Машинная графика

1 Теоретическая часть

1.1 Основные понятия

Рассмотрим понятие **экранной системы координат**. Экранная система координат - **дискретная прямоугольная** система координат, то есть координата каждой точки на экране - **два целых числа**. Помимо этого, экранная система координат **ограничена**, то есть все точки имеют координаты по горизонтальной оси в диапазоне $\left[x_s^{min}, x_s^{max}\right]$ и по вертикальной оси в диапазоне $\left[y_s^{min}, y_s^{max}\right]$ (здесь и далее обозначение x_s, y_s - координаты в экранной системе координат, индекс s обозначает *screen*.

Как правило, начало экранной системы координат связывают с верхним левым углом экрана или окна, в котором происходит отображение графики. При этом координаты верхнего левого угла принимаются за $\left(x_s^{min}, y_s^{min}\right) = (0, 0)$, а правого нижнего - за $(x_s^{max}, y_s^{max}) = (w-1, h-1)$, где w - *width* - ширина окна отображения, h - *height* - высота окна отображения.

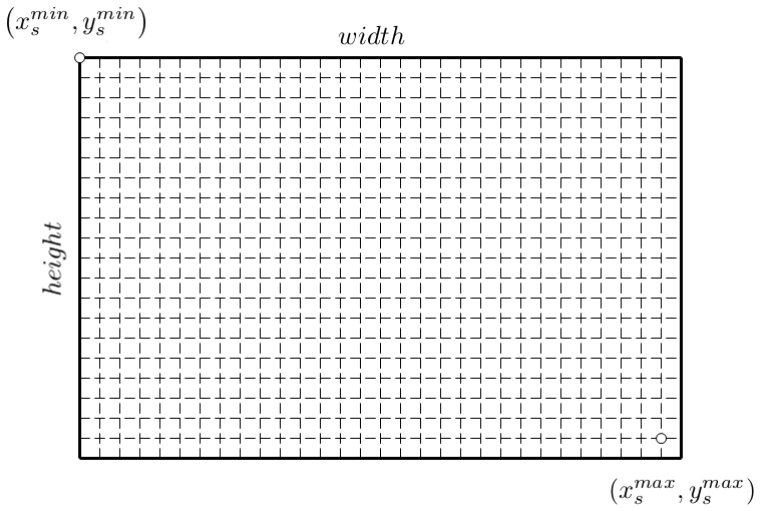

Рис. 1: Экранная система координат

При таком выборе граничных точек вертикальная ось получается направлена вниз.

У точек в экранной системе координат помимо расположения есть ещё **конечный размер** и **цвет**. Будем говорить, что множество всех точек данной экранной системы координат составляют **изображение**. Поскольку такое изображение состоит из конечного множества точек конечного размера, будем называть такое изображение **растровым**.

В связи с этим введем термин **растеризация** - построение **растрового изображения** путем проецирования геометрических объектов на экранную систему координат.

Следует обратить внимание на расположение правой нижней граничной точки на рисунке 1: выглядит так, как будто мы ошиблись и вместо граничной точки выбрали какую-то другую. Однако это не так: отмеченные «точки» на самом деле показывают верхний левый угол той точки, которая будет закрашена. Если сделать цвет крайних точек серым, то можно получить следующее изображение, как на рисунке 2.

Рассмотрим задачу растеризации графика, построенного в декартовой системе координат.

1.2 Преобразования координат

Пусть нам заданы координаты x_c^{min} и x_c^{max} в декартовой системе координат (индекс c означает *cartesian* - декартова) - начало и конец интервала по оси oX, который требуется растеризовать. Пусть также заданы (или вычислимы) координаты y_c^{min} и y_c^{max} - начало и конец интервала по оси oY, который требуется растеризовать. «Совместим» экранную и декартову системы координат так, чтобы точка x_c^{min} «совпала» с x_s^{min}, x_c^{max} - с x_s^{max}, y_c^{max} - с y_s^{min}, y_c^{min} - с y_s^{max}:

Введём дополнительные обозначения:

