КЕМЕРОВСКИЙ ИНСТИТУТ (филиал)
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ТОРГОВО-ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Информатика и программирование

Лебедева Т.Ф.

Управление экраном в графическом режиме

В графическом режиме программист получает возможность управлять каждой точкой (пикселем) экрана. Координаты точки определяются относительно верхнего левого угла. Каждая точка экрана при этом может высвечиваться одним из доступных цветов. Информация о цвете точки хранится в видеобуфере.

Количество цветов зависит от количества бит, отведенных в видеобуфере под одну точку. Рассмотрим основные варианты.

1. «1 точка - 1 бит» - монохромный режим: каждая точка высвечивается либо основным цветом, если в видеобуфере для точки записана 1, либо цветом фона, если в видеобуфере записан 0.
2. «1 точка - 2 бита» - режим с двумя трехцветными палитрами:

Палитра 0:	**Палитра1:**
01-зеленый;	01-светло-голубой;
10 - красный;	10 - сиреневый;
11 - коричневый.	11 - белый.

Если в буфере записана комбинация 00, то точка высвечивается цветом фона.

3. «1 точка - 4 бита» - режим, использующий 16-цветную палитру. В этом режиме в отличие от предыдущих в видеобуфер заносится не цвет точки, а номер регистра палитры, в котором записан нужный цвет.
Для записи цвета используется 6 бит по схеме RGBrgb, причем первые три бита кодируют 2/3 яркости цвета, а вторые три бита - оставшуюся 1/3 яркости.
Так, максимально яркий красный цвет будет кодироваться двумя единицами в первом и четвертом битах:

R G B r g b
1 0 0 1 0 0

Таким образом, каждый цвет имеет четыре градации яркости: 0, 1/3, 2/3, 1, что позволяет кодировать 4^3 = 64 варианта цвета. На экране в этом режиме одновременно может присутствовать не более 16 цветов, так как имеется всего 16 регистров палитры.

4. «1 точка - 8 бит» - режим, использующий палитру на 256 цветов. В этом режиме используется та же идея записи кода цвета не в видеобуфер, а в регистры палитры, что и в предыдущем режиме, но используется 256 регистров палитры, в которых под запись цвета используется 18 бит. Из этих 18 бит под кодирование яркости каждого цвета используется 6 бит, обеспечивая 64 градации яркости каждого цвета. Таким образом, на экране можно одновременно видеть 256 цветов из 64^3 = 262144 возможных.

Количество точек на экране, набор возможных цветов и количество страниц изображения, которые могут одновременно храниться в памяти, зависят от используемых технических средств (типа монитора и блока управления монитором - адаптера) и режимов их работы.

Для каждого типа оборудования, существовавшего на момент разработки среды программирования, среда программирования Turbo Pascal 7.0 включает свою программу управления дисплеем - драйвер, которая обеспечивает работу в нескольких доступных режимах, различающихся количеством точек на экране и количеством страниц, информация о которых может храниться в видеобуфере.

Драйвер – это специальная программа, которая управляет техническими средствами компьютера. Для всех существующих типов адаптеров фирма Borland разработала графические драйверы (они имеют расширение .bgi и находятся на диске в одноименном подкаталоге).

Необходимые процедуры и функции для работы с графикой собраны стандартном модуле – *Graph (более 80 процедур, функций, типов, констант)*.

I. Процедуры и функции переключения режимов управления экраном.

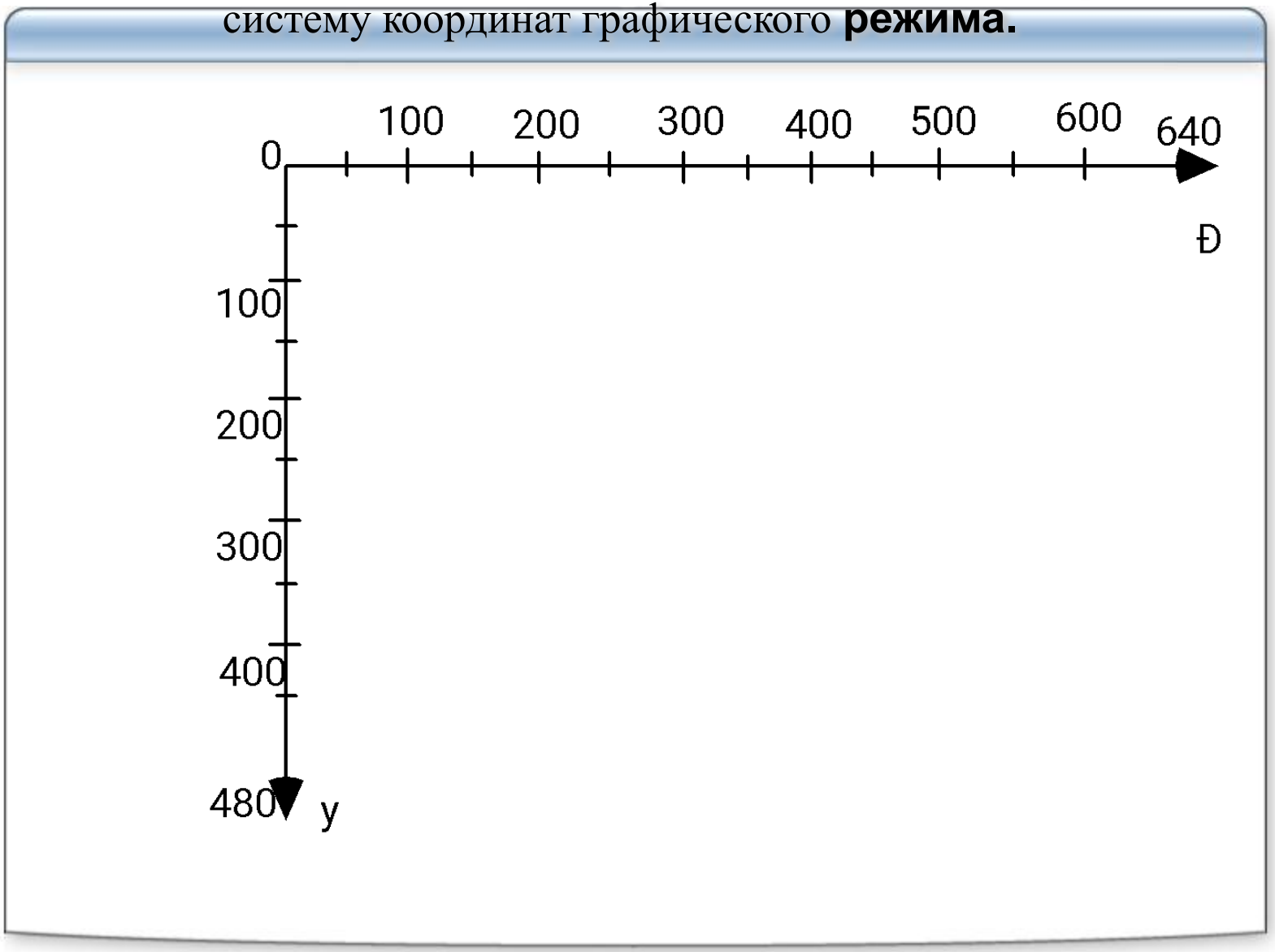
Модуль Graph содержит средства, обеспечивающие различные варианты переключения текстового и графического режимов.

1. Процедура **InitGraph(Var driver, mode:integer; path:string)** – переключает экран в графический режим. При вызове процедуры следует объявить специальные переменные, куда занести константу драйвера и константу режима, и указать эти переменные в качестве параметров процедуры. Существует возможность запросить автоматическое определение драйвера и режима: для этого необходимо вместо константы драйвера в переменную, используемую в качестве параметра, записать константу detect = 0.

