МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра информатики и вычислительной математики

РЕШЕНИЕ ГРАФИЧЕСКИХ ЗАДАЧ В СРЕДЕ TURBO PASCAL V 7.0

Методические указания для студентов естественных специальностей

Издательство «Самарский университет»
1997

Методические указания «Решение графических задач в среде Turbo Pascal v 7.0» представляют собой краткое руководство по программированию на языке Pascal применительно к графическим задачам. Пособие поясняет и раскрывает назначение и возможности модулей CRT и GRAPH **Turbo Pascal**я и содержит описание основных процедур и функций этих модулей. Многочисленные примеры, приведенные в пособии, демонстрируют технику программирования при решении графических задач. Задачи различной сложности, оформленные в пособии как лабораторные работы, помогут студентам закрепить изученный материал.

Данные указания адресованы, в первую очередь, студентам естественных факультетов вузов. Изложенный материал также будет полезен в работе научным работникам, инженерам и учителям информатики в школах с углубленным изучением программирования на языке Pascal.

Составители: А.С.Луканов, кандидат физико-математических наук, доцент кафедры «Информатика и вычислительная математика» СамГУ, **В.М.Сиников,** доцент кафедры «Информатика и вычислительная математика» СамГУ.

Рецензенты: зам. зав. каф. «Высшей и прикладной математики» СамГТУ, д. ф.-м. н., профессор Радченко В.П., доцент каф. «Высшей и прикладной математики» СамГТУ, к. ф.-м. н. Попов Н.Н.

1. ПОДПРОГРАММЫ УПРАВЛЕНИЯ МОНИТОРОМ (МОДУЛЬ CRT)

Подпрограммы модуля CRT предназначены для обеспечения работы с монитором в текстовом режиме. Отметим, что указанный режим является стандартным для MS-DOS. Указанный модуль также содержит описания типов, констант и переменных, используемых указанными подпрограммами.

1.1.Понятие текстового окна

В текстовом режиме, после начала выполнения программы, по умолчанию текстовое окно устанавливается во весь экран. Экран в этом режиме разбивается на строки, а строки на столбцы. Строки нумеруются начиная с 1 сверху вниз, а колонки - слева направо тоже с 1.

В каждой позиции экрана, заданной номером строки и номером столбца, может отображаться любой алфавитно-цифровой символ. Количество строк зависит от используемого контроллера, самого монитора и установленного текстового режима. В зависимости от установленного драйвера, при загрузке ЭВМ устанавливается некоторый принятый по умолчанию текстовой режим. Для изменения этого режима используется подпрограмма:

Text Mode (режим: Integer),

где параметр ***режим*** задается одним из следующих значений:

Константа	Режим
BW40 = 0	25 стр.((25*40) символов черно-белый режим
C40 = 1	25*40 режим 25*40 цветной режим
BW80 = 2	25*80 символов черно-белый режим
C80 = 3	25*80 символов цветной режим
MONO = 7	25*80 монохромный режим
Font8x8= 256	для EGA/VGA 43*80/50*80 цветной режим

Здесь и в дальнейшем предполагается, что значение соответствующего параметра может задаваться допустимым числовым значением или именем константы. Рекомендуется использовать имена констант, т.к. это делает программу более понятной и наглядной. При вызове процедуры ***Text Mode*** сбрасывается текущее окно и в качестве окна выбирается вся область экрана. Текущий текстовой атрибут устанавливается в нормальное состояние, соответствующее обращению к процедуре ***Norm Video***, а текущий видеорежим сохраняется в переменной ***Last Mode.*** Вызов ***Text Mode (Last Mode)*** приводит к тому, что вновь будет установлен последний активный текстовой режим. Это бывает полезно в том случае, когда после использования графических средств необходимо вернуться в текстовой режим.

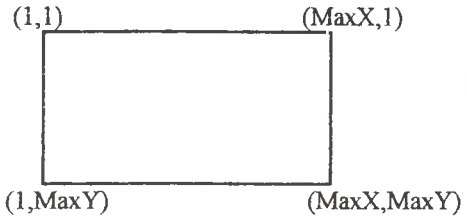
По умолчанию текущее окно устанавливается во весь экран. Под текущим окном здесь понимается область экрана, на которую будут распространяться все последующие текстовые операции (в частности, вывод).

Окно можно изменить с помощью подпрограммы:

Window (x1,y1,x2,y2: byte),

где ***(x1,y1)***- координаты левого верхнего угла, а ***(x2,y2)***- правого нижнего угла.

Наименьшее окно имеет размеры 1*1 символов. С текстовым окном связано понятие текстового курсора, который указывает текущую позицию в окне. Координаты позиций внутри окна задаются относительно его верхнего угла, который имеет координаты (1,1).



Здесь MaxX - количество столбцов, а MaxY - количество строк окна.

Окно обладает всеми свойствами монитора. Следует помнить, что в каждый момент времени на экране может быть установлено только одно окно.

После определения окна курсор размещается в левом верхнем его углу. Для перемещения текстового курсора в позицию окна с координатами ***(x,y)*** можно использовать процедуру:

GoToXY(x,y : byte)

Текущее положение курсора можно определить с помощью функций:

WhereX: byte

WhereY: byte

которые выдают соответствующие координаты текущего положения курсора.

1.2. Установка цвета

Для выводимого на экран текста можно задать атрибуты цвета символов и цвета фона. Можно также задать режим вывода с миганием символов, а также увеличить или уменьшить яркость его отображения. Для использования цвета необходимо наличие адаптера и монитора,обеспечивающих вывод цветных изображений.

Для установки цвета выводимых символов используется процедура:

TextColor(c: byte).

Здесь параметр ***c*** задает цвет выводимых символов и может принимать значения от 0 до 15. Значения этого параметра могут задаваться следующими константами:

Константа	Цвет	Константа	Цвет
Black =0	черный	DarkGray=8	темно-серый
Blue =1	синий	LightBlue=9	голубой
Green=2	зеленый	LightGreen=10	светло-зеленый
Cyan =3	бирюзовый	LightCyan=11	светло-бирюзовый
Red =4	красный	LightRed=12	светло-красный
Magenta=5	малиновый	LihtMagenta =13	светло-малиновый
Brown =6	коричневый	Yellow=14	желтый
LightGray=7	светло-серый	White=15	белый

В модуле **CRT** имеется переменная ***TextAttr***, которая используется для сохранения текущего видеоатрибута в соответствии с параметром ***c***. Если значение параметра превышает 15, то устанавливается бит мерцания (7). Процедура устанавливает биты 0..3 в соответствии со значением параметра ***c***. Режим мерцания можно задать, прибавляя к параметру процедуры значение 128 или константу ***Blink***.

Для задания цвета фона используется стандартная процедура:

TextBackGround(цвет: byte).

Параметр ***цвет*** может принимать следующие значения:

Константа	Цвет	Константа	Цвет
Black=0	черный	Read=4	красный
Blue=1	синий	Magenta=5	малиновый
Green=2	зеленый	Brown=6	коричневый
Cyan=3	бирюзовый	LightGray=7	светло-серый

Процедура ***TextBackGround*** устанавливает биты **4..6** атрибута в соответствии с параметром ***цвет***.

