10 Графика во Free Pascal

Данная глава посвящена графическим средствам Free Pascal. Рассмотрены основные процедуры и функции работы с графикой. Приведен пример построения графика математической зависимости.

10.1 Средства рисования в Lazarus

При разработке оконного приложения Lazarus есть форма, на которой можно рисовать. В распоряжении программиста находится полотно (холст) — свойство `Canvas`, карандаш — свойство `Pen`, и кисть — свойство `Brush`.

Свойством `Canvas` обладают следующие компоненты:

- форма (класс `Tform`);
- таблица (класс `TSringGrid`);
- растровая картинка (класс `Timage`);
- принтер (класс `TPrinter`).

При рисовании компонента, обладающего свойством `Canvas`, сам компонент рассматривается как прямоугольная сетка, состоящая из отдельных точек, называемых пикселями.

Положение пикселя характеризуется его вертикальной `(X)` и горизонтальной `(Y)` координатами. Левый верхний пиксель имеет координаты `(0,0)`. Вертикальная координата возрастает сверху вниз, горизонтальная — слева направо. Общее количество пикселей по вертикали определяется свойством `Height`, а по горизонтали — свойством `Weight`. Каждый пиксель может иметь свой цвет. Для доступа к любой точке полотна используется свойство

```
Pixels[X,Y]:TColor.
```

Это свойство определяет цвет пикселя с координатами `X(integer), Y(integer).`

Изменить цвета любого пикселя полотна можно с помощью следующего оператора присваивания

```
Компонент.Canvas. Pixels[X,Y]:=Color;
```

где `Color` — переменная или константа типа `Tcolor`. Определены следующие константы цветов (тал. 10.1):

Таблица 10.1. Значение свойств Color

Константа	Цвет	Константа	Цвет
clBlack	Черный	clSilve	Серебристый
clMaroon	Каштановый	clRed	Красный
clGreen	Зеленый	clLime	Салатовый
clOlive	Оливковый	clBlue	Синий
clNavy	Темно-синий	clFuchsia	Ярко-розовый
clPurple	Розовый	clAqua	Бирюзовый
clTeal	Лазурный	clWhite	Белый
clGray	Серый		

Цвет любого пикселя можно получить с помощью следующего оператора присваивания:

```
Color:=Компонент.Canvas. Pixels[X,Y];
```

где Color — переменная типа Tcolor.

Класс цвета точки Tcolor определяется как длинное целое longint. Переменные этого типа занимают в памяти четыре байта. Четыре байта переменных этого типа содержат информацию о долях синего (B), зеленого (G) и красного(R) цветов и устроены следующим образом: $00BBGGRR.

Для рисования используются методы класса TCanvas, позволяющие изобразить фигуру (линию, прямоугольник и т.д.) или вывести текст в графическом режиме, и три класса, определяющие инструменты вывода фигур и текстов:

- TFont (шрифты);
- TPen (карандаш, перо);
- TBrush (кисть).

Класс TFONT

Можно выделить следующие свойства объекта Canvas.TFont:

- Name (тип string) — имя используемого шрифта.
- Size (тип integer) — размер шрифта в пунктах (points). Пункт — это единица измерения шрифта, равная 0,353 мм, или 1/72 дюйма.
- Style — стиль начертания символов, который может быть обычным, полужирным (fsBold), курсивным (fsItalic), под-

черкнутым (`fsUnderline`) и перечеркнутым (`fsStrikeOut`). В программе можно комбинировать необходимые стили, например, чтобы установить стиль «полужирный курсив», необходимо написать следующий оператор:

```
Объект.Canvas. Font. Style:=[fsItalic,fsBold]
```

- `Color` (тип `Tcolor`) — цвет символов.
- `Charset` (тип 0..255) — набор символов шрифта. Каждый вид шрифта, определяемый его именем, поддерживает один или более наборов символов. В табл. 10.2 приведены некоторые значения `Charset`.

Таблица 10.2. Значения свойства `Charset`

Константа	Значение	Описание
ANSI_CHARSET	0	Символы ANSI
DEFAULT_CHARSET	1	Задается по умолчанию. Шрифт выбирается только по его имени Name и размеру Size. Если описанный шрифт недоступен в системе, будет заменен другим
SYMBOL_CHARSET	2	Стандартный набор символов
MAC_CHARSET	77	Символы Macintosh
GREEK_CHARSET	161	Греческие символы
RUSSIAN_CHARSET	204	Символы кириллицы
EASTEUROPE_CHARSET	238	Включает диалектические знаки (знаки, добавляемые к буквам и характеризующие их произношение) для восточно-европейских языков

Класс TPEN

Карандаш (перо) используется как инструмент для рисования точек, линий, контуров геометрических фигур. Основные свойства объекта `Canvas.TPen`:

- `Color` (тип Tcolor) – определяет цвет линии;
- `Width` (тип Integer) – задает толщину линии в пикселях;

• `Style` – дает возможность выбрать вид линии. Это свойство может принимать значение, указанное в таблице 10.3.

Таблица 10.3. Виды линий

Значение	Описание
`psSolid`	Сплошная линия
`psDash`	Штриховая линия
`psDot`	Пунктирная линия
`psDashDot`	Штрих-пунктирная линия
`psDashDodDot`	Линия, чередующая шрих и два пунктира
`psClear`	Нет линии

• `Mode` – определяет, каким образом взаимодействуют цвета пера и полотна. Выбор значения этого свойства позволяет получать различные эффекты, возможные значения `Mode` приведены в табл. 10.4. По умолчанию вся линия вычерчивается цветом, определяемым значением `Pen.Color`, но можно определять инверсный цвет линии по отношению к цвету фона. В этом случае независимо от цвета фона, даже если цвет линии и фона одинаков, линия будет видна.

Таблица 10.4. Возможные значения свойства Mode

Режим	Операция	Цвет пикселя
`pmBlack`	`Black`	Всегда черный
`pmWhite`	`White`	Всегда белый
`pmNop`	`-`	Неизменный
`pmNot`	`Not Screen`	Инверсный цвет по отношению к цвету фона
`pmCopy`	`Pen`	Цвет, указанный в свойствах Color пера Pen (это значение принято по умолчанию)
`pmNotCopy`	`Not Pen`	Инверсия цвета пера

Режим	Операция	Цвет пикселя
pmMergePenNot	Pen or Not Pen	Дизъюнкция цвета пера и инверсного цвета фона
pmMaskPenNot	Pen and Not Screen	Конъюнкция цвета пера и инверсного цвета фона
pmMergeNotPen	Not Pen or Screen	Дизъюнкция цвета фона и инверсного цвета пера
PmMaskNotPen	Not Pen and Screen	Конъюнкция цвета фона и инверсного цвета пера
pmMerge	Pen or Screen	Дизъюнкция цвета пера и цвета фона
pmNotMerge	Not (Pen or Screen)	Инверсия режима pmMerge
pmMask	Pen and Screen	Конъюнкция цвета пера и цвета фона
pmNotMask	Not (Pen and Screen)	Инверсия режима pmMask
pmXor	Pen xor Screen	Операция xor над цветом пера и цветом фона
pmNotXor	Not (Pen xor Screen)	Инверсия режима pmXor

Класс TBRUSH

Кисть (`Canvas.Brush`) используется методами, обеспечивающими вычерчивание замкнутых фигур для заливки. Кисть обладает двумя основными свойствами:

- `Color` (тип Tcolor) – цвет закрашивания замкнутой области;
- `Style` – стиль заполнения области.