Целая переменная *mode* задает режим работы графического адаптера. Многие адаптеры могут работать в нескольких режимах.
Например, переменная *Mode* в момент обращения к InitGraph может иметь одно из следующих значений для адаптера VGA: VGALo =0; VGAMed = 1; VGAHi = 2;
Параметр path должен описывать путь, определяющий местоположение файла, который содержит требуемый драйвер (обычно все драйверы находятся в подкаталоге BGI основного каталога среды Turbo Pascal). Если драйвер находится в текущем каталоге, то в качестве параметра передается пустая строка - ".
Для нашего графического режима обычно значение максимальной координаты X равно 640 пикселям, а Y– 480 пикселям.
2. Функция **GraphResult: integer** - возвращает номер ошибки, обнаруженной при инициализации графического режима.

3. Функция **GraphErrorMSG(ErrNum:integer):string** - позволяет по номеру ошибки определить ее причину.

Чтобы точно строить изображение на экране, Вам нужно сначала нарисовать, как это должно выглядеть на бумаге. Для этого начертите в тетради систему координат графического **режима.**



Полностью инициализация графического режима может быть выполнена следующим образом:

```
Var driver,mode,error: integer;
Begin
driver: =detect; {или driver: =0;}
InitGraph (driver, mode, 'd: \BP \BGI');
error: =GraphResult;
If error <>0 then {если обнаружены ошибки}
begin
WriteLn('Ошибка инициализации графического режима',
GraphErrorMSG(error));
Halt(1); {аварийное завершение программы}
end;
{работа в графическом режиме}...
```

4. Процедура **CloseGraph** - завершает работу в графическом режиме: выгружает драйвер и восстанавливает текстовый режим. Если завершить программу, не выходя из графического режима, то нормальная работа MS DOS нарушается, так как MS DOS функционирует в текстовом режиме.

5. Процедура **RestoreCrtMode** - осуществляет временный возврат в текстовый режим с сохранением графического изображения в видеобуфере.

6. Процедура **SetGraphMode(mode:integer)** - осуществляет возврат в графический режим после временного выхода, осуществленного процедурой RestoreCrtMode.

7. Функция **GetGraphMode:integer** - возвращает номер активного графического режима.

Таким образом, временный выход в текстовый режим обычно оформляется следующим образом:

RestoreCrtMode; {переход в текстовый режим}
... {выполнение обработки в текстовом режиме}
SetGraphMode(GetGraphMode); {возврат в графический режим}

II. Процедуры и функции управления цветом. Модуль Graph содержит специальные процедуры и функции управления цветом изображения.

1. Процедура **SetBkColor(Color:word) -** устанавливает новый цвет фона. При инициализации графического режима видеобуфер очищается (обнуляется). При этом все точки оказываются связаны с 0 регистром палитры, который и считается регистром, хранящим цвет фона. Процедура SetBkColor записывает в 0 регистр палитры цвет, указанный в параметре Color.

2. Процедура **SetColor(ColorNum:word) -** объявляет цвет в регистре с номером ColorNum текущим цветом рисования.

III. Процедуры и функции управления типом, толщиной линии и видом штриховки. Модуль Graph содержит средства, позволяющие управлять типом и толщиной линии, а также образцом закраски замкнутых контуров.

1. Процедура **SetLineStyle(style, pattern, thickness: word) -** устанавливает стиль **style** или образец **pattern** линии и ее толщину **thickness**.

Для установки стиля используют следующие константы:

SolidLn=0; {сплошная} DottedLn=l; {точечная}

CenterLn=2; {штрихпунктирная} DashedLn=3; {пунктирная}

UserBitLn=4; {определенная программистом в образце}

Если указаны стили 0..3, то образец игнорируется. Если установлен стиль 4, то вид линии определяется образцом.
Толщину линии можно установить одинарной и тройной, соответствующие константы выглядят следующим образом:
NormWidth=l; {нормальная толщина линии}
ThickWidth=3; {тройная толщина линии }
2. Процедура **SetFillStyle(fillstyle, color: word)** - устанавливает образец fillstyle и цвет color штриховки замкнутых контуров. Для определения образцов штриховки можно использовать следующие константы:
EmptyFill=0; {без штриховки - прозрачный контур}
SolidFill=1; {сплошная заливка}
LineFill=2; { горизонтальные линии}
LtSlashFill=3; {//// тонкие линии}
SlashFiU=4; {\\\\ - толстые линии}
BkSlashFill=5; {\\\\ - толстые линии}
LtBkSlashFill=6; {////- тонкие линии}
HatchFill=7; {-f+++ - клетка горизонтальная}
и т.д.

3. Процедура **FloodFill(x, y, color: word)** - закрашивает текущим образцом и цветом закраски, установленными процедурой SetFillStyle, область, ограниченную контуром цвета color, внутри которой находится точка с координатами (x,y). Если контур не замкнут, то будет закрашен весь экран

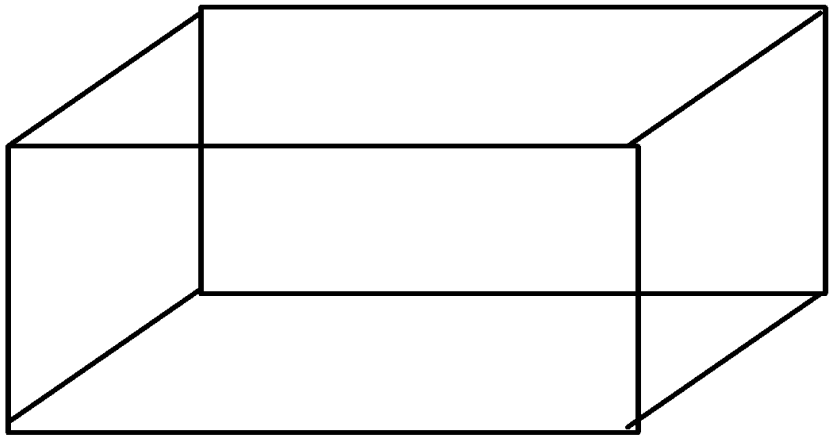
IV. Процедуры и функции рисования точек и линий, фигур

Модуль Graph предоставляет программисту большой набор процедур и функций, рисующих различные примитивы (простейшие фигуры, размеры и местоположение которых определяются параметрами).

1. Процедура **PutPixel(x, y, color: word)** - рисует точку цветом, определенным color, в точке экрана с координатами (x, y).
2. Функция **GetPixel(x,y:word):word** – возвращает цвет точки с указанными координатами.
3. Функция **MoveTo(x,y:word):word-** задает положение текущей точки - точки, используемой в качестве исходной при рисовании линий или выводе текста.
4. Процедура **MoveRel(dx,dy:word)** - задает положение текущей точки относительно ее предыдущего положения.
5. Функции **GetX:word** и **GetY:word-** возвращают соответственно координаты x и y текущей точки.

6. Функции **GetMaxX:word** и **GetMaxY:word** - возвращают соответственно максимальные размеры экрана в точках для текущего режима.
7. Процедура **Line(x1,y1,x2,y2:word)** - рисует линию из точки (x1,y1) в точку (x2,y2).
8. Процедура **LineTo(x,y:word)** - рисует линию из текущей точки в точку (x, y).
9. Процедура **LineRel(dXydy:word)** - рисует линию из текущей точки в точку, отстоящую от текущей на dx, dy.
10. Процедура **Rectangle(xl,yl,x2,y2:word)** - рисует прямоугольник по координатам левого верхнего и правого нижнего углов.
11. Процедура **Bar(xlyyl,x2,y2:word)** - рисует заштрихованный прямоугольник с заданными координатами текущим цветом и закрашивает его текущим образцом закраски.
12. Процедура **Bar3D(xl,yl,x2,y2, depth:word; top: boolean)** - рисует параллелепипед, у которого: размер передней грани определяется координатами x1, y 1, x2, y2; глубина **depth** обычно задается равной 25% ширины, а параметр **top** определяет, рисовать ли верхнюю грань (да, если true, и нет, если false). Верхнюю грань не рисуют, если необходимо изобразить несколько параллелепипедов, стоящих друг на друге. Передняя грань закрашивается текущим образцом и цветом закраски.