В текстовом режиме имеется возможность управления яркостью выводимых символов . Для этого имеются процедуры:

LowVideo - понижение яркости,
HighVideo - повышение яркости,
NormVideo - установка обычной яркости .

1.3.Работа с текстом

Для удобства работы с текстом имеется ряд дополнительных процедур:
ClrScr - очищает строку, начиная с позиции курсора на экране;
InsLine - начиная с текущей позиции курсора вставляет пустую строку;
DelLine - удаляет строку, в которой расположен курсор.

1.4.Дополнительные процедуры модуля CRT.

В модуле CRT имеется еще ряд дополнительных процедур. Наиболее часто используемыми являются следующие:

KeyPressed:Boolean-возвращает значение **true**, если на клавиатуре нажата клавиша, а в противном случае значение **false**.

ReadKey : Char- считывает с клавиатуры символ, который на экране не отображается. Если перед обращением к ней функция ***KeyPressed*** выдавала значение **true**, то символ берется из буфера клавиатуры немедленно. В противном случае, функция ожидает нажатия клавиши.

Следует иметь в виду, что если нажата функциональная клавиша, то в начале передается символ с кодом 0, а затем уже код нажатой клавиши. Приведем коды наиболее часто используемых клавиш:

Константа	10-ный код	Клавиша
UpArrow	72	стрелка верх
DownArrow	80	стрелка вниз
PageUp	73	на страницу вверх
PageDown	81	на страницу вниз
LeftArrow	75	стрелка влево
RightArrow	77	стрелка вправо
EscKey	27	клавиша ESC

Все клавиши, за исключением последней, являются функциональными.

Для иллюстрации подпрограмм модуля **CRT** рассмотрим следующий пример.

Пример 1. Составить программу, которая очищала бы заданную прямоугольную область экрана с использованием заданного цвета фона. При каждом нажатии клавиши **Enter** она изменяла бы цвет закраски области на следующий по порядку, при нажатии клавиши **UpArrow** возвращалась к

предыдущему цвету закраски. Завершение работы должно осуществляться при нажатии клавиши ***Esc***.

```
uses Crt;
const
  UpArrow=#72;
  DownArrow=#80;
  EscKey=#27;
var i:byte;
nc,c:byte;
kl:char;x1,y1,x2,y2:byte;
begin
  ClrScr;{очистка экрана}
  {ввод координат прямоугольной области}
  write('x1=');readln(x1);
  write('y1=');readln(y1);
  write('x2=');readln(x2);
  write('y2=');readln(y2);
  {ввод номера цвета закраски}
  writeln('Задайте номер начального цвета=');
  readln(nc);
  ClrScr;
  c:=nc;
  window(x1,y1,x2,y2);
  repeat
   if c>7 then c:=0;{ c задает текущий цвет фона для окна}
   TextBackGround(c);
   ClrScr;
     kl:=ReadKey;  {в переменную kl заносится символ нажатой
  клавиши }
   if kl=#0 then   {проверка нажатия функциональной клавиши}
     begin
       kl:=ReadKey;
       case kl of
       UpArrow:  if c>0 then c:=c-1;
       DownArrow: if c<7 then c:=c+1
       end
     end;
  until kl=EscKey;
end.
```

2. ПРОЦЕДУРЫ ДЛЯ РАБОТЫ С ГРАФИКОЙ (МОДУЛЬ GRAPH)

2.1. Общие сведения

Если персональный компьютер оснащен графическим адаптером, то при работе с дисплеем можно использовать графический режим. В этом режиме имеется возможность выводить на экран, помимо текстовой информации, различные графические примитивы, т.е. отдельные точки, линии и геометрические фигуры. Причем, для вывода текста можно использовать большое число различных штриховых шрифтов. Это позволяет повысить качество отображения информации на экране и принципиально изменить стиль интерфейса.

Модуль ***Graph*** содержит библиотеку из более чем 50 графических подпрограмм - от процедур и функций высокого уровня, таких например, как ***SetViewPort, Bar3D, DrawPoly,*** до подпрограмм «низкого» уровня, ориентированных на работу с битами, таких, например, как ***GetImage*** или ***PutImage***. В модуле реализовано несколько типов закрашивания и типов линий. Поддерживается также множество графических шрифтов. При выводе текста имеется возможность изменять размер символов, выполнять операции выравнивания, а также осуществлять вывод горизонтально или вертикально.

Для компиляции программы, использующей модуль ***Graph***, Вам не потребуется никаких внешних файлов (кроме, разумеется, исходного текста Вашей программы, компилятора и доступа к стандартным модулям в библиотеке исполняющей системы). А именно:

Тип программы	Библиотека	Имя модуля
Реальный режим	TURBO.TPL	GRAPH.TPU
Защищенный режим	TPP.TPL	GRAPH.TPP

Для запуска программы, использующей модуль ***Graph***, Вам потребуется один или более графических драйверов (см. ниже файлы с расширением ***.bgi***), а также, если в Вашей программе используются шрифты, потребуется один или более файлов шрифтов (расширение ***.chr***).

Для переключения системы в графический режим необходимо выполнить процедуру ***InitGraph:***

InitGraph(var ***граф_драйвер***, ***граф_реж***: integer; ***путь_драйвера***: string)

Параметр ***граф_драйвер*** задает номер загружаемого драйвера, обеспечивающего работу в графическом режиме. Драйверы хранятся в файлах типа ***.bgi***.

Если задать значение этого параметра константой **DetectGraph**=0, то автоматически будет вызвана процедура ***DetectGraph***, которая определит нужный драйвер и выберет нужный графический режим. Иначе, будет загружен указанный драйвер и система будет переведена в режим заданный вторым параметром ***граф_реж***.

Параметр ***путь_драйвера*** задает маршрут к каталогу, в котором расположен на диске графический драйвер. Если значение этого параметра задать пустой строкой, то подразумевается, что файл драйвера находится в текущем каталоге.

Для проверки наличия соответствующих аппаратных средств и определения драйвера и графического режима можно использовать процедуру

DetectGraph(var ***драйвер***, ***режим***: ***Integer***)

Она возвращает значения ***драйвер***а и ***режим***а, которые могут затем передаваться в качестве параметров процедуре ***InitGraph*** для загрузки соответствующего драйвера и установки режима. Если графическая аппаратура не найдена, то параметры ***драйвер и режим*** содержать значения -2. Для задания драйверов используются следующие предопределенные константы:

Имя константы	Значение	Имя константы	Значение
CGA	1	IBM8614	6
MCGA	2	HercMono	7
EGA	3	ATT400	8
EGA64	4	VGA	9
EgaMono	5	PC3270	10

Проверить правильность установления графического режима можно с помощью функции

GraphResult: Integer.

