y:integer; GrColor:word);

Начиная с указанной точки, процедура закрашивает соседние пикселы установленным шаблоном, затем перебирает пикселы, соседние с ними, и так далее, пока не дойдет до граничных пикселов, имеющих цвет, указанный в параметре GrColor.

Если область не имеет замкнутой границы, этот процесс приведет к заливке всего экрана.

Вопросы для самопроверки

1. Сколько вариантов стандартных шаблонов заполнения имеется в языке Турбо-Паскаль.
2. Какова последовательность действия при создании своего шаблона заполнения.
3. Какой константой будет описываться шаблон заполнения, в котором высвечены все точки, соседние с граничными.
4. Перечислите процедуры изображения стандартных закрашенных фигур.

Задачи

Составить процедуру построения в центре экрана заданного изображения. Численные значения параметров задачи вводить с клавиатуры. В процедуре предусмотреть контроль за размещением изображения в области экрана. В верхней части экрана вывести текст условия варианта. Закрасить различные элементы изображения разным цветом, используя свой шаблон заполнения, являющийся изображением буквы латинского алфавита (номер буквы равен номеру варианта) в блоке 8*8 точек.

1. Квадрат со стороной h и касающиеся каждой его стороны 4 окружности радиуса R.
2. Прямоугольник со сторонами a,b и 4 окружности радиуса R, центры которых находятся в вершинах прямоугольника.
3. Куб со стороной A в изометрической проекции.
4. Две замкнутые полуокружности радиуса R, одна направлена дугой вверх, другая - дугой вниз, касающиеся в точке (x0,y0).

ТЕМА 4. РАЗРАБОТКА СОБСТВЕННЫХ ГРАФИЧЕСКИХ ЭЛЕМЕНТОВ

4.1 Экранные и объектные координаты
4.2 Вычисление координат точек на экране
4.3 Ввод параметров изображения в режиме диалога

4.1 Экранные и объектные координаты

Набор стандартных средств языка Турбо-Паскаль не является полным, так как многие элементы изображений можно задать различными способами. Например, прямоугольник определяется:

- указанием координат левой верхней и правой нижней вершин (как в процедуре Rectangle);
- указанием координат левой верхней вершины, высоты и ширины;
- указанием координат центра, высоты и ширины;
- указанием координат левой нижней вершины, высоты и ширины;
- указанием расстояний сторон от границ экрана, и так далее.

Возможность создавать собственные процедуры позволяет расширить возможности языка программирования и создать новые процедуры, выводящие на экран нужные в данной задаче элементы изображений.

Роль процедур велика еще и потому, что многократный вызов процедуры позволяет вывести на экран нужное количество таких изображений. Отдельные изображения при этом могут отличаться друг от друга теми характеристиками, которые определены в графической процедуре как параметры.

Как обычно, при программировании подпрограмм, необходимо для себя ответить на три вопроса: а) что должно быть известно для выполнения нужного действия (входные параметры); б) что должно получиться (в данном случае – изображение); в) какие действия нужно выполнить для получения результата (текст подпрограммы).

Конфигурация объекта определяется расположением его точек относительно друг друга. При расположении объекта в разных местах экрана относительное расположение его точек не

изменится. Поэтому удобно проектировать изображение в так называемых объектных координатах, начало которых связано с выбранной, так называемой базовой, точкой объекта. Для окружности это может быть ее центр, для прямоугольника - левая верхняя точка, и т.д. Координаты всех узловых точек изображения в этом случае выражаются относительно базовой точки (через приращения). При выводе изображения на экран достаточно указать координаты базовой точки, тогда координаты всех остальных точек автоматически пересчитываются. Изменяя координаты базовой точки, получим изображение одного и того же рисунка в разных местах экрана.

Такой подход удобен при разработке собственной процедуры построения объекта, параметрами которой и будут являться координаты базовой точки и линейные размеры. Перед составление процедуры нужно спланировать изображение на бумаге и выписать математические соотношения, определяющие координаты узловых точек.

Пример. Составить процедуру, выводящую на экран прямоугольник и его диагонали, если заданы координаты левой верхней вершины, высота, ширина и цвет линий.

```
Procedure RDiag(x1,y1,h,L,cv:integer);
Var x2,y2:integer;
Begin
  SetColor(cv);
  x2:=x1+L;    y2:=y1+h;
  Rectangle(x1,y1,x2,y2);
  Line(x1,y1,x2,y2);  Line(x1,y2,x2,y1);
End;
```

Вызывая эту процедуру в тексте программы несколько раз, получим на экране:

```
RDiag(10,100,240,80,11);
RDiag(400,300,90,180,12);
RDiag(310,360,200,120,14);
```

4.2 Вычисление координат точек на экране

При построении параметрических изображений узловые точки довольно часто определяются посредством расчетных формул. Например, требуется изобразить окружность с центром в

точке, являющейся серединой отрезка (x1,y1)-(x2,y2), причем значения координат вводятся с клавиатуры. Результат:

```
xr=(x2+x1)/2;    yr=(y2+y1)/2;
```

не является корректным, так как координаты центра окружности должны иметь целый тип. В таких случаях необходимо после выполнения вычислений на множестве вещественных чисел преобразовать тип результата к целому с помощью функции ROUND(X). В программе использованному примеру должны соответствовать операторы:

```
xr:=Round((x2+x1)/2);
yr:=Round((y2+y1)/2);
```

При построении правильного многоугольника его вершины рассматриваются как точки, полученные при делении окружности на N равных дуг. Угол между соседними точками w=2*Pi/N. Координаты таких точек несложно вычислить по формулам:

```
xi=x0+R*cos(w*i);    yi=y0+R*sin(w*i);
```

где в качестве базовой точки (x0,y0) использован центр многоугольника. Сам многоугольник получается соединением поочередно точек отрезками прямых. Соответствующая процедура может быть записана в таком виде:

```
Procedure N_ugolnik(x0,y0,R,n,cv:integer);
{построение правильного N-угольника}
var i,x,y:integer; fi:real;
begin
 SetColor(cv);
 fi:=2*Pi/n;
 MoveTo(x0+R,y0);
 For i:=1 to n do begin
  x:=x0+Round(R*cos(fi*i));
  y:=y0+Round(R*sin(fi*i));
  LineTo(x,y);
 end;
end;
```

Вызывая процедуру с различными значениями параметров, можно получить изображение треугольника, квадрата, шестиугольника и т.д.

```
N_ugolnik(100,200,50,3,10);
N_ugolnik(200,200,70,4,11);
N_ugolnik(300,200,80,5,12);
N_ugolnik(400,200,90,6,13);
```

```
N_ugolnik(500,200,90,7,14);
```

Легко заметить, что при больших N многоугольник выглядит практически как окружность.

Можно предложить несколько модификаций этой процедуры. Так, если при вычислении x- и y-координат точек использовать разные значения радиуса R1, R2, то получим вытянутый многоугольник (вписанный в эллипс). Если в процедуре выполнять цикл по i от 1 до 10*N и при этом изменять координаты центра x0, y0, то для больших значений N получим изображение спирали.

4.3 Ввод параметров изображения в режиме диалога

При использовании подпрограмм графический элемент может быть построен при любых допустимых значениях параметров.

В этом случае выбор параметров можно осуществить в диалоге с программой по желанию пользователя. Параметры вводятся с клавиатуры в текстовом режиме, а затем в графическом режиме строится изображение.

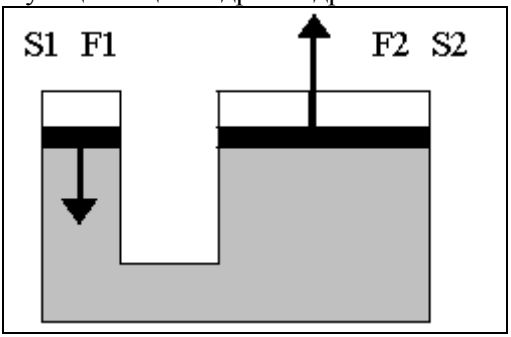
Вопросы для самопроверки

1. По каким формулам вычисляются координаты точек при равномерном разбиении окружности.
2. Что нужно изменить в процедуре N_ugolnik, чтобы рисунок всегда начинался с самой верхней вершины ?
2. Перечислите различные варианты определения окружности.
3. Какие координаты имеют точки, расположенные на верхней и нижней границах экрана и делящие эти границы на N равных отрезков.

Задачи

1. Составить процедуру, изображающую квадрат по заданным координатам его центра и длине стороны.
2. Составить процедуру, изображающую треугольник по заданным координатам его вершин.
3. Составить процедуру, изображающую на экране вектор. Параметры процедуры: координаты исходной и конечной точек x1, y1, x2, y2, цвет вектора.

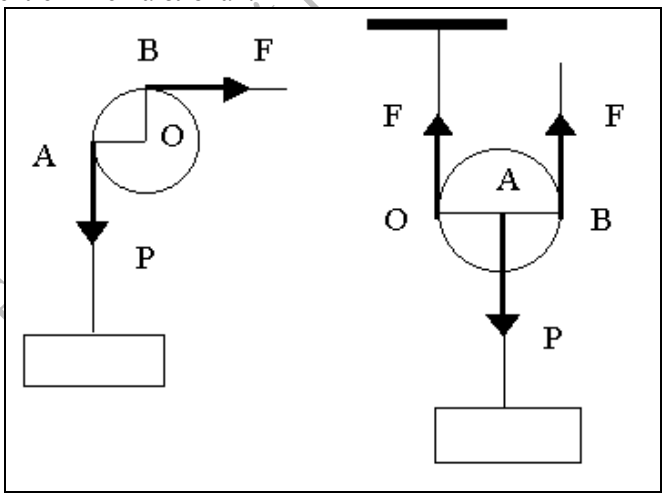
4. Используя процедуру построения вектора, составить программу, изображающую на экране соотношение сил, действующих в цилиндрах гидравлического пресса.



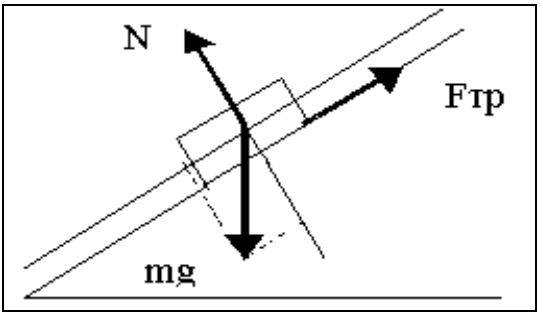
5. Составить процедуру, изображающую на экране вектор с заданными координатами исходной точки x1, y1, заданным углом по отношению к оси OX и заданным цветом.

6. Составить процедуру, изображающую на экране вектор с заданными координатами исходной точки x1, y1, заданными проекциями на оси OX и OY и заданным цветом.

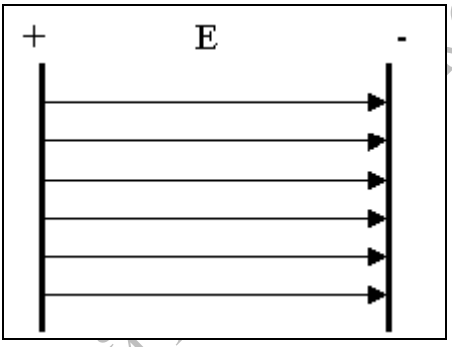
7. Используя процедуру построения вектора, составить программу, изображающую на экране силы, действующие на тела, закрепленные на блоках.



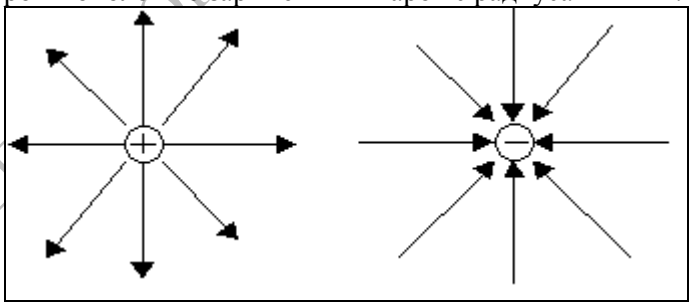
8. Составить программу, изображающую на экране силы, действующие на тело, движущееся по наклонной плоскости.



9. Используя процедуру построения вектора, составьте программу, изображающую на экране силовые линии однородного электрического поля.



10. Используя процедуру построения вектора, составить программу, изображающую на экране силовые линии системы двух противоположно заряженных шаров с радиусами R и r.



11. Составить процедуру, изображающую правильный N-угольник по заданным координатам его центра и радиуса описанной окружности.

12. Составить процедуру, изображающую на экране определенный интеграл. Параметры процедуры: координаты левого нижнего угла используемой прямоугольной области,

нижний предел интегрирования (символ), верхний предел интегрирования (символ), подинтегральное выражение (строка).