Класс TCANVAS

Это класс является основным инструментом для рисования гра-

фики. Рассмотрим наиболее часто используемые методы этого класса.

```
Procedure MoveTo(X, Y : Integer);
```

Метод `MoveTo` изменяет текущую позицию пера на позицию, заданную точкой `(X, Y)`. Текущая позиция хранится в переменной `PenPos` типа `Tpoint`. Определение типа `TPoint` следующее:

```
type TPoint =record
 X: Longint;
 Y: Longint;
end;
```

Текущую позицию пера можно считывать с помощью свойства `PenPos` следующим образом:

```
X:=PenPos.X;
Y:=PenPos.Y;
Procedure LineTo(X, Y :Integer);
```

Метод `LineTo` соединяет прямой линией текущую позицию пера и точку с координатами `(X, Y)`. При этом текущая позиция пера перемещается в точку с координатами `(X, Y)`.

Рассмотрим работу процедуры на примере. Расположим на форме кнопку и рассмотрим процедуру обработки события `TForm1.Button1Click`, которая рисует прямые линии:

```
Procedure TfForm1.Button1Click(Sender: TObject)
begin
 Form1.Canvas.LineTo(30,50);
end;
```

В результате щелчка по кнопке на форме возникнет прямая линия, соединяющая точку с координатами (0,0) и точку с координатами (30,50). Теперь перепишем процедуру обработки события следующим образом:

```
Procedure TForm1.Button1Click(Sender: TObject)
begin
Form1.Canvas.LineTo(Canvas.PenPos.x+30,
                                Canvas.PenPos.y+50);
end;
```

При первом щелчке по кнопке на экране прорисуется аналогичная линия. Но при повторном щелчке первая процедура продолжает рисовать линию, соединяющую точки (0,0) и (30,50). Вторая процедура рисует линию, которая соединяет текущую точку с точкой, полу-

чившейся из текущей добавлением к координате X числа 30, а к координате Y – числа 50. Т.е. при повторном щелчке по кнопке процедура соединяет прямой линией точки (30,50) и (60,100). При третьем щелчке по кнопке будут соединяться прямой линией точки (60,100) и (90,150) и т.д.

```
Procedure PolyLine(const Points array of TPoint);
```

Метод `PolyLine` рисует ломаную линию, координаты вершин которой определяются массивом `Points`.

Рассмотрим работу процедуры на примере. Расположим на форме кнопки **Рисовать** и **Выход** и запишем следующие операторы процедур обработки события:

```
procedure TForm1.Button1Click(Sender: TObject);
var temp :array [1..25] of TPoint;
 i : byte;
 j: integer;
begin
 j:=1;
 for i:=1 to 25 do
 begin
//вычисление координат вершин ломаной линии
     temp[i].x:=25+(i-1)*10;
     temp[i].y:=150-j*(i-1)*5;
     j:=-j;
 end;
 Form1.Canvas.Polyline (temp);
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
 Form1.Close;
end;
```

После запуска программы и щелчка по кнопке «Рисовать» окно формы будет выглядеть, как на рисунке 10.1.

```
Procedure Ellipse (X1, Y1, X2, Y2 :Integer);
```

Метод `Ellipse` вычерчивает на холсте эллипс или окружность. `X1, Y1, X2, Y2` — это координаты прямоугольника, внутри которого вычерчивается эллипс. Если прямоугольник является квадратом, то вычерчивается окружность.

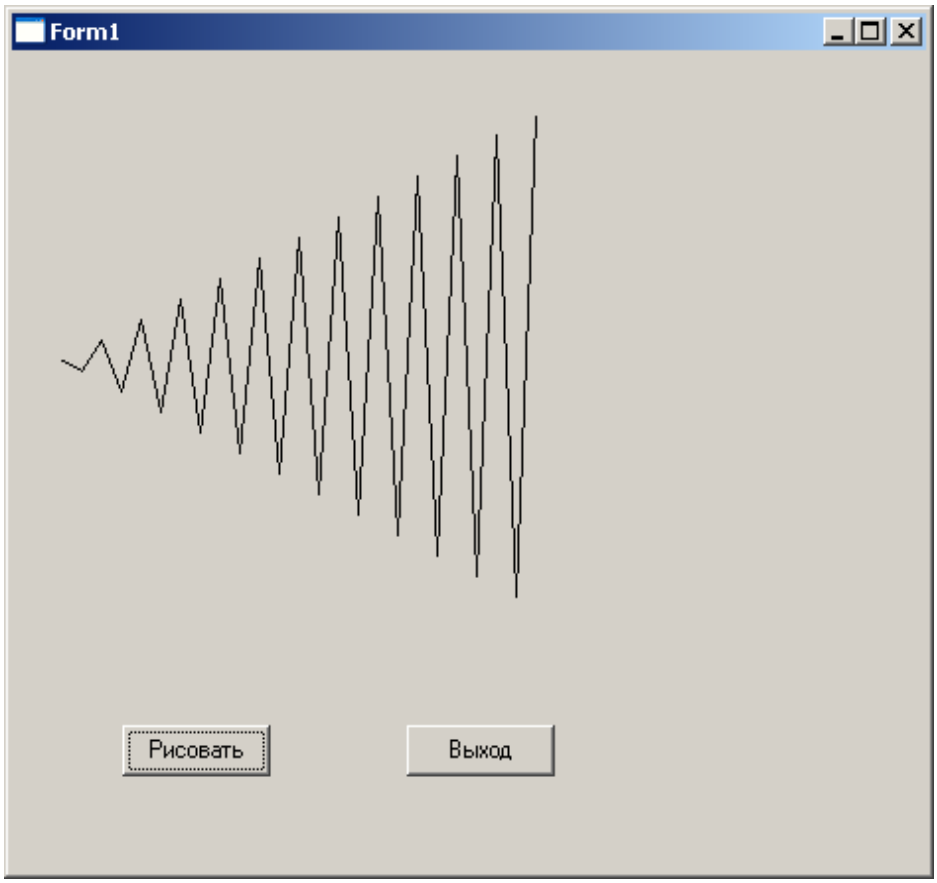

Рисунок 10.1: Пример использования процедуры `PolyLine`

Метод

```
Procedure Arc (X1,Y1,X2,Y2,X3,Y3,X4,Y4:Integer);
```

вычерчивает дугу эллипса. `X1, Y1, X2, Y2` — это координаты, определяющие эллипс, частью которого является дуга. `X3, Y3` — координаты, определяющие начальную точку дуги, `X4, Y4` — координаты, определяющие конечную точку дуги. Дуга рисуется против часовой стрелки.

Метод

```
Procedure Rectangle (X1, Y1, X2, Y2 :Integer);
```

рисует прямоугольник. `X1, Y1, X2, Y2` — координаты верхнего левого и нижнего правого углов прямоугольника.

Метод

```
Procedure RoundRect (X1,Y1,X2,Y2,X3,Y3:Integer);
```

вычерчивает прямоугольник со скругленными углами. `X1, Y1, X2, Y2` — координаты верхнего левого и нижнего правого углов прямоугольника, а `X3, Y3` — размер эллипса, одна четверть которого используется для вычерчивания скругленного угла.

