автор неизвестен

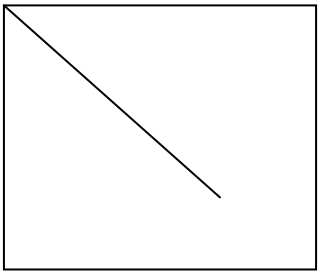
Графика на Паскале

В языке TURBO Pascal имеется значительное количество графических процедур и функций. Нам понадобятся лишь некоторые из них. Для того, чтобы компилятор "узнавал" их названия, мы должны после заголовка программы разместить строчку следующего вида: uses Graph; (что в переводе на русский означает "используется графика").

До сих пор во время нашей работы за компьютером экран всегда находился в текстовом режиме, поэтому на экране можно было видеть только лишь символы (правда, включая такие изыски, как так называемые "символы псевдографики"). Для рисования прямых, окружностей и пр. необходимо перевести экран в графический режим. Для включения графического режима используется процедура InitGraph. Простейшая программа может иметь вид:

Пример 1.

```
PROGRAM  Primer_1;
  uses Graph;
  var  Gd,Gm: Integer;
BEGIN
  Gd:=VGA;   { Графический адаптер - VGA        }
  Gm:=VGAHi; { Графический режим VGAHi (640x480) }
  InitGraph (Gd,Gm,'..\bgi'); { Включить графический режим }
  If  GraphResult=grOk
    then { Если режим включился успешно }
       { Нарисовать отрезок прямой }
       Line (0,0,639,479); ReadLn
END.
```



Мы видим, что у процедуры InitGraph три параметра. В качестве первых двух фактических параметров должны стоять имена целых (integer) переменных. Не будем вдаваться в подробности, почему это так; вместо этого выясним их предназначение.

Первый параметр является кодом графического адаптера (т.е. электронной схемы, управляющей выводом информации на экран). (Дело в том, что на IBM-совместимых компьютерах применяется ряд стандартных графических адаптеров, носящих названия CGA, EGA, VGA.) По нашей программе можно догадаться, что в используемых нами компьютерах используется адаптер VGA, и компилятор сам "узнает" слово VGA и заменит его на нужное целое число (на самом деле равное 9, но мы этого можем и не запоминать). Каждый графический адаптер позволяет использовать несколько графических режимов, отличающихся количеством цветов и разрешающей способностью (в дальнейшем мы узнаем, что это такое). И второй из параметров как раз предназначен для того, чтобы указать, какой из графических режимов следует включить. Пока что мы ограничимся лишь одним графическим режимом VGAHi. Третий параметр является строкой, содержащей путь к файлу, который называется EGAVGA.BGI. В этом файле содержится драйвер (такая специальная программа), необходимый для работы с адаптерами EGA и VGA. И, как легко увидеть из нашего примера, файл этот находится в подкаталоге TPBGI.

Все вышеизложенное необходимо знать каждому грамотному пользователю IBM-совместимых компьютеров. Однако в нашей лабораторной работе достаточно использовать конструкцию, использованную в первом примере, для включения графического режима. (И не страшно, если в ней не все понятно.) Для того, чтобы мы могли что-либо нарисовать на экране, нам нужно уметь задавать положение на экране того, что мы рисуем. Для этого с экраном связывается система координат следующего вида:

```
(0,0)                              X
  + ------------------------------------------->
  |
  |
  |
Y|
```

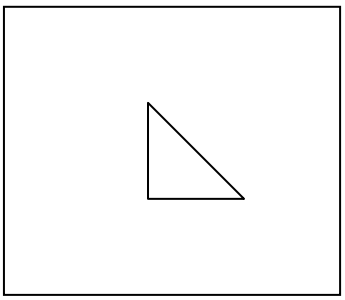
Каждая точка на экране на самом деле представляет собой очень маленький прямоугольник (и поскольку это не совсем точка, то иногда используют специальный термин - "пиксел"). Количество точек (пикселов), умещающихся на экране по вертикали и горизонтали, называют разрешающей способностью. Разрешающая способность экрана в режиме VGAHi - 640x480. Это означает, что по горизонтали на экране умещается 640 точек, а по вертикали - 480.

Точка в левом верхнем углу экрана имеет координаты (0,0). Координата X любой точки экрана лежит в пределах от 0 до 639, а координата Y - в пределах от 0 до 479. Как Вы уже догадались, процедура Line (x1,y1,x2,y2) рисует на экране прямую, соединяющую точки (x1,y1) и (x2,y2).