13. Составить процедуру, изображающую на экране квадратный корень из заданного числа. Параметры процедуры: координаты левого нижнего угла используемой прямоугольной области, подкоренное число.

ТЕМА 5. ПРОГРАММИРОВАНИЕ ДВИЖУЩИХСЯ ИЗОБРАЖЕНИЙ

5.1 Организация движения объекта с помощью переключения цвета

5.2 Метод копирования изображения в оперативную память и вывода из памяти на экран

5.3 Метод переключения графических страниц

5.1 Организация движения объекта с помощью переключения цвета

Программирование движущихся изображений, помимо развлекательного, игрового, имеет и большое практическое применение. При разработке моделирующих программ важно использовать динамическую графику, то есть, строить на экране монитора постоянно изменяющееся изображение. При этом графические построения должны адекватно отражать соответствующие модели и процессы. Примерами являются перемещение указателей по диаграмме или графику, а также последовательное перемещение элементов устройства для объяснения подробностей его работы, или движение элементов физической системы при иллюстрации физического процесса.

Движущееся изображение программируется как последовательность кадров, в каждом из которых объект имеет новые координаты или конфигурацию, причем изменения в соседних кадрых не должны быть значительными. Каждый кадр выводится на экран, после некоторой задержки стирается, затем

выводится следующий. В целом это циклический процесс, причем параметр цикла определяет конфигурацию текущего кадра.

Алгоритм построения движущегося изображения:

```
Вычислить начальные координаты, нарисовать объект
Повторять
  вычислить новые координаты объекта
  выполнить задержку
  стереть объект на старом месте
  нарисовать объект на новом месте
До получения конечных координат объекта
```

В методе переключения цвета стирание линейного объекта получается в результате повторного рисования его на исходном месте цветом фона.

Для того, чтобы создать впечатление движущегося по экрану линейного объекта, следует каждый раз после построения изображения стирать его с экрана (на практике это означает рисование его цветом фона), а затем снова выводить на экран в новом месте заданным цветом.

Перечисленные действия нетрудно организовать в одном цикле с использованием рассмотренных ранее операторов графических примитивов.

```
Program Prim1;
{расходящиеся круговые волны}
Uses graph,crt;
Var j,i,Gm,Gd,n:integer;
Begin
   writeln('Введите количество повторений ');
   readln(n);
   Gd:=detect;
   InitGraph(Gd,Gm,'');
   OuttextXY(150,300,'Перед вами расходящиеся
круговые волны');
   for j:=1 to n do begin
      For i:=1 to 50 do begin
        SetColor(7);  Circle(310,170,I);
        delay(20);
        SetColor(0);  Circle(310,170,I);
      end;
      delay(1000);
```

```
   end;
Closegraph;
End.

Program Prim2;
{движущаяся строка}
 Uses  graph,crt;
 Var r:string;x,y,i,Gm,Gd:integer;
 Begin
   writeln('Введите нужную строку');
   readln(R);
   Gd:=detect;
   InitGraph(Gd,Gm,'');
   ClearDevice;
   For i:=1 to 600 do begin
      SetColor(3);   OutTextXY (I,140,r);
      DELAY(30);
      SetColor(0);   OutTextXY (I,140,r);
   end;
  CloseGraph;
  End.
```

5.2 Метод копирования изображения в оперативную память и вывода из памяти на экран

Для объектов, состоящих из большого числа элементов различной окраски и штриховки, простая организация движения не всегда обеспечивает приемлемый эффект, так как на прорисовку элементов на каждом шаге затрачивается много времени. Если кадры сменяют друг друга недостаточно быстро, изображение становится пульсирующим, или же движение выглядит как появление объекта по частям. Эффект движения утрачивается.

В таких случаях используется другой подход для организации движения. Объект рисуется один раз, затем состояние точек экрана записывается в оперативной памяти. Изображение может быть выведено в любом месте экрана многократно, без повторного рисования элементов. Перечислим необходимые для реализации этого подхода программные элементы в порядке их применения:

◊ ImageSize(X1,Y1,X2,Y2:integer):word; - функция вычисляет объем памяти (байт), требуемый для запоминания фрагмента изображения прямоугольного участка экрана (X1, Y1)-(X2, Y2).

◊ GetMem(var Pt:pointer; Size:word); - процедура выделяет в динамической памяти непрерывный участок для хранения изображения объемом Size байт. Адрес участка в памяти хранится в переменной Pt типа "указатель". (Процедура GetMem входит в состав модуля System).

◊ GetImage(X1,Y1,X2,Y2:integer; var Pt:pointer); - процедура, которая записывает цвет всех точек заданной прямоугольной области экрана с координатами (X1,Y1)-(X2,Y2) как числовой массив в области памяти, на которую указывает указатель Pt;

◊ PutImage(X,Y:integer; var Pt:pointer; Mode:word); - процедура, которая выводит на экран изображение из области памяти, связанной с указателем Pt, в произвольном месте экрана (верхний левый угол изображения располагается в точке (X,Y)). Mode - режим наложения на существующее в этом месте экрана изображение, задаваемый константами

```
    CopyPut    (0)    - прямой вывод изображения;
    XORPut     (1)    - исключающее наложение
изображения;
    NotPut     (4)    - инвертирование.
```

Для организации на экране движения объекта, данные об изображении которого хранятся в динамической памяти, необходимо выполнить следующие действия:

- построить изображение объекта в произвольном месте экрана;
- определить объем памяти, необходимый для запоминания состояния точек экрана функцией ImageSize;
- создать в памяти динамическую переменную процедурой GetMem. В списке переменных она должна быть описана как переменная типа Pointer.
- записать состояние точек экрана выбранной прямоугольной области в оперативную память процедурой GetImage;
- Повторять:
  - воспроизвести изображение объекта на экране на новом месте процедурой PutImage с режимом XORPut;
  - вычислить координаты нового положения объекта на экране;
  - стереть с помощью повторного вызова процедуры PutImage с прежними параметрами текущее изображение объекта;

до выполнения условия окончания движения.

```
Program prim5;
{копирование изображения в память и из памяти}
uses Graph;
var
 Gd, Gm, i: Integer;
 P: Pointer;
 Size: Word;
begin
 Gd := Detect;
 InitGraph(Gd, Gm, ' ');
 if GraphResult <> grOk then Halt(1);
 For i:=1 to 10 do begin
   SetColor(i+4);
   Circle(200,200,i*10);
 end;
 Size := ImageSize(150, 150, 250, 250);
 GetMem(P, Size);
 GetImage(150, 150, 250, 250, P^);
 Readln;
 ClearDevice;
   {вывод изображения в различных местах экрана:}
   PutImage(50, 50, P^, NormalPut);   Readln;
   PutImage(450, 50, P^, NormalPut);  Readln;
   PutImage(350, 250, P^, NormalPut); Readln;
 For i:=1 to 50 do begin         {-движение
изображения}
   PutImage(50+i, 50, P^, 0);
   Delay(100);
   PutImage(50+i, 50, P^, 1);
 End;
 Readln;  CloseGraph;
end.
```


5.3 Метод переключения графических страниц

В зависимости от выбранного графического режима видеопамять разделяется на несколько (от 1 до 4) областей ("страниц"). В каждый момент времени лишь одна видеостраница является видимой, передающей изображение на экран дисплея, а формировать изображение можно на любой странице (активная страница). Видимая и активная страницы могут совпадать и не совпадать, так что можно, например, сначала подготовить

изображение на какой-либо странице, а затем вывести его на экран.

Еще один прием организации движущихся изображений основан на использовании двух страниц видеопамяти, которые поочередно становятся то видимой (на экране - исходное положение объекта), то активной (на ней рисуется новое положение объекта, которое затем станет видимым). Так как положение объекта меняется на шаг h при переходе от одной страницы к другой, в пределах одной страницы объект смещается на 2*h.

Для управления видеостраницами ТП содержит процедуры:

SetActivePage(n); - сделать активной видеостраницу с номером n. При этом весь вывод будет осуществляться на эту страницу видеопамяти;

SetVisualPage(n); - сделать видимой видеостраницу с номером n.

Чтобы процесс построения рисунка не был виден, можно делать графические построения на невидимой странице, а потом ее отображать уже с рисунком.

```
Program Prim4_3;
Uses Crt, Graph;
Var gd,gm,ee,i,x,y:integer;
u:real;
Begin
. . . {-установка графического режима}
x:=GetMaxX div 2;
y:=GetMaxY div 2;   {- координаты центра экрана}
For i:=1 to 5 do begin
  SetActivePage(0);
  PieSlice(x,y,0,360,10);
  SetActivePage(1);
  PieSlice(x+30,y+30,0,360,10);
  SettVisualPage(0);
  Readln;
 SettVisualPage(1);
  Readln;
End;
CloseGraph;
End.
```

```
Program DV_PAGE;
{Построение движущегося изображения переключением
графических
страниц. Параметры движения:
t - задержка видимой страницы, подбирается для
каждого ПК, должно
превышать время закрашивания, чтобы на изображении
не было полос;
h - шаг смещения изображения, определяет скорость
движения}
Uses Crt, Graph;
Var  d,r,i,nmax: integer;
Const t=50;    h=2;

Procedure Ris(cv:byte; dx:integer);
{Построение изображения; cv-цвет; dx-смещение}
begin
    Setcolor(cv);
    SetfillStyle(1,cv);
    Circle(30+dx,180,30);  Bar(dx,200,60+dx,280);
    Circle(610-dx,50,30);  Bar(580-dx,70,640-
dx,150);
end;

Begin
 d:=Detect;
 InitGraph(d,r,'');
 if GraphResult <>grOk then Halt(1);
 SetGraphmode(1);
 nmax:=630 div h;
 i:=1;
While i<nmax do begin
    SetActivePage(0);  Ris(0,(i-2)*h);
    Ris(14,i*h);       SetVisualPage(0);
    delay(t);
     SetActivepage(1); Ris(0,(i-1)*h);
     Ris(14,(i+1)*h);  SetVisualPage(1);
     delay(t);
 i:=i+2;
end;
Readln;
Closegraph;
End.
```

Вопросы для самопроверки

1. Как организуется движение изображения с помощью изменения цвета.

2. Назначение и параметры процедур PutImage и GetImage.

3. Как организуется движение изображения с помощью операторов PutImage и GetImage.

4. Как организуется движение объектов с помощью чередования видеостраниц.

СОДЕРЖАНИЕ ЗАДАНИЯ

Задача 1. Организовать движение заданных графических объектов в пределах экрана с помощью чередования цвета. Перед построением изображения задать количество повторений (оборотов), геометрические и физические параметры изображения. Изображение на экране пояснить текстом.

1) Пульсирующая окружность.

2) Квадрат, вращающийся относительно своего центра.

3) Пульсирующий квадрат.

4) Последовательность N точек, появляющихся в центре экрана и движущихся к случайной точке на границе экрана.

5) Колебания математического маятника.

6) Ядро, вылетающее от середины левой границы экрана в точку с координатой Y на правой границе.

7) Пружина с жесткостью k, сжимающаяся под тяжестью груза массы m.

Задача 2. Организовать движение закрашенный графических объектов в пределах экрана с помощью операторов PutImage и GetImage. Перед построением изображения задать количество повторений (оборотов), геометрические и физические параметры изображения. Изображение на экране пояснить текстом.

1) Падающие закрашенные прямоугольники. Новый начинает падать, когда предыдущий достигнет нижней границы экрана.

2) Закрашенный круг радиуса R1, движущийся по окружности радиуса R2.

3) Закрашенная окружность, перемещающаяся от левого верхнего угла экрана к правому нижнему вдоль границ экрана.

4) Движение двух тел массы m1,m2, связанных нитью, перекинутой через блок.

5) Отскоки мяча от горизонтальной поверхности. При каждом отскоке теряется 10% механической энергии.

6) Центральное столкновение шара массы m1, движущегося со скоростью v0, с неподвижным шаром массы m2. Радиусы шаров одинаковы.

7) Поднимающийся аэростат с грузом.

8) Мяч, брошенный под углом к горизонту в поле силы тяжести.

9) Упругое столкновения шара с начальной скоростью vx0,vy0 с границами экрана.

ТЕМА 6. ПРОЦЕДУРА ИЗОБРАЖЕНИЯ ГРАФИКА ФУНКЦИИ

6.1 Математические и экранные координаты

6.2 Построение и разметка осей

6.3 Алгоритм и процедура изображения графика функции на экране ПК

6.4 Использование процедуры Grafik1 при задании фунции массивом значений

6.1 Математические и экранные координаты

Большинство физических задач содержит этапы исследования заданных или вычисленных в ходе решения функциональных зависимостей. При этом графический способ представления функции является наиболее наглядным и удобным для изучения.