Метод
`Procedure PolyGon(const Points array of TPoint);`
рисует замкнутую фигуру (многоугольник) по множеству угловых точек, заданному массивом `Points`. При этом первая точка соединяется прямой линией с последней. Этим метод `PolyGon` отличается от метода `Poliline`, который не замыкает конечные точки. Рисование осуществляется текущим пером `Pen`, а внутренняя область фигуры закрашивается текущей кистью `Brush`.

Метод
`Procedure Pie (X1,Y1,X2,Y2,X3,Y3,X4,Y4 :Integer);`
рисует замкнутую фигуру — сектор окружности или эллипса с помощью текущих параметров пера `Pen`, внутренняя область закрашивается текущей кистью Brush. Точки `(X1, Y1)` и `(X2, Y2)` задают прямоугольник, описывающий эллипс. Начальная точка дуги определяется пересечением эллипса с прямой, проходящей через его его центр и точку `(X3, Y3)`. Конечная точка дуги определяется пересечением эллипса с прямой, проходящей через его центр и точку `(X4, Y4)`. Дуга рисуется против часовой стрелки от начальной до конечной точки. Рисуются прямые, ограничивающие сегмент и проходящие через центр эллипса и точки `(X3, Y3)` и `(X4, Y4)`.

Создадим форму, установим ей размеры `Heigth` — 500, `Width` — 500. Внизу разместим кнопку, зададим ей свойство `Caption` — «`Рисовать`». При запуске программы и щелчке по этой кнопке на форме прорисуются различные фигуры (см. рис. 10.2). Ниже приведен листинг программы, демонстрирующий работу перечисленных методов. Результат работы программы приведен на рис. 10.2.

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
 Classes,    SysUtils,    LResources,    Forms,
Controls, Graphics, Dialogs, StdCtrls;
type
 { TForm1 }
 TForm1 = class(TForm)
 Button1: Tbutton;
```

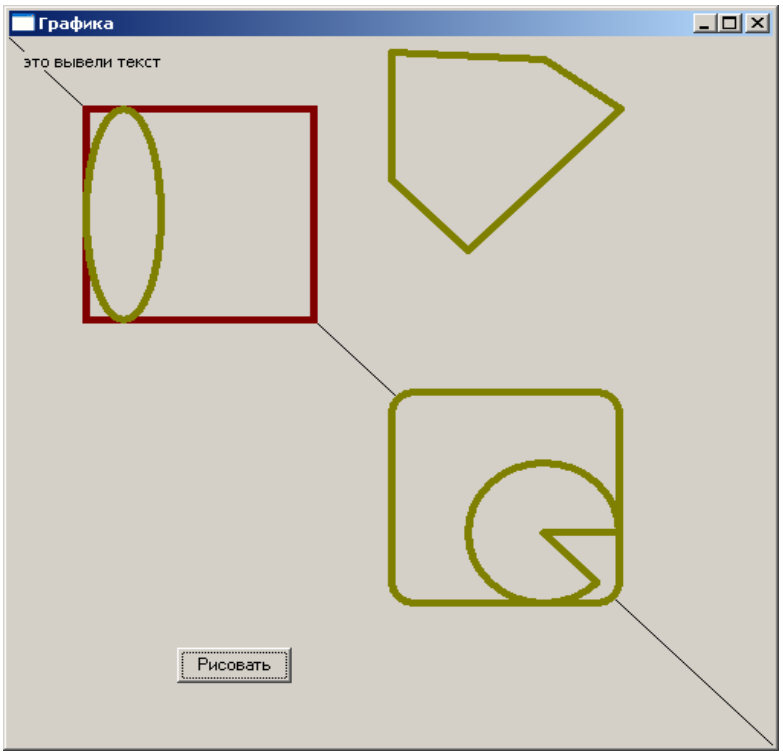

Рисунок 10.2: Пример использования методов рисования фигур

```
 procedure Button1Click(Sender: TObject);
private
 { private declarations }
public
 { public declarations }
end;
var
 Form1: TForm1;
implementation
{ TForm1 }
procedure TForm1.Button1Click(Sender: TObject);
var t: array [1..5] of TPoint;
begin
 Form1.Canvas.LineTo(500,500);//Рисование линии
//Изменение цвета и толщины линии.
 Form1.Canvas.Pen.Color:= clMaroon;
 Form1.Canvas.Pen.Width:= 5;
//Рисование прямоугольника.
 Form1.Canvas.Rectangle(50,50,200,200);
 Form1.Canvas.Pen.Color:= clolive;
```

```
    //Рисование эллипса.
     Form1.Canvas.Ellipse(50,50,100,200);
//Рисование прямоугольника с скругленными углами.
     Form1.Canvas.RoundRect(250,250,400,400,30,30);
//Рисование сектора окружности.
Form1.Canvas.Pie(300,300,400,400,350,350,500,500);
//Массив координат вершин пятиугольника.
     t[1].x:=250; t[1].y:=10;
     t[2].x:=350; t[2].y:=15;
     t[3].x:=400; t[3].y:=50;
     t[4].x:=300; t[4].y:=150;
     t[5].x:=250; t[5].y:=100;
     Form1.Canvas.Polygon (t) ;
    Form1.Canvas.TextOut(10,10,'это вывели текст');
    end;
    initialization
     {$I unit1.lrs}
    end.
```

Функция

```
procedure TextOut(X,Y:Integer;const Text:String);
```

пишет строку текста `Text`, начиная с позиции с координатами `(X, Y)`. Текущая позиция `PenPos` пера `Pen` перемещается на конец выведенного текста. Надпись выводится в соответствии с текущими установками шрифта `Font`, фон надписи определяется установками текущей кисти. Для выравнивания позиции текста на канве можно использовать методы, позволяющие определить высоту и длину текста в пикселях — `TextExtent`, `TextHeight` и `TextWidth`. Рассмотрим эти функции.

Функция

```
function TextExtent (const Text : String): Tsize;
```

возвращает структуру типа `Tsize`, содержащую длину и высоту в пикселях текста `Text`, который предполагается написать на канве текущим шрифтом.