Если произошла ошибка, то функция ***GraphResult*** будет возвращать значение, отличное от **GrOK**=0:

Имя константы	Значение	Тип ошибки
grNoInitGraph	1	не прошла инициализация графики
grNotDetected	2	не обнаружена графическая плата;
grFileNotFound	3	не найден файл
grInvalidDriver	4	неверный драйвер
grNoLoadMem	5	не загружен в память
grNoScanMem	6	не хватает памяти
grNoFloodMem	7	буфера заполнения
rFontNotFound	8	не найден фонт
grFontMem	9	нет памяти для фонта
grInvalidMode	10	неверный графический режим
grError	11	грубая ошибка
grIOerror	12	ошибка ввода
grInvalidFont	13	неверный тип фонта
grInvalidFontNum	14	неверный номер фонта
rInvalidDeviceNum	15	неверный номер устройства

Для расшифровки смысла кода, возвращаемого функцией ***Graph_Result***, используется функция

***GraphErrorMsg(errorcode:*Integer*):*String**

которая выдает пояснение ошибки в виде текстовой строки.

Для задания нужного графического режима может использоваться процедура

***SetGraphMode(режим:* Integer)**,

которая устанавливает заданный графический режим и очищает экран. Параметр может принимать значения от **0** до **5**. Возможные режимы зависят от установленного графического драйвера. Тип графического режима определяет разрешение экрана, т.е. количество точек в строке и количество строк на экране, а также доступную гамму цветов и число графических страниц.

Наиболее широко используются следующие режимы работы:

Тип адаптера	Графический режим	Разрешение	Число страниц	Палитра
CGA	CGAC0=0	320*200	1	C0
	CGAC1=1	320*200	1	C1
	CGAC2=2	320*200	1	C2
	CGAC3=3	320*200	1	C3
	CGAHi=4	640*200	1	2 цвета
EGA	EGALo=0	640*200	1	16 цветов
	EGAHi =1	640*350	2	16 цветов
EGA64	EGA64Lo=0	640*200	1	16 цветов
	EGA64Hi=1	640*350	1	4 цвета
VGA	VGALo=0	640*200	2	16 цветов
	VGAMED=1	640*350	2	16 цветов
	VGAHi=2	640*480	1	16 цветов

Для получения текущего графического режима можно использовать процедуру ***GetGraphMode,*** которая не имеет параметров и выдает в виде целого значения текущий графический режим.

Для завершения работы в графическом режиме можно использовать процедуры ***CloseGraph*** или ***RestoreCrtMode***. Эти процедуры не имеют параметров.

CloseGraph восстанавливает исходный текстовой режим экрана и освобождает память занимаемую графическим драйвером.

RestoreCrtMode возвращает режим экрана в исходное состояние (то, которое действовало до инициализации графики, т.е. восстанавливает видеорежим, обнаруженный процедурой ***InitGraph***.

Пример 2. Приведем описание функции, которая будет инициализировать графический режим. При успешном завершении будет возвращать значение **true**, а в противном случае значение **false.**

```
function initgr: Boolean;
var grdriver,grmode,errcode:Integer;
begin
  grdriver:=Detect;
  InitGraph(grdriver,grmode,'');
  errcode:=GraphResult;
  if errcode=GrOK then
    initgr:=true
  else
    begin
      writeln('Ошибка:',GraphErrorMsg(errcode));
      initgr:=false
    end
end;
```

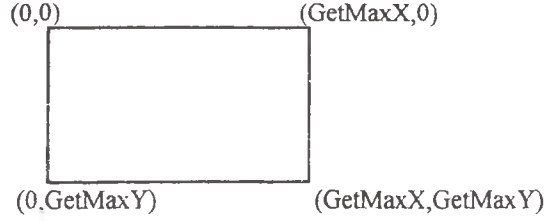
Пример 3. Рассмотрим программу переключения из графического режима в текстовой и наоборот.

```
Uses Graph;
var
   GraphDriver, GraphMode: integer;
begin
   GraphDriver:=Detect;
   InitGraph(GraphDriver,GraphMode, '');
   if GraphResult <> grOk then
      Halt(1);
   OutText('Для выхода из графики нажмите Enter: ');
   Readln;
   RestoreCrtMode;
   Writeln( ' Теперь Вы в текстовом режиме');
   Write(' Нажмите Enter для перехода в графический режим');
   Readln;
   SetGraphMode(GetGraphMode);
   OutTextXY(0,0, ' Вы снова в графическом режиме');
   OutTextXY(0,TextHeight('H')+2, 'Для выхода нажмите Enter');
   ReadLn;
   CloseGraph;
end.
```

Здесь вызов функции SetGraphMode(GetGraphMode) используется для возврата в режим, установленный InitGraph.

2.2.Графические окна

После установления графического режима по умолчанию графическим окном будет весь экран. Левый верхний угол экрана имеет координаты **(0,0)**, а правый нижний - ***(GetMaxX,GetMaxY)***, где ***GetMaxX*** - это функция, выдающая максимальное значение номера позиции строки, а ***GetMaxY***- максимальный номер координаты у, т.е. номера строки экрана. Таким образом, координаты каждого из четырех углов максимально развернутого окна экрана будут выглядеть следующим образом:



Размеры графического окна можно изменить с помощью процедуры:

SetViewPort(x1,y1,x2,y2:Word;Clip:Boolean).

Здесь **(x1,y1),(x2,y2)** - координаты левого верхнего и правого нижнего углов прямоугольной области экрана, соответственно. Параметр ***Clip*** определяет, будет ли для выводимого изображения выполняться операция отсечения границами окна. Операция отсечения заключается в определении видимой в текущем графическом окне части выводимого изображения. Если ***Clip*=true**, то отсечение будет выполняться. Эта операция замедляет вывод, но обеспечивает корректное отображение видимой части рисунка в окне, когда он не полностью виден в нем. Процедура ***SetViewPOrt*** устанавливает заданное графическое окно, очищает его, т.е. заполняет цветом фона, и перемещает графический курсор в левый верхний угол окна, который имеет координаты **(0,0)**. После задания графического окна весь последующий вывод будет осуществляться в область этого окна.

С графическим окном связано понятие графического курсора, который на экране невидим, но любой графический вывод осуществляется с учетом позиции его положения. Координаты текущего положения графического курсора можно определить с помощью функций:

GetX: Integer
GetY: Integer.

Графический курсор можно перемещать в нужную позицию экрана с помощью процедур:

MoveRel(dx,dy:Integer)
MoveTo(x,y:Integer).

Первая перемещает курсор из текущей позиции **(xt,yt)** в позицию экрана с координатами **(xt+dx,yt+dy)**, а вторая в точку окна с координатами **(x,y)**.

Для очистки экрана/окна могут использоваться следующие процедуры:

ClearDevice

ClearViewPort.

Первая - сбрасывает состояние графического экрана и подготавливает его к графическому выводу. При этом все параметры устанавливаются в значения, принятые по умолчанию. Экран очищается. а курсор устанавливается в позицию **(0,0)**.

Вторая - очищает область текущего окна, используя установленный цвет фона, а также перемещает курсор в позицию **(0,0)** окна.

2.3.Установка цвета

Изображение в графическом режиме может выводиться как цветное, так и черно-белое. Все зависит от монитора, графического адаптера и установленных цветов. Высвечиваемые на экране точки называются пикселами. Цвет выводимых символов и пикселов может устанавливаться с помощью процедуры:

SetColor(цвет: Word),

где параметр процедуры задает цвет. Значение этого параметра является индексом в таблице допустимых цветов, называемой палитрой. Значение этого параметра может изменяться от 0 до 15. Можно получить текущее значение цвета, вызвав функцию

GetColor:Word.