Пример 2.

Изобразить на экране прямоугольный треугольник с вершинами (320, 10), (120,210), (520,210).

```
PROGRAM  Primer_2;
  uses Graph;
  var  Gd,Gm: Integer;
BEGIN
  Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');
  If  GraphResult=grOk
    then  begin
            Line (120,210,520,210); { Горизонтальный отрезок }
            Line (120,210,320,10);  { Левый катет           }
            Line (320,10,520,210);  { Правый катет          }
            ReadLn
          end;
END.
```



Довольно обидно работать на цветном мониторе только с черно-белыми

изображениями. Для задания цвета рисования прямых, окружностей, точек и пр. используется процедура SetColor. В качестве единственного ее параметра нужно указать целое число, являющееся кодом цвета. Цвета кодируются в соответствии со следующей лькодовой таблицейль: Black=0-черныйDarkGray=8-темно-серый
Blue=1-синийLightBlue=9-голубо
Green=2-зеленыйLightGreen=10- светло-зеленый
Cyan=3-цвет морской волныLightCyan=11-светло-цианový
Red=4-красныйLightRed=12-розовый
Magenta=5-сиреневыйLightMagenta=13-светлосиреневый
Brown=6-коричневыйYellow=14- желтый
LightGray=7-светло-серыйWhite=15-белый

Если Вы хорошо знаете английский язык, то Вам будет удобнее использовать не числа, а соответствующие им идентификаторы; если же Вы английский знаете плохо, то все равно полезнее запомнить английские названия цветов, чем запоминать числа, кодирующие цвета.

Пример 3.

Изобразить тот же треугольник, что и в предыдущем примере, но сделать стороны треугольника разноцветным.

```
PROGRAM  Primer_3;
  uses Graph;
  var  Gd,Gm: Integer;
BEGIN
  Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');
  If  GraphResult=grOk
    then begin
          SetColor (LightMagenta); { Цвет - светло-сиреневый }
          Line (120,210,520,210);  { Горизонтальный отрезок }
          SetColor (LightCyan);    { Цвет - светло-циановый }
          Line (120,210,320,10);   { Левый катет           }
          Set Color (Green);       { Цвет - зеленый        }
          Line (320,10,520,210);   { Правый катет          }
          ReadLn
        end
END.
```

Пример 4

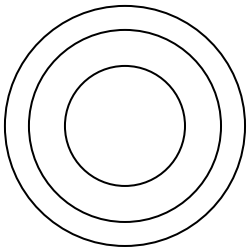
Разноцветные лучи.

```
PROGRAM  Primer_4;
  uses  Graph;
  const CenterX=320;
        CenterY=240;
        Radius=200;
  var   Gd,Gm: Integer;
        i    : Integer;
        dx,dy: Integer;
BEGIN
  Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');
  If  GraphResult=grOk
    then  begin
            For i:=0 to 160  do
              begin
                dx:=Round (Radius*sin(i*pi/80));
                dy:=Round (Radius*cos(i*pi/80));
                SetColor (i MOD 16);
                Line(CenterX,CenterY,CenterX+dx,CenterY+dy)
              end;
            ReadLn
          end
END.
```

Пример 5

Концентрические окружности.
Для рисования окружностей используется процедура Circle с тремя целочисленными параметрами, задающими координаты центра окружности и радиус.

```
PROGRAM  Primer_5;
   uses Graph;
   const CenterX=320;
         CenterY=240;
   var   Gd,Gm: Integer;
         i    : Integer;
BEGIN
   Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');
   If  GraphResult=grOk
     then  begin
             For i:=0 to 23  do
               Circle (CenterX,CenterY,i*10);
             ReadLn
           end
END.
```



Пример 6

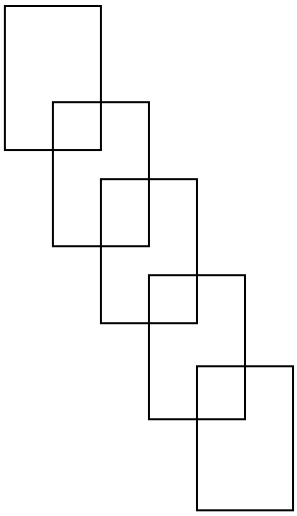
Разноцветные концентрические окружности.