```
    type Tsize = record
     cx:Longint; cy:Longint;
    end;
```

Функция

```
function TextHeight(const Text:String):Integer;
```
возвращает высоту в пикселях текста `Text`, который предполагается написать на канве текущим шрифтом.

Функция
```
function TextWidth(const Text:String):Integer;
```
возвращает длину в пикселях текста `Text`, который предполагается написать на канве текущим шрифтом. Это позволяет перед выводом текста на канву определить размер надписи и расположить ее и другие элементы изображения наилучшим образом.

Если цвет кисти в момент вывода текста отличается от того, которым закрашена канва, то текст будет выведен в цветной прямоугольной рамке, но ее размеры будут точно равны размерам надписи.

Мы рассмотрели основные функции рисования. Прежде чем перейти непосредственно к рисованию, обратите внимание на то, что если вы свернете окно с графикой, а затем его восстановите, то картинка на форме исчезнет. Изменение размеров окна также может испортить графическое изображение в окне. Для решения этой проблемы существуют процедуры обработки событий `Объект.FormPaint` и `Объект.FormResize`.

Процедура `Объект.FormPaint` выполняется после появления формы на экране.

Процедура `Объект.FormResize` выполняется после изменения размера формы.

Следовательно, для того чтобы не пропадала картинка, все операторы рисования нужно помещать внутрь `Объект.FormPaint` и дублировать в процедуре `Объект.FormResize`.

10.2 Построение графиков

Алгоритм построения графика непрерывной функции $y=f(x)$ на отрезке $[a;b]$ состоит в следующем: необходимо построить точки $(x_i, f(x_i))$ в декартовой системе координат и соединить их прямым линиями. Координаты точек определяются по следующим формулам:

$hx=(b-a)/N,$

где N — количество отрезков на отрезке $[a;b]$.

$x_i=a+(i-1)hx; \quad y_i=f(x_i), \quad$ где $i=0,N$

Чем больше точек будет изображено, тем более плавным будет построенный график.

При переносе этого алгоритма на форму или другой компонент Lazarus учитывает размеры и особенности компонента (ось `OX` направлена слева направо, ее координаты лежат в пределах от 0 до `Width`; ось `OY` направлена вниз, ее координаты находятся в пределах от 0 до `Height`). Значения координат *X* и *Y* должны быть целыми.

Необходимо пересчитать все точки из «бумажной» системы координат (*X* изменяется в пределах от a до b, *Y* изменяется от минимального до максимального значения функции) в «компонентную[79]» (в этой системе координат ось абсцисс обозначим буквой U, `0` ≤ U ≤ `Wigth`, а ось ординат — буквой *V*, `0` ≤ `V` ≤ `Height`).

Для преобразования координаты *X* в координату *U* построим линейную функцию *cX+d*, которая переведет точки из интервала *(a;b)* в точки интервала (`X0,Width-Xk`)[80]. Поскольку точка *a* «бумажной» системы координат перейдет в точку *X0* «экранной», а точка *b* — в точку `Width-Xk`, то система линейных уравнений для нахождения коэффициентов *c* и *d* имеет вид:

$$\begin{cases} c \cdot a + d = X0 \\ c \cdot b + d = Width - Xk \end{cases}$$

Решив ее, найдем коэффициенты *c, d*:

$$\begin{cases} d = X0 - c \cdot a \\ c = \dfrac{Width - Xk - X0}{b - a} \end{cases}$$

Для преобразования координаты *Y* в координату *V* построим линейную функцию *V=gY+h.* Точка `min` «бумажной» системы координат перейдет в точку `Height-YK` «компонентную», а точка `max` – в точку *Y0.* Для нахождения коэффициентов *g* и *h* решим систему линейных алгебраических уравнений:

$$\begin{cases} g \cdot min + h = Heignt - Yk \\ g \cdot max + h = Y0 \end{cases}$$

Ее решение позволит найти нам коэффициенты *g* и *h*

$$\begin{cases} h = Y0 - g \cdot max \\ g = \dfrac{Heigth - Yk - Y0}{min - max} \end{cases}$$

Перед описанием алгоритма построения графика давайте

79 Система координат, связанная с конкретным компонентом класса Tform, Timage, Tprinter и т.д.
80 X0, XK, Y0, YK — Отступы от левой, правой, нижней и верхней границы компонента.

уточним формулы для расчета коэффициентов *c, d, g* и *h*. Дело в том, что `Width` – это ширина компонента с учетом рамок слева и справа, а `Height` – полная высота компонента с учетом рамки, а если в качестве компонента будет использоваться форма, то необходимо учесть заголовок окна. Однако, для изображения графика нам нужны вертикальные и горизонтальные размеры компонента без учета рамок и заголовка. Эти размеры хранятся в свойствах `ClientWidth` (ширина клиентской области компонента без учета ширины рамки) и `ClientHeight` (высота клиентской области компонента без учета ширины рамки и ширины заголовка) компонента. Поэтому для расчета коэффициентов *c, d, g* и *h* логичнее использовать следующие формулы:

$$\begin{cases} d = X0 - c \cdot a \\ c = \dfrac{ClientWidth - Xk - X0}{b - a} \end{cases} \qquad (10.1)$$

$$\begin{cases} h = Y0 - g \cdot max \\ c = \dfrac{ClientHeigth - Yk - Y0}{min - max} \end{cases} \qquad (10.2)$$

Алгоритм построения графика на экране дисплея можно разделить на следующие этапы:

1. Определить число отрезков *N*, шаг изменения переменной *X*.
2. Сформировать массивы *X, Y*, вычислить максимальное (`max`) и минимальное (`min`) значения Y.