Если функция возвращает значение 0, то это означает, что для рисования используется первый цвет текущей палитры.

Для задания цвета фона, т.е. цвета поля на котором выводятся символы, используется процедура:

SetBkColor(цвет:Word).

Цвет фона может задаваться значениями от 0 до 15. Действующий цвет фона можно получить с помощью функции:

GetBkColor:Word.

2.4.Вывод текста в графическом режиме

Для вывода текста в графическом режиме используются процедуры:

OutText(строка:String)

OutTextXY(x,y:Integer;срока: Strig).

Первая осуществляет вывод заданной текстовой строки с учетом текущего положения графического курсора. А вторая - в заданную позицию экрана. Размещение текста относительно курсора может определяться с помощью процедуры:

SetTextJustify(Horiz,Vert:Word),

где первый параметр задает метод выравнивания по горизонтали, а второй - повертикали. Выравнивание для процедуры **OutText** осуществляется относительно текущей позиции графического курсора, а для процедуры **OutTextXY**- относительно заданной точки экрана.

Горизонтальное выравнивание может задаваться значениями:

LeftTop = 0 - выравнивание по левому краю(по умолчанию)
CenterText=1 - выравнивание по центру
RightText = 2 - выравнивание по правому краю.

Выравнивание по вертикали может задаваться значениями:

BottomText=0 - выравнивание по нижнему краю (действует по умолчанию)
CenterText=1 - выравнивание по центру
TopText =2 - выравнивание по верхнему краю.

Для вывода строки используется текущий цвет, а также текущий шрифт. Тип шрифта может изменяться с помощью процедуры:

SetTextStyle(font,direction,charsize : Word),

где первый параметр задает тип шрифта и может принимать следующие значения:

DefaultFont =0 - шрифт принятый по умолчанию, определяемый шаблоном 8*8.
TriplexFont = 1 - тройной штриховой шрифт
SmallFont = 2 - индексный штриховой
SansSerifFont=3 - гротесковый штриховой
GothicFont = 4 - готический штриховой

Штриховые шрифты характеризуются тем, что при увеличении размера букв качество их изображения не ухудшается.

Второй параметр задает направление вывода и может принимать значения:

HorizDir = 0 - горизонтально слева-направо
VertDir = 1 - вертикальное снизу-вверх.

Третий параметр задает коэффициент растягивания выводимых букв по вертикали и горизонтали:

UserCharSize=0 - размер задается с помощью процедуры ***SetUserCharSize***;

1 - символы будут размером 8*8 пикселов,

2 - размером 16*16 и т.д. вплоть до 10 кратного увеличения.

Можно задать свои размеры символов. Для этого можно использовать процедуру:

SetUsertCharSize(xmul,xdiv,ymul,ydiv:Byte).

Коэффициент растяжения по горизонтали будет равен **xmul/xdiv**, а по вертикали **ymul/ydiv**. Например, чтобы сделать шрифт вдвое шире, нужно использовать для ***xmul*** значение 2, а для ***xdiv*** значение, равное 1. После вызова ***SetUs-***

erCharSize для того, чтобы шрифт получил новый размер, необходимо вызвать процедуру ***SetTextStyle***.

При задании шрифта с использованием процедуры ***SetTextStyle*** он считывается в память с магнитного диска. Пользователь может сам считать содержимое файла шрифта в динамически распределяемую память и зарегистрировать его с помощью процедуры

RegisterBGIfont(font:pointer):integer

Здесь параметр ***font*** является указателем на соответствующую область динамической памяти. Функция подключает к графической системе фонт, загружаемый пользователем или входящий в базовый графический интерфейс. В результате последующие переключения на этот шрифт не вызывают повторные считывания его с магнитного диска. Функция возвращает внутренний номер фонта или код ошибки. В последующем этот номер может использоваться в процедуре ***SetTextStyle*** для подключения соответствующего фонта для вывода текста. Зарегистрированный пользователем фонт может быть преобразован в ***.OBJ*** файл (используя **Binobj.exe**) и связан с программой в ***.EXE*** файл.

При выводе текста полезными являются следующие функции:

TextHight(строка:String):Word,

TextWidth(строка:String):Word.

Первая на основе текущего шрифта, размера и коэффициента растяжения определяет высоту строки, заданной параметром, в пикселах, а вторая - ширину.

Пример 4. Составить программу вывода заданной текстовой строки заданным шрифтом с использованием символов всех допустимых размеров.

```
uses Graph;
var gd,gm:Integer;
row,size,nfont:Integer;
title:String;
begin
  gd:=Detect;
  InitGraph(gd,gm, '');
  if GraphResult<>GrOK then
   begin
     writeln('Ошибка: ');Halt(1)
   end
  else
   begin
     write('Строка=');readln(title);
     write('Номер строки экрана=');readln(row);
     write('Задайте номер шрифта(0..4)= ');readln(nfont);
     size:=1;
```

```
    while TextWidth(title)<GetMaxX do
      begin
        OutTextXY(0,row,title);
        Inc(row,TextHight('M'));
        Inc(size);
        SetTextStyle(nfont,HorizDir,size)
      end;
  end;
 readln;
 CloseGraph
end.
```

2.5.Рисование графических объектов

К основным типам графических объектов относятся точки, линии, геометрические фигуры, закрашенные геометрические фигуры и текстовые символы. Очевидно, что любое геометрическое изображение может быть построено из этих более простых объектов, называемых графическими примитивами.

Для вывода точек используется процедура:

PutPixel(x,y:Integer;color:word),

где ***(x,y)***- координаты выводимой точки, а ***color*** задает ее цвет.

Для получения значения цвета заданной точки изображения можно использовать функцию:

GetPixel(x,y:Integer):Word,

которая возвращает индекс цвета.

Для рисования линий используются процедуры:

Line(x1,y1,x2,y2: Word) - соединяет точки с координатами ***(x1,y1)*** и ***(x2,y2)*** отрезком прямой;

LineTo(x,y:Word) - проводит отрезок от текущей точки в точку с координатами ***(x,y)***;

LineRel(dx,dy:Word) - от текущей точки в точку, отстоящую от текущей на ***dx,dy*** вдоль координатных осей.

Рисование линии осуществляется ранее установленным цветом и стилем линии.

Стиль линии можно изменять с помощью процедуры:

SetLineStyle(стиль,образец,толщина:Word),

где первый параметр задает тип линии и может принимать следующие значения:

SolidLn = 0 -сплошная линия (по умолчанию);
DottedLn =1 - пунктирная линия;
CenterLn =2 - осевая линия;
DashedLn =3 - штриховая линия;

UserBitLn =4 - линия, задаваемая пользователем.

Третий параметр задает толщину линии. Допустимы его значения:

NormWidth=1 - тонкая линия (по умолчанию);

ThickWidth=3 - толстая линия.

Второй параметр используется, если только значение первого равно **UserBitLn**. Этот параметр задает образец проводимой линии. Например, если его значение равно $55, то образец закраски имеет вид 01010101, где 1 соответствует высвечиваемой точке, а 0 - нет.