Bar3D (x1,y1,x2,y2, Depth, Top), где переменные X1, X2, Y1, Y2 типа Integer, Depth типа Word, а Top типа Boolean – рисуется параллелепипед, закрашенный текущим стилем и цветом. Здесь переменные X1, X2, Y1, Y2 являются координатами левого верхнего и правого нижнего углов передней грани, Depth – ширина боковой грани (отсчитываются по горизонтали), Top – признак включения верхней грани: TopOn = true – верхняя грань изображается, TopOff = false – верхняя грань не изображается.



13. Процедура **DrawPoly(numPoints:word; var PolyPoints)** - рисует ломаную линию. Первый параметр определяет количество вершин ломаной, а второй - координаты этих вершин в массиве элементов специального типа PointType:

Type PointType = record
x,y:word;
end;...

Например:

Var MP:array[1..5] of PointType; {объявление массива координат точек}
DrawPoly(5,MP); {рисуем ломаную линию}

14. Процедура **FillPoly (numPoints:word; Var PolyPoints)** - рисует закрашенный многоугольник с указанными вершинами.

15. Процедура **Circle (x, y, radius:word)** - рисует окружность заданного радиуса radius с центром в точке (x, y).

16. Процедура **Arc(x,y, stangle, endangle, radius: word)** - рисует дугу окружности указанного радиуса radius от угла stangle до угла endangle. Углы отсчитываются от положительного направления оси абсцисс против часовой стрелки и указываются в градусах.

17. Процедура **Pieslice (x, y, stangle, endangle, radius:word)** - рисует заштрихованный сектор или окружность (если конечный угол равен начальному). Углы отсчитываются от положительного направления оси абсцисс против часовой стрелки и указываются в градусах.

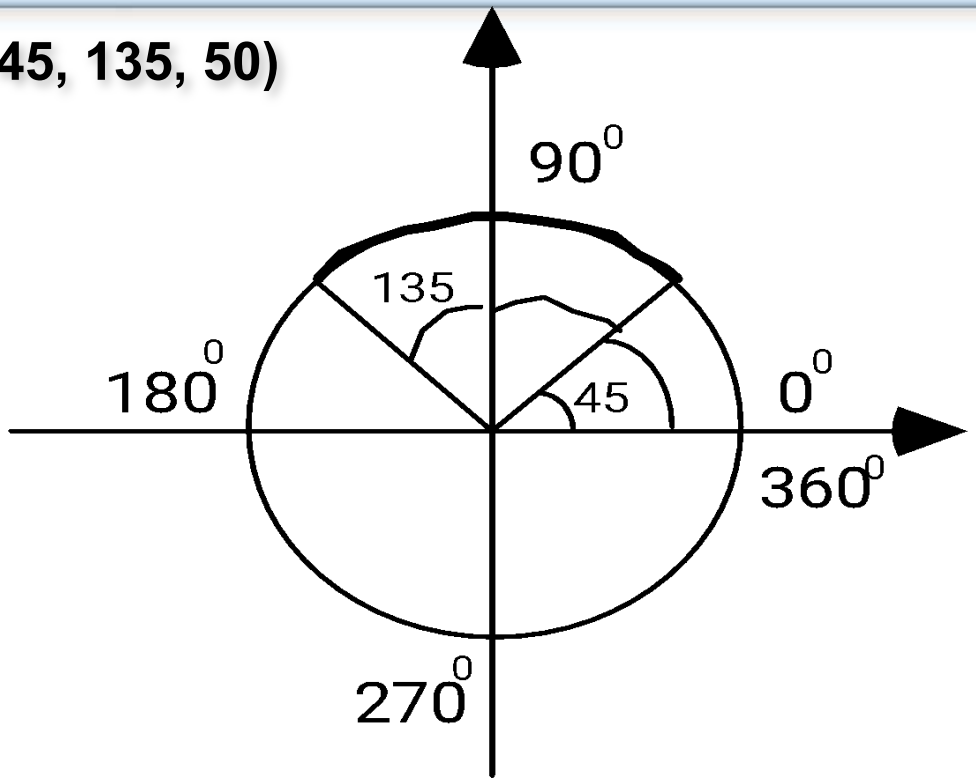
18. Процедура **Ellipse (x, y, stangle, endangle, radiusX, radiusY: word)** - рисует эллипс или его дугу.

19. Процедура **Sector (x, y, stangle, endangle, radiusX, radiusY: word)** - рисует заштрихованный эллипс или его сектор.

20. Процедура **FillEllipse (x, y, radiusX, radiusY: word)** - рисует заштрихованный эллипс.

Например, чтобы начертить дугу (смотри рисунок) от окружности радиуса 50 пикселей и координатами центра (100, 230) надо набрать команду:

Arc (100, 230, 45, 135, 50)



V. Процедуры и функции управления текстом. Текст в графическом режиме может быть выведен процедурами Write и WriteLn, но при этом управление выполняется аналогично текстовому режиму. Специально для работы с текстом в графическом режиме модуль Graph предлагает следующие процедуры и функции.

1. Процедура **SetTextStyle(font,direction,charsize:word)** - устанавливает текущий шрифт **font**, направление вывода строки **direction** и размер шрифта **charsize.**

Для установки шрифта используют специальные константы:

DefaultFont=0; {обычный}
TriplexFont=1; {полужирн ы й}
SmallFont=2; {мел кий}
SanserifFont=3; {прямой}
GothicFont=4; {готический}

Направление вывода может быть либо горизонтальным, либо вертикальным: HorizDir=0; {горизонтальное направление}
VertDir=I; {вертикальное направление}

Размер шрифта указывается значением, на которое необходимо умножить исходный размер шрифта (8x8 точек). Таким образом, шрифт можно увеличивать в целое число раз.

2. Процедура **SetTextJustify(goriz, vert :word)** - устанавливает способ выравнивания для горизонтального goriz и вертикального vert направлений вывода. Способ выравнивания определяется относительно некоторой точки с помощью специальных констант:

LeftText=0; {точка слева от текста}
CenterText=1; {точка по центру текста при горизонтальном направлении вывода}
RightText=2; {точка справа от текста}
BottomText=0; {точка под текстом}
CenterText=1; {точка по центру текста при вертикальном направлении вывода}
TopText=2; {точка над текстом}

3. Процедура **OutText (st:string)** - выводит строку, выравнивая ее заданным способом относительно текущей точки.

4. Процедура **OutTextXY(x,y:word; st: string)** - выводит строку **st**, выравнивая ее в соответствии с установленными режимами вертикального и горизонтального выравнивания относительно точки (x, y).