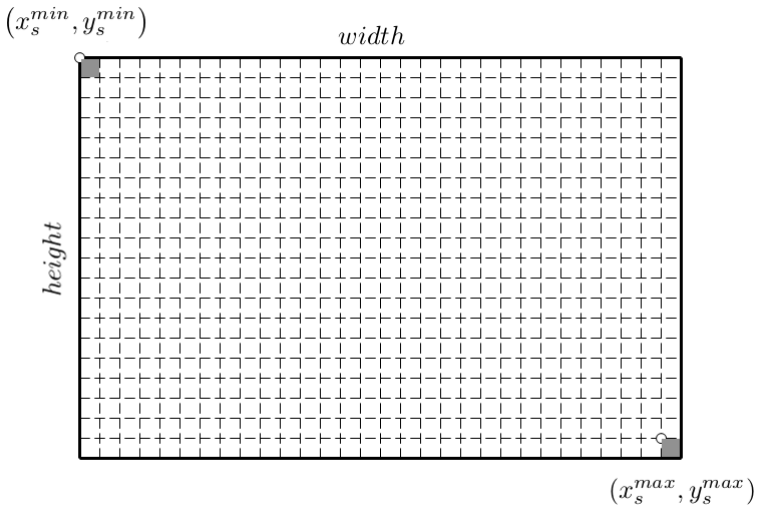

Рис. 2: Экранная система координат с отмеченными крайними точками

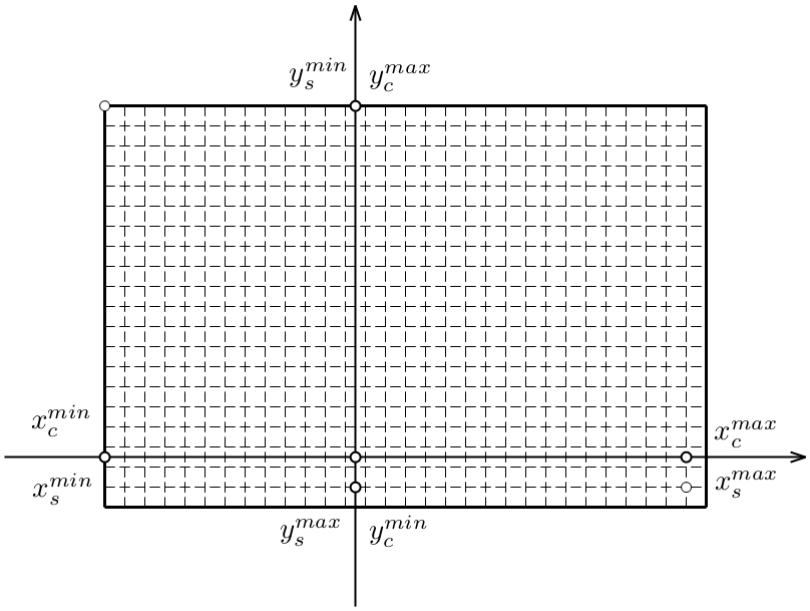

Рис. 3: Декартова и экранная системы координат

(x_s^0, y_s^0) - расположение начала координат декартовой системы координат в экранной системе координат,

m_x - масштаб для перевода из декартовой системы координат в экранную по оси oX,

m_y - масштаб для перевода из декартовой системы координат в экранную по оси oY.

В этих обозначениях перевод из декартовой системы координат будет выглядеть так:

$$\begin{cases} x_s = \lfloor x_s^0 + m_x \cdot x_c \rfloor \\ y_s = \lfloor y_s^0 - m_y \cdot y_c \rfloor \end{cases} \tag{1}$$

В системе (1) скобки $\lfloor \rfloor$ означают округление в меньшую сторону.

Исходя из этого уравнения и соответствия точек, найдём выражения для неизвестных величин.

1. Поскольку x_s^{min} совпадает с x_c^{min}, а x_s^{max}- с x_c^{max}, запишем:

$$\begin{cases} x_s^{min} = \lfloor x_s^0 + m_x \cdot x_c^{min} \rfloor \\ x_s^{max} = \lfloor x_s^0 + m_x \cdot x_c^{max} \rfloor \end{cases}$$

Допуская, что в правых частях $\lfloor a \rfloor = a$, получаем:

$$\begin{cases} x_s^{min} = x_s^0 + m_x \cdot x_c^{min} \\ x_s^{max} = x_s^0 + m_x \cdot x_c^{max} \end{cases} \tag{2}$$