Построение графика с помощью компьютера выполняется обычным образом: изображаются отдельные точки графика, которые соединяются затем отрезками прямых.

На экране, наряду с линией графика, должны присутствовать элементы оформления: название графика, численные значения делений графика (оцифровка), значения параметров графика.

Поэтому для размещения точек графика используется не весь экран, а его часть. Будем считать, что это прямоугольник в пределах [*Xe1,Xe2*] по оси OX и [*Ye1,Ye2*] по оси OY.

Экранный график является увеличенной или уменьшенной копией математического графика функции *Y(X)*, так как положение точки графика на экране определяется границами экранной области. Поэтому для каждой точки графика следует различать ее математические координаты (реальные значения *Xm* и *Ym*) и экранные координаты – координаты положения этой точки на экране дисплея *Xe* и *Ye*./

Связь экранных координат с математическими устанавливается из требования пропорциональности отрезков, отсекаемых координатами точки на осях в математической и экранной областях. Так, для произвольной точки графика (*Xm,Ym*) отрезок (*Xe-Xe1*) на оси OX должен составлять такую же часть длины экранной области (*Xe2-Xe1*), что и отрезок (*Xm-Xmin*) от полной математической области (*Xmax-Xmin*). На основании этого получаем пропорцию

$$\frac{Xm - X\min}{X\max - X\min} = \frac{Xe - Xe1}{Xe2 - Xe1} \qquad (1)$$

Аналогичную пропорцию можно составить для координат точки по оси Y, с учетом различной направленности математической и экранной осей (*Ye2<Ye1*):

$$\frac{Ym - Y\min}{Y\max - Y\min} = \frac{Ye1 - Ye}{Ye1 - Ye2} \qquad (2)$$

В записанных соотношениях *Xmin, Xmax, Ymin, Ymax* - параметры, определяющие пределы изменения математических координат точек; *Xe1, Xe2, Ye1,Ye2,* - константы, определяющие границы области экрана, внутри которой должны находиться точки графика. Все эти величины должны быть определены перед построением графика. Предполагается, что координаты точек по оси OX упорядочены по возрастанию, так что *Xmin=X(1), Xmax=X(NT).* Числа *Xm, Ym* для каждой точки также должны быть известны.

Задача заключается в том, чтобы выразить числа *Xe,Ye* координат точки графика на экране. Как следует из (1), (2), положение точки на экране, соответствующей точке графика с координатами *Xm,Ym*, определяется по формулам

$$Xe = Xe1 + \frac{Xm - X\min}{X\max - X\min}(Xe2 - Xe1);$$

$$Ye = Ye1 + \frac{Ym - Y\min}{Y\max - Y\min}(Ye2 - Ye1). \qquad (3)$$

Удобно обозначить постоянные множители (коэффициенты масштабирования осей):

$$kx = \frac{Xe2 - Xe1}{X\max - X\min}; \qquad ky = \frac{Ye2 - Ye1}{Y\max - Y\min}.$$

Тогда формулы (3) вычисления экранных координат принимают вид:

$$Xe = Xe1 + kx \cdot (Xm - X\min); \qquad Ye = Ye1 + kxy \cdot (Ym - Y\min)$$

6.2 Построение и разметка осей

Для удобства использования оси графика должны быть подробно размечены и оцифрованы. При этом расположение текстовых и графических элементов строго согласовано, и границы экранной области нельзя изменять произвольно. Поэтому они не являются параметрами процедуры, а используются как внутренние переменные. Название графика является параметром строкового типа и может содержать до 80 символов.

Необходимо также предусмотреть изображение оси OX в виде горизонтальной пунктирной линии на уровне *Ym=0*, то есть, *Ye=Y1+Ky*Ymin* в случае, если *Ymin* и *Ymax* имеют разные знаки.

Операторы построения графика удобно оформить в виде процедуры, исходной информацией для которой будут массивы X- и Y-координат всех точек графика. Тогда в основной программе необходимо будет организовать заполнение этих массивов (вычислением значений аргумента и функции или вводом их с клавиатуры, или вычислением в ходе решения некоторой расчетной задачи) и вызвать процедуру построения графика.

6.3 Алгоритм и процедура изображения графика функции на экране ПК

Таким образом, алгоритм построения графика функции в заданной области экрана состоит из следующих этапов:

- для заданной функции y=f(x) разделить отрезок [A,B] на NI интервалов и для каждой из NT точек вычислить значения аргумента и функции, заполнив таким образом массивы X(NT), Y(NT) математических координат точек графика;

- вызвать процедуру построения графика функции, передав с помощью параметров: количество точек, имена массивов, название графика, номера цветов для оформления графика, режим использования точечной сетки;

- процедура построения графика должна выполнить следующие действия:

- найти максимальное и минимальное значения функции;

- задать экранные границы графика и вычислить коэффициенты масштабирования осей;

- ограничить область графика, разметить и оцифровать оси;

- вывести название графика;

- если параметр grid равен 1, нанести на область графика точечную сетку;

- для каждой точки графика вычислить ее экранные координаты и соединить отрезком прямой с предыдущей точкой.

Для передачи подпрограмме одномерных массивов произвольного размера используются параметры-переменные без указания их типа. Бестиповыми могут быть только параметры-переменные, (т.е. те, которые передаются как адрес, а не как значение, с указанием VAR). При этом внутри подпрограммы необходимо явно указать с помощью собственного описания *Type*, в качестве какого типа она будет обрабатывать переданные ей бестиповые параметры. Кроме того, следует предусмотреть наличие параметра, указывающего фактическое количество используемых элементов. При использовании таких параметров подпрограмма будет корректно работать только с массивами, элементы которых имеют тип Real. Это ограничение задано внутренним типом GrMas, к которому приводятся элементы бестиповых параметров x и y с помощью операций приведения

типа GrMas(x)[i]. Количество фактически обрабатываемых элементов передается через параметр n.

Следует, однако, учитывать, что увеличение количества точек не всегда приводит к улучшению качества графика, так как количество точек в области графика фиксировано, а вычисленные экранные координаты все равно округляются до целых значений. Поэтому количество точек рекомендуется выбирать в пределах 20-500.

Перечисленные выше действия реализованы в процедуре GRAFIK1, включенной в состав модуля GRAFIK_D (установлен на компьютеры в дисплейном классе). Заголовок процедуры:

```
Procedure Grafik1(n:integer; var x,y;
          nazv:string; cv1,cv2,grid:byte);
```

Процедура предназначена для изображение графика одной функции с построением прямоугольных границ графика и их разметкой в качестве осей OY и OX на 20 делений. Параметры: n - количество точек графика; x, y - бестиповые массивы значений аргумента и функции; nazv$ - название графика (до 80 символов); cv1 - цвет изображения названия графика и линий графика; cv2 - цвет разметки осей и цифровых надписей; grid - параметр построения точечной сетки (1 - нанести сетку на график, 0 - нет). Ниже приведен текст процедуры.

```
Procedure Grafik1(n:integer; var x,y; nazv:string;
cv1,cv2,grid:byte);
{(C) Дей Е.А. 2005}
-------------------------------------------------}
type
  GrType = array[0..2000] of real;
const          {--границы экранной области:}
  xe1=90; xe2=590; ye1=308; ye2=28; ndel=20;
var
  grDriver, grMode :integer;
  i,j,k :integer;
  xmin,xmax,ymin,ymax,x0,y0 :real;
  dy,dx,z,kx,ky,cdx,cdy, xt,yt,xt0,yt0 :real;
  sdx,sdy :string;
begin
  grDriver := 3; grMode:=1;
  InitGraph(grDriver,grMode,'');
  if GraphResult <> grOk then Halt(1);
```

```
        {---Определение графического окна:}
  SetViewPort(xe1,ye1,xe2,ye2,True);
  SetBkColor(0);   ClearViewPort;
        {---Вывод названия графика:}
  SetColor(cv1);   OutTextXY(xe1,1,nazv);
     {--прямоугольная область графика:}
  SetColor(cv2);  rectangle(xe1,ye1,xe2,ye2);
      {--нахождение ymin,ymax:}
  xmin:=GrType(x)[0]; xmax:=GrType(x)[n];
  ymin:=GrType(y)[0]; ymax:=GrType(y)[0];
  for i:=1 to n do begin
    if GrType(y)[i]>ymax then ymax:=GrType(y)[i];
    if GrType(y)[i]<ymin then ymin:=GrType(y)[i];
  end;
      {--коэффициенты масштабирования осей:}
  kx:=(xe2-xe1)/(xmax-xmin);
  ky:=(ye1-ye2)/(ymax-ymin);
  dy:=(ye1-ye2)/ndel;
      {--разметка и оцифровка оси OY:}
  for i:=0 to ndel do begin
    z:=ye2+dy*i;
    line(xe1-2,Round(z),xe1+2,Round(z));
    line(xe2-2,Round(z),xe2+2,Round(z));
    z:=ymax-(ymax-ymin)*i/ndel;
    Str(z:7:3,sdx); OutTextXY(24,ye2-3+i*14,sdx);
  end;
       {--разметка и оцифровка оси OX:}
  dx:=(xe2-xe1)/ndel;
  for i:=0 to ndel do begin
    z:=xe1+dx*i;
    line(Round(z),ye1-2,Round(z),ye1+2);
    line(Round(z),ye2-2,Round(z),ye2+2);
    if (i mod 5) = 0 then begin   {каждое 5-е
деление}
      z:=xe1+dx*i;
      line(Round(z),ye1-2,Round(z),ye1+7);
      Str((xmin+(xmax-xmin)*i/ndel):7:3,sdx);
      OutTextXY(xe1-28+i*25,ye1+12,sdx);
    end;
  end;
   cdx:=(xmax-xmin)/ndel;  {--математич. цена
делений}
   cdy:=(ymax-ymin)/ndel;
   Str(cdx:6:3,sdx); Str(cdy:6:3,sdy);
```

```
   OutTextXY(375,340,'Dx= '+sdx);
   OutTextXY(250,340,'Dy= '+sdy);
          {--изображение оси OX на уровне Y=0:}
  if ymin*ymax<0 then begin
    y0:=ye1+ymin*ky;
    SetLineStyle(1,1,1);
    line(xe1,Round(y0),xe2,Round(y0));
{**}    OutTextXY(xe2+7,Round(y0)-3,'OX');
  end;
{**}          {--изображение оси OY на уровне X=0:}
  if xmin*xmax<0 then begin
    x0:=xe1-xmin*kx;
    SetLineStyle(1,1,1);
    line(Round(x0),ye1,Round(x0),ye2);
    OutTextXY(Round(x0)-7,ye2-11,'OY');
  end;
        {-- построение точечной сетки в области
графика:}
  if grid=1 then begin
    for i:=1 to ndel do
    for j:=1 to ndel do
      PutPixel(Round(xe1+dx*i),Round(ye1-dy*j),cv2);
  end;
        {-- изображение линий графика:}
  SetColor(cv1);
  SetLineStyle(0,1,1);
  for i:=0 to n do begin
    xt:=xe1+(GrType(x)[i]-xmin)*kx;
    yt:=ye1-(GrType(y)[i]-ymin)*ky;
    if i>0 then
line(Round(xt0),Round(yt0),Round(xt),Round(yt));
    xt0:=xt; yt0:=yt;
  end;
  Readln;
  CloseGraph;
end;
```

6.4 Использование процедуры Grafik1 при задании фунции массивом значений

Для использования процедуры GRAFIK1 достаточно подключить модуль GRAFIK_D в описании USES и задать в программе значения всех параметров процедуры. Например:

```
Program Use_Grafik1;
{==================
  Построения графика функции для IBM PC AT-286
  в режиме EGA с помощью процедуры GRAFIK1
 ============================================== }
Uses  Grafik_D, Crt;
Type  Massiv=array [0..500] of real;
Var
   nazv:string;
   x,y:massiv;
   hx,a,b:real;
   i,ni,nt:integer;
Function FUN1(a:real):real;
{ вычисление значений исследуемой функции}
begin
  fun1:=a*a*sin(a)*exp(-0.01*a*a);
end;
BEGIN
  TextColor(11); TextBackGround(1); ClrScr;
      {--задание интервала и количества точек:}
  nazv:='ГРАФИК ФУНКЦИИ Y=a*a*sin(a)*exp(-
0.01*a*a)';
  writeln;                    {a:=-4; b:=4; nt:=250;}
  writeln(nazv);
  write('Введите границы интервала a,b: ');
  readln(a,b);
  writeln('Введите число точек графика {<=500}: ');
  readln(nt);
      {--заполнение массивов:}
  hx:=(b-a)/nt;
  for i:=0 to nt do begin
    x[i]:=a+i*hx;
    y[i]:=fun1(x[i]);
  end;
  Grafik1(nt,x,y,nazv,14,12,1);
end.
```