3. Найти коэффициенты *c, d, g* и *h* по формулам (10.1), (10.2).
4. Создать массивы $U_i = cX_i + d$, $V_i = gY_i + h$.
5. Последовательно соединить соседние точки прямыми линиями с помощью функции `LineTo`.
6. Изобразить систему координат, линий сетки и подписи.

При построении графиков нескольких непрерывных функций вместо массивов *Y* и *V* рационально использовать матрицы размером `k×n`, где `k` – количество функций. Элементы каждой строки матриц являются координатами соответствующего графика в «бумажной» (*Y*) и «компонентной» (*V*) системе координат.

Блок-схема алгоритма изображения графика показана на рис.10.3. Блок-схема построения графиков `k` непрерывных функций представлена на рис. 10.4.

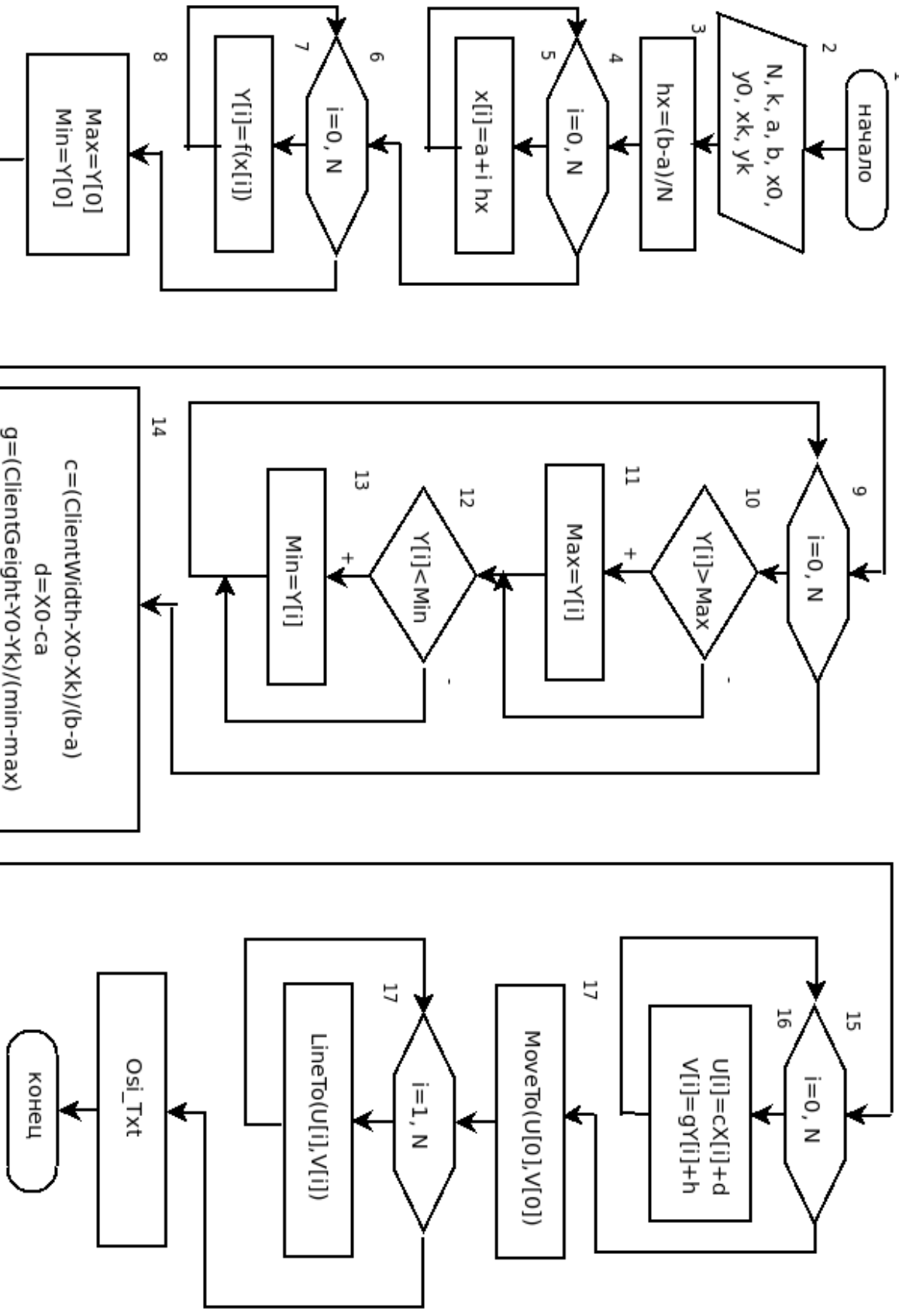

Рисунок 10.3: Блок-схема алгоритма построения графика функции

Рассмотрим поэтапно каждую блок-схему.

Для рис. 10.3 первый этап алгоритма представлен в блоках 2 и 3. Второй этап реализуется в блоках 4 – 13. Коэффициенты третьего этапа рассчитываются в блоке 14. В блоках 15 – 16 формируются массивы значений U и V «компонентной системы координат (этап 4). Блоки 17 – 19 – это вывод графиков на экран. И блок 20 предназначен для шестого этапа.

Для второй схемы рис. 10.4 реализация второго этапа представлена в блоках 4 – 15. В блоке 16 рассчитываются коэффициенты *c, d, g* и *h*. В блоках 17 – 20 формируются массивы четвертого этапа. Блоки 21 – 24 предназначены для вывода графиков, а блок 25 – для построения осей.

ЗАДАЧА 10.1. Построить график функции *f(x)* на интервале *[a, b]*. Функция задана следующим образом:

$$f(x)=\begin{cases}\sin\dfrac{x}{2}, \text{если } x\leqslant 0\\ \sqrt{\dfrac{1+x}{3}}, \text{если } x>0\end{cases}$$

Создадим новый проект, изменим высоту и ширину формы до размеров, достаточных для отображения на ней графика. Например, можно установить следующие свойства: `Width` – 800, `Height` – 700. Разместим на форме кнопку и компонент класса `TImage`. Объект `TImage1` – это растровая картинка, которая будет использоваться для отображения графика после щелчка по кнопке `Button1`. Размеры растровой картинки сделаем чуть меньше размеров формы.

Установим в качестве свойства `Caption` формы сроку «График функции». Чтобы избавиться от проблемы перерисовки будущего графика при изменении размера формы, запретим изменение формы и уберем кнопки минимизации и максимизации окна. Свойство формы `BorderStyle` определяет внешний види поведение рамки вокруг окна формы. Для запрета изменения формы установим значение свойства `BorderStyle` в `bsSingle`, это значение определяет стандартную рамку вокруг формы и запрещает изменение размера формы. Чтобы убрать кнопки минимизации и максимизации формы, установим ее свойства `BolderIcons.BiMaximize` и `BolderIcons.BiMinimize` в `False`.

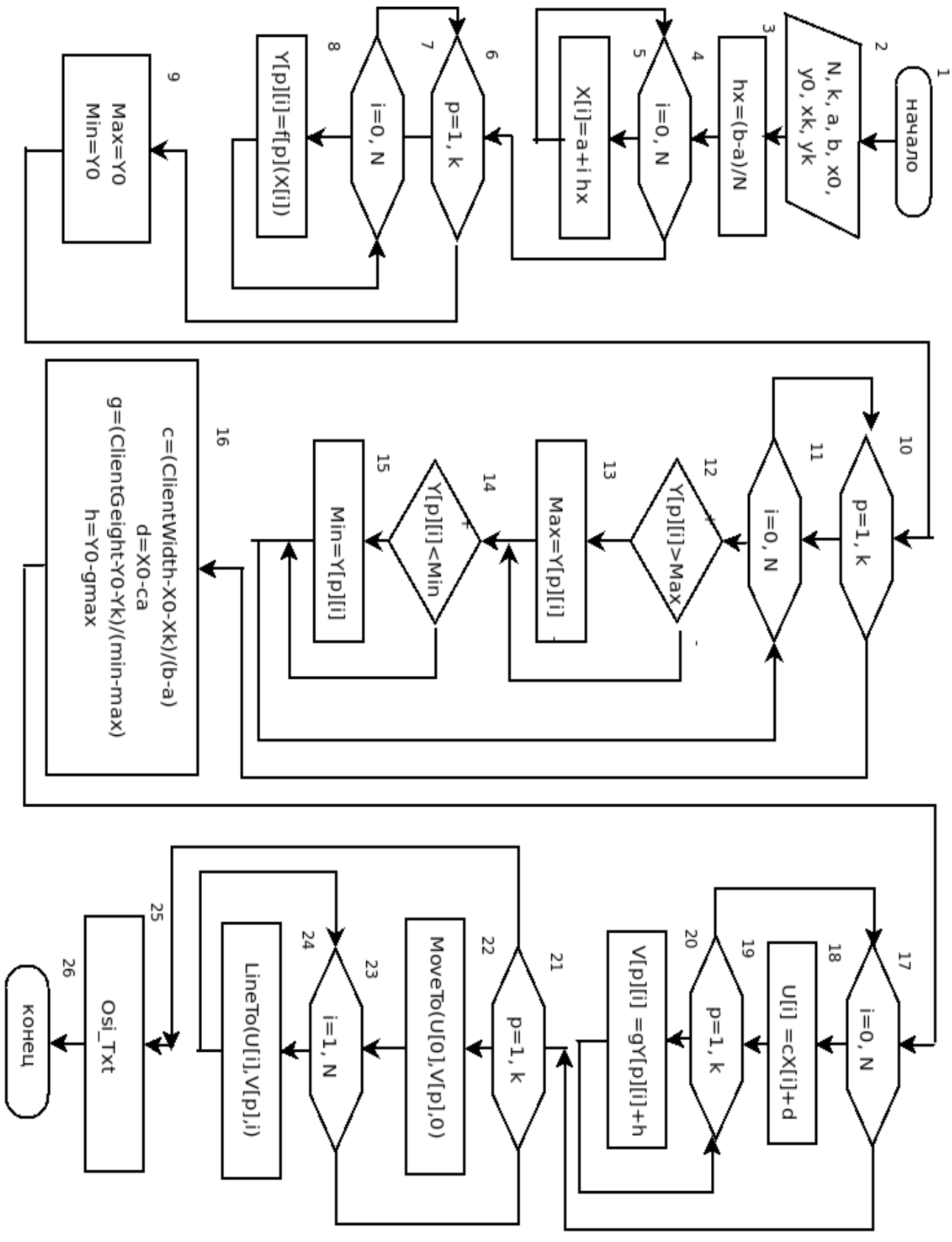

Рисунок 10.4: Блок-схема алгоритма построения графиков нескольких функций

Для кнопки установим свойство `Caption` – фразу «График».

После установки всех описанных свойств окно формы должно стать подобным представленному на рис.10.5.

Рисунок 10.5: Окно формы после установки свойств

Ввод интервала *a* и *b* сделаем с помощью запроса в момент создания формы (в методе инициализации формы `FormCreate`). Ниже приведен листинг модуля проекта рисования графика с комментариями:

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
    Classes,    SysUtils,    LResources,    Forms,
Controls, Graphics, Dialogs, ExtCtrls, StdCtrls;
//Функция, которая аналитически определяет график.
function f(x:real):real;
//Функция, которая изображает график.
procedure Graphika(a, b: real);
type
    { TForm1 }
    TForm1 = class(TForm)
```

```
        Button1: TButton;
        Image1: TImage;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { private declarations }
    public
        { public declarations }
    end;
var
    Form1: TForm1;
    //Объявление переменных:
    //x0, xk, y0, yk - отступы от границ
    //компонента слева, справа, сверху и снизу;
    //x, y - массивы, определяющие координаты
    //точек графика в "бумажной"
    //системе координат;
    //u, v - массивы, определяющие координаты
    //точек графика в "компонентной"
    //системе координат;
    //N - количество точек.
    x0, y0, xk, yk, a, b :real;
    x, y : array [0..1000] of real;
    u, v : array [0..1000] of integer;
    N : integer;
implementation
   // функция, которая будет изображена
   //на компоненте Image1
function f (x : real) : real;
begin
    if x<=0 then Result:=sin(x/2)
    else Result:=sqrt((1+x)/3);
end;
//функция, которая рисует график
//заданной функции на компоненте Image1.
procedure Graphika(a, b: real);
//Kx+1 - количество линий сетки,
//перпендикулярных оси OX.
```

```
//Ky+1 - количество линий сетки,
//перпендикулярных оси OY.
const Kx=5;Ky=5;
var dx, dy, c, d, g, h, max, min :real;
    i, tempx, tempy :integer; s : string;
begin
//Вычисление шага изменения по оси X.
    h:=(b-a)/(N-1);
//Формирование массивов x и y.
    x[0]:=a;
    y[0]:=f(x[0]);
    for i:=1 to N do
    begin
        x[i]:=x[i-1]+h;
        y[i]:=f(x[i]);
    end;
//Вычисление максимального
//и минимального значений массива Y.