Для закраски замкнутой области используется процедура FloodFill, три целочисленных параметра которой задают начальную точку закраски и код цвета ограничивающей область линии. Цвет, которым будет производиться закраска, ничего общего не имеет с цветом, задаваемым процедурой SetColor. Цвет закраски задается вторым параметром процедуры SetFillStyle. Первый параметр этой процедуры (задающий узор для закраски) на первых порах будем задавать равным единице (что соответствует сплошной закраске).

```
PROGRAM  Primer_6;
  uses Graph;
  const CX=320;
       CY=240;
  var   Gd,Gm: Integer;
       i    : Integer;
BEGIN
  Gd:=VGA; Gm:=VGAHi; InitGraph (Gd,Gm,'..\bgi');
  If  GraphResult=grOk
    then  begin
            For i:=0 to 23  do
              Circle (CX,CY,i*10);
            For i:=0 to 23  do
              begin
                SetFillStyle (1,i MOD 16);
                { Закрашивать до границы белого цвета }
                FloodFill (CX,CY+i*10-5,White)
              end;
            Readln
          end
END.
```

Пример 7.

```
PROGRAM  Primer_7;
  uses Graph;
  var  grDriver: Integer;
       grMode  : Integer;
       i,x,y   : Integer;
{ ---------------------------------- }
PROCEDURE  Rect (x,y,x1,y1: Integer);
{ Рисует прямоугольник, у которого левый нижний угол }
{ имеет координаты (x,y), а правый верхний - (x1,y1) }
BEGIN
  Line (x,y,x,y1);   { Левая сторона   }
  Line (x1,y,x1,y1); { Правая сторона  }
  Line (x,y1,x1,y1); { Верхняя сторона }
  Line (x,y,x1,y)    { Нижняя сторона  }
END;
{ --- }
BEGIN
  GrDriver:=VGA; GrMode:=VGAHi;
  InitGraph (grDriver,grMode,'..\bgi');
  If  GraphResult=grOk
    then  begin
            For i:=1 to 15  do
              begin
                x:=i*30; y:=i*25; SetColor (i);
                Rect (x,y,x+50,y+60)
              end;
            ReadLn
          end
END.
```



Рисование прямоугольников - часто встречающаяся проблема, и поэтому неудивительно, что существует стандартная процедура, работающая так же, как и созданная нами ниже процедура Rect. Она называется Rectangle.

Пример 8.

Для рисования "заполненных" прямоугольников используется процедура Bar. Так же, как и для процедуры Rectangle, мы должны указать четыре числа - координаты двух противоположных углов прямоугольника. (Для процедуры Bar цвет задается не с помощью SetColor, а с помощью SetFillStyle!).

```
PROGRAM  Primer_8;
  uses Graph;
  const Step=35;
  var   grDriver: Integer;
        grMode  : Integer;
        i,x,y   : Integer;
{ ------------------------------- }
PROCEDURE  Square (x,y: Integer);
{ Рисует цветастый квадрат, центр которого }
{ имеет координаты (x,y)                   }
   var  i,d: Integer;
BEGIN
  For i:=15 downto 0  do
    begin
      SetFillStyle (SolidFill,i); d:=i*3+2;
      Bar (x-d,y-d,x+d,y+d)
    end
END;
{ --- }
BEGIN
  GrDriver:=VGA; grMode:=VGAHi;
  InitGraph (grDriver,grMode,'..\bgi');
  If  GraphResult=grOk
    then  begin
            For i:=0 to 10  do
              { На "побочной" диагонали - 11 точек }
              begin
                x:=50+i*Step; y:=50+(10-i)*Step;
                SetColor(i); Square(x,y)
              end;
            ReadLn
          end
END.
```

При рисовании сложных изображений, содержащих много отрезков, возникает довольно противная проблема - вычислять координаты всех точек. Если использовать процедуру LineRel, то достаточно указывать смещения по обеим координатам относительно текущей точки. Для относительного перемещения без рисования используется процедура MoveRel Для задания начальных значений координат текущей точки используется процедура MoveTo

Пример 9.