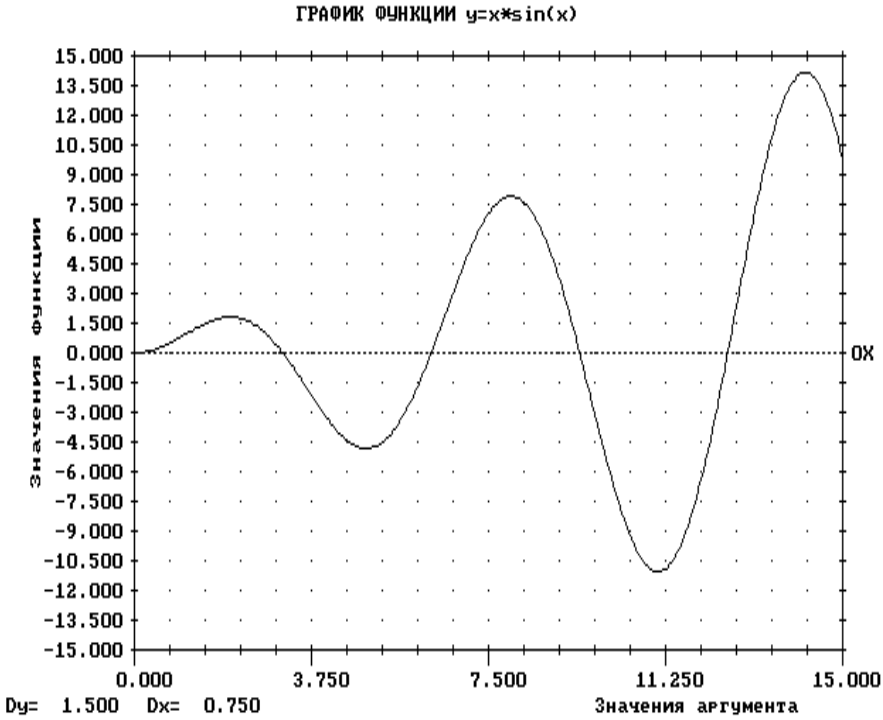


Рис 6.1 Вид графика функции y=x*sin(x), построенного процедурой Grafik1

Следует особо отметить тот факт, что обычно при построении графика задается функциональная зависимость. Однако канонические уравнения многих кривых не позволяют получить такую зависимость. Примерами подобных уравнений могут служить уравнения окружности $x^2+y^2=R^2$, эллипса $x^2/a^2+y^2/b^2=1$, гиперболы $x^2/a^2-y^2/b^2=1$ и многие другие. В этих случаях удобно пользоваться параметрическими уравнениями кривых, где x и y представлены как функции некоторого параметра t.

При использовании N точек графика формулы для расчета координат отдельных точек будут следующими:

Эллипс: $x=x0+A\cos(i\pi/N)$; $y=y0+B\sin(i\pi/N)$;

Спираль: $x=0.5t\cos(t)+A$; $y=t\sin(t)+B$;

Фигуры Лиссажу:

$$x=A_1\cos(\omega_1 t+\varphi_1)+B_1; \quad y=A_2\sin(\omega_2 t+\varphi_2)+B_2;$$

Используя обычные формулы преобразования полярных координат в декартовы, нетрудно состовить программу, в которой кривая задается уравнением в полярных координатах.

Для построения графика двух функций в одной общей области можно использовать процедуру GRAFIK2, которая содержится в модуле Grafik_D. Заголовок процедуры:

```
Procedure Grafik2(n:integer; var x,yl,y2;
            nazv:string; cvl,cv2,cv3, grid: byte);
```

Процедура имеет следующие параметры (все - входные):

n - количество точек графика;

x, yl, y2 - бестиповые массивы значений аргумента, первой и второй функций соответственно;

nazv - название графика (до 80 символов)

cvl-цвет названия графика и линии графика первой функции

cy2-цвет разметки осей и цифровых надписей

cv3 - цвет линии графика второй функции

grid - параметр построения точечной сетки (1 или 0)

Перед обращением к процедуре GRAFIK2 необходимо заполнить массивы x, yl, y2 по аналогии с примером п.4.

Вопросы для самопроверки

1. Каков порядок построения кусочно-линейного графика функции

2. Повторите вывод формул для коэффициентов масштабирования осей

3. Перечислите варианты построения и разметки осей

4. Каковы шаги алгоритма изображения графика функции на экране ПК

5. Как в программе организуется передача подпрограмме массивов произвольного размера.

Задачи

1. Набрать и отладить программу из п.5.3 Создать EXE-файл. Выполнить программу при различных значениях параметров. Проанализировать, как влияет на качество графика выбор количества точек, построив график той же функции при nt=5,10,50,100. Сравнить удобство отсчета координат точек при наличии точечной сетки на графике и без нее (изменяя

фактическое значение параметра grid при вызове процедуры). С помощью параметров cvl,cv2 подобрать удобное цветовое оформление графика. Рассмотреть глобальное поведение функции в области [0,100] и локальное поведение в области [10,12].

2. С помощью процедуры GRAFIK1 построить график заданной функции на указанном интервале при nt=200. По графику определить характер изменения функции, точки ее максимумов и минимумов, а также нули функции.

1. y(x) = $0.1*x^3/(x+1)*\exp(-0.4x)$; [0,15]
2. a(b) = $\ln(1+b^2)/\mathrm{sqr}(1+\sin(3b)^2)$; [0,5]
3. s(z) = $3\sin(z+1)/(z^2+2)$; [-3,5]
4. f(x) = $2\sin(\cos(2x)+x)$; [-5,5]
5. w(t) = $\sin(t)-\ln(1+t^2)$; [-8,8]
6. r(d) = arctg(2cos(d)); [-4,4]

3. Составить диалоговую программу для изображения графиков квадратных трехчленов $y=ax^2+bx+c$. Значения коэффициентов a, b, c вводить с клавиатуры в текстовом режиме. После заполнения массивов x и y выполнить вызов процедуры Grafik1. Перечисленные действия организовать в цикле, признаком окончания работы является ввод нулевых значений для a, b, c. С помощью программы выполнить построение парабол с различными, в том числе отрицательными, коэффициентами.

4. На основании аналитического выражения для исследуемой физической величины составить программу на языке Турбо-Паскаль, в которой выполняется запрос и ввод значений параметров в текстовом режиме, расчет данных для построения графика величины и вызывается процедура GRAFIK1, изображающая график. С помощью графика функции исследовать поведение физической системы (определить характер поведения функции, нули, точки максимума и минимума). Напечатать график.

1) В последовательном резонансном контуре зависимость силы тока от частоты приложенного напряжения определяется выражением

$$I = \frac{U_0}{Z} = \frac{I_p}{\sqrt{1+Q^2\left(\frac{\omega}{\omega_0}-\frac{\omega_0}{\omega}\right)^2}}$$

где Q - добротность контура, ω_0 - резонансная частота контура, I_p - амплитуда силы тока при резонансе, U_0 и ω - амплитуда и частота внешнего напряжения. Построить графики зависимости силы тока I/I_p в последовательном резонансном контуре от частоты приложенного сигнала при различных значениях добротности контура Q= 1.0; 5.0; 10.0. В качестве аргумента выбрать параметр $0.1 \le \omega/\omega_0 \le 2.1$.

2) Значение давления в атмосфере Земли при изменении высоты описывается следующей эмпирической зависимостью:

$$P(г/см^2) = \begin{cases} 1030\left(1 - \dfrac{h}{44308}\right)^{5.25} & 0 \le h(м) \le 10^4 \\ 268\,exp\left(\dfrac{10^4 - h}{6340}\right) & 10^4 \le h(м) \le 3\cdot 10^4 \\ \dfrac{5\cdot 10^4 - h}{2020} & 3\cdot 10^4 \le h(м) \le 5\cdot 10^4 \\ 0 & h(м) \ge 5\cdot 10^4 \end{cases}$$

Для аналитических исследований используют т.н. барометрическую формулу $P(г/см^2) = 1030\,exp(-\dfrac{h(м)}{7991})$. Построить графики обеих зависимостей для $0<h<5*10^4$. Определить максимальное отличие (в процентах) аналитической зависимости от результатов эксперимента.

3) В момент t=0 точке x_0=0 бесконечного стержня с начальной температурой T_0 сообщили количество тепла Q_0. В этом случае температура любой точки стержня в произвольный момент времени определяется формулой

$$T(x,t) = \frac{Q_0}{2C\rho\sqrt{\pi t}} e^{-\frac{x^2}{4t}},$$

где C - теплоемкость, ρ - плотность материала стержня. Построить графики температуры участка бесконечного стержня $-3 \le x \le 3$ в моменты времени t = 0.1; 0.5; 1.0; 5.0 секунд. Для каждого момента определить по графику область стержня, в которой $T>T_1$. Параметры задачи: Q_0 = 1500 Дж; C = $1{,}5\cdot 10^3$ Дж/(кг·град); ρ = $2{,}7\cdot 10^3$ кг/м^3; T_0=20 град; T1=28 град.

4) Колебания тела, обусловленные одновременным действием квазиупругой силы и силы трения, пропорциональной мгновенной

скорости, являются затухающими и описывается формулой $U = Ae^{-\delta t}\sin(\omega t+\varphi)$. Здесь A - амплитуда, $\delta=\alpha/(2m)$ - коэффициент затухания, α – коэффициент сопротивления, $\omega = \sqrt{\omega_0^2 - \delta^2}$ - циклическая частота, ω_0 - собственная частота физической системы, φ - начальная фаза. Построить график затухающих колебаний для $\omega_0=0.5$, $\delta=0.2$, $A=3$, $\varphi=\pi/6$. Определить по графику, за какое время максимальное отклонение уменьшится до значения $U = 0,6A$.

ТЕМА 7. ПЕЧАТЬ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

7.1 Формирование графических изображений при печати

7.2 Команды управления работой принтера и режимами графической печати

7.3 Программный контроль состояния принтера

7.4 Процедура графической печати прямоугольного фрагмента экрана

7.1 Формирование графических изображений при печати

Логическим завершением работы по программированию и построению изображения на экране дисплея является сохранение результатов – печать изображения на бумаге или запись его в файле на диск в одном из графических форматов.

Рабочим элементом матричного принтера является печатающая головка, которая содержит 9 вертикально расположенных иголок. Текст или графическое изображение, формируемые матричным принтером, состоит из горизонтальных строк, набранных из близко расположенных вертикальных колонок, состоящих, в свою очередь, из отдельных точек.

Матричный принтер подачей управляющих кодов (специальная последовательность байтов) может быть переведен в растровый (графический) режим работы, в котором подаваемые на принтер байты интерпретируются уже не как коды символов, а как данные, непосредственно управляющие иглами печатающей головки. Следующие байты, поступающие на принтер, управляют

каждой из 8 игл печатающей головки. Байты обрабатываются таким образом: срабатывают иголки печатающей головки принтера, соответствующие «установленным в 1» битам в двоичном представлении байта.

Графическое изображение выводится матричным принтером построчно, графическая строка образуется из вертикальных колонок, состоящих обычно из 8 точек, которые соответствуют состоянию восьми вертикально расположенных пикселов экрана. Если цвет пиксела отличен от фона (≠0), то при печати ставится точка, если совпадает с фоном (=0), точка не ставится. В графических данных каждая колонка строки описывается одним байтом, причем верхняя точка колонки соответствует состоянию 7-го бита байта данных.

Состояние точек экрана	Установка битов по состоянию точек экрана	Вклад битов В значение Байта	Значение байта, описывающего состояние точек экрана
●	1	1*128=128	
·	0	0* 64= 0	
·	0	0* 32= 0	1+2+4+0+
●	1	1* 16= 16	+16+0+0+128=
·	0	0* 8= 0	=151
●	1	1* 4= 4	
●	1	1* 2= 2	
●	1	1* 1= 1	

В 8-точечной графике существует восемь режимов, отличающихся горизонтальной плотностью (точек на дюйм) [15]:

0. Одинарная плотность печати - 60.
1. Двойная плотность печати - 120.
2. Двойная плотность с высокой скоростью – 120.
3. Четырехкратная плотность печати – 240.
4. Печать с плотностью ЭЛТ1 - 80.
5. Одинарная печать с плотностью графо-построителя - 72.
6. Печать с плотностью ЭЛТ2 - 90.
7. Двойная печать с плотностью графопостроителя 144 точек/дюйм.

В режимах 3) и 4) печать соседних точек невозможна. Если эти точки встречаются в описании графического изображения, то они исключаются автоматически.