Пример 5. В качестве примера работы с отдельными пикселами и строками текста рассмотрим программу, которая выполняет зеркальное преобразование заданного слова 'Samara'.

```
program Example;
uses graph;
var  r,c:word;
   i,j,n:integer;
   x1,x2,y1,y2:integer;
   grDriver,grMode,g:integer;
   s:string;
begin                              {Инициализация графического}
 grDriver  :=Detect;                    {режима}
 InitGraph(grDriver,grMode,'e:\tp\bgi');
 g := GraphResult;
 if g =grOk  then
 begin
    SetBkColor(White);
    SetColor(blue);
   SetUserCharSize(1,3,1,8);   {Задание высоты и ширины символов}
   SetTextStyle(triplexfont,horizdir,8); {Задание типа начертания шрифта и
 коэффициента увеличения символов}
   SetTextJustify(centertext,centertext);{Выравнивание}
   s:='Samara';
   OutTextXY(280,120,s);              {Вывод строки}
      x1:=280-(TextWidth(s)  div  2);      {Вычисление координат прямо
угольника, в котором находится строка}}
   x2:=280+(TextWidth(s)  div  2);
   y1:=120-TextHeight(s);
   y2:=120+TextHeight(s);
         {Преобразование строки}
   for i:=y1 to y2 do
    for j:=x1 to x1+((x2-x1)  div  2) do
     Begin
     n:=x2-j+x1;
```

```
      c:= GetPixel(j,i);  {Получаем цвет двух точек, симметричных}
      r:=GetPixel(n,i);   {относительно оси симметрии строки     }
      if c<>r  then
        begin    {Если точки разного цвета,то меняем их местами}
          PutPixel(j,i,r);
          PutPixel(n,i,c);
        end;
      end;
end;
readln;
CloseGraph;          {Закрываем графический режим}
end.
```

2.6. Процедуры рисования геометрических фигур

Для отображения на экран контуров геометрических фигур используются следующие процедуры:

Rectangle(x1,y1,x2,y2:Word) - вывод прямоугольника, где ***(x1,y1)*** - координаты его левого верхнего угла, а ***(x2,y2)*** - правого нижнего;

Circle(x,y,R:Word) - рисует окружность с центром ***(x,y)*** радиуса ***R***;

Arc(x,y:Integer;StAngle,EndAngle,R:Word) - рисует дугу окружности заданного радиуса ***R*** с центром в точке ***(x,y)***, начиная с угла ***StAngle*** по угол ***EndAngle***. Угол отсчитывается против часовой стрелки. Для ***StAngle***=0 и ***EndAngle***=360 будет рисоваться полная окружность. Углы задаются в градусах.

Ellipse(x,y:Integer;StAngle,EndAngle,XRadius,YRadius:Word) - рисует дугу эллипса. Смысл параметров тот же, что в предыдущей процедуре. ***XRadius,YRadius*** - радиусы вдоль осей x и y, соответственно.

DrowPoly(numpoint:Word;var polypoint) - рисует ломанную линию. Первый параметр задает число вершин, а второй является нетипизированным параметром, содержащим координаты каждой вершины. Данная процедура использует текущие тип линии и цвет, а также установленный режим вывода при вычерчивании линии. Режим вывода можно изменить с помощью процедуры ***SetWriteMode***. Заметим, что для вычерчивания замкнутой фигуры с n вершинами необходимо передать при обращении к ***DrowPoly*** n+1 координат вершин, где координата вершины с номером n+1 должны совпадать с координатами 1-ой вершины.

Пример 6.

```
Uses Graph;
Const triangle:array [1..4] of PoitType =
   ((x:10;y:100),(x:100;y:100),(x:150;y:150),(x:10;y:100));
```

```
var Gd,Gm:Integer;
begin
  Gd:=Detect;
  InitGraph(Gd,GM, '');
  if GraphResult <> grOK then Halt(1);
  DrowPoly(sizeof(tiangle) div sizeof(PoinType),triangle);
  ReadLn;
   CloseGraph;
end.
```

В этом примере на экран будет выведен контур треугольника с координатами (10,100), (100,100), (150,150). Здесь используется тип данных ***PoinType*** определенный в модуле ***Graph***:

```
PoinType=record
        x,y:word
       end;
```

Пример 7.

Рассмотрим программу, которая выводит 10 концетрических окружностей в центре екрана.

```
program Example;
 uses graph,crt;
 var
 grDriver,j,x,y,  ErrCode  :  Integer;
 grMode:integer;
begin
  grDriver  :=Detect;                          {Задан текущий}
 InitGraph(grDriver,grMode,'');  {графический режим}
 ErrCode  := GraphResult;
 if ErrCode  =grOk  then
 begin
    x:=GetMaxX  div  2;     {Определяем центр окружностей}
    y:=GetMaxY  div  2;
    SetBkColor(Blue);       {Задаем  цвет фона}
    SetColor(White);        {Задаем цвет окружностей}
    for j:=1 to 10 do
      Circle(x,y,(x  div 15)*j); {Рисуем окружность}
  end;
 readln;
 CloseGraph;          {Закрываем графический режим}
 end.
```

2.7.Рисование заполненных фигур

Все последующие процедуры служат для рисования закрашенных фигур. Следует отметить, что закраска осуществляется текущими цветом и образцом заполнения. Цвет и шаблон закрашивания могут быть определены с помощью процедур ***SetFillStyle*** и ***SetFillPattern***.

Первая из этих процедур вызывается в соответствии с заголовком

SetFillStyle(Pattern:Word;Color:Word),

где первый параметр задает шаблон для заполнения области. Он может принимать следующие значения:

Константа	Шаблон
EmptyFill = 0	заполнение текущим цветом фона;
SolidFill = 1	сплошное закрашивание;
LineFill = 2	заполнение толстыми горизонтальными линиями;
LtSlashFill = 3	заполнение наклонными линиями ////;
SlashFill = 4	заполнение толстыми наклонными линиями ////;
BkSlashFill = 5	заполнение толстыми косыми линиями \\\\;
LtBkSlashFill = 6	заполнение косыми линиями \\\\;
HatchFill = 7	заполнение редкой вертикальной сеткой;
XhatchFill = 8	заполнение наклонной часто пересекающейся сеткой;
InterLeaveFill= 9	заполнение прерывистыми линиями;
WideDotFill =10	редкое заполнение точками;
CloseDotFill = 11	плотное заполнение точками.

Процедура ***SetFillPattern*** вызывается в соответствии с заголовком

SetFillPattern(pattern:FillPatternType; color:Word)

и предназначена для задания цвета и шаблона закраски, которые задаются пользователем. Первый параметр задает шаблон и имеет следующий тип ***FillPatternType=array [1..8] of Byte***. Заполняющий шаблон основывается на внутренних байтовых значениях , содержащихся в массиве ***pattern***. Этот массив имеет длину в 8 байт, каждый из которых соответствует отдельной строке матрицы образца заполнения. Если бит в байте образца закраски равен 1, то соответствующая точка будет отображаться. Например, закраска с расположением точек в виде шахматной доски задается массивом значений:

```
10101010=$AA
01010101=$55
10101010=$AA
01010101=$55
. . . . . .
```

Имеются две процедуры, которые позволяют получить установленный шаблон. Это процедуры

GetFillPattern(var pattern:FillPattternType)

GetFillSetting(var FillInfo:FillSettingType).