Квадратная спираль.

```
PROGRAM  Primer_9;
  uses Graph;
  const CenterX=320;
        CenterY=240;
        d=12;
  var   grDriver: Integer;
        grMode  : Integer;
        i,L     : Integer;
{ ------------------------------- }
PROCEDURE  Vitok (L,dL: Integer);
{ Начиная от текущей точки, рисует виток спирали          }
{ из четырех отрезков увеличивающейся длины               }
{ L - длина первого отрезка                               }
{ dL - приращение длины для каждого из следующих отрезков }
BEGIN
  LineRel (L,0);           { Сдвинуться вправо }
  LineRel (0,-(L+dL));     { Сдвинуться вверх  }
  LineRel (-(L+2*dL),0);   { Сдвинуться влево  }
  LineRel (0,L+3*dL);      { Сдвинуться вниз   }
END;
{ --- }
BEGIN
  grDriver:=VGA; grMode:=VGAHi;
  InitGraph (grDriver,grMode,'..\bgi');
  If  GraphResult=grOk
    then  begin
            { Сделать текущей точкой центр экрана }
            MoveTo (CenterX,CenterY);
            L:=1;   { Начальная длина отрезка }

            For i:=1 to 10  do  { 10 витков спирали }
              begin  Vitok (L,d); L:=L+4*d  end;
            ReadLn
          end
END.
```

Пример 10.

Небольшой городок.

```
PROGRAM  Domiki;
  uses Graph;
  var  grDriver: Integer;
       grMode  : Integer;
       i,j     : Integer;
{ ------------------------------ }
PROCEDURE  Domik (x,y: Integer);
{ Рисует домик, у которого левый нижний угол }
{ имеет координаты (x,y)                     }
  const dx=60;     { Ширина фасада }
        dy=40;     { Высота фасада }
        dx2=dx DIV 2;
        dy2=dy DIV 2;
        wx=16;     { Ширина окна }
        wy=22;     { Высота окна }
        wx2=wx DIV 2;
        wy2=wy DIV 2;
BEGIN
  Rectangle (x,y,x+dx,y-dy); MoveTo (x,y-dy);
  Linerel (dx2,-dx2);  { Левый скат крыши }
  Linerel (dx2,dx2);   { Левый скат крыши }
  Rectangle (x+dx2-wx2,y-dy2-wy2,x+dx2+wx2,y-dy2+wy2);  { Окно }
  MoveTo (x+dx2,y-dy2);       { Центр фасада (и окна)          }
  LineRel (0,wy2);            { Вертикальная часть рамы окна   }
  MoveTo (x+dx2-wx2,y-dy2);   { Центр левой части рамы окна    }
  LineRel (wx,0);             { Горизонтальная часть рамы окна }
  SetFillStyle (SolidFill,Red);
  FloodFill (x+1,y-1,White);
  SetFillStyle (SolidFill,Blue);
  FloodFill (x+dx2,y-dy-1,White)
END;
{ --- }
BEGIN
  grDriver:=VGA; grMode:=VGAHi;
  InitGraph (grDriver,grMode,'..\bgi');
  If  GraphResult=grOk
    then  begin
            For i:=1 to 6  do
              For j:=1 to 5  do
                Domik (i*80,j*80);
            ReadLn
          end
END.
```

Если Вас интересуют другие графические процедуры или функции, то Вам следует обратиться к системе "помощи" (Help). Для этого в меню Help выберите пункт Standard units. Среди всего прочего Вы увидите список названий стандартных модулей (неважно, что вы не знаете, что это такое). Если установить курсор на слово Graph (этого можно достигнуть и с помощью клавиши Tab) и нажать Enter, то на экране появится общая информация о модуле Graph. Для продолжения нам предлагается два выбора:

Go to GRAPH.TPU Functions and Procedures
Go to GRAPH.TPU Constants and Types

Если выбрать пункт "Перейти к константам и типам GRAPH.TPU", то Вы сможете добраться до такой полезной информации, как названия графических режимов и их разрешение, названия цветов, названия стилей закраски и т.п. На случай, если Вам захочется там что-нибудь поискать, приведем перевод названий некоторых пунктов соответствующего меню:

Color Constants	Цветовые константы
Fill Pattern Constants	Константы для образцов закраски
Graphics Drivers	Графические драйверы
Graphics Modes for Each Driver	Графические режимы для каждого драйвера

Выбрав же пункт "Перейти к процедурам и функциям GRAPH.TPU", мы увидим внушительный список названий графических процедур и функций. Если поместить курсор на название приглянувшейся Вам процедуры (или функции), и нажать "перевод строки", то Вы увидите краткое описание этой процедуры (функции), и в

конце - предложение посмотреть пример. (Если, скажем, Вы читаете информацию о процедуре Ellipse, то в самом конце Вы увидите слова Sample Code: Ellipse.PAS

Если установить курсор на слова Ellipse.PAS и нажать "перевод строки", то на экране Вы увидите текст программы, иллюстрирующий применение этой процедуры). Эта информация может пригодиться даже тем, кто не знает ни одного английского слова - Вы увидите, сколько у процедуры параметров и какие у этих параметров типы; Вы можете обычным образом (клавишами управления курсором при нажатой клавише Shift) отметить текст примера, скопировать в пустое окно и запустить.