7.2 Команды управления работой принтера и режимами графической печати

Управление работой матричных EPSON-совместимых принтеров осуществляется с помощью символов кодовой таблицы с номерами 0...31, 128...159 или специальных последовательностей байтов (команд). Первый байт команды имеет значение #27 (ESC-признак команды), следующий байт - код команды, далее - необходимое число байт - параметров команды, после которых следуют байты данных. При печати эти коды не будут появляться в выводимых документах.

Приведем несколько наиболее употребляемых кодов (в десятичной форме) управления печатью для матричных принтеров:

7 - принтер выдает звуковой сигнал;

14 - установка расширенного шрифта;

15 - выбор уплотненного шрифта;

18 - отмена уплотненного шрифта;

20 - отмена расширенного шрифта:

27 77 - выбор шрифта "элит"(12 знаков печатается на 2,56 см);

27 80 - отмена шрифта "элит" и установка значения по умолчанию (10 знаков на 2,56 см);

27 69 - установка подчеркнутого шрифта;

27 70 - отмена подчеркнутого шрифта;

27 71 - установка режима двойного удара;

27 72 - отмена режима двойного удара.

Примеры:

```
writeln (Lst,#14,'-РАСШИРЕННЫЙ ШРИФТ');
writeln (Lst,#15,'-УПЛОТНЕННЫЙ ШРИФТ');
writeln (Lst,#27,'x','1'); {включение режима
                            высококачественной печати}
```

Все команды управления принтером можно разделить на две группы: глобальные и локальные. Команды первой группы посылаются перед тем, как содержимое файла будет выводиться на печать, и действуют на весь выводимый текст. Команды второй

группы используются непосредственно в выводимом тексте, они могут включаться и отменяться в тексте любое количество раз.

Команды 8-точечной графики имеют следующий формат:

```
ESC <тип графики> n1 n2 <данные>
```

где <тип графики> — определяет горизонтальную плотность графического изображения и скорость печати; числа n1 и n2 указывают общее число колонок в графической строке, <данные> — байты, описывающие колонки в графической строке. Числа n1 и n2 вычисляются по следующему правилу:

```
n1=<число колонок данных> MOD 256;
n2=<число колонок данных>  DIV 256;
```

Например, если число колонок равно 400, то n2=1, n1=144.

Общее число графических байтов равно, таким образом, N=256*n2+n1. Лишние байты будут рассматриваться как текстовая информация. Если байтов окажется недостаточно, принтер приостановит печать.

При ширине полного экрана 640 пиксел ширина напечатанного изображения будет варьироваться от 6,77 см при четырехкратной плотности печати до 27,1 см при одинарной плотности печати. Число посылаемых графических данных должно точно соответствовать числу данных, указанных в команде графической печати. Если графическое изображение выходит за пределы бумаги, то оставшаяся часть игнорируется.

Строки графики должны быть расположены вплотную одна под другой. По вертикали расстояние между двумя соседними точками равно 1/72 дюйма. Поэтому при печати столбца из 8 точек необходимо расстояние между графическими строками в 8/72 дюйма, чтобы между графическими строками не было промежутка. Поэтому при работе с графикой необходимо установить межстрочный интервал равным 8/72=24/216=1/9 дюйма. После завершения графической печати необходимо установить стандартное значение интервала 1/6 дюйма.

7.3 Программный контроль состояния принтера

Возможность печати изображения во время выполнения программы зависит от текущего состояния принтера. Для получения информации о состоянии принтера следует использовать функции обслуживания периферийного

оборудования, доступ к которым осуществляется через прерывания BIOS [15].

Работу с параллельным портом ввода-вывода обеспечивает прерывание номер $17. Оно выполняет три функции: вывод символа на печать, инициализацию параллельного порта и получение слова состояния принтера.

Для опроса состояния ПУ необходимо занести в регистр AH значение 2, в регистр DX — номер порта параллельного ввода-вывода (0, 1 или 2). После выполнения прерывания в регистре AH будет слово состояния ПУ.

Назначение битов слова состояния ПУ прерывания int 17

0. Тайм-аут
1. Не используется
2. Не используется
3. Ошибка ввода-вывода
4. Выбран
5. Нет бумаги
6. Подтверждение
7. Не занято (0 - занято)

Для определения текущего состояния принтера необходимо анализировать все биты слова состояния ПУ комплексно. Существует пять часто встречающихся ситуаций:

1. Устройство готово к работе. Установлены биты 4 — устройство выбрано (SLCT) и 7 — ПУ не занято, значение слова состояния 144d (90h).

2. Принтер не подключен к ПЭВМ. Слово состояния показывает, что ПУ занято и обнаружен конец бумаги, значение слова состояния 48d (30h), это значение может зависеть от конкретной реализации адаптера печати ПЭВМ.

3. Принтер выключен, но подключен к ПЭВМ с помощью интерфейсного кабеля. В этом случае имеется ошибка ввода-вывода и ПУ находится в состоянии "Не выбрано", значение слова состояния 136d (88h).

4. Принтер находится в автономном режиме (состояние "off-line"). В этой ситуации слово состояния показывает, что устройство занято и имеется ошибка ввода-вывода; значение слова состояния 24d (18h).

5. В принтере нет бумаги. Слово состояния показывает, что есть ошибка ввода-вывода, нет бумаги в устройстве и ПУ занято, значение слова состояния 56d (38h).

С учетом изложенного можно предложить следующий вариант текста подпрограммы-функции проверки состояния принтера:

```
Function TestPrinter:byte;
{ Получение состояния порта печати LPT1 (PRN)
  При работе требуется подключение модуля DOS}
var rg:registers;
begin
  TestPrinter:=0;
  with rg do begin
    AH:=$02;    DX:=$00;
    Intr($17,rg);
    TestPrinter:=5;
    case AH of
      144: TestPrinter:=0;
       48: TestPrinter:=1;
      136: TestPrinter:=2;
       24: TestPrinter:=3;
       56: TestPrinter:=4;
    end;  {case}
  end;  {with}
end;  {TestPrinter}
```

Выходные значения функции: 0 - принтер готов к работе; 1 - нет принтера; 2 - принтер выключен; 3 - не включен режим On Line; 4 - нет бумаги; 5 - прочее.

7.4 Процедура графической печати прямоугольного фрагмента экрана

```
Program ScreenPrint;
{ Копирование изображения с графического экрана
на матричный принтер
{(C) Дей Е.А. 2005}
-------------------------------------------}
Uses Crt,Printer,Graph;
```

```
Procedure GrInit;
var
  GrDriver,GrMode,ErrCode:integer;
begin
  GrDriver:=Detect;
  InitGraph(GrDriver,GrMode,'');
  ErrCode:=GraphResult;
  If ErrCode<>grOk then begin
    Writeln('Ошибка графики:',
            GraphErrorMsg(ErrCode));
    Writeln('Программа остановлена...');
    Halt(1);
  end;
end;

Procedure CopyToPrn(x1,y1,x2,y2:integer;
bk1,bk2:word;
                    inverse:boolean; mode:byte);
  {Mode  =1 - двойная плотность}
Var
 ScanLine:integer;  n1,n2:byte;

 Function ConstructByte(x,y:integer):byte;
 Const
   Bits: array [0..7] of byte =
(128,64,32,16,8,4,2,1);
 Var
  p:word; cbyte,bit:byte; yy:integer;
 Begin
  cbyte:=0;
  for bit:=0 to 7 do begin
    yy:=y+bit;
    p:=GetPixel(x,yy);
    if (yy<=y2) and (p<>bk1) and (p<>bk2)
      then Inc(cbyte,bits[bit]);
  end; {for}
  ConstructByte:=Cbyte;
 end; {ConstructByte}

 Procedure Doline;
 var
   Xpixel:integer;
   PrintByte:byte;
 begin
```

```
  if Mode=1
    then Write(Lst, #27'L')
    else Write(Lst, #27'*',Chr(Mode));
  Write(Lst,Chr(n1),Chr(n2));
   for Xpixel:=x1 to x2 do begin
     PrintByte:=ConstructByte(Xpixel,ScanLine);
     if inverse then PrintByte:= not PrintByte;
     Write(Lst,Chr(PrintByte));
   end; {for}
   Writeln(Lst);
 end;

 Label  Quit;
 Begin
  Mode:=Mode mod 7;
  if Mode in [0,5] then Mode:=4;
  Write(Lst,#27'3'#24);
  n1:=Lo(Succ(x2-x1));
  n2:=Hi(Succ(x2-x1));
  ScanLine:=y1;
  while ScanLine<y2 do begin
   { if KeyPressed and (ReadKey=#27) then Goto
Quit;}
    DoLine;
    Inc(ScanLine,8);
  end;
Quit:
  Write(Lst,#27#2);
End;
{-----------------------------------------------}
Begin
  GrInit;  SetBkColor(0);
  Rectangle(50,50,500,300);
  SetFillStyle(HatchFill,2);
  FillEllipse(200,100,60,40);
  SetFillStyle(4,3);
  Bar(300,150,400,200);
  SetFillStyle(8,7);
  Bar(100,200,180,280);
  CopyToPrn(40,40,520,320,0,0,False,0);
  Readln;
  CloseGraph;
End.
```

Иногда бывает удобно организовать вывод так, что если принтер готов к работе, то изображение выводится на принтер. Если же принтер не готов к работе, то выполняется "печать в файл", т.е. байты, предназначенные для посылки в принтер, записываются в нетипизированный файл. В последующем командой MS DOS

```
>COPY name PRN
```

где name - имя файла, можно напечатать изображение в любое время и любое количество раз.

Для реализации этого подхода в приведенных выше процедурах печати изображения не используется модуль Printer, а осуществляется вывод информации в файл. Если в качестве имени файла указывается системное имя принтера 'PRN' или связанного с ним порта 'LPT1', то изображение будет напечатано. Если указывается другое имя файла, информация будет записана на диск как обычный файл, при копировании которого на устройство PRN будет выполнена печать изображения.

Для отличия файлов, содержащих графическую информацию в формате принтера, будем использовать расширение *.pri.

Вопросы для самопроверки

1. Алгоритм построения графика функции
2. Построение графика функции, заданной параметрически
3. Построение графика функции, заданной в полярных координатах.

Задачи

1. Составить процедуру для печати всего графического экрана вдоль страницы (с поворотом страницы на 90° по часовой стрелке).
2. Составить процедуру для печати прямоугольного фрагмента экрана вдоль страницы (с поворотом страницы на 90° по часовой стрелке).

ТЕМА 8. ПРОГРАММИРОВАНИЕ КЛАВИШНЫХ КОМАНД

8.1 Технические особенности клавиатуры
8.2 Встроенные функции для работы с клавиатурой
8.3 Программа для изучения символьных кодов клавиш

8.1 Технические особенности клавиатуры

Прикладные графические программы работают в интерактивном режиме, когда координаты и размеры графических примитивов не задаются в программе, а вводятся напрямую в графическом режиме с помощью клавиатуры или манипулятора «мышь». Интерактивный ввод означает, что на экране рисуется «проект» элемента, и клавишными командами уточняются его параметры и размеры. Именно такой принцип работы реализуется в программах – графических редакторах, а также в современных системах автоматизированного проектирования.

Управление работой программы основано на том, что символьный код нажатой клавиши может быть проанализирован в программе, и в зависимости от того, какая клавиша была нажата, в программе заранее предусматривается выполнение какого-то действия.

Первый шаг в реализации таких действий – изучение программных средств работы с клавиатурой.

Стандартная клавиатура имеет четыре типа клавиш: символьные (буквы, цифры); специальные (функциональные клавиши [F1]..[F12], клавиши перемещения курсора, клавиши [Del], [Home], [End], [PgUp] и т.д.); клавиши модификации ([Ctrl], [Alt], [Shift]); клавиши-переключатели ([Ins], [NumLock], [CapsLock], [ScrollLock]).

Несмотря на кажущуюся простоту, клавиатура ПЭВМ представляет собой достаточно сложное устройство, содержащее в том числе собственную область памяти (т.н. буфер клавиатуры объемом 16 байт) и встроенный микропроцессор Intel 8048, который следит за нажатиями клавиш и передает их состояние в системный блок. При нажатии любой клавиши в системный блок посылается числовой код, который соответствует порядковому

номеру ее расположения на клавиатуре, и не связан с изображенным на клавише символом - код сканирования (скан-код).

Наиболее существенным с точки зрения программирования является то обстоятельство, что скэн-коды и коды, передаваемые в программу, - это разные коды.

Программа приема и обработки скэн-кодов (драйвер клавиатуры) входит в состав базовой системы ввода-вывода (BIOS) и размещается в ПЗУ. Драйвер клавиатуры осуществляет преобразование кода сканирования с учетом того, что могут быть одновременно нажаты две или более клавиш, например, клавиша верхнего регистра SHIFT и какая-нибудь алфавитно-цифровая клавиша. В результате такого преобразования на выходе драйвера клавиатуры формируется т.н. символьный код, который и воспринимается программой.