Пример 1. Приведем пример программы, при выполнении которой создается интересный визуальный эффект путем заполнения экрана множеством точек различных цветов.

```
Uses Crt, Graph;
Var   Driver, Mode : integer; Color: word; X,Y:word;
Procedure GrInit;
  Begin
Driver:= Detect; {автораспознанание драйвера} InitGraph(Driver, Mode,'');
If GraphResult<>0 then Begin Writeln ('Ошибка  инициализации!');
Writeln ('Работа  программы  прервана'); Halt (1) {остановить выполнение программы с кодом завершения — 1}
          End;  End;
Begin  GrInit;
 Color: =0; Randomize;
  Repeat
{выберем случайным образом координаты точки}
   X: = Random(GetMaxX);  Y: = Random(GetMaxY);
   PutPixel(X,Y, Color); {вывод точки}  Inc(Color); {изменение цвета}
{проверим, не превышает ли значение цвета максимального значения, которое определяет функция GetMaxColor}
    If Color = GetMaxColor Then Color: =0;
 Until  KeyPressed; {повторять до нажатия любой клавиши}
ClearDevice;
CloseGraph
End.
```

Пример 2 Будильник.

```
program Budil;
uses graph;
var   grDriver:integer;    grMode:integer;
Begin
grDriver:=Detect;   InitGraph(grDriver,grMode,'');
{Смена цвета фона}
SetBkColor(14);
{Смена цвета линии}
SetColor(5);
{Рисование окружности}
Circle(250,180,60);  Circle(270,180,60);
Circle(210,235,7);    Circle(310,235,7);
Circle(260,180,7);     Circle(260,115,10);
SetColor(9);
{Рисование линий}
Line(260,180,260,140);    Line(260,180,270,150);
Circle(260,230,5);          Circle(260,125,5);
Circle(230,180,5);          Circle(290,180,5);
readln;
End.
```

Пример 3. Программа рисует 5 концентрических окружностей в центре экрана.

```
Program Krug2;
Uses Graph;
Var Gd, Gm, rad,xc,yc:integer;
begin Gd:=Detect; { функция InitGraph должна выполнить } { автоопределение
типа монитора (Gd) и его } { максимального разрешения (Gm)}
InitGraph(Gd,Gm,''); { инициализация графического режима }
if GraphResult <> 0 then { в случае ошибки инициализации } begin
Writeln('Ошибка инициализации графического режима');
Writeln('В каталоге программы должен присутствовать драйвер
egavga.bgi'); Writeln('или укажите путь к нему в IniGraph(Gd,Gm,<путь>)');
Halt(1); end;
Case Gd of { анализируем тип дисплея } { и вычисляем координаты центра
экрана }
 9: begin { VGA монитор } xc:=(640-1) div 2; yc:=(480-1) div 2; end;
3: begin { EGA монитор } xc:=(640-1) div 2; yc:=(350-1) div 2; end; end;
 for   rad:=1 to 5 do Circle(xc, yc, rad*20); { рисуем окружности } Readln;
CloseGraph; { возврат в текстовый режим }
 end.
```

Пример 4. Программа выводит десять вложенных друг в друга прямоугольников, после чего циклически меняет цвет фона. Для выхода из программы достаточно нажать на любую клавишу.

```
Uses Graph, CRT;
const   NC: array [0..15] of String [12] = ('Black', 'Blue', 'Green', 'Cyan‘ ,'Red', 'Magenta',
 ' Brown', 'LightGray‘ ,'DarkGray', 'LightBlue',  'LightGreen’,'LightCyan1,'LightRed',
'LightMagenta‘ ,'Yellow', 'White');
var  d, r, e, k, color, dx, dy: Integer;
Begin {Инициируем графику}
d := Detect; InitGraph(d, r, ' ') ;
e := GraphResult; if e <> grOK then WriteLn(GraphErrorMsg(e))
else  begin {Выводим текст в центре экрана}
OutTextXY(200,GetMaxY div 2  ,'BACKGROUND COLOR');
dx := GetMaxX div 30; {Приращение длины}
dy := GetMaxY div 25; {Приращение высоты}
for k := 0 to 9 do        {Выводим 10 прямоугольников}
Rectangle(k*dx,k*dy,GetMaxX-k*dx,GetMaxY-k*dy);
color := black; {Начальный цвет фона}
repeat {Цикл смены фона}
SetBkColor(color) ; SetFillStyle(0,Color);
Bar(345,GetMaxY div 2, 440 , GetMaxY div 2+8);
OutTextXY(345,GetMaxY div 2,NC[color]); delay(1000); inc(color);
if color > White then color := Black until KeyPressed;
if ReadKey=#0 then  k := ord(ReadKey);
CloseGraph  end
end.
```

Построение графика элементарных функций

Пусть дана некоторая функция y = f(x). Рассмотрим прямоугольную область координатной плоскости, которая определяется значениями:

xMin ≤ x ≤ xMax; yMin ≤ y ≤ yMax;

Эту область назовём областью функции, а координаты (x, y) — мировыми координатами. Графиком функции f(x) является некоторое конечное множество точек (x_k,y_k), $y_k = f(x_k)$, k=1, ..., n из области функции.

Выделим на экране компьютера прямоугольную область, которую будем называть окном, в котором поместим точечный график функции. В графическом режиме экран имеет другую систему координат U и V.

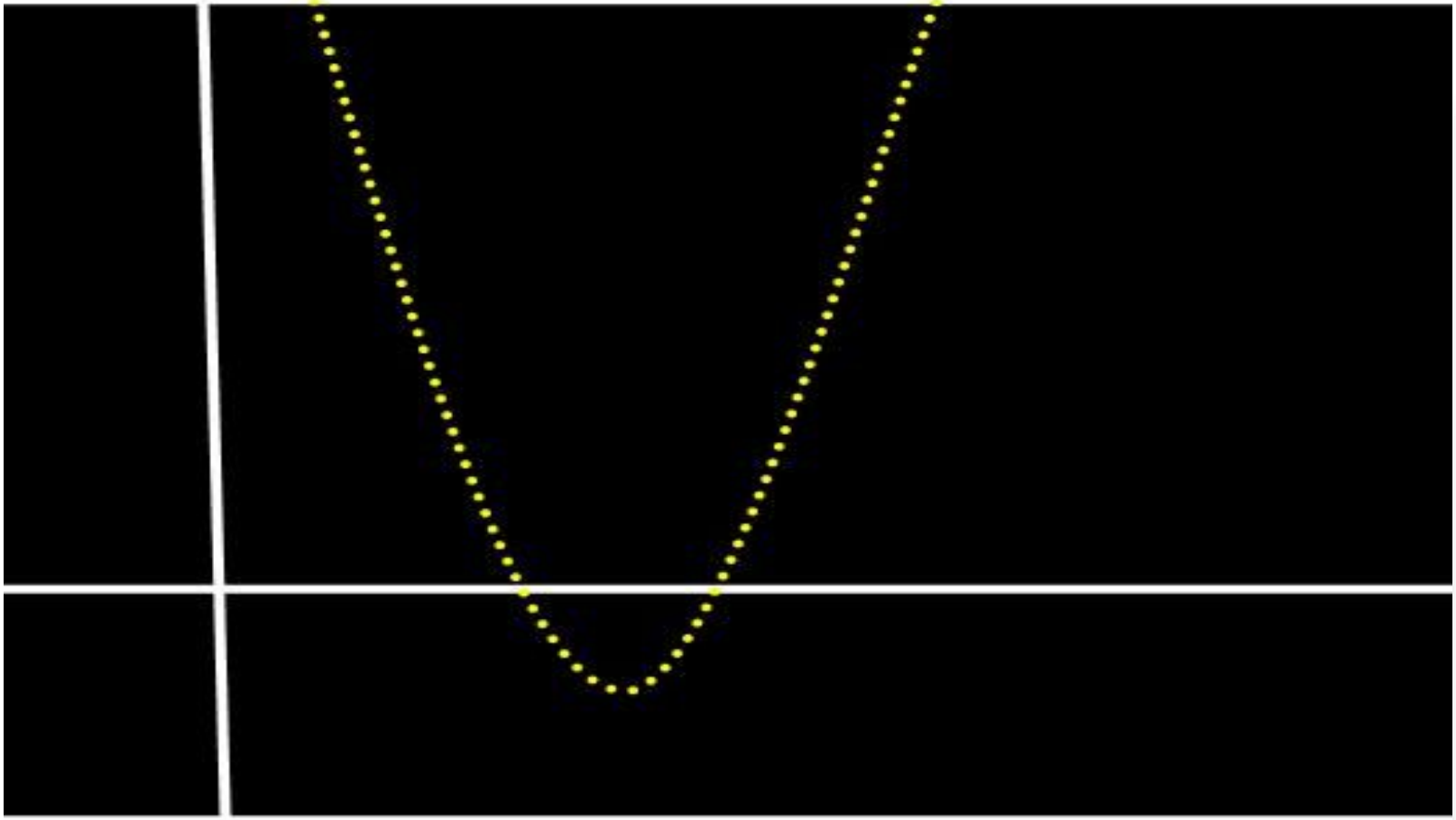
Шаг 1. Определим исходные данные uMin, uMax, vMin, vMax, xMin, xMax, yMin, yMax, а также количество точек n=uMax — uMin и шаг изменения x xStep=(xMax — xMin)/n.