Решая систему (2), получаем:

$$\begin{cases} m_x \left(x_c^{max} - x_c^{min} \right) = x_s^{max} - x_s^{min} \\ x_s^0 = x_s^{min} - m_x \cdot x_c^{min} \end{cases}$$

$$\begin{cases} m_x = \frac{x_s^{max} - x_s^{min}}{x_c^{max} - x_c^{min}} \\ x_s^0 = x_s^{min} - m_x \cdot x_c^{min} \end{cases} \tag{3}$$

Заметим, что вторая строка системы (3) содержит величину m_x, формально неизвестную, однако её выражение указано в первой строке системы, и для упрощения реализации следует вычислять x_0^s по указанной формуле

2. Заметим, что по вертикальной оси y_s^{min} будет совпадать с y_c^{max}, а y_s^{max} - с y_c^{min}. Это обусловлено различным направлением вертикальных осей в декартовой и экранной системах координат. Запишем:

$$\begin{cases} y_s^{min} = \lfloor y_s^0 - m_y \cdot y_c^{max} \rfloor \\ y_s^{max} = \lfloor y_s^0 - m_y \cdot y_c^{min} \rfloor \end{cases}$$

Принимая аналогичное п.1 допущение, получаем:

$$\begin{cases} y_s^{min} = y_s^0 - m_y \cdot y_c^{max} \\ y_s^{max} = y_s^0 - m_y \cdot y_c^{min} \end{cases} \tag{4}$$

Решая систему, получаем:

$$\begin{cases} m_y \left(y_c^{max} - y_c^{min} \right) = y_s^{max} - y_s^{min} \\ y_s^0 = y_s^{min} + m_y \cdot y_c^{max} \end{cases}$$

$$\begin{cases} m_y = \frac{y_s^{max} - y_s^{min}}{y_c^{max} - y_c^{min}} \\ y_s^0 = y_s^{min} + m_y \cdot y_c^{max} \end{cases} \tag{5}$$

3. Переход от экранных координат к декартовым, строго говоря, должен давать не одну точку, а некоторую область, поскольку значения в точках $\left[x_c, x_c + \frac{1}{m_x} \right)$ будут проецироваться в одну и ту же точку на горизонтальной оси экранных координат (при условии, конечно же, что $x_c \cdot m_x \in \mathbb{Z}$). Тем не менее, мы можем указать, в какую точку в декартовых координатах будет проецироваться левый верхний угол точки экранных координат:

$$\begin{cases} x_c = \frac{x_s - x_s^0}{m_x} \\ y_c = \frac{y_s^0 - y_s}{m_y} \end{cases}$$

Заметим, что в общем случае величины y_c^{min} и y_c^{max} могут быть не заданы. Как правило, в таких случаях прибегают к одному из следующих методов:

- Если требуется построить график функции $y = f(x)$, то вычисляется (точно или приближенно) наименьшее и наибольшее значения заданной функции на отрезке $\left[x_c^{min}, x_c^{max} \right]$
- Если требуется выполнить условие $m_x = m_y$, то значения y_c^{min} и y_c^{max} допускается не вычислять. При этом значение y_s^0 принимается равным $\frac{y_s^{max} - y_s^{min}}{2}$ или вычисляется из значений функции (например, как некоторое усредненное значение). В последнем случае можно принять, что среднее значение функции на отрезке должно соответствовать центру изображения, и тогда вычислить значение y_s^0 как

$$\frac{y_s^{max} - y_s^{min}}{2} = y_s^0 - m_y \frac{\int_{x_c^{min}}^{x_c^{max}} f(x) dx}{x_c^{max} - x_c^{min}}$$

$$y_s^0 = \frac{y_s^{max} - y_s^{min}}{2} + m_y \frac{\int_{x_c^{min}}^{x_c^{max}} f(x)dx}{x_c^{max} - x_c^{min}} \quad (6)$$

Заметим, что в зависимости (6) можно заменить интеграл на среднее арифметическое значений в точках, соответствующих экранным координатам:

$$y_s^0 = \frac{y_s^{max} - y_s^{min}}{2} + m_y \frac{\sum_{x_s=x_s^{min}}^{x_s^{max}} f(x_c(x_s))}{x_s^{max} - x_s^{min}}$$

где

$$x_c(x_s) = \frac{x_s - x_s^0}{m_x}$$

2 Практические рекомендации

2.1 Модули в языке Pascal. Модуль GraphABC

В современных редакциях языка программирования Pascal существует концепция подключаемых **модулей**. Модулем может являться файл с расширением .pas, в начале которого стоит конструкция unit название_модуля;, или предварительно скомпилированный модуль (в среде PascalABC.Net - с расширением .pcu). В качестве названия модуля могут выступать латинские буквы, цифры и символы подчёркивания. Название модуля, указанное после ключевого слова unit, должно совпадать с именем файла (без расширения).