Драйвер клавиатуры преобразовывает скэн-код нажатой клавиши или комбинации клавиш в символьный код и заносит его в буфер клавиатуры. Каждой обычной клавише соответствует один байт, содержащий строго определенный символ.

При нажатии специальных клавиш или комбинаций клавиш (в частности, с управляющиими клавишами [Ctrl] или [Alt]) в буфер заносится т.н. расширенный код, состоящий уже из двух байтов, причем первый байт этого кода содержит 0. Это является признаком того, что нажата специальная клавиша или специальная комбинация клавиш.

Не все комбинации клавиш распознаются драйвером клавиатуры. Те, которые не являются стандартными, игнорируются, то есть, в буфер ничего не передается. Кроме этого, в буфере может храниться информация о 16 символах. Если из буфера символы не считываются процедурами Read, Readln, ReadKey, то после заполнения 16 символами остальные символы, набранные на клавиатуре, теряются, о чем свидетельствует специальный звуковой сигнал.

При построении символьного кода используется следующий принцип. Для всех обычных алфавитно-цифровых клавиш как в нижнем, так и в верхнем регистрах, а также клавиш Esc, Enter и некоторых комбинаций Ctrl+клавиша в драйвере формируется однобайтный числовой код. Этому коду соответствует некоторый символ, причем соответствие задается т.н. кодовой таблицей

(таблицей ASCII-кодов). Эта однозначная связь между числовым кодом и символом используется и в данном случае, так что драйвер передает в программу значение символьного типа. Поэтому выражение "символьный код" означает символ, которому в кодовой таблице сответствует числовой код.

При нажатии одной из специальных клавиш, таких как [F1]..[F10], клавиш направлений и других, а также комбинаций всех клавиш с модифицирующей клавишей [ALT] и большинства клавиш с управляющей клавишей [CTRL] драйвер вырабатывает код из так называемого расширенного набора. Этот код передается последовательностью двух символов, причем первый символ всегда имеет нулевой код. Это служит признаком того, что нажатая в данный момент клавиша или комбинация клавиш является специальной.

Приведем в качестве примера символьные коды наиболее часто используемых клавиш:

```
[Esc]    #27                [F1]     #0+#59
[Enter]  #13                [PgUp]   #0+#73
[->]     #0+#77             [PgDn]   #0+#81
[<-]     #0+#75             [Del]    #0+#83
```

На практике используются далеко не все коды и возможны не все комбинации клавиш. При вводе непредусмотренных комбинаций драйвер их просто игнорирует. Некоторые комбинации имеют специальное назначение: они не передаются на выход драйвера и, следовательно, не могут восприниматься программой, но обрабатываются операционной системой. В частности, комбинация [CTRL]-[BREAK] прекращает исполнение программы, [CTRL]-[ALT]-DEL] вызывает перезагрузку резидентной части DOS, [PrintScreen] приводит к копированию содержимого экрана на принтер.

Основная роль символьных кодов клавиатуры, получаемых программой, заключается в том, что код нажатой клавиши может быть проанализирован и, в зависимости от того, какая клавиша была нажата, в программе заранее может быть предусмотрено то или иное действие. На этом основано управление работой программы с помощью клавиатуры.

8.2 Встроенные функции для работы с клавиатурой

В языке Турбо-Паскаль для работы с клавиатурой реализованы две функции, находящиеся в модуле Crt. Функции не имеют параметров. Одна из них - функция опроса буфера клавиатуры - выдает логическое значение TRUE, если в буфере ввода с клавиатуры имеется хотя бы один символ (т.е. до обращения к функции была нажата какая-либо клавиша), и значение FALSE, если буфер пуст.

```
Function KeyPressed: Boolean;
```

Возвращает значение True, если в буфере есть хотя бы один символ, и False, если символов нет.

Применяют эту функцию для реализации режима "нажмите любую клавишу". При этом опрос клавиатуры должен происходить непрерывно, для чего организуется цикл Repeat..Until, например

```
GotoXY(35,24); Write('Нажмите любую клавишу');
Repeat Until KeyPressed;
```

Другая функция предназначена для чтения одного символа из буфера клавиатуры:

```
Function ReadKey: Char;
```

Считывает символ из буфера клавиатуры, при этом считанный символ на экране не отображается (в отличие от Read, Readln).

Чтобы воспользоваться ею, нужно присвоить результат ее работы некоторой символьной переменной, например:

```
var ch:char;
...
ch:=ReadKey;
```

Используя результат работы этой функции, можно определить клавишу, которая была нажата перед обращением к ней, и предусмотреть в программе действия, связанные с таким нажатием (тем самым нажатие клавиши будет играть роль введенной команды). Анализ осуществляется с помощью операторов выбора IF. При этом следует различать случаи однобайтовых и двухбайтовых символьных кодов.

Пример анализа однобайтного кода:

```
ch:=ReadKey;
If ch='Q' then ...{реакция на нажатие клавиши [Q]}
If ch=#13 then ...{     нажатие клавишиEnter}
...
```

Если таких клавишных команд достаточно много, реакцию на них удобнее программировать с использованием оператора CASE:

```
 c:=ReadKey;
 Case c of
  '3':  z:=100;  {выбор значения переменной}
  '4':  begin
          TextColor(14); GotoXY(70,1);
          Write(#7,'%%%%%%%');
        end;        {выполнение группы операторов}
  #65:  ClrScr;     {вызов процедуры}
 end;  {case}
 ...
```

Если в результате ch:=ReadKey; получено значение #0, то это означает, что нажималась какая-то из специальных клавиш, - либо функциональная, либо клавиша направления, и т.п. При этом первый символ всегда имеет нулевой код. Значит, следует выделить второй символ кода, еще раз применив функцию ReadKey и анализировать именно второй символ.

Пример анализа двухбайтного кода:

```
 ch:=ReadKey;     {читается первый символ кода}
 If ch=#0 then begin
  ch:=ReadKey;     {читается второй символ кода}
  If ch=#61 then ... {реакция на нажатие клавиши
[F1] }
  If ch=#78 then ... {реакция на нажатие клавиши [-
>] }
 end;
```

Таким образом, нажатие конкретной клавиши может привести к выполнению заранее запланированного действия программы, то есть, клавиша играет роль команды. Так как символы не отображаются на экране, реакция программы на их появление может быть самой разнообразной. Например, при нажатии клавиш

Alt-M можно запрограммировать или вывод нужного меню, или текстовое сообщение, или смещение графического изображения в центр экрана, или вызов процедуры, и т.п. Эти особенности работы с клавиатурой и лежат в основе организации широко используемого в профес-сиональных программах управления с помощью клавишных команд.

8.3 Программа для изучения символьных кодов клавиш

Для изучения символьных кодов наиболее часто используемых клавиш можно воспользоваться следующей программой, в которой, кроме того, при нажатии клавиш направлений выводится текстовое сообщение:

```
Program Use_key;
 uses Crt;
 var  Ch: Char;
Begin
 Repeat
  ClrScr;
  GotoXY(20,4);  Write('ПРОГРАММА РАСПОЗНАВАНИЯ
НАЖАТЫХ КЛАВИШ');
  GotoXY(30,24); Write('ESC ESC  - выход');
  GotoXY(28,7);  Write('Нажмите любую клавишу');
  Ch := Readkey;
  If ch<>#0 then begin
   GotoXY(20,10);
   Write('Вы нажали клавишу с однобайтовым кодом:
         [',Ch,']');
   GotoXY(20,11);
   Write('ей соответствует ASCII-код: ', Ord(Ch));
  end
  else begin
    Ch:= Readkey;
    GotoXY(20,10); Write('Вы нажали управляющую
                          клавишу или группу
клавиш');
    GotoXY(20,11);
    Write('ей соответствует ASCII-код: 0+',
Ord(Ch));
    GotoXY(10,16); Write(' ':40); GotoXY(10,16);
```

```
    Case ch of
      #75: Write('Нажата клавиша СТРЕЛКА ВЛЕВО');
      #77: Write('Нажата клавиша СТРЕЛКА ВПРАВО');
      #72: Write('Нажата клавиша СТРЕЛКА ВВЕРХ');
      #80: Write('Нажата клавиша СТРЕЛКА ВНИЗ');
   end;  {case}
  end;  {else}
  Repeat Until KeyPressed;
Until ch=#27;
End.
```

Вопросы для самопроверки

1. Технические особенности клавиатуры
2. Символьные коды клавиатуры и типы клавиш.
3. Функции опроса клавиатуры.
4. Выделение кодов символьных и управляющих клавиш.
5. Методы распознавания нажатых клавиш.

Задачи

Задача 1. На основе примера из п.3 с использованием процедур оформления диалога составить программу, анализирующую код нажатой на клавиатуре клавиши и выводящей его на экран. Указанные действия организовать в цикле Repeat..Until, выход из которого осуществить при нажатии клавиши Esc. Создать EXE-файл этой программы. С помощью программы получить символьные коды клавиш и заполнить таблицу:

Клавиша	Код Клавиши	Код Shift+ Клавиша	Код Ctrl+ Клавиша	Код Alt+ Клавиша
F1				
F10				
<-				
->				
Tab				
Esc				
Enter				
Home				
End				
PgUp				
PgDn				

Ins Del A Q X Пробел				

Задача 2. Составить программу, в которой в графическом режиме строится заданное изображение (параметры подобрать самостоятельно), и при нажатии указанной клавиши выполняется заданное действие.

а) В центре экрана изображен квадрат белого цвета. При нажатии Ctrl-C изменяется цвет квадрата. При нажатии F10 изменяется цвет фона.

б) Область экрана ограничена прямоугольником. При нажатии Alt-3 на экран выводится окружность, параметры которой заданы случайным образом. При нажатии Ctrl-F1 на экран выводится прямоугольник, параметры которого заданы случайным образом.

в) В центре экрана изображена окружность радиуса R=50. При нажатии клавиши Alt-R она сдвигается на 20 точек вправо, при нажатии клавиши Alt-L она сдвигается на 20 точек влево.

3. Дополнить программу Задачи 2 новой клавишной командой Alt-P, по которой прямоугольная область экрана, содержащая построенное изображение, выводится на принтер в режиме печати «в файл». Напечатать изображение на матричном принтере.

ТЕМА 9. ПРОГРАММНЫЕ ЭЛЕМЕНТЫ ИНТЕРАКТИВНОЙ ГРАФИКИ

9.1 Программирование клавишных команд перемещения курсора

9.2 Пример программы управления движением графического курсора

9.1 Программирование клавишных команд перемещения графического курсора

Точку на графическом экране будем обозначать специальным символом, называемым графическим курсором. Форма курсора не имеет особого значения, поэтому изобразим его в виде небольшого перекрестия. После запуска программы курсор изображается в центре рабочей области экрана. В ходе работы необходимо обеспечить возможность перемещения курсора в любое место на экране. Для этого используем четыре клавиши-стрелки (клавиши направлений). При перемещении курсор может наложиться на участок графического изображения. Чтобы при этом участок не был испорчен, курсор изображается оператором PutImage в режиме XOR. Это позволяет сохранить существующее изображение неповрежденным, если сначала на него наложить курсор, а затем его удалить.

При нажатии любой из клавиш направления курсор нужно передвинуть на несколько точек в указанном направлении. Это делается в три этапа:

1. выполняется оператор PutImage для удаления курсора в исходной позиции;

2. изменяется значение переменных x или y в зависимости от нажатой клавиши на величину текущего значения шага;

3. выполняется оператор PutImage для изображения курсора на новом месте.

Предусмотрим возможность изменения шага. Вначале для переменной dx,dy присваивается значение 2, которое означает, что при нажатии клавиши управления курсором последний перемещается на 2 точки в заданном направлении. Иногда желательно иметь более крупные шаги. Для этой цели

используем клавишу PgUp, PgDn. Нажатие клавиши PgUp приведет к увеличению шага на 1, PgDn - к уменьшению шага на 1. Однако нельзя допускать неограниченное увеличение шага, как и его отрицательных значений, так как тогда движение по рабочей области будет непредсказуемым. Будем полагать, что число 20 является максимально допустимым значением шага, а 1 - минимально допустимым. Аналогичные ограничения налагаются и на значения координат курсора xkurs, ykurs для предотвращения выхода курсора за границы экрана. Рабочая область должна быть несколько меньшей полного экрана. Например, нельзя задавать координаты курсора xkurs=0, поскольку это означало бы, что левая половина курсора должна выйти за границы экрана. Кроме клавиш направлений используем команду Alt-Q, подача которой будет означать завершение работы. Нажатие клавиши Esc при построении графического элемента будет означать отказ от построения и возврат в исходное состояние. Приведенная ниже программа выполняет все эти операции.