Шаг 2. Нарисуем рамку, обрамляющую окно (это будет прямоугольник, левый верхний угол которого имеет координаты (uMin, vMin), а правый нижний угол — (uMax, vMax).

Шаг 3. Вычислим кооэффициенты сжатия px и py по формулам .

Шаг 4. Нарисуем оси координат.

Шаг 5. Строим график функции y = f(x) из n точек, преобразовывая x в u, а y - в v.



Приведем программу построения графика функции $y=x^2 — 5x + 6$, используя следующие исходные данные:
xMin= —2; yMin= —2; xMax = 10; yMax=10.

```
uses Graph;
Const   uMin=50;  uMax=600;  vMin=50;  vMax=450;
{Окно графика функции}
Var  grDriver: integer;
grMode: integer;
xStep:real; {Шаг по x}
n, k: integer; {Число точек графика}
x, y, px, py:real;
xMin,  yMin, yMax,  xMax: real;{Область функции}
Procedure GrInit;
Begin
grDriver:=Detect;
InitGraph(grDriver,grMode,'');
IF GraphResult<>0 then
begin
    Writeln('Ошибка инициализации!');
    writeln('Работа программы прервана');
    Halt(1)
end
End;
```

```
Function xScr(x: real):integer;
{Преобразование координаты x в u}
begin
xScr:=Round((x-xMin)*px+uMin);
end;
Function yScr(y:real):integer;
{Преобразование координаты y в v}
begin
yScr:=Round((y-yMax)*py+vMin);
end;
Function f(x:real):real;
{Вид функции}
begin
f:=X*X-5*X+6;
end;
```

```
Begin
{Ввод исходных значений области функции}
write('xMin='); readln(xMin);
write('xMax='); readln(xMax);
write('yMin='); readln(yMin);
write('yMax='); readln(yMax);
px:=(uMax-uMin)/(xMax-xMin);
py:=-(vMax-vMin)/(yMax-yMin);
GrInit;
{Построение рамки прямоугольника}
SetLineStyle(0,0,3);
Rectangle(uMin,vMin,uMax,vMax);
{Построение осей координат}
SetLineStyle(0,0,1);
If (xMin<0) and (xMax>0) then
Line (xScr(0),vMin,xScr(0),vMax);
If(yMin<0) and(yMax>0) then
Line(uMin,yScr(0),uMax,yScr(0));
```

```
{Определение количества точек графика}
n:=uMax-uMin;
{Вычисление шага}
xStep:=(xMax-xMin)/n;
x:=xMin;
{Вывод графика в виде n точек желтого цвета, преобразовывая при
этом x в u, y в v}
for k:=1 to n do
   begin
      y:=f(x);
      if(y> yMin) and (y<yMax) then PutPixel(xScr(x),yScr(y),Yellow);
     x:=x+xStep
end;
readln;
Closegraph
End.
```

Этапы проектирования программ:

I. техническое задание;
II. разработка технического проекта;
III. разработка рабочего проекта;
IV. отладка отдельных модулей и программы в целом;
V. разработка документации

I. Техническое задание (примерный план)

1.Организационно-экономическая сущность задачи:

- наименование задачи, место ее решения;
- цель решения;
- назначение (для каких объектов подразделений и пользователей она предназначена);
- периодичность решения и требования к срокам решения;
- источники и способы поступления данных;
- потребители результатной информации и способы ее отправки
- информационная связь с другими задачами.

2. Описание исходной (входной) информации:

- перечень исходной информации;
- формы представления (электронная форма или бумажный документ) по каждой позиции перечня; примеры заполнения форм (документов); описание структурных единиц информации (каждого элемента данных, реквизита);
- способы контроля ввода исходной информации;

3. Описание выходной информации:

- перечень результатной информации;
- формы представления, (печатная сводка, видеограмма, машинный носитель и его макет и т.д.);
- перечень пользователей результатной информации (подразделение и персонал)
- перечень регламентной и запросной информации;
- описание структурных единиц информации (каждого элемента данных, реквизита) по аналогии с исходными данными;
- способы контроля результатной информации;
- контроль разрядности;
- контроль интервала значений реквизита;
- контроль соответствия списку значений.

4.Описание алгоритма решения задачи (последовательности действий и логики решения задачи):

• описание способов формирования результатной информации с указанием последовательности выполнения логических и арифметических действий;

• описание связей между частями, операциями, формулами алгоритма;

• требования к порядку расположения (сортировке) ключевых реквизитов, например; по возрастанию значений табельных номеров.

5.Описание используемой условно-постоянной и справочной информации:

• перечень единиц условно-постоянной информации, классификаторов, справочников, таблиц, списков с указанием их полных наименований;

• формы представления;

• описание структурных единиц информации (по аналогии с исходными записями;

• способы взаимодействия с переменной информацией;

• содержание справочной информации, ее структура и способы обращения к ней.

Описание данных (входной, выходной и справочной информации) свести в таблицу 1.

Таблица 1 - Описание данных, используемых в программе.

Имя переменной	Физический смысл переменной	Назначение переменной	Ограничения

В столбце «*Назначение переменной*» может быть указан один из трёх вариантов:

- **исходные данные;**
- **вспомогательная переменная;**
- **результат.**

В столбце «*Ограничения*» необходимо указать тип переменной и ее вид (простая или элемент массива или записи), а также ограничения на значения, например, «*значение больше нуля*».

II. Разработка технического проекта

На данном этапе выполняется комплекс наиболее важных работ:

- конкретизируются данные об объекте программы,
- производится декомпозиция задачи,
- разрабатывается детальный алгоритм обработки данных.

Задачи обладают внутренней структурой.

Необходимо разбить сложную задачу на структурные элементарные составляющие и задать их комбинирование.

Далее требуется детализация задачи, т.е. переход от описания задачи в крупных понятиях к стоящим за этими понятиями объектам более низкого уровня.

Структурная методология включает следующие методы:

- Метод нисходящего проектирования («сверху-вниз»);
- Метод восходящего проектирования («снизу-вверх»)

Метод нисходящего проектирования («сверху-вниз») **– подход функциональной декомпозиции на основе 2 стратегий:**

- **пошагового уточнения, при котором на каждом следующем этапе декомпозиции определяются модули очередного более низкого уровня;**
- **анализа сообщений, при котором анализируются потоки данных, обрабатываемых модулями.**

Метод восходящего проектирования («снизу-вверх») **– подход, при котором в первую очередь определяются вспомогательные модули, которые потребуются при проектировании программы.**

Программные продукты обладают внутренней структурой, образованной взаимосвязанными программными модулями

Модуль **– это самостоятельная часть программы, имеющая определенное назначение и обеспечивающая заданные функции обработки.**

При модульном подходе работа программного продукта представляется в виде совокупности действий по преобразованию данных от «исходных» к «выходным». Под исходными данными подразумеваются данные, которые задаются об объекте при инициализации программы.

При проектировании программы необходимо разделять ее на все меньшие и меньшие подпрограммы, т.е. производить декомпозицию задачи. Под *декомпозицией* подразумевается разделение задачи на составляющие модули.