Модули используются для выделения и группирования функций и процедур, а также констант, пользовательских типов и глобальных объектов, в отдельные единицы компиляции. Это может делаться для ускорения процесса компиляции программы (модули, в которых не произошло изменений, можно использовать повторно), для разделения программы на осмысленные элементы (например, модуль работы с файлами, модуль обработки математической модели, модуль вывода отчёта на экран), для упрощения командной работы над проектом (в команде каждый человек может работать со своим модулем, это упрощает процесс слияния изменений). Кроме того, модули используются для взаимодействия со **сторонними библиотеками**, возможно, написанными на других языках программирования.

Для того, чтобы указать, что в программе используется тот или иной модуль, следует в начале программы (перед объявлением собственных типов, констант, функций и процедур) указать:

```
1 { Пример подключения модуля }
2 uses
3   PABCSystem, System, GraphABC;
4 { далее идёт исходный код программы, }
5 { использующей те или иные элементы  }
6 { указанных модулей.                 }
```

Подключение модуля позволяет использовать все **экспортируемые** (то есть являющиеся интерфейсом модуля) элементы этого модуля: функции, процедуры, типы данных, константы и переменные. Кроме того, при загрузке модуля может выполняться некоторый код, называемый кодом **инициализации** модуля: так, при подключенном модуля GraphABC, при запуске программы будет создаваться окно графического вывода. При этом в самой программе будут доступны функции и процедуры для работы с этим окном.

Рассмотрим подробнее модуль GraphABC, распространяющийся с компилятором PascalABC.Net.

Этот модуль используется для вывода графики на экран. Поскольку современные операционные системы используют концепцию **окон** для представления своих приложений, данный модуль позволяет выводить графику не в эксклюзивном режиме (когда всё экранное пространство занято одной программой), а в специально создаваемое для этого окно. Это окно создаётся автоматически при запуске программы. При этом вывод с помощью процедуры writeln будет производиться в это окно, а чтение с помощью процедуры readln будет создавать в окне специальную строку ввода

При выводе графики на экран с помощью модуля GraphABC используются концепции **пера** и **кисти**. Перо используется для задания параметров отображения **линий**, кисть - для задания параметров **заливки**. Помимо этого, присутствует возможность задавать цвет для каждого пикселя окна индивидуально.

Будем рассматривать часть процедур и функций, доступных при подключении модуля GraphABC.