Как и большинстве интерактивных программ, центральной частью программы является цикл, в котором каждый раз считываются и обрабатываются символьные коды нажатых клавиш.

9.2 Пример программы управления движением графического курсора

Перечисленные возможности реализованы в программе управления курсором с помощью клавишных команд.

```
Program UPR_KEY1;
{ управление движением курсора
 с помощью клавишных команд (C) Дей Е.А.
-------------------------------------}
Uses   Graph,Crt;
Var
  Gd,Gm,ErrorCode:Integer;
  P:Pointer; Size:Word;   ch:char;
  x,y,dx,dy,xc,yc,xmin,xmax,ymin,ymax :integer;
  sx,sy,sdx:string;
Begin
 Gd := Detect;                    {установка
графического режима}
```

```
 InitGraph(Gd,Gm,'');
 if GraphResult <> grOK then Halt(1);
 SetBkColor(1);          {установка цвета фона}
 OutTextXY(250,40,'УПРАВЛЕНИЕ КУРСОРОМ');
 OutTextXY(220,60,'С ПОМОЩЬЮ КЛАВИШНЫХ КОМАНД');
 xmin:=80; xmax:=560; {-границы области изображения}
 ymin:=80; ymax:=360;
SetColor(11);
Rectangle(xmin,ymin,xmax,ymax);
xc:=xmin+((xmax-xmin) div 2);
yc:=ymin+((ymax-ymin) div 2);
x:=xc;  y:=yc;
dx:=4;  dy:=dx;
 SetColor(15);
 line(x-5,y,x+5,y);  {-построение курсора и запись в
память}
 line(x,y-5,x,y+5);
Size:=ImageSize(x-5,y-5,x+5,y+5);
GetMem(p,Size);
GetImage(x-5,y-5,x+5,y+5,p^);
PutImage(x-5,y-5,p^,XORPut);
SetFillStyle(1,1);
Repeat
OutTextXY(80,420,'Координаты  x:    y:
Шаг:
          <Esc>-Выход ');
Str(x,sx);  Str(y,sy);  Str(dx,sdx);
OutTextXY(170,430,sx+'   '+sy+'            '+sdx);
PutImage(x-5,y-5,p^,XORPut);
  ch:=Readkey;
  PutImage(x-5,y-5,p^,XORPut);
  if ch=#0 then begin
    ch:=Readkey;
    case ch of
      #75: x:=x-dx;
      #77: x:=x+dx;
      #72: y:=y-dy;
      #80: y:=y+dy;
      #73: begin dx:=dx+1; dy:=dy+1; end;  {PgUp}
      #81: begin dx:=dx-1; dy:=dy-1; end;  {PgDn}
    end;
  end; {if}
  if dx<=0 then begin dx:=1; dy:=1; end;
  if x<xmin+6 then x:=xmin+6;
```

```
  if x>xmax-6 then x:=xmax-6;
  if y<ymin+6 then y:=ymin+6;
  if y>ymax-6 then y:=ymax-6;
Bar(100,430,400,440);
Until ch=#27;
END.
```

Вопросы для самопроверки

1. Принцип работы интерактивных графических программ.

2. Организация перемещения курсора в рабочей области экрана.

3. Оформление экрана при реализации интерактивного ввода графических примитивов.

Задачи

Используя пример п.9.2 в качестве основы, разработать программу, выполняющую заданные действия в графическом режиме под управлением указанных дополнительных клавишных команд. На экране рядом с рабочей областью должен присутствовать список используемых клавишных команд и текущие значения рабочих параметров.

1) Редактор отрезков. На экране присутствует графический курсор. Система команд: фиксация 1-й точки: - Enter; фиксация 2-й точки: - Enter; изменение цвета линии: - Alt-1, Alt-2. После фиксации 2-й точки между точками проводится отрезок.

2) Редактор прямоугольников. По экрану передвигается графический курсор. При нажатии клавиши F1 появляется индикация режима "прямоугольник" и в рабочей области изображается прямоугольник предварительным размером 10*10 с центром в точке (x,y). В этом режиме нажатие клавиш F1,F2 приводит к увеличению, уменьшению горизонтальной стороны прямоугольника, а нажатие клавиш F3,F4 - к изменению вертикальной стороны прямоугольника. Клавиша Enter фиксирует выбранный прямоугольник и отключает режим.

3) Редактор окружностей. Аналогично п.2, но для окружности, с регулированием ее радиуса.

4) Редактор эллипсов. Аналогично п.2, но для эллипса, с регулированием коэффициента эллиптичности.

5) Редактор ломаных линий. Аналогично п.1, но конечная точка построенного отрезка считается начальной для следующего отрезка. Командами Alt-1, Alt-2, Alt-0 можно переключать цвет линий.

6) Редактор треугольников. Аналогично п.1 и п.5.

7) Конструктор RC-схем. Клавишные команды служат для изображения следующих объектов:

Alt-C - условное обозначение конденсатора (горизонтально);

Alt-V - условное обозначение конденсатора (вертикально);

Alt-R - условное обозначение резистора (горизонтально);

Alt-T - условное обозначение резистора (вертикально);

Alt-E - условное обозначение источника тока;

Alt-P - условное обозначение точки соединения.

8) Набор текста в графическом режиме. Курсор указывает, в каком месте экрана появится символ, соответствующий нажатой символьной клавише.

9) Редактор продольно-поперечных линий. При движении курсора по горизонтали или вертикали по его следу рисуется линия. Командами Alt-1, Alt-2, Alt-0 можно переключать цвет линии. Клавиша F5, F6 включает, отключает рисование линии.

10) Редактор цветных полос. При движении курсора по его следу рисуется полоса шириной h. Командами Alt-1, Alt-2, Alt-0 можно переключать цвет полосы. Клавиша F5, F6 включает, отключает рисование полосы. Клавиши Ctrl-A, Ctrl-Z увеличивают, уменьшают ширину полосы.

ТЕМА 10. ФАЙЛОВЫЙ ФОРМАТ ГРАФИЧЕСКИХ ДАННЫХ «.BMP»

10.1 Заголовок и структура графических данных формата BMP

10.2 Получение информации о параметрах BMP-файла

10.3 Преобразование палитры к графическому режиму и к BMP-формату

10.4. Программирование записи изображения в формате BMP с использованием функции GetPixel

10.1 Заголовок и структура графических данных формата BMP

При изучении графических средств языка Турбо-Паскаль [1-5] важно не только правильно составить программу, формирующую изображение, но и сохранить полученную картинку для последующего использования, например, в тексте отчета. В настоящее время разработан ряд файловых форматов графических данных, реализующих эффективные и достаточно сложные алгоритмы сжатия информации [6]. Существует и сравнительно простой и распространенный файловый формат - формат BMP, широко используемый в Windows-приложениях [7].

В данном параграфе изложены основные элементы и принципы формирования формата BMP. Приведены примеры процедур, выполняющих, при вызове их из самой программы, запись нужного участка графического экрана в файле в 16-цветном формате BMP, а также вывод 16-цветного изображения из BMP-файла на экран. Использован более наглядный и простой для изучения, хотя и не самый быстрый, способ обработки отдельных пикселов изображения - с помощью функции GetPixel и процедуры PutPixel.

Файл в формате BMP является нетипизированным и состоит из заголовка стандартной структуры и последовательности байтов, хранящих информацию об изображении в прямоугольной области экрана.

Информация об изображении, записанная в формате BMP, представляет собой подробное перечисление цвета каждого

пиксела прямоугольного фрагмента изображения, причем количество битов, описывающих пиксел, минимально и зависит от количества используемых цветов. Для монохромного изображения состояние каждой точки графического экрана фиксируется с помощью одного бита (цвет или фон), так что участку строки из 8 последовательных пикселов в файле BMP будет соответствовать всего один байт. Для 16-цветных изображений на каждый пиксел будет отводиться 4 бита, то есть, один байт на два соседних пиксела строки, для 256-цветных – 8 бит (один байт) на пиксел.

Графические данные записываются в файл последовательно, причем первой строкой изображения считается нижняя строка экранной области. Количество байтов, соответствующих строке изображения, дополняется нулями до числа, кратного 4 (выравнивание по 32-битной границе). Этот дополненный набор байтов называется файловым образом изображения.

Заголовок BMP-файла состоит из трех частей: главная, информационная и таблица цветов. Первая часть заголовка - структура BITMAPFILEHEADER, которой в языке Турбо-Паскаль соответствует определение типа:

```
Type BITMAPFILEHEADER=record          {14 байт}
  bfType:word;                  {тип файла BM = 19778}
  bfSize:LongInt;               {размер файла, байт}
  bfReserved1,bfReserved2:word; {не используется}
  bfOffBits:LongInt;    {размер заголовка в байтах}
end;
```

Непосредственно за ней располагается информационная структура BITMAPINFOHEADER, описывающая параметры изображения:

```
Type BITMAPINFOHEADER=record    {40 байт}
 biSize:LongInt;     {размер структуры
BITMAPINFOHEADER}
 biWidth:LongInt;    {ширина изображения, пиксел}
 biHeight:LongInt;  {высота изображения, пиксел}
 biPlanes:word;     {число битовых плоскостей
формата}
 biBitCount:word;    {число битов на пиксель}
 biCompression:LongInt;    {тип сжатия}
 biSizeImage:LongInt;    {размер BMP-образа, байт}
```

```
 biXPelsPerMeter:LongInt; {горизонтальное
                      разрешение устр-ва, пиксел/м}
 biYPelsPerMeter:LongInt; {вертикальное
                      разрешение устр-ва, пиксел/м}
 biClrUsed:LongInt;    {число используемых цветов}
 biClrImportant:LongInt;  {число "важных" цветов}
end;
```

Рассмотрим более подробно назначение некоторых параметров информационной части:

- biPlanes - определяет число битовых плоскостей формата BMP. Поскольку цвет кодируется последовательными битами, это число всегда равно 1.
- biBitCount - указывает, сколько битов используется для описания каждого пиксела изображения в BMP-образе: 1 - монохромный образ (таблица цветов содержит два элемента); 4 - максимум 16 цветов, таблица цветов имеет до 16 элементов; 8 - максимум 256 цветов, таблица цветов имеет до 256 элементов.
- biCompression – вариант сжатия на основе подсчета групп повторяющихся байтов. На практике сжатие в BMP-формате используется крайне редко.
- biXPelsPerMeter и biYPelsPerMeter - позволяют выбрать пиктограмму, наиболее соответствующую данному устройству; могут быть заданы как 0;
- biClrUsed - число цветов, используемых данным битовым образом. Если 0, то используются все цвета, указанные в таблице цветов.
- biClrImportant – число важных цветов, используется при выводе нескольких картинок в Windows.

Вслед за описанием параметров битового образа располагается таблица цветов - массив, состоящий из элементов типа

```
Type RGBQUAD=record      {4 байта}
  rgbBlue:byte;      {интенсивность синего}
  rgbGreen:byte;     {интенсивность зеленого}
  rgbRed:byte;       {интенсивность красного}
  rgbReserved:byte; {не используется}
end;
```

Значения элементов для каждого цвета указывают интенсивность r, g, b – составляющих (0..255) и таким образом определяют цветовую палитру. Количество элементов соответствует количеству используемых цветов.

Перечисленные определения типов и описание переменных, имеющих эти типы, должны присутствовать в любой процедуре, работающей с форматом BMP, например, в таком виде:

```
Type                    {Фрагмент 1}
  BITMAPFILEHEADER= . . .
  BITMAPINFOHEADER= . . .
  RGBQUAD= . . .
  PaletteBMP16=array[0..15] of RGBQUAD;
Var
  Head:BITMAPFILEHEADER;
  Info:BITMAPINFOHEADER;
  PaletBMP: PaletteBMP16;
  fbmp:file;
```

Если в программе несколько таких процедур, определения следует сделать глобальными, расположив их в описательной части программы.