Пример 11

(Е.В.Баранова, РГПУ им.А.И.Герцена).

Изобразить график функцииљ0 y=cos(x).

При изображении графика функции необходимо иметь ввиду, что начало графических координат находится в левом верхнем углу экрана и что графические координаты принимают целые неотрицательные значения в диапазоне (0,maxx) и (0,maxy). Значения maxx и maxy для каждого режима можно определить, используя соответствующие функции. Таким образом, для получения "хорошего" графика необходимо выполнить поворот и масштабирование. Пусть, xmax - максимальное значение по оси x; ymax - максимальное значение по оси y;

(x0,y0) - графические координаты центра - точки (0,0);
(xg,yg) - графические координаты точки (x,y);
mx - масштаб по оси x, т.е. величина Abs((xg-x0)/x);
my - масштаб по оси y, т.е. величина Abs((yg-y0)/y).

Графические координаты точки (x,y): xg=x0+mx*x; yg=y0-my*f(x).

```
PROGRAM  Primer_11;
  Uses Graph;
  var  x,y,a,b,h                    : Real;
       x0,y0,xg,yg,xmax,ymax,mx,my,grd,grm,c: Integer;
BEGIN
  WriteLn ('Введите координаты центра: '); ReadLn (x0,y0);
  WriteLn ('Введите масштаб по x и y: '); ReadLn (mx,my);
  WriteLn ('Введите область задания функции по x и шаг: ');
  ReadLn (a,b,h); WriteLn ('Введите цвет изображения: ');
  ReadLn (c); grd:=0; grm:=0; InitGraph (grd,grm,'');
  c:=getcolor; xmax:=getmaxx; ymax:=getmaxy;
  Line (10,y0,xmax-10,y0);  { Ось OX }
  Line (x0,10,x0,ymax-10);  { Ось OY }
  x:=a;
  While  x<=b  do
    begin
      xg:=x0+Trunc(mx*x); yg:=y0-Trunc(my*f(x));
      If  (xg>=0) AND (xg<=xmax) AND (yg>=0) AND (yg<=ymax)
        then  putpixel (xg,yg,c);
      x:=x+h
    end;
  ReadLn;
  closegraph
END.
```

Пример 12

(Е.В.Баранова, РГПУ им.А.И.Герцена).

Изобразить движение шарика по синусоиде.

Движение реализуется с помощью процедур GetImage и PutImage. Процедура GetImage запоминает образ изображаемого объекта и образ области экрана такого же размера, закрашенной цветом фона. Процедура PutImage на каждом шаге последовательно заменяет старое изображение цветом фона и создает изображение на новом месте.

```
PROGRAM  Primer_12;
{ Программа движения шарика со следом по синусоиде }
  uses Graph;
  var  p1,p2: Pointer;
       { p1 - указатель на образ "следа",
         p2 - указатель на  образ шарика }
       grm,grd,x,y,x1,y1: Integer;
       size,c           : Word;
BEGIN
  grd:=0; InitGraph (grd,grm,'D:\Tp\Bgi');
  { Инициализация графического режима с автоматическим
    определением подходящего драйвера }
  c:=GetColor;  { c - цвет изображения               }
  x1:=0;y1:=90; { x1,y1 - начальные координаты шарика }
  PutPixel (0,y1+5,c);          { Изображение "следа" }
  size:=ImageSize(0,0,10,10); GetMem (p1,size);
  { size - количество байтов для изображения квадрата 11x11 }
  GetImage (0,y1,10,y1+10,p1^);
  { p1 указывает на область памяти с изображением следа }
  SetFillStyle (11,c);  { Устанавливается тип и цвет закраски }
  Circle (x1+5,y1+5,5); { Окружность с центром в (x1,y1)      }
  FloodFill (x1+5,y1+5,c); { Закраска окружности }
  GetMem (p2,size); GetImage (x1,y1,x1+10,y1+10,p2^);
  { p2 указывает на область памяти с изображением шарика }
  For x:=1 to 300  do
    begin
      y:=Trunc (40*sin(0.1*x)+90);
      { x,y - графические координаты нового положения шарика }
      PutImage (x1,y1,p1^,0); { На месте шарика изображается след }
      PutImage (x,y,p2^,0); { Шарик изображается на новом месте }
      x1:=x; y1:=y    { Запоминаются новые координаты шарика }
    end;
  ReadLn; Closegraph
END.
```

Пример 13

(Е.В.Баранова, РГПУ им.А.И.Герцена).