Последняя выдает значение типа:

```
type
  FillSettingType=record
                    pattern:Word;
                    color: Word
                  end;
```

Здесь поле ***pattern*** служит для передачи номера шаблона, а ***color***-индекса цвета.

Процедура ***FloodFill*** предназначена для заливки действующими цветом и образцом заполнения ограниченной области. Ее заголовок имеет вид

FloodFill(x,y,border: Word).

Она осуществляет заполнение области, содержащей точку с координатами **(x,y)**, и ограниченной непрерывной линией, индекс цвета которой задается параметром **border**. Если точка с координатами **(x,y)** находится вне области, то заполняется все окно, кроме заданной области.

Процедура ***Bar*** имеет заголовок

Bar(x1,y1,x2,y2:Integer)

и рисует закрашенный прямоугольник, заданный координатами **(x1,y1)** и **(x2,y2)**, используя текущие цвет и шаблон.

Процедура ***Bar3D*** вызывается в соответствии с заголовком

Bar3D(x1,y1,x2,y2,depth:Word;topflag:Boolean)

и предназначена для рисования параллелепипеда, передняя грань которого закрашена. Параметры ***(x1,y1),(x2,y2)*** задают координаты передней грани, ***depth***- глубину параллелепипеда, а ***topflag*** указывает, должна ли быть нарисована его верхняя грань. Верхняя грань не рисуется, если на этот параллелепипед нужно поставить другой. Контур рисуется установленным типом линии.

Процедура ***PiSlice*** имеет заголовок

PieSlice(x,y:Integer;stangle,endangle,radius:Word)

и предназначена для отображения закрашенного сектора круга с центром **(x,y)** и радиуса ***radius***, начиная с угла ***stangle*** и кончая углом ***tndangle***. Углы задаются в градусах.

Процедура ***FillPoly***:

FillPoly(numpoints:Word;var polypoints)

предназначена для заполнения установленными цветом и шаблоном многоугольника с ***numpoints*** вершинами, координаты которых находятся в области памяти, заданной параметром ***polypoints***. Второй параметр имеет тип

array [1..numpoits] of poittype.

Например,

```
const pentagon:array [1..6] of poittype=
    ((x:50,y:50),(x:75,y:75),(x:62,y:100),(x:48,y:100),
      (x:25,y:75),(x:50,y:50));
  .....
begin
   ......
```

```
    drawpoly(sizeof(pentagon) div sizeof(poittype),pentagon);
    fillpoly(sizeof(pentagon) div sizeof(pointype),pentagon)
    . . . . . .
  end.
```

Процедура ***FillEllipse:***

FillEllipse(x,y:Integer;xRadius,yRadius:Word)

предназначена для рисования закрашенного эллипса. Первые два параметра задают координаты центра, а два другие - длину полуосей вдоль осей Ox и Oy соответственно.

Процедура ***Sector:***

Sector(x,y:Integer;StAngle,EnAngle,xRadius,YRadius:Word)

рисует заполненный эллиптический сектор. Углы задаются против часовой стрелки в градусах. Если **StAngle**=0,а **EnAngle**=360, то рисуется полный эллипс.

2.8.Масштабирование изображения на экране

Получаемое на экране изображение будет несколько искаженным. Это связано с тем, что расстояние между точками по горизонтали отличается от расстояния между точками по вертикали. Для устранения этого искажения необходимо производить масштабирование изображения с использованием коэффициента сжатия, который определяется с помощью процедуры:

GetAspectRation(var xAsp,yAsp:Word),

которая присваивает параметрам ***xAsp, yAsp*** значения типа **Word**, на основании которых можно определить коэффициент сжатия экрана по формуле **xAsp/yAsp**.

Пример 8. Рассмотрим программу отображения на экране квадрата с учетом возможного искажения.

```
uses Graph;
var gd,gm,xsidelen,ysidelen:Integer;
xasp,yasp:Word;
begin
  gd:=Detect;
  InitGraph(gd,gm, '');
  if GraphResult<>grOk theh Halt(1);
  GetAspectRation(xasp,yasp);
  xsidelen:=20;
  ysidelen:=Round(xasp/yasp*xsidelen);
  Rectangle(0,0,xsidelen,ysidelen);
  readln;
  CloseGraph
end.
```

2.9.Графические страницы

Графический вывод можно направить на любую графическую страницу, а затем отобразить соответствующую страницу на экран. Этот метод можно использовать для отображения на экран быстроизменяющихся изображений (примером может служить процесс анимации). Страницу можно сделать активной с помощью процедуры

SetActivePage(Page:Word)

В результате весь последующий графический вывод будет производиться в заданную страницу. Следует помнить, что на экране по умолчанию отображается нулевая страница. Для того, чтобы отобразить содержимое страницы с номером ***Page*** используется процедура

SetVisualPage(Page:Word).

2.10.Запоминание и восстановление изображений

Для построения перемещающихся на экране изображений можно использовать процедуры запоминания и восстановления изображений. В их состав входят следующие.

Процедура определения размера памяти, необходимой для сохранения графического изображения, имеет заголовок

ImageSize(x1,y1,x2,y2:Word):Word.

Функция возвращает величину типа **Word**, значение которой равно числу байтов, необходимых для запоминания изображения в области экрана с координатами **(x1,y1)** и **(x2,y2)**.

Для сохранения в оперативной двоичного образа заданной области экрана используется процедура

GetImage(x1,y1,x2,y2:Word;var BitMap)

Параметры **x1,y1,x2,y2** определяют прямоугольную область экрана, которая запоминается в оперативной памяти. Параметр **BitMap** является нетипизированным. Он задает область памяти, в которой запоминается заданная часть экрана. Размер этой области должен быть не меньше величины, равной <размер памяти, необходимый для сохранения области экрана> + 4. Первые два слова **BitMap** зарезервированы для ширины и высоты области экрана:

Ширина=x2-x1+1

Высота =y2-y1- 1

Для определения нужного размера участка памяти можно использовать функцию ***ImageSize***.

Для восстановления из буфера в ОЗУ на экран двоичного образа изображения используется процедура

PutImage(x,y:Integer;var BitMap;op:Word)

Здесь (x,y) определяет координату левого верхнего угла прямоугольной области экрана - куда будет выводиться изображение из буфера **BitMap**.

Параметр *op* определяет, какой двоичный оператор будет использоваться для вывода двоичного образа области экрана.

Этот параметр может принимать следующие значения:

NormalPut=0 {Операция mov- вывод с затиранием старого изображения}
XorPut =1 {**Xor** - с сохраненим текущего изображения}
OrPut =2 {**Or** }
AndPut =3 {**And**}
NotPut =4 {**Not** - инвертированный вывод}

Каждая константа соответствует двоичной операции, выполняемой при выводе изображения. Указанные операции выполняются попарно к атрибутам выводимой точки и точки, отображенной на экране.