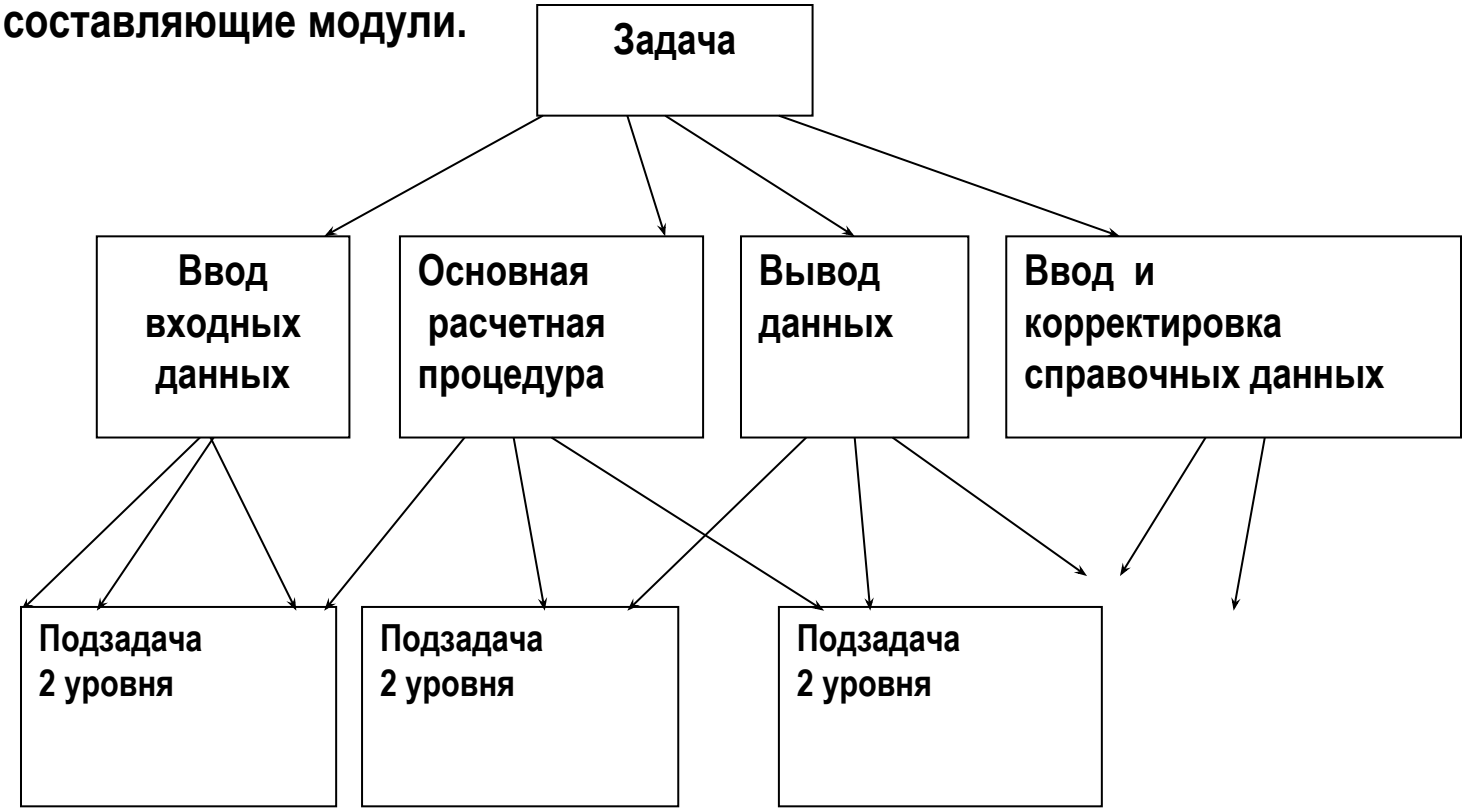


Рис. 1 Примерный состав модулей, выделяемых в программном продукте при декомпозиции

На этом этапе также необходимо:

- установить какие справочные данные требуются для решения поставленной задачи,
- разработать форму представления и ввода для них.
- определить в виде каких объектов данных будут представлена используемая информация.

Под *объектом данных* программы подразумевается переменные, структуры (записи), массивы, файлы, используемые в программе.

Обычно объекты данных, используемые в программе, формируются на основе объектов предметной области, участвующих в задаче.

Выделенные подзадачи (рис.1) являются прообразами будущих программных модулей.

Для разработки программных модулей необходимо разработать алгоритм для каждой выделенной подзадачи.

На рис. 2 приведена типовая структура программного продукта, состоящего из отдельных программных модулей, процедур, встроенных функций.

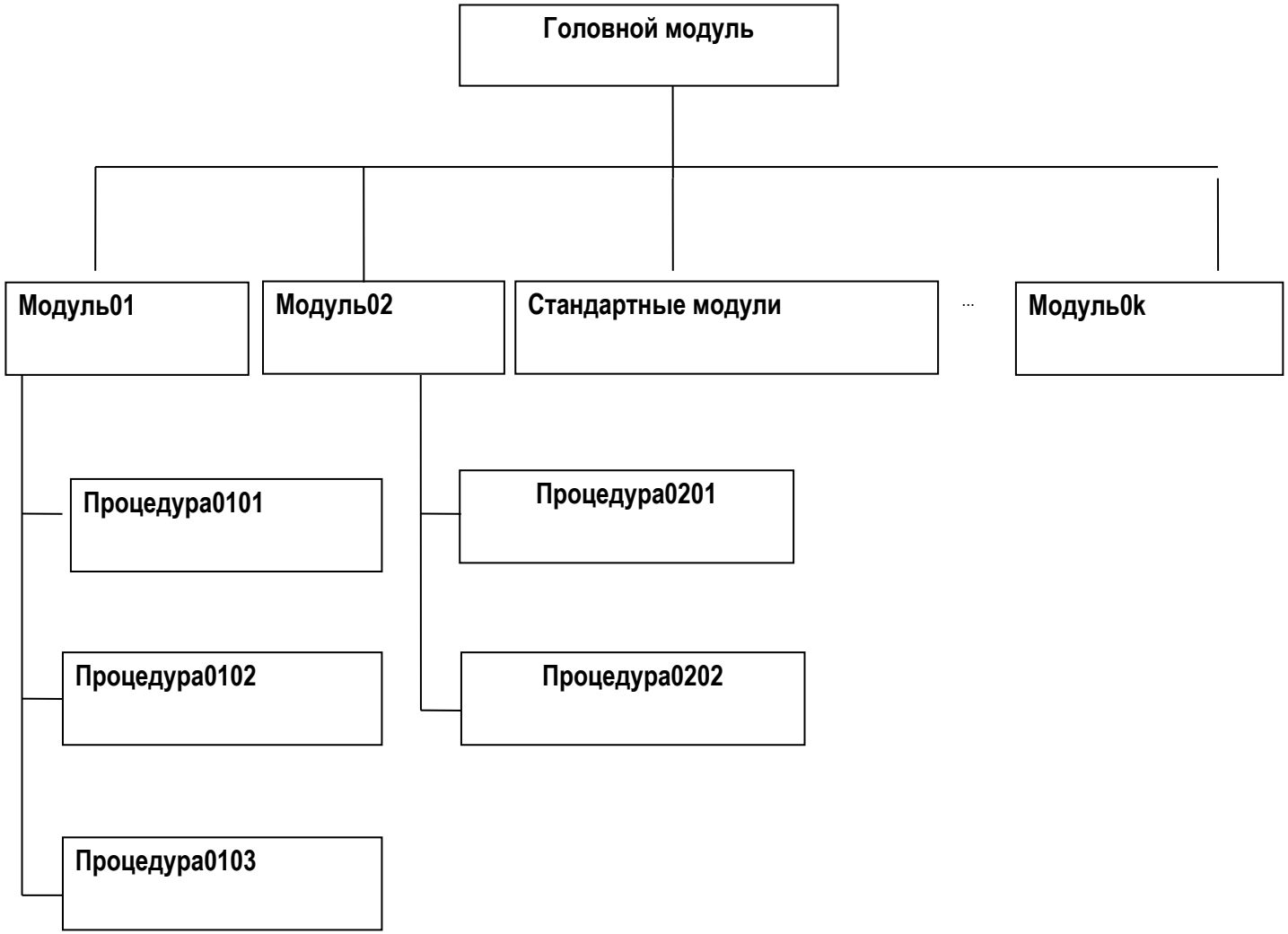


Рис. 2 Типовая структура программного продукта

Среди множества модулей различают:

- ***головной модуль*** **(вызывающая программа) – управляет запуском программного продукта (существует в единственном числе);**
- **управляющий модуль – обеспечивает вызов других модулей на обработку;**
- **рабочие модули выполняют функции обработки;**
- **сервисные модули и библиотеки, утилиты – осуществляют обслуживающие функции.**

На данном этапе необходимо, прежде всего, для каждой подзадачи определить выполняемые в ней действий, то есть какие основные операции по преобразованию данных от установленных входных к выходным данным выполняются в модуле. Необходимо выбрать тип используемого алгоритма и произвести его детальную разработку.