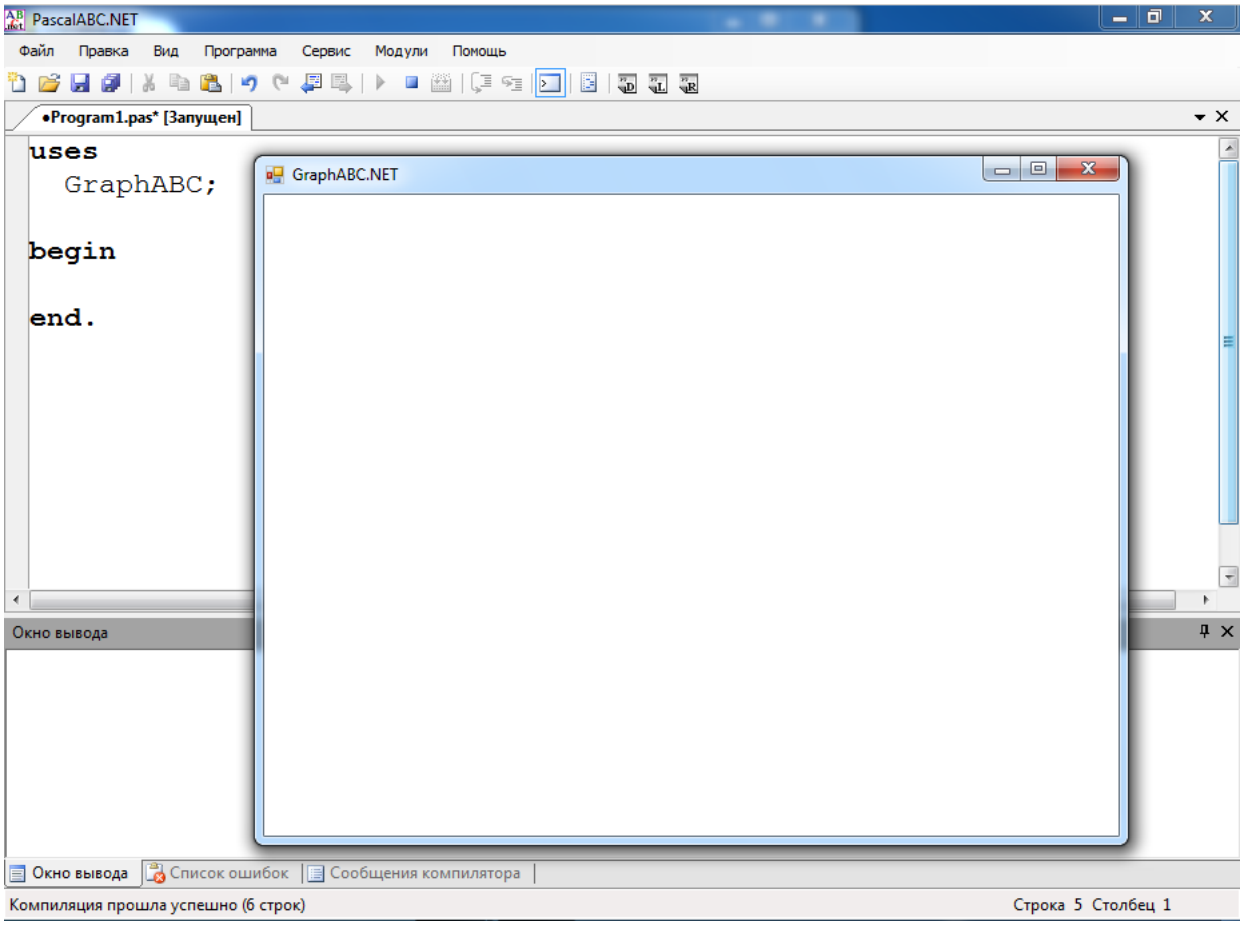

Рис. 4: Запуск программы с подключенным модулем GraphABC

Функция window возвращает **объект**, описывающий текущее состояние окна. **Свойствами** этого объекта являются размеры окна, его положение на экране, его название (заголовок), а также флаг, указывающий, может ли пользователь изменять размеры окна. Доступ к этим свойствам осуществляется с помощью оператора «точка». Покажем, как можно пользоваться этими свойствами:

```
1  uses
2    GraphABC;
3
4  begin
5    { Устанавливаем название окна      }
6    window.Caption := 'Hello GraphABC';
7    { Устанавливаем расположение окна }
8    window.Top := 200;
9    window.Left := 300;
10   { Выводим размер окна на экран     }
11   writeln('Window size: ',
12            window.Width, 'x',
13            window.Height);
14   for var i := 1 to 10 do begin
15     var a : Integer;
16     readln(a);
17     writeln('You entered: ', a);
18   end;
19 end.
```

Обращение к пикселям (точкам в экранных координатах) можно производить с помощью функции GetPixel и процедуры PutPixel. Их описание:

function GetPixel(x, y: Integer): Color - функция, возвращающая значение цвета точки с координатами x, y.

procedure PutPixel(x, y: Integer; c: Color) - процедура, устанавливающая цвет точки с коорди-