10.2 Получение информации о параметрах BMP-файла

В целях изучения структуры и возможных вариантов существующих BMP-файлов полезно составить программу, выводящую на экран информацию о параметрах заданного файла. Имя файла вводится с клавиатуры. Программа может иметь следующий вид (описание типов приведено выше):

```
PROGRAM HEAD_BMP; {чтение заголовка BMP-файла}
Uses Graph;
(* .. Фрагмент 1 .. *)
Var   i,cv:integer;  fname:string;
Begin
 Writeln('Введите имя файла формата BMP: ');
 Readln(fname);
 If Pos('.bmp',fname)=0 then fname:=fname+'.bmp';
 Assign(fbmp,fname);   Reset(fbmp,1);
 BlockRead(fbmp,Head,SizeOf(Head));
```

```
 BlockRead(fbmp,Info,SizeOf(Info));
With Head do begin
  Writeln('Тип файла: ',bfType);
  Writeln('Размер файла: ',bfSize,' байт');
  Writeln('Длина заголовка: ',bfOffBits,' байт');
end;
With Info do begin
  Writeln('Размер Info-части: ',biSize,' байт');
  Writeln('Ширина изображения: ',biWidth,' пиксел');
  Writeln('Высота изображения: ',biHeight,'
пиксел');
  Writeln('Число битов на пиксел: ',biBitCount);
  Writeln('Тип сжатия: ',biCompression);
  Writeln('Число используемых цветов: '
            ,biClrUsed);
  If biClrUsed=0 then cv:=16 else cv:=biClrUsed;
end;
readln;
If Info.biBitCount=4 then begin
  BlockRead(fbmp,PaletBMP,SizeOf(RGBQUAD)*cv);
  Writeln('Палитра, записанная в файле: ');
  Writeln('      N       R     G    B');
  Writeln('-------------------------');
  For i:=0 to cv-1 do begin
    Write('цвет ',i:2,':    ');
    With PaletBMP[i] do
    Writeln(rgbRed:3,'  ',rgbGreen:3,'  '
                              ,rgbBlue:3);
  end;
  Readln;
end;
Close(fbmp);
End.
```

Несложно дополнить этот пример операторами вывода и других параметров файла.

10.3 Преобразование палитры к графическому режиму и к BMP-формату

При записи заголовка BMP-файла необходимо преобразовать информацию о программной палитре от формы, в которой она

хранится в графических режимах EGA/VGA, к форме, которая принята в BMP-формате.

Как известно [1-5], для 16-цветных режимов EGA/VGA каждый из активных цветов палитры кодируется одним байтом формата 00rgbRGB, где R, G, B - младшие биты, определяющие вклад 2/3 интенсивности красного, зеленого и синего цветов, а r, g, b - биты, определяющие вклад 1/3 интенсивности [5]. Таким образом, для каждого основного цвета могут быть установлены 4 уровня: 0, 1, 2 и 3. В языке Турбо-Паскаль для хранения информации о цветах палитры используется тип PaletteType, определенный следующим образом:

```
  Type  PaletteType=record
    Size:byte; Colors:array[0..MaxColors] of
ShortInt;
  end;
```

Поле Size определяет количество цветов в палитре, а массив Colors содержит EGA/VGA-коды действующих цветов палитры. Информацию о программной палитре можно получить с помощью процедуры GetPalette(var ProgrPalette:PaletteType).

Как уже отмечалось, в формате BMP коды цветов запоминаются в массиве элементов типа RGBQUAD со значениями интенсивности красного (rgbRed), зеленого (rgbGreen) и синего (rgbBlue) цветов. Значения цветовых компонент могут находиться в диапазоне от 0 до 255. На практике это означает необходимость установить соответствие:

Уровень EGA/VGA	0	1	2	3
Значение BMP	0	85	170	255

Для преобразования информации от формы 00rgbRGB необходимо проверить значение битов в байтах, выдаваемых процедурой GetPalette, с помощью операций битовой арифметики.

Для определенности будем записывать в BMP-файл всю палитру графического режима. В этом случае размер заголовка составит 118 байт. Текст соответствующей процедуры может быть, например, таким:

```
Procedure CodePaletteToBMP16(var
palet:paletteBMP16);
var   pal: PaletteType;
      cv, i, blue, green, red: byte;
begin
 GetPalette(Pal);
```

```
 for i:=0 to pal.size-1 do begin
  cv:=pal.colors[i];  {-код i-го цвета}
  blue:= ((cv AND 8)  DIV 8)+(cv AND 1)*2; {0..3}
  green:=((cv AND 16) DIV 16)+(cv AND 2) ;
  red:=  ((cv AND 32) DIV 32)+((cv AND 4) DIV 2);
  with palet[i] do begin
    rgbBlue:=  blue*85;    rgbGreen:=green*85;
    rgbRed:=    red*85;    rgbReserved:=0;
  end;
 end; {for i}
end; {CodePaletteToBMP16}
```

Соответственно, при чтении BMP-файла понадобится преобразовать информацию об используемых цветах к палитре графического режима. Для этого можно использовать следующую процедуру (параметр Nc – количество используемых цветов):

```
Procedure GetPaletteFromBMP16(var pal:paletteBMP16;
Nc: byte);
var   palet:PaletteType;   cv:shortint;
      blue, green, red,i:byte;
begin
 for i:=0 to Nc-1 do begin
  red:=  pal[i].rgbRed   DIV 85;     {-пересчет}
  green:=pal[i].rgbGreen DIV 85;     {к диапазону}
  blue:= pal[i].rgbBlue  DIV 85;     {0..3}
  cv:=0;                {-установка нужных битов:}
  cv:=cv+(blue  MOD 2)*8  + (blue  DIV 2);
  cv:=cv+(green MOD 2)*16 + (green DIV 2)*2;
  cv:=cv+(red   MOD 2)*32 + (red   DIV 2)*4;
  palet.colors[i]:=cv;
  SetPalette(i,cv);
 end; {for i}
end; {GetPaletteFromBMP16}
```

При выводе на экран файлов, созданных без учета 4-уровневой градации основных цветов, возможны некоторые изменения цветопередачи.

10.4. Программирование записи изображения в формате BMP с использованием функции GetPixel

На основе сделанного описания составим процедуру, записывающую в файл формата BMP прямоугольный участок экрана. Информацию о цвете каждой точки экрана будем получать с помощью функции GetPixel.

При заданных координатах (x1,y1)-(x2,y2) в строке изображения будет (x2-x1+1) пикселов. Для дополнения до числа, кратного 8, введем переменную nbytes. Тогда для каждой графической строки, начиная с y2, в BMP-файл будет записано 4*nbytes байт.

Запись данных в файл будем производить участками по 40 строк, с текущими номерами от yb до ya, записывая все байты участка во вспомогательный массив bmpBytes. Для текущей строки с номером y организуем цикл по парам точек, формирующим один байт данных (used - число полных пар). Переменная L фиксирует это число байтов, соответствующих реальным точкам экрана.

При этом 4-битовый код цвета левой точки в каждой паре умножением на 16 заносится в старшие биты байта данных, а код цвета правой точки сложением – в младшие биты. Например, если первый пиксел имеет 14-й цвет (1110), а второй пиксел 9-й цвет (1001), то байт данных имеет значение 14*16+9=233 (11101001).

Возможен случай, когда число реально имеющихся в строке пикселов x2-x1+1 нечетное. Тогда по оставшемуся пикселу формируется еще один байт данных, а переменная L увеличивается на 1. Наконец, в BMP-файл при необходимости следует записать нулевые байты, чтобы их полное число в строке данных было кратно 4. Так, строке изображения, пикселы которой имеют цвет 6, 14, 11, 1, 12, 2 будет соответствовать (дополненная) последовательность байтов в BMP-файле: 110, 177, 194, 0.

С учетом сделанных пояснений можно предложить следующий текст процедуры GetPixelToBMP16 для записи прямоугольного фрагмента 16-цветного изображения в файл формата BMP

```
Procedure GetPixelToBMP16(x1,y1,x2,y2:integer;
name:string);
```

```
 {Запись 16-цветного изображения в формате BMP с
использованием   функции GetPixel (C)Дей Е.А.}
Var
  BmpBytes:array[1..12800] of byte;
  ya,yb,y,nbytes,L,nx,used,i,bmpNom:integer;
  FSize:longint;
Begin
  nbytes:=(x2-x1+1) div 8;     {групп по 8 пиксел}
  If ((x2-x1+1) mod 8)<>0 then nbytes:=nbytes+1;
                               {-кратно 8}
   FSize:= nbytes*4*word(y2-y1+1)+118;
  With Head do begin
    bfType:=19778;   bfSize:=FSize;
bfReserved1:=0;
    bfReserved2:=0;  bfOffBits:=118;
  end;
  With Info do begin
    biSize:=40;   biWidth:=x2-x1+1;  biHeight:=y2-
y1+1;
    biPlanes:=1;  biBitCount:=4;
biCompression:=0;
    biSizeImage:=FSize-118;
    biXPelsPerMeter:=0;  biYPelsPerMeter:=0;
    biClrUsed:=0;        biClrImportant:=0;
  end;
 CodePaletteToBMP16(paletBMP);

 Assign(fbmp,name);  Rewrite(fbmp,1);
 BlockWrite(fbmp,Head,SizeOf(Head));
 BlockWrite(fbmp,Info,SizeOf(Info));
 BlockWrite(fbmp,PaletBMP,SizeOf(PaletBMP));
used:=((x2-x1+1) div 2);   {- полных пар пикселов}
yb:=y2;                    {-по 40 строк}
Repeat
  ya:=yb-39;  If ya<y1 then ya:=y1;
   bmpNom:=0;                {номер байта данных}
 For y:=yb downto ya do begin
     For nx:=1 to used do begin
       bmpNom:=bmpNom+1;
       BmpBytes[bmpNom]:=16*GetPixel(x1+2*nx-2,y);
       BmpBytes[bmpNom]:=
         BmpBytes[bmpNom]+GetPixel(x1+2*nx-1,y);
     end;
     L:=used;
```

```
    If Odd(x2-x1+1) then begin {-одиночный пиксел}
       bmpNom:=bmpNom+1;
       BmpBytes[bmpNom]:=16*GetPixel(x2,y);
       L:=used+1;
     end;
   For i:=1 to nbytes*4-L do begin {-дополн. до
                                    32-битной границы}
     bmpNom:=bmpNom+1;  BmpBytes[bmpNom]:=0;
   end;
 end; {For y}
   BlockWrite(fbmp,bmpBytes,bmpNom);
   yb:=ya-1;
 Until yb<y1;
 Close(fbmp);
end; {GetPixelToBMP16}
```

Порядок использования процедуры: 1) установить графический режим; 2) построить изображение на экране; 3) вызвать процедуру, указав в качестве параметров границы участка, записываемого в файл, и имя файла. Запись полного экрана 640*480, 16 цветов в BMP-файл может занять несколько секунд. Размер файла - 153718 байт.

Вопросы для самопроверки

1. Заголовок формата BMP.
2. Описание палитры в формате BMP.
3. Алгоритм кодирования графической информации в формате BMP.

Задачи

1. Составить процедуру для записи монохромной версии изображения в формат BMP.
2. Составить процедуру для чтения файла BMP и вывода изображения на экран.
3. Использовать процедуру записи изображения в файл формата bmp для сохранения состояния в рабочей области в интерактивной графической программе по команде Alt-S.

ЛИТЕРАТУРА

1. Вальвачев А.Н. Графическое программирование на языке Паскаль: Справ. gособие – Мн.: Выш. шк., 1992. – 143 с.

2. Епанешников А.М., Епанешников В.А. Программирование в среде Турбо Паскаль 7.0.– 3-е изд.– М.: Диалог-МИФИ, 1996. –288 с.

3. Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль. - М.: Изд-во МАИ, 1992. - 576 с.

4. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Уч. пособие. Изд. 7-е. – М.: Изд-во «Нолидж», 2001. – 576 с.

5. Фаронов В.В. Турбо Паскаль 7.0. Практика программирования. Уч. пособие. Изд. 7-е. – М.: Изд-во «Нолидж», 2001. – 416 с.

6. Дей Е.А. Разработка программно-методического обеспечения вычислительного практикума. – Деп. БелИСА. 17.09.2003, № Д200374. – 35 с.

7. Дей Е.А. Байты и биты формата BMP / Информатизация образования. 2004. № 3. с. 29-40.

УЧЕБНОЕ ИЗДАНИЕ

ДЕЙ Евгений Александрович

ИЗУЧЕНИЕ И ПРИМЕНЕНИЕ ГРАФИЧЕСКИХ СРЕДСТВ ЯЗЫКА ТУРБО-ПАСКАЛЬ

ПРАКТИЧЕСКОЕ ПОСОБИЕ
для студентов
специальности 1- 31 04 01 – 02 – «Физика»

В авторской редакции

Лицензия ЛИ № 02330/0133208 от 30.04.03.

Подписано в печать ____ Формат 60x84 1/16. Бумага писчая №1.
Печать на ризографе. Усл.-печ.л.5,6. Уч-изд.л. 5,9. Тираж 50 экз.

Учреждение образования «Гомельский государственный университет имени Франциска Скорины»,
246019, г. Гомель, ул. Советская, 104.

Отпечатано в учреждении образования «Гомельский государственный университет имени Франциска Скорины»,
246019, г. Гомель, ул. Советская, 104.