Общие рекомендации при декомпозиции:

- не пытайтесь разбивать на модули очень простые задачи;
- рекомендуемый размер модуля от 15 до 60 программных строк;
- обязательно выделяйте в виде модуля ту часть программы, которая может быть полезна в других программах или в нескольких частях данной программы;
- переменные, общие для нескольких модулей, должны быть определены в вызывающей программе;
- стремитесь к независимости модулей.

Выделим преимущества модульного программирования:

1. единичный модуль легче написать, отладить, проверить;
2. один модуль можно использовать в разных частях программы и в других программах;
3. изменения в программе могут затрагивать только один модуль, а не всю программу;
4. разработку отдельных модулей можно поручить различным людям.

***Пример*. – Разработка информационно-поисковой системы «Склад».**

Созданная программа должна обеспечивать:

- **ввод исходных данных;**
- **редактирование и модификацию данных;**
- **просмотр результатов;**
- **выдачу данных по запросам пользователя (поиск товара по наименованию, подсчет общей стоимости продуктов);**
- **выход в ОС.**

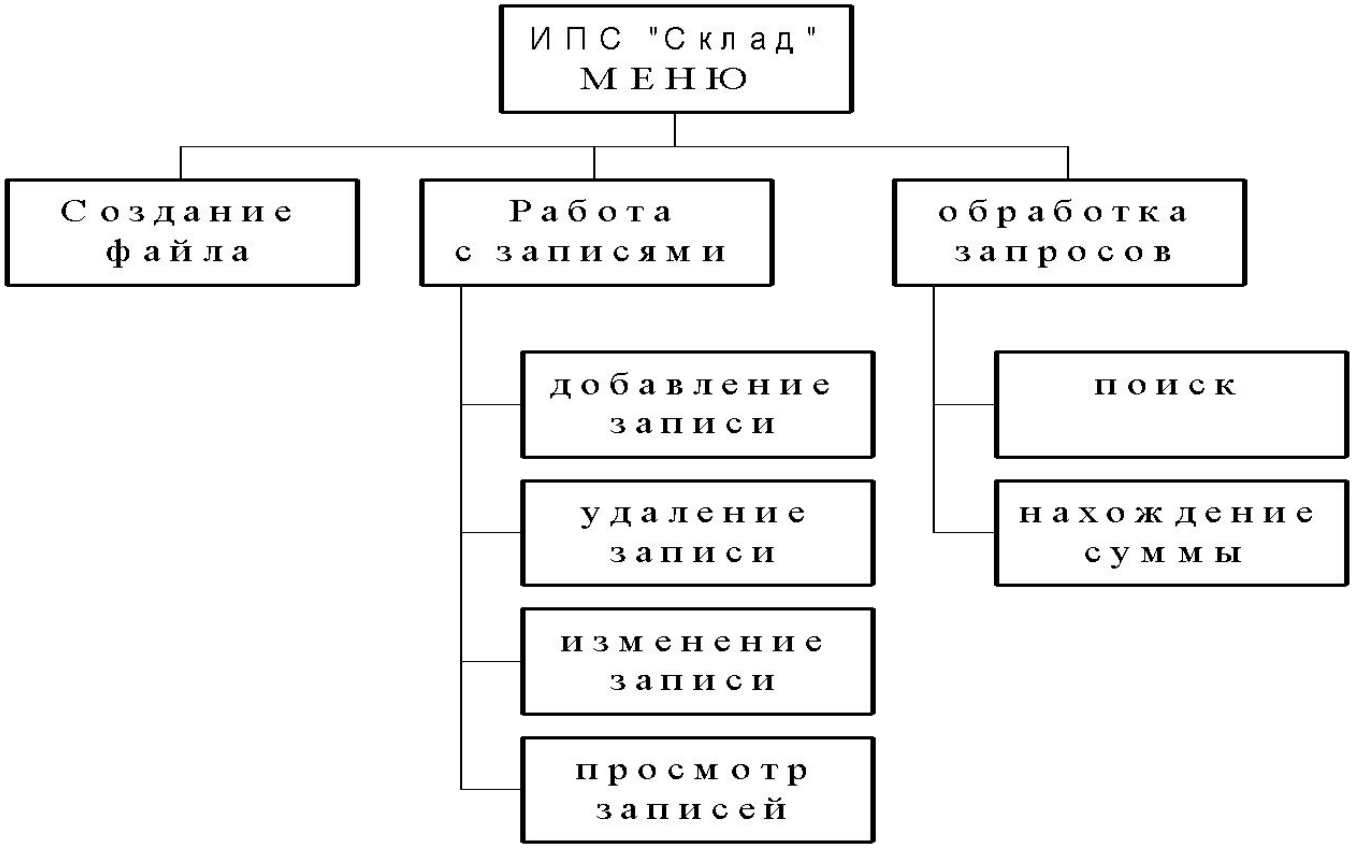
Выделим сущность предметной области - *Товар*. В качестве объекта данных, соответствующего сущности предметной области, выбираем структуру данных – *запись*, включающую поля:

- ***номер,***
- ***наименование,***
- ***стоимость единицы товара,***
- ***количество единицы товара.***

Для хранения и организации последовательной обработки информации о товарах наиболее удобной структурой представляется файл из записей.

Зададим описания основных структур данных:

```
Type tovar =record
nom: 0..100; {номер или индекс товара}
 nt: string[20]; {наименование товара}
 ed_iz : string[10]; {единица измерения }
 s_ed, kol : real; {цена 1 единицы, количество единиц}
     end;
tov=tovar;
file_type= file of tov; {файл из записей}
var t :tov; { описание переменной типа запись}
copf, f: file_type; { описание двух переменных типа файл}
p: byte; {пункт меню}
```



На рис.3 приведена примерная структурная схема информационно-поисковой системы (ИПС) «Склад», полученная в результате декомпозиции поставленной задачи на подзадачи первого и второго уровней.

III. Разработка рабочего проекта

На данном этапе выполняется описание программы, разработка программных модулей - собственно программирование или создание программного кода на языке программирования.

Описание программы должно включать:

- общие сведения;
- функциональное назначение;
- описание логической структуры;
- описание данных;
- используемые технические средства.

В общих сведениях о программе необходимо указать обозначение и наименование программы; программное обеспечение, необходимое для функционирования программы; язык программирования, на котором будет написана программа.

Функциональное назначение должно содержать описание классов решаемых задач и (или) назначение программы и сведения о функциональных ограничениях на применение.

В описании логической структуры должны быть указаны используемые методы, представлена модульная структура программы с описанием функций составных частей и связей между ними.

Для одного из модулей должна быть представлена блок-схема алгоритма его реализации, описаны виды организации диалога.

В описании логической структуры также должны быть представлены описания созданных процедур и функций, а также описания используемых данных.