    max:=y[0]; min:=y[0];
    for i:=1 to N do
    begin
        if y[i]>max then max:=y[i];
            if y[i]<min then min:=y[i];
    end;
//формирование коэффициентов пересчета
//из "бумажной" в "компонентную"
//систему координат.
    c:=(Form1.Image1.ClientWidth-x0-xk)/(b-a);
    d:=x0-c*x[0];
    g:=(Form1.Image1.ClientHeight-y0-yk)/
                                        (min-max);
    h:=yk-g*max;
//Формирование массивов точек
//в экранной системе координат.
    for i:=0 to N do
    begin
        u[i]:=trunc(c*x[i]+d);
        v[i]:=trunc(g*y[i]+h);
```

```
    end;
    Form1.Image1.Canvas.Color:= clGray;
    Form1.Image1.Canvas .Pen .Mode:= pmNot;
//Рисование графика функции на компоненте Image1.
    Form1.Image1.Canvas.MoveTo(u[0],v[0]);
    Form1.Image1.Canvas.Pen.Width:= 2;
    Form1.Image1.Canvas.Pen.Color:= clGreen;
    for i:=1 to N do
        Form1.Image1.Canvas.LineTo(u[i],v[i]);
    Form1.Image1.Canvas.Pen.Width:= 1 ;
    Form1.Image1.Canvas.Pen.Color:= clBlack;
//Рисование осей координат, если они попадают
//в область графика.
Form1.Image1.Canvas.MoveTo(trunc(x0),trunc(h));
    if (trunc(h)>yk) and
    (trunc(h)<trunc(Form1.Image1.ClientHeight-y0))
    then
Form1.Image1.Canvas.LineTo(trunc
        (Form1.Image1.ClientWidth -xk),trunc(h));
Form1.Image1.Canvas.MoveTo(trunc(d),trunc(yk));
    if (trunc(d)>x0) and
    (trunc(d)<trunc(Form1.Image1.ClientWidth-xk))
    then
   Form1.Image1.Canvas.LineTo(trunc(d),
           trunc(Form1.Image1.ClientHeight -y0));
//Рисование линий сетки,
//вычисление расстояния между линиями сетки в
//"компонентной" системе координат,
//перпендикулярных оси OX
    dx:=(Form1.Image1.ClientWidth -x0-xk)/KX;
//Выбираем тип линии для линий сетки,
    for i:=0 to KX do
    begin
//первую и последнюю линии сетки рисуем обычной
//сплошной линией,
        if (i=0) or (i=KX) then
           Form1.Image1.Canvas.Pen.Style:=psSolid
//остальные - рисуем пунктирными линиями.
```

```
        else
            Form1.Image1.Canvas.Pen.Style:=psDash;
//Рисование линии сетки, перпендикулярной оси Ox.
Form1.Image1.Canvas.MoveTo(trunc(x0+i*dx),
                                    trunc(yk));
Form1.Image1.Canvas.LineTo(trunc(x0+i*dx),
            trunc(Form1.Image1.ClientHeight -y0));
    end;
//Вычисление расстояния между линиями сетки,
//перпендикулярными оси OY,
//в "компонентной" системе координат.
    dy:=(Form1.Image1.ClientHeight  -y0-yk)/KY;
    for i:=0 to KY do
    begin
//Первую и последнюю линии сетки
//рисуем обычной сплошной линией,
    if (i=0) or (i=KY) then
        Form1.Image1.Canvas.Pen.Style:=psSolid
//остальные - рисуем пунктирными линиями.
    else
        Form1.Image1.Canvas.Pen.Style:=psDash;
//Рисование линии сетки, перпендикулярной оси OY.
    Form1.Image1.Canvas.MoveTo(trunc(x0),
                                  trunc(yk+i*dy));
Form1.Image1.Canvas.LineTo(trunc
    (Form1.Image1.ClientWidth-xk),trunc(yk+i*dy));
    end;
    Form1.Image1.Canvas.Pen.Style:=psSolid;
//Вывод подписей под осями,
//определяем dx - расстояние между выводимыми
//под осью OX значениями.
    dx:=(b-a)/KX;
    tempy:=
        trunc(Form1.Image1.ClientHeight -y0+10);
    for i:=0 to KX do
    begin
//Преобразование выводимого значения в строку.
        Str(a+i*dx:5:2,s);
```

```
//Вычисление x-координаты выводимого
//под осью OX значения в "компонентной" системе.
    tempx:=trunc(x0+i*(Form1.Image1.ClientWidth -
                                    x0-xk)/KX)-10;
//Вывод значения под осью OX
    Form1.Image1.Canvas.TextOut(tempx,tempy,s);
    end;
    if (trunc(d)>x0) and
     (trunc(d)<Form1.Image1.ClientWidth-xk) then
Form1.Image1.Canvas.TextOut(trunc(d)-5,tempy,'0');
//Определяем dy - расстояние между
//выводимыми левее оси OY значениями.
    dy:=(max-min)/KY;
    tempx:=5;
    for i:=0 to KY do
    begin
//Преобразование выводимого значения в строку.
        Str(max-i*dy:5:2,s);
//Вычисление y-координаты выводимого левее оси OY
//значения в "компонентной" системе.
        tempy:=trunc(yk-5+
        i*(Form1.Image1.ClientHeight- y0-yk)/KY);
//Вывод значения левее оси OY.

    Form1.Image1.Canvas.TextOut(tempx,tempy,s);
    end;
    if (trunc(h)>yk) and (trunc(h)<
        Form1.Image1.ClientHeight-y0) then
        Form1.Image1.Canvas.TextOut(tempx+10,
                                    runc(h)-5,'0');
    tempx:=trunc(x0+i*(Form1.Image1.ClientWidth -
                                    x0-xk)/2);
    Form1.Image1.Canvas.TextOut(tempx,10,
                                'График функции');
end;
{ TForm1 }
procedure TForm1.FormCreate(Sender: TObject);
var s : string; kod : integer;
```

```
begin
    N:=100;
    x0:=40;
    xk:=40;
    y0:=40;
    yk:=40;
    repeat
        s:=InputBox('График непрерывной функции',
                    'Введите левую границу','-10');
        Val(s,a,kod);
    until kod=0;
    repeat
        s:=InputBox('График непрерывной функции',
                    'Введите правую границу','10');
        Val(s,b,kod);
    until kod=0;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Graphika(a, b);
end;
initialization
   {$I unit1.lrs}
end.
```