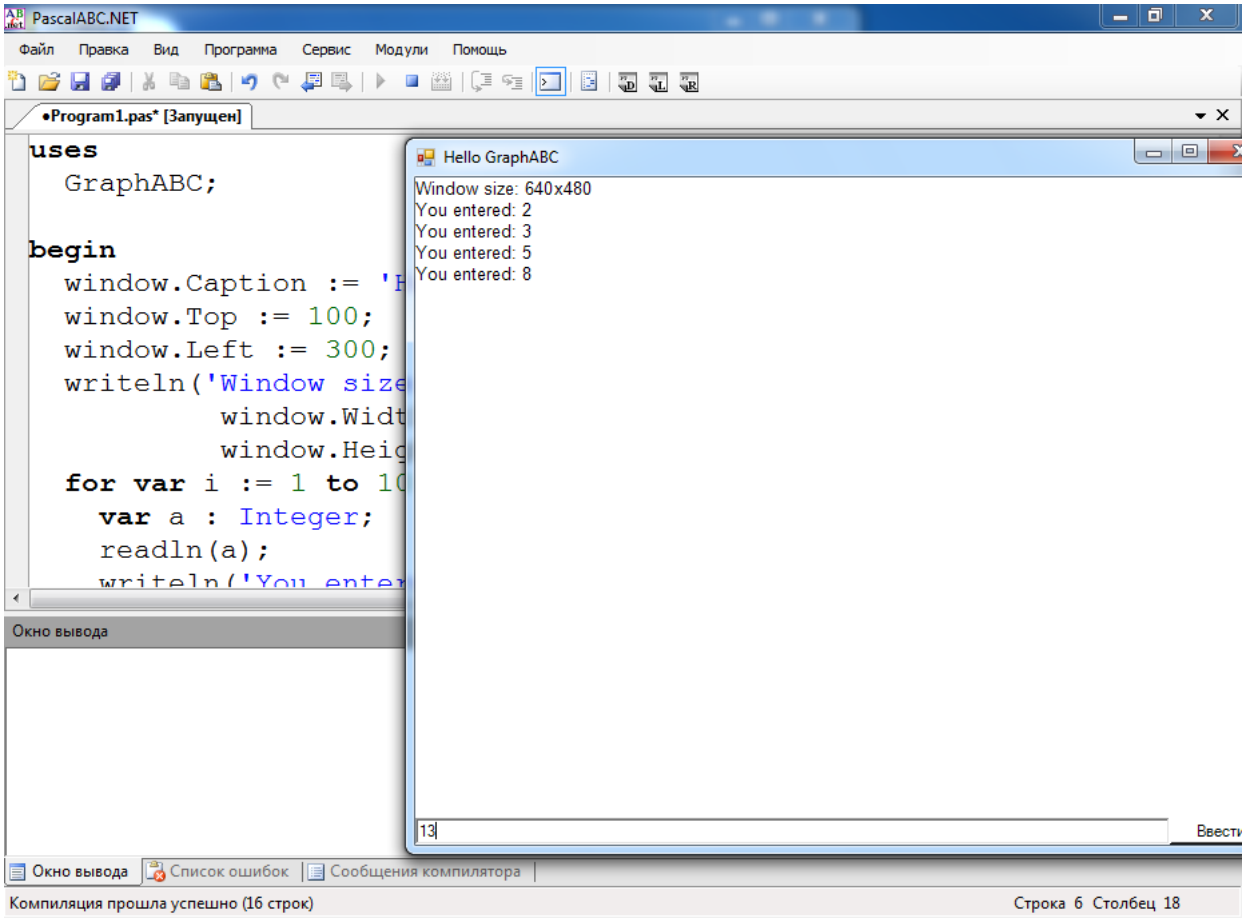

Рис. 5: Пример работы программы

натами x, y в значение c.

В качестве цветов можно использовать определённые в модуле GraphABC константы, либо задавать его, указывая интенсивности красного, зелёного и синего цветов.

Для рисования линий используются процедуры Line, LineTo и MoveTo. Их описание:

procedure Line(x1, y1, x2, y2: Integer) - процедура, рисующая линию от точки (x1, y1) до точки (x2, y2). В качестве параметров линии будет использоваться текущее состояние объекта Pen.

procedure LineTo(x, y: Integer) - процедура, проводящая линию от текущего положения **графического курсора** до указанной точки (x, y). При этом новым положением графического курсора станет точка (x, y). По умолчанию в начале работы программы графический курсор находится в точке (0, 0).