Описания созданных процедур и функций должны быть представлены в виде таблицы 2.

Таблица 2 - Характеристика созданных подпрограмм.

наименование	тип подпрограммы	назначение

Продолжение примера – Головной модуль

```
BEGIN
Assign(f, 'sclad'); { назначение переменной типа файл имени физического
{файла}
Writeln('Нужно ли создавать файл? (д/н)'); Readln(c);
If (c='д') or (c='Д') Then sozdfile(f);{вызов п/п создания файла}
Repeat
      Quit:=false; {переменная типа boolean}
      Clrscr;
      Writeln (' Программа "СКЛАД" ');
      Writeln ('Что будете делать?');
      Writeln ('1. Работа с записями о товарах'); {вывод пунктов меню}
      Writeln ('2.Обработка запросов');
      Writeln ('3. Выход из программы');
      Readln(p); {ввод пункта }
      Case p of
       1: RabZap;{вызов п/п записей}
       2: ObrZapr;
       3: Quit:=True; {выбран пункт выхода из программы}
     End; {Case}
Untile quit;
END.
```

Пример 1 **– Управляющий модуль работы с записями**

```
Procedure  RabZap;
Var  m:byte; qrab: boolean;
BEGIN
Repeat
   qrab :=false; {переменная типа boolean}
   Clrscr;
  Writeln (' Работа с записями');
  Writeln (‘Что будете делать?');
  Writeln (‘1. Добавление записи'); {вывод пунктов меню}
  Writeln (‘2.Просмотр записей');
  Writeln (‘3.Изменение записи');
  Writeln (‘4.Удаление записи');
  Writeln (‘5. Выход из режима работы с записями');
  Readln(p); {ввод пункта }
  Case m of
      1: dobav(f); {вызов п/п записей}
       ……..
       5: qrab :=True; {выбран пункт выхода из управляющего модуля}
 End; {Case}
Untile   qrab;
END.
```

Пример 2 **– Управляющий модуль обработки запросов**

```
Procedure  ObrZapr;
Var  m:byte; qobr: boolean;
BEGIN
Repeat
   Qobr:=false; {переменная типа boolean}
   Clrscr;
  Writeln (' Обработка запросов');
  Writeln (‘Что будете делать?');
  Writeln (‘1. Поиск товара  по наименованию '); {вывод пунктов меню}
  Writeln (‘2.Подсчет общей стоимости товаров');
  Writeln (‘3. Выход из режима  обработки запросов');
  Readln(p); {ввод пункта }
 Case m of
    1: poisk (f); {вызов п/п записей}
    ……..
    3: Qobr:=True; {выбран пункт выхода из управляющего модуля}
 End; {Case}
Untile qobr;
END.
```

Пример 3 **- Процедура ввода одной записи.**

```
Procedure vvodzap( Var t: tov);
begin
Clrscr;
With t do begin
Writeln('Конец ввода номер=0');
Write('Номер товара'); Readln(nom);
Write('Название товара'); Readln(nt);
..... {ввод значений остальных полей}
end;
end; {конец n/n}
```

Пример 4 - Процедура вывода одной записи.

```
Procedure vivodzap(Var t: tov);
Begin
With t do begin
 Writeln('Номер',nom:4, 'название ', nt:20);
..... {вывод остальных полей с пояснениями}
end;
end; { конец n/n}
```

Пример 5 **– Процедура создания файла и записи в него.**

```
Procedure sozdfile( var  f: file_type);
Var  t: tov;
Begin
Rewrite(f); {установка указателя на начало файла для записи }
Vvodzap(t); {ввод одной записи с клавиатуры}
While t.nom<>0 do {выполнять, пока номер записи не будет равен 0}
Begin
Write(f,t); {запись в файл}
Vvodzap(t); {ввод следующей записи с клавиатуры}
End;
Close(f); {закрытие файла}
End; {конец п/п}
```

***Пример 6** – Процедура удаления записи из файла.*

```
Procedure udal(var i:file_type);
 Var t: tov; k: 1..100; c: char;  found: boolean;
Begin
Assign(copf, 'copsc');
Writeln('Какую запись удалить? Введите номер товара'); Readln(k);
Reset(f); Rewrite(copf); {позиционирование файлов}
Found:=False;
While Not Eof(f) do begin
   Read(f,t);   {читаем запись из файла}
   If  t.nom < > k  then write(copf, t) {если номер записи не k, записываем ее в файл-{копию}
  Else  begin  found:=true; {запись для удаления найдена}
  Vivod(t); write('Именно эту запись удалить?');  Writeln('Д/Н'); Readln(c);
  Case c of
'   д','Д','у','У : ; {пустой оператор, запись k в файл-копию не записываем}
   Else Write(copf,t); {раздумали удалять, записываем запись k в файл-копию }
  End; {case}
End; {else}
 End; {while}
 Close(f); Close(copf); {закрываем файлы }
 If not found { если запись не найдена} Then begin
Writeln('Запись с номером ', k : 3, ' не найдена'); Erase(copf); {удаляем файл-копию}
End
Else begin  Erase(f); Rename(copf,'sclad'); End;
End; {конец п/п}
```

Пример 7– Процедура изменения. записи

```
procedure izmen( var f : file_type); {изменение  записи в файле}
var  t:tov;  k: 1..100;  c: char;
begin
  writeln('Какую запись нужно изменить (номер товара)?');  readln(k);
   reset(f);
while   not eof(f)   and (t.nom < >k) do read(f,t); {считываем записи из
файла, пока не достигнем искомой записи}
if  t.nom = k  then begin
              vivod(t); write('новая цена ед.'?);readln(t.s_ed);
              write('новое количество?'); readln(t.kol);……..
                seek(f,filepos(f)-1); {поместить указатель перед изменяемой
записью}
              write(f,t); {записываем в файл исправленную запись}
                      end
                 else writeln('Запись с номером ', k : 3, ' не найдена');
close(f);
end; {конец п/п}
```

Пример 8 **– Процедура просмотра всех записей файла.**

```
Procedure prosmotr(var f:file_type);
Var  t:tov;
Begin
Reset(f); {установка указателя на начало файла для чтения из него}
While not eof(f) do begin  {выполнять, пока не достигнут конец файла}
  Read(f, t); {чтение записи из файла}
  Vivodzap(t); {вывод записи}
end;
End; {конец п/п}
```

Пример 9 **– Процедура добавления новой записи в файл.**

```
Procedure dobav( Var f: file_type);
Var  t:Tov; c: Char;
Begin
    Reset(f);
    Seek(f,Filesize(f)); {указатель перемещаем на конец файла}
    Vvodzap(t);
     Write(f, t); Close(f);
End; {конец п/п}
```

Пример 10 **– Процедура нахождения общей стоимости товаров**

```
Procedure Sum(Var f: File_type); { }
Var  t:Tov; s:Real;
Begin
    Reset(f); s:=0;
    While not eof(f) do begin
          Read(f, t);   {читаем запись из файла}
          s:=s+t.s_ed * t.kol; {суммируем стоимости
отдельных товаров}
     End;
     Writeln('На складе хранятся товары на сумму – ',
s:8:2, 'руб');
     Readln;
End; {конец п/п}
```

Пример 11 **– Процедура поиска данных о товаре по наименованию.**

```
Procedure poisk(var f: file_type);
 Var t:tov; name: String[20]; z: Boolean;
Begin
Writeln('Введите название искомого товара?'); Readln(name);
Reset(f); z:=false;
While not eof(f) do Begin
   Read(f,t);   {читаем запись из файла}
   If  t.nt=name {товар с искомым наименованием найден}
       Then begin {выводим данную запись}
                z:=true; vivod(t);
              end;
  end; {while}
if not z then Writeln('Запись с наименованием ', name:20, ' не найдена');
readln;
close(f);
end; {конец п/п}
```

Вопросы?