Например, **PutImage(x,y,BitMap,NormalPut)** - выводит на экран двоичный образ, сохраненный в массиве **BitMap**, начиная с точки **(x,y)** на экране. При этом при выводе каждого байта двоичного образа используется инструкция Ассемблера **Mov**. Это означает, что восстанавливаемое изображение затирает ранее выведенное на экране.

PutImage(x,y,BitMap,XorPut) - осуществляет вывод с использованием операции **Xor**. В результате выводимое изображение накладывается на существующее. Это часто используется в мультипликации для стирания изображения, т.к. повторный вывод изображения приводит к его затиранию.

PutImage(x,y,BitMap,NotPut) - инвертирует биты в **BitMap** и выводит полученный двоичный образ. В результате выводимое изображение будет инвертировано.

Следует отметить, что процедура **PutImage** отсечение не выполняет, поэтому - если какая-либо часть изображения не умещается в окне вывода, то оно вообще не выводится.

Пример 9. Составить программу, которая выводит изображение падающего прямоугольного тела.

```
program Pgraph;
 uses Crt,graph;
 const deltat=100;{Константа задержки}
 var P:Pointer;
    size:Word;
    gd,gm,x1,y1,x2,y2,hy,y,ht:integer;
 begin
  writeln('Введите координаты левого верхнего и правого нижнего углов
 прямоугольника);
  write('x1=');readln(x1);
  write('y1=');readln(y1);
  write('x2=');readln(x2);
```

```
 write('y2=');readln(y2);
 write('Шаг смещения по вертикали:');readln(hy);
 gd:=Detect;
 InitGraph(gd,gm,'');
 if GraphResult<>GrOK then
  halt(1);
  setcolor(white);
  setbkcolor(blue);
  cleardevice;
  setfillstyle(1,Yellow);
  Bar(x1,y1,x2,y2);
  Size:=ImageSize(x1,y1,x2,y2);
  GetMem(p,Size);
  GetImage(x1,y1,x2,y2,p^);
  cleardevice;
  rectangle(0,0,getmaxx,getmaxy);
  outtextxy(120,20,'Example');
  y:=y1;
  ht:=y1-y2+1;
  while y+ht<=getmaxy do
  begin
   PutImage(x1,y,p^,NormalPut);
   Delay(deltat);
   PutImage(x1,y,p^,XorPut);
   y:=y+hy;
  end;
  ReadLn;
  CloseGraph
 end.
```

Здесь предполагается, что прямоугольник падает с постоянной скоростью. Для задержки изображения на экране задается константа **deltat**, величина которой зависит от быстродействия компьютера и графической платы.

3. ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1.

Построение статических графических изображений

Варианты заданий:

1. Построить изображения графиков функций:

1) Астроиды

$x = b \cdot cos^3 t, y = b \cdot sin^3 t, t \in x \in [0,2\pi]$

2) Улитки Паскаля

$x = a \cdot cos^2 t + b \cdot cos\, t, y = a \cdot cos\, t \cdot sin\, t + b \cdot sin\, t$

$a > 0, b > 0, t \in [0,2\pi]$

Рассмотреть случаи: $b \geq 2a, a < b < 2a, a > b$

3) Овала Кассини, заданного в полярных координатах

$\rho^2 = c^2 \cdot cos\, 2\varphi \pm \sqrt{c^4 \cdot cos^2 2\varphi + (a^4 - c^4)}$

Рассмотреть случаи $a > c\sqrt{2} > 0, 0 < c < a < c\sqrt{2}, 0 < a < c.$

4) Лемнискаты

$\rho = a\sqrt{2\, cos\, 2\varphi}, a > 0$

где ρ и μ полярные координаты

5) Эпициклоиды

$x = (a + b)cos\, t - a\, cos((a + b)t / a)$

$y = (a + b)sin\, t - a\, sin((a + b)t / a)$

где a,b>0,b/a=p/q, где p,q-целые простые числа, $t \in [0,2\pi]$

5) Циклоиды Диоклеса

$\rho = a(\frac{1}{cos\,\varphi} - cos\,\varphi)$

6) Кардиоиды

$\rho = a(1 + cos\,\varphi)$

7) Трисектрисы

$\rho = a(4\, cos\,\varphi - \frac{1}{cos\,\varphi})$

8) Декартов лист

$x = \frac{3at}{t^3 + 1}, y = \frac{3at^2}{t^3 + 1}, t \in [p,q]$

9) Логарифмическая спираль

$\rho = ae^{b\varphi}$

10) Гипоциклоида

$x = (b - a)sin\frac{a}{b}t - a_1\, sin\frac{b - a}{b}t$

$y = (b - a)cos\frac{a}{b}t + a_1\, cos\frac{b - a}{b}t$

11) Трактриса

$x = a(\cos t + \ln tg\frac{t}{2}), y = a\sin t$

12) Спираль Архимеда

ρ=aφ

13) Параболическая спираль

$\rho^2 = 2p\varphi$

При выводе графиков нужно изобразить оцифрованные оси системы координат

2. Построить области:

1) Нарисовать квадрат с заданной стороной и вписать в него окружность. Закрасить область внутри окружности в заданный цвет.

2) Вывести на экран треугольник с заданными координатами вершин и вписать в него окружность. Область треугольника вне окружности закрасить в заданный цвет.

3) В заданной квадратной области экрана нарисовать фрагмент шахматной доски размером в 5*5 ячеек, раскрасить ее заданными цветами.

4) Вывести на экран изображение трех касающихся окружностей заданных радиусов.

5) Вывести на экран область, заданную системами неравенств:

a) $\begin{matrix} x^2 + y^2 \le R^2 \\ y \ge x^2, y \le 0.75 \cdot R \end{matrix}$ b) $\begin{matrix} y \le x^2 \\ y >= a \cdot x + b, x \le c \end{matrix}$ c) $\begin{matrix} y \ge x^2, x <= 0.5R \\ x^2 + y^2 \le R^2 \end{matrix}$

d) $|x| + |y| \le b, y \le x^2$ e) $\begin{matrix} |x + y| <= b \\ c \le x \le d, x \ge 0 \end{matrix}$ f) $\begin{matrix} y = e^{|x|} \\ x^2 + y^2 \le R^2 \end{matrix}$

6) Построить график функции, заданной массивами координат точек $\{x_i\}, \{y_i\}, i = 1, n$, где n заданная константа.

7) Пусть массив $\{y_{ij}\}, i = 1, n, j = 1.4$, содержит информацию о помесячных продажах фирмой автомобилей j-ой марки. Построить диаграммы, описывающие динамику изменения количества продаж:

а) столбчатую диаграмму(вертикальную)

б) столбчатую диаграмму горизонтальную

в) кольцевую диаграмму

г) в виде совокупности помесячных круговых диаграмм

д) в виде совокупности наложенных графиков, описывающих динамику изменения по каждому виду продукции.

Лабораторная работа № 2.

Пошаговое построение геометрических изображений

Требуется составить программу, которая

1. Выводила бы на экран прямоугольник с заданными координатами вершин (x1,y1) и (x2,y2). При каждом нажатии клавиш управления курсором ⇒⇓⇑⇐ осуществляла бы перемещение прямоугольника на экране, а при каждом нажатии клавиш PgUp, PgDn изменяла бы цвет его закраски на следующий(циклически). Выход из программы по нажатию клавиши ESC.