При запуске проекта появится запрос ввода левой (рис. 10.6) и правой (рис. 10.7) границ интервала. После этого появится окно формы с кнопкой `График`. После щелчка по кнопке на форме прорисуется график функции (рис. 10.8).

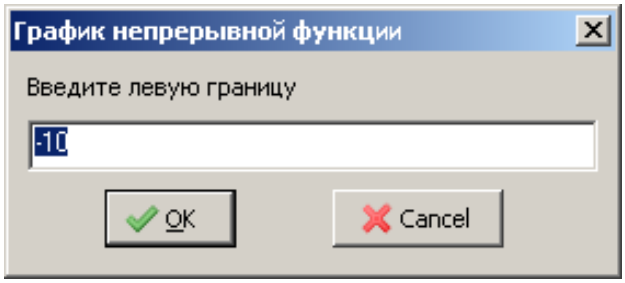

Рисунок 10.6: Окно ввода левой границы

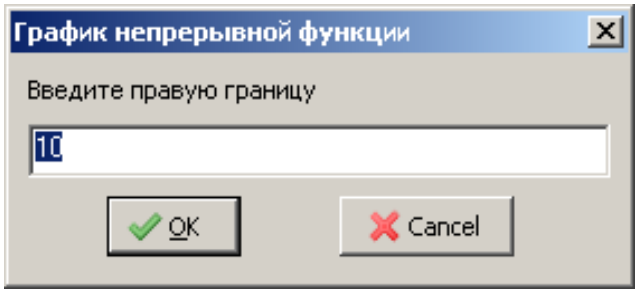

Рисунок 10.7: Окно ввода правой границы интервала

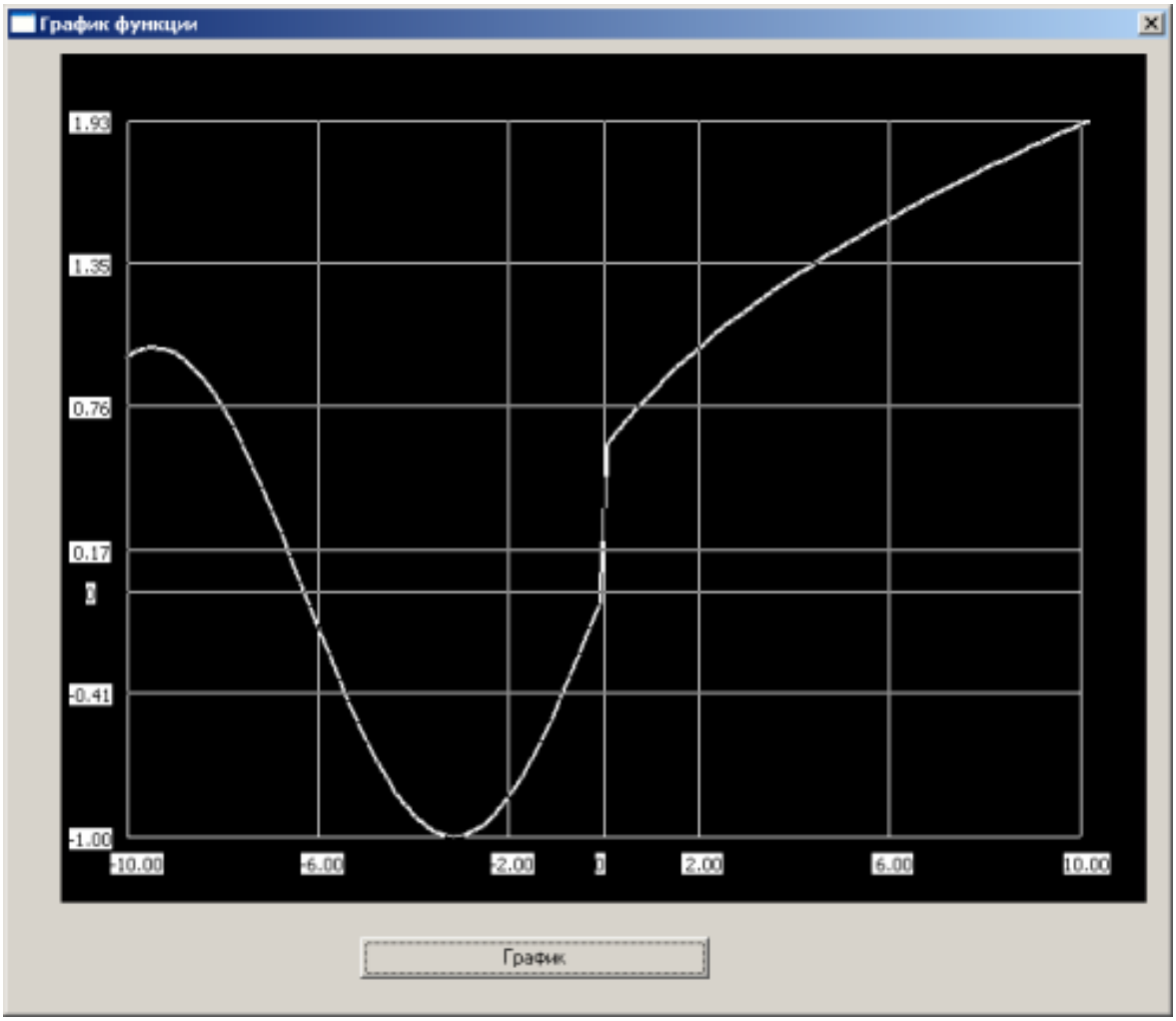

Рисунок 10.8: График функции

10.3 Задачи для самостоятельного решения

Построить график функции *f(x)* на интервале *[a;b]*. Функции заданы в табл. 10.5.