Управление движением объекта.

Направление движения определяется нажатой клавишей (стрелки влево, вправо, вверх, вниз). Шаг перемещения вводится. Реализация движения характеризуется тем, что на каждом шаге запоминается образ области экрана, куда помещается курсор, затем при смещении курсора изображение восстанавливается.

```
PROGRAM  Primer_13;
{ Программа управления движением курсора.
  Курсор - прямоугольный объект, двигающийся вверх, вниз,
  вправо, влево при нажатии соответствующих стрелок.
  Выход при нажатии клавиши ESC }
   uses Crt,Graph;
        { Модуль Crt необходим для использования Readkey }
PROCEDURE  BadKey;
{ Процедура формирует звук при нажатии неправильной клавиши }
BEGIN
   Sound (500); Delay (100); Sound (400);
   Delay (200); Nosound
END;
var p,pc: Pointer;
{pc - указатель на образ курсора,
 p - указатель на образ изображения "под" курсором }
  grm,grd,curx,cury,curx0,cury0,lx,ly,hx,hy:integer;
  size,c:word; ch:char;
{ grd,grm - переменные для номеров графических драйверов и режима
  curx,cury - координаты текущего положения курсора
  curx0,cury0 - переменные для запоминания координат курсора
  lx,ly - ширина и длина курсора прямоугольного вида
  hx,hy - шаги движения курсора по горизонтали и вертикали }
BEGIN
   WriteLn ('Введите размеры курсора и шаги движения');
   ReadLn (lx,ly,hx,hy);
   { Установка значения системной переменной для обеспечения
     совместимости работы модулей Crt и Graph }
   DirectVideo:=FALSE;
   grd:=0; InitGraph (grd,grm,'D:\Tp\Bgi');
   { Инициализация графического режима с автоматическим
     определением подходящего драйвера                }
   c:=GetColor; { c - цвет изображения }
   size:=ImageSize (0,0,lx,ly);
   { size - количество байтов для изображения курсора }
   GetMem (pc,size); GetMem (p,size);
   { Выделяются области для хранения образа курсора,
     и образа изображения под курсором }
   SetFillStyle (1,c); { устанавливается тип и цвет закраски курсора }
   GetImage (0,0,lx,ly,p^);
   { p указывает на область памяти, где хранится изображение,
     которое будет "закрыто" курсором }
   curx:=0; cury:=0;
   Bar (0,0,lx,ly); GetImage (0,0,lx,ly,pc^);
   { pc указывает на область памяти с изображением курсора }

   SetColor (6); SetFillStyle (1,2);
   Bar3d (150,150,200,30,10,TRUE);
   { Параллелограмм, на фоне которой происходит движение }
   Repeat { Цикл по вводу символа }
      ch:=ReadKey; { Ввод очередного символа }
      If  Ord(ch)=0
        then { Нажата управляющая клавиша }
             begin
               ch:=ReadKey;
               curx0:=curx; cury0:=cury;
               { В переменных curx,cury запоминаются координаты курсора }
               Case  Ord(ch)  of
```

```
            77: If  curx<getmaxx-hx
                  then curx:=curx+hx;  { Шаг вправо }
            75: If  curx>hx
                  then  curx:=curx-hx; { Шаг влево }
            72: If  cury>hy
                  then  cury:=cury-hy; { Шаг вверх }
            80: If  cury<getmaxy-hy
                  then  cury:=cury+hy  { Шаг вниз }
                else  BadKey { Нажата "неправильная" клавиша }
            end;
            If  (curx<>curx0) OR (cury<>cury0)
              then  begin
                      PutImage (curx0,cury0,p^,0);
                      { Восстановить изображение, которое было "закры-
                        то" курсором }
                      GetImage (curx,cury,curx+lx, cury+ly,p^);
                      { Запомнить то изображение, которое будет "зак-
                        рыто" курсором }
                      PutImage (curx,cury,pc^,0);
                      { Установить курсор в новое положение }
                    end
          end
        else  BadKey
  until  Ord(ch)=27;
  CloseGraph
END.
```