2. То же, что в предыдущей задаче, но должен изменяться образец заполнения внутренней области прямоугольника.

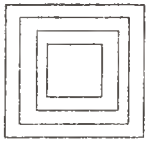
3. Выводила бы на экран окружность заданного радиуса и с заданным центром, со вписанным в нее равносторонним заштрихованным треугольником, а затем по каждому нажатию клавиши PgDn удваивала бы количество вершин вписанного многоугольника, а клавиши PgUp - уменьшала бы в два раза.

4. Строила бы на экране окружность заданного радиуса R с центром (x,y) (x,y,R- целые). При каждом нажатии клавиши '>' ее радиус должен изменяться на заданную величину, а клавиши '<' - уменьшаться. Выход из программы по нажатию клавиши ESC.

5. Выводила бы заданную строку текста, начиная с заданной позиции экрана. При каждом нажатии клавиши Enter шрифт вывода строки должен изменяться на следующий (циклически), а с помощью клавиш управления курсором можно было бы изменять положение строки на экране. Выход из программы по нажатию клавиши ESC.

6. Выводила бы на экран, начиная с заданной позиции, заданную строку текста, заключив ее в двойную рамку. При каждом нажатии клавиш '>' размер символов должен увеличиваться в 2 раза, а при нажатии клавиши '<' - уменьшаться в 2 раза. Выход по нажатию клавиши Esc.

7. Выводила бы на экран картинку.



При этом каждый очередной прямоугольник должен выводиться при нажатии клавиши Enter, а при нажатии клавиши 'D' убирался бы последний построенный. Выход по ESC. Прямоугольники должны выводиться с заданными шагами уменьшения размера его сторон hx, hy.

8. Выводила бы совокупность вложенных концентрических окружностей, аналогично предыдущей задачи.

9. Выводила бы совокупность вложенных равносторонних треугольников, аналогично предыдущим задачам.

10. Строила бы на экране прямоугольник с заданными координатами левой верхней вершины с помощью клавиш управления курсором. При каждом нажатии клавиш ⇒⇓⇐⇑ осуществляла бы построение прямоугольника с про-

извольными сторонами, длина которых зависит от числа нажатий указанных клавиш.

11. Строила бы на экране прямоугольный треугольник с заданными координатами вершины прямого угла с помощью клавиш управления курсором. При каждом нажатии клавиш ⇒⇑⇓ осуществляла бы построение прямоугольника с произвольными сторонами, длина которых зависит от числа нажатий указанных клавиш.

12. Строила бы на экране квадрат с заданными координатами левой верхней вершины с помощью клавиш управления курсором. По нажатии клавиш ⇒⇓⇐⇑ осуществляла бы построение фигуры со сторонами, длина которых зависит от числа нажатий клавиши ⇒.

13. По заданной координате центра фигуры и заданной длине отрезка строила бы на экране по нажатии клавиши «S» - квадрат, клавиши «T» - равносторонний треугольник, клавиши «C» - окружность.

14. Раскрашивала бы кольца, образованные концентрическими окружностями, начиная с центрального круга произвольными цветами, в зависимости от комбинаций нажатых клавиш, которые соответствуют определенным цветам.

15. Демонстрировала бы простейшую игру «Попади из пушки в танк». Траектория полета снаряда регулируется массой снаряда **m** и углом α наклона ствола.

Лабораторная работа № 3.

Вывод движущихся изображений

Требуется составить программу, демонстрирующую:

1. Вращение заданного координатами концов отрезка относительно заданной точки, в плоскости экрана, с заданной угловой скорость. Выход по ESC.

2. Вращение заданного треугольника в плоскости экрана с заданной угловой скоростью.

3. Процесс падения упругого шарика с заданной высоты на абсолютно твердую горизонтальную плоскость с последующими отражениями. Коэффициент отражения равен k. Программа должна завершаться по нажатию клавиши ESC или когда высота поднятия отскакивающего шарика станет меньше заданной величины h.

4. Колебания математического маятника, подвешенного на нити длиной L. Шарик был отклонен от вертикали на угол α и отброшен.

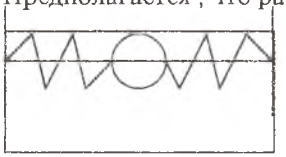
5. Вертикальные колебания шарика массой m, подвешенного на упругой пружине. Известна длина недеформированной пружины L и ее коэффициент упругости C. Шарик был перемещен в начальный момент времени из состояния равновесия, по вертикали вниз на h см.

6. Движение снаряда, который выпущен с начальной скоростью V под углом α к горизонту. Программа должна циклически запрашивать новые значения скорости V и угла α. Выход из программы по клавише ESC.

7. Движение абсолютно упругого шарика в прямоугольной области заданной координатами (x1,y1) и (x2,y2) и ограниченной твердыми стенками. Предполагается, что в начальный момент времени шарик был выпущен из точки с координатами (x,y) со скоростью V. От стенок шарик отражается абсолютно упруго, а силы тяжести отсутствуют.

8. Колебания шарика между двумя упругими пружинами:

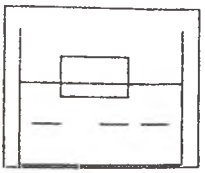
Предполагается , что расстояние между стенками равно L=l1+l2+2r, где l1 и l2 длины нерастянутых пружин, r - радиус шарика, c1 и c2 - заданные коэффициенты упругости пружин.



В начальный момет шарик был выведен из равновесия на d единиц влево и затем отпущен. Силы тяжести отсутствуют.

9. Скатывания цилиндра по наклонной плоскости. Заданы угол наклона плоскости и начальное положение цилиндра. Демонстрация до момента достижения горизонтальной плоскости.

10. Соударения двух шариков с заданными радиусами и массой. Шарики движутся вдоль оси x с заданными начальными скоростями. От вертикальных границ окна отражение идеально упругое.



11. Колебания плавающего тела, погруженного в резервуар с водой. В начальный момент оно находилось в равновесии, а затем было смещено вниз на d единиц и отпущено.Тело имеет форму прямоугольного параллелепипеда. Все необходимые размеры и физические параметры считать заданными.

12. Иллюзию движения кораблика по волнам.

13. Иллюзию движения кораблика на фоне берега.

14. Движение стилизованного автомобиля на фоне деревьев.

15. Бесконечные отражения шарика от стенок произвольного задаваемого своими координатами прямоугольника с «мягкими» стенками. «Мягкие» стенки означают, что угол отражения шарика задается по произвольной формуле в зависимости от угла падения и времени процесса. Выход из программы по клавише ESC.

Корректор Н.В.Голубева
Компьютерная верстка, макет Н.В.Бутырева

ЛР № 020316 от 4.12.96 г. Подписано в печать 17.10.97. Формат 60×84/16.
Бумага офсетная. Печать офсетная. Объем 2 печ.л. Тираж 350 экз.
Издательство "Самарский университет", 443011, г.Самара, ул.Акад.Павлова,1.
Ризограф