procedure MoveTo(x, y: Integer) - процедура, перемещающая графический курсор в точку (x, y). Перемещение происходит без рисования линии.

```
1  uses
2    GraphABC;
3  { Шаг изменения координат }
4  var step: Integer;
5  begin
6    { Указываем размер окна и шаг }
7    window.Width := 400;
8    window.Height := 400;
9    step := 20;
10   for var i := 0 to window.Width div step do begin
11     var x := i * step;
12     var y := window.Height - i * step;
13     var x2 := window.Width - x;
14     var y2 := window.Height - y;
15     { Верхняя часть рисунка - красная }
16     Pen.Color := clRed;
17     Line(0, y, x, 0);
```

```
18        { Нижняя - синяя }
19        Pen.Color := clBlue;
20        MoveTo(x2, window.Height);
21        LineTo(window.Width, y2);
22      end;
23    end.
```

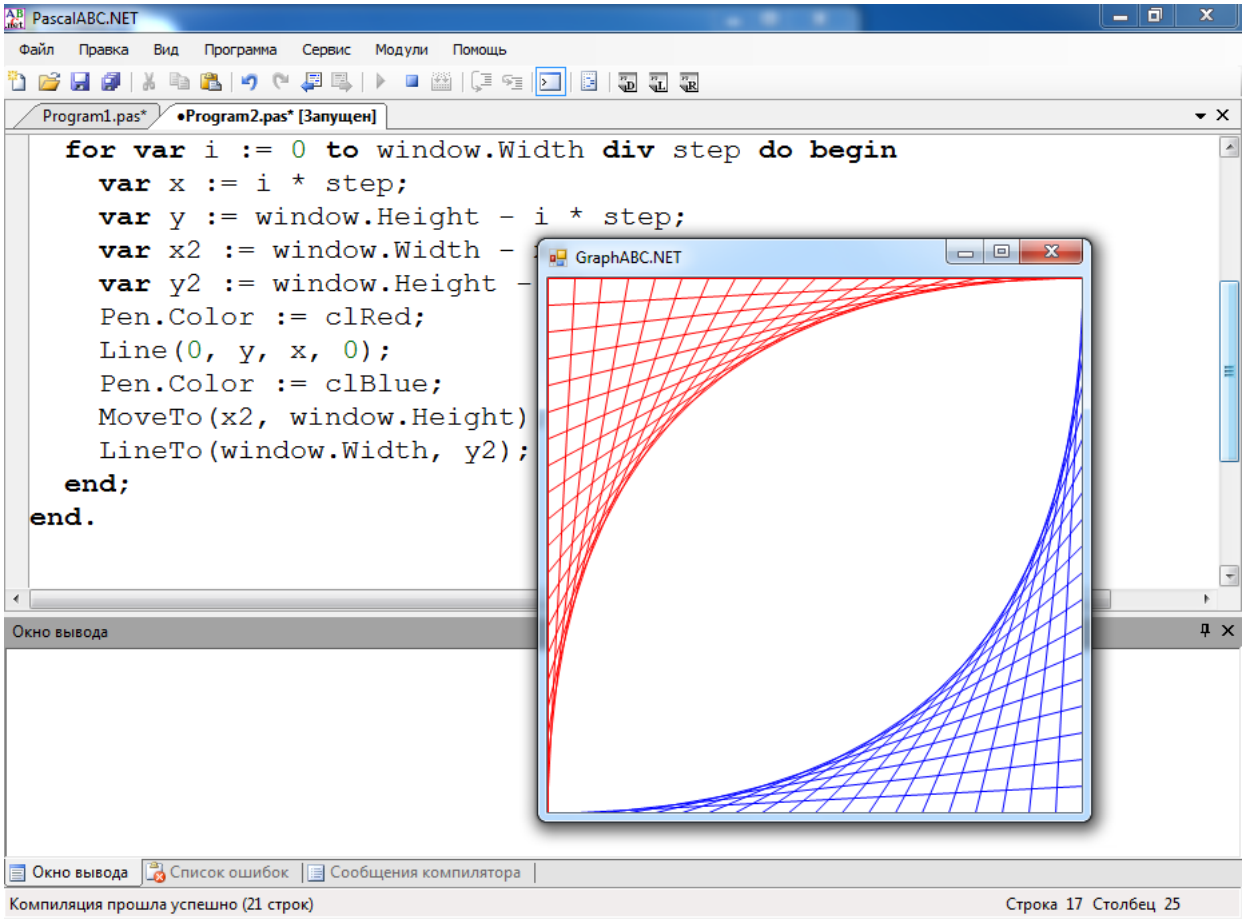

Рис. 6: Пример работы программы

Это не является полным описанием всех возможностей модуля GraphABC, однако для текущего задания этого должно быть достаточно.