Таблица 10.5. Варианты заданий

№ варианта	f(x)	№ варианта	f(x)
1	$\begin{cases} \sin\dfrac{x^3+2}{3}, если\ x \leqslant 5 \\ \dfrac{\sqrt[5]{1+x}}{3}, если\ x > 5 \end{cases}$	9	$\begin{cases} \dfrac{x^2-x+2}{x^3}, если\ x \geq 1 \\ \sqrt[5]{\dfrac{e^x}{2}+x^2}, если\ x < 1 \end{cases}$
2	$\begin{cases} \tan(\pi+x), если\ x \leqslant -1 \\ \sqrt[3]{e^{x+1}}, если\ x > -1 \end{cases}$	10	$\begin{cases} \sin\dfrac{x+3,6}{x^3}, если\ x < 0 \\ \sqrt[3]{1+x}, если\ x \geq 0 \end{cases}$

№ варианта	f(x)	№ варианта	f(x)
3	$\begin{cases}\log(\frac{1+x}{3}), если\ x>2\\ \sqrt[3]{1+x^3}, если\ x\le 2\end{cases}$	11	$\begin{cases}\frac{x^3+3x^2}{3}, если\ x\leqslant -2\\ \sqrt[3]{\log(1{,}5+x^2)}, если\ x>-2\end{cases}$
4	$\begin{cases}\frac{\sqrt[2]{\cos(x+2)^3}}{3{,}5}, если\ x>2\\ \sin(x+2)^2, если\ x\le 2\end{cases}$	12	$\begin{cases}e^{\frac{x}{4}}, если\ x>-2{,}5\\ \sqrt[5]{x^2}, если\ x\le -2{,}5\end{cases}$
5	$\begin{cases}\cos\frac{x-6}{x-3}, если\ x>5\\ \sqrt{1+x^4}, если\ x\le 5\end{cases}$	13	$\begin{cases}\cos\frac{x+2}{x^3}, если\ x\ge 4\\ \sqrt[3]{e^x+x^2}, если\ x<4\end{cases}$
6	$\begin{cases}x^3+3x^2, если\ x\leqslant -1\\ \sqrt[3]{\ln(10{,}5+x)}, если\ x>-1\end{cases}$	14	$\begin{cases}x^2-x+2, если\ x\ge -1\\ \sqrt[5]{e^x+7}, если\ x<-1\end{cases}$
7	$\begin{cases}\cos\frac{x^2+2}{x}, если\ x>3\\ \sqrt{12+x^2}, если\ x\le 3\end{cases}$	15	$\begin{cases}\log(\frac{1+x^3}{2}), если\ x>2{,}5\\ \sqrt[3]{1+2x^2-x^3}, если\ x\le 2{,}5\end{cases}$
8	$\begin{cases}x^3\cdot\sin(x), если\ x>-3{,}5\\ \sqrt[5]{\left\|\frac{x}{2}\right\|}, если\ x\leqslant -3{,}5\end{cases}$	16	$\begin{cases}x^3+3(x+2)^2, если\ x\leqslant 3\\ \sqrt[5]{(\sin(10{,}5+x))^2}, если\ x>3\end{cases}$

Построить графики функций *f1(x), f2(x), f2(x)* в одной системе координат на интервале *[a;b]*. Функции заданы в табл. 10.6.

Таблица 10.6. Варианты заданий

№ варианта	f1(x)	f2(x)	f3(x)
17	$\cos\frac{x-6}{x^2+3}$	$x^3\cdot\sin(x)$	$\sqrt{(5x^3)}\cdot\sin(x^2)$
18	$1+2x^2-x^3$	$e^{\frac{x}{2}}+7$	$\sin\left(\frac{x}{3}\right)$
19	$\sqrt[3]{1+x^3}$	$14+2x^2-3x^3$	$\cos(\sin(x))$
20	$\sin\left(\frac{x}{3}+e^x\right)$	$\sqrt{7+2x^4}$	$\sqrt[3]{(3+5x^2-x^3)^2}$

№ варианта	f1(x)	f2(x)	f3(x)
21	$\sqrt[5]{7-x^3}$	$\cos\left(\frac{x}{2}+\pi\right)$	$\cos(\sin(x^2))$
22	$\sqrt[3]{\ln(12+x^2)}$	$e^{\frac{x}{5}}$	$\cos\left(\frac{x}{\pi}\right)$
23	$\cos\left(\frac{x+2}{\pi}\right)$	$\sqrt[4]{1+x^6+2x^2}$	$3x^4-5x^2+7x-2$
24	$\sqrt[3]{(1+x^2-x^3)^2}$	$2\cdot\sin\left(\frac{x}{2}+\pi\right)$	$5x^2-3x^3$
25	$\frac{x}{2}+\sin(2\cdot x\cdot\pi)$	$\sqrt[3]{(1+x)(x^3-4)^2}$	$e^{\frac{x}{7}}+4$

Вместо заключения

Перевернута последняя страница книги. Что теперь? Авторы надеются, что знакомство с языком Free Pascal будет только первым этапом в изучении программирования. Желание читателя что-то исправить в книге — переписать приведенные в книге программы, предложить более простые и быстро работающие алгоритмы, написать свои программы и модули — будет лучшей благодарностью авторам. Если у вас, читатели, появилось подобное желание, то мы выполнили свою задачу — научили вас основам программирования.

Следующим этапом в освоении программирования будет разработка ваших алгоритмов и написание реально работающих программ для различных операционных систем.

Алфавитный указатель

Литература

1. Алексеев Е.Р., Чеснокова О.В. Турбо Паскаль 7.0. - М.:НТ Пресс, 2006. - 320 с.:ил. - (Полная версия).

2. Алексеев Е.Р. Учимся программировать на Microsoft Visual C++ и Turbo C++ Explorer (под общей редакцией Чесноковой О.В.)/Алексеев Е.Р. - М.:НТ Пресс, 2007. - 352 с.:ил. - (Самоучитель).

3. Фаронов В.В. Delphi. Программирование на языке высокого уровня: Учебник для вузов. - Спб.:Питер, 2005.-640 с.:ил.

4. Чеснокова О.В. Delphi 2007. Алгоритмы и программы. Учимся программировать на Delphi 2007/Чеснокова О.В. Под общ. ред. Алексеева Е.Р.- М.:НТ Пресс, 2008. - 368 с.:ил.

5. Бронштейн И.Н., Семендяев К.А. Справочник по математике для инженеров и учащихся вузов.- М.:Наука, 1981. - 720 с.

6. *GNU Pascal — Википедия.* URL: .http://ru.wikipedia.org/wiki/GNU_Pascal (дата обращения 03.08.2009).

7. *Free Pascal - Advanced open source Pascal compiler for Pascal and Object Pascal - Home Page.* URL: http://www.freepascal.org (дата обращения 03.08.2009).

8. *GNU Pascal.* URL: http://www.gnu-pascal.de/gpc/h-index.html (дата обращения 03.08.2009).

9. Main Page/ru – Free Pascal wiki. URL: http://wiki.freepascal.org/Main_Page/ru (дата обращения 03.11.2009).

10. Free Pascal.ru - Информационный портал для разработчиков на Free Pascal & Lazarus & MSE. URL http://www.freepascal.ru (дата обращения: 03.11.2009).

11. Lazarus – News. URL: http://www.lazarus.freepascal.org (дата обращения 03.11.2009).

12. Lazarus – Википедия. URL: http://ru.wikipedia.org/wiki/Lazarus (дата обращения 03.11.2009).