2.2 Внутренние функции

В языках программирования семейства Pascal существует возможность определять процедуры и функции внутри других процедур и функций. Такие «внутренние» функции получают доступ к локальным переменным заключающей их функции или процедуры. При этом вызывать эти «внутренние» функции возможно только из тела заключающей функции или из функций, расположенном на том же уровне.

Использование таких функций позволяет реализовать принцип **инкапсуляции** (сокрытия деталей реализации от остальных частей программы) и тем самым ограничить количество потенциальных ошибок в программе. С другой стороны, использование таких внутренних функций позволяет упростить код программы.

Приведём пример использования данной возможности:

```
1  uses
2    GraphABC;
3
4  { Функция, график которой мы будем строить }
5  function plotted_function(x: Real): Real;
6  begin
7    result := x * x - 3 * x;
```

```
 8 end;
 9
10 { Процедура построения графика функции f }
11 { Параметры:                               }
12 {   f: функция одной переменной, график    }
13 {      которой требуется построить         }
14 {   xmin, xmax: границы области построения}
15 {      графика функции по горизонтальной   }
16 {      оси                                 }
17 {   ymin, ymax: границы области построения}
18 {      графика функции по вертикальной оси}
19 procedure Plot(f: function(x: Real): Real;
20                 xmin, xmax: Real;
21                 ymin, ymax: Real);
22 var
23   { Масштабы по осям }
24   mx, my: Real;
25   { Начало координат ДСК }
26   x0s, y0s: Real;
27
28   { Вспомогательные функции для перевода }
29   { в различные системы координат:       }
30
31   { экранная -> декартова по оси oX:     }
32   function toXs(xc: Real): Integer;
33   begin
34     result := floor(x0s + mx * xc);
35   end;
36
37   { экранная -> декартова по оси oY:     }
38   function toYs(yc: Real): Integer;
39   begin
40     result := floor(y0s - my * yc);
41   end;
42
43   { декартова -> экранная по оси oX:     }
44   function toXc(xs: Integer): Real;
45   begin
46     result := (xs - x0s) / mx;
47   end;
48
49   { декартова -> экранная по оси oY:     }
50   function toYc(ys: Integer): Real;
51   begin
52     result := (y0s - ys) / my;
53   end;
54
55 begin
56   { Задаём границы в экранных координатах }
57   var xsmin := 0; var xsmax := window.Width - 1;
58   var ysmin := 0; var ysmax := window.Height - 1;
59
60   { Пересчитываем масштаб }
61   mx := (xsmax - xsmin) / (xmax - xmin);
62   my := (ysmax - ysmin) / (ymax - ymin);
63
64   { Находим начало координат ДСК }
65   x0s := xsmin - mx * xmin;
66   y0s := ysmin + my * ymax;
67
68   { Находим координаты первой точки }
69   var xs, ys: Integer;
```

```
70      xs := toXs(xmin);
71      ys := toYs(f(xmin));
72      MoveTo(xs, ys);
73      { Строим весь оставшийся график }
74      for xs := xsmin + 1 to xsmax do begin
75        var xc := toXc(xs);
76        var yc := f(xc);
77        ys := toYs(yc);
78        LineTo(xs, ys);
79      end;
80
81  end;
82
83  begin
84      Plot(plotted_function, -2, 3, -4, 4);
85  end.
```

Здесь внутренние функции использованы для упрощения написания кода процедуры, строящей график. Вместо написания сравнительно громоздкого выражения для проецирования точки из декартовой системы координат в экранную теперь достаточно использовать функции toXs() и toYs(), описанные на строках 32-35 и 38-41 приведенного листинга. Обратный перевод осуществляется функциями toXc() и toYc(). Заметим, что вне указанной процедуры использование данных функций бессмысленно (поскольку вне этой процедуры нет масштабов и положения начала координат декартовой системы координат), и реализация их в виде внутренних функций оправдана и очевидна. Заметим также, что если бы мы нашли ошибку в выводе выражений для этих функций, нам достаточно было бы изменить только реализацию одной или нескольких функций, а не искать по коду вхождение неправильного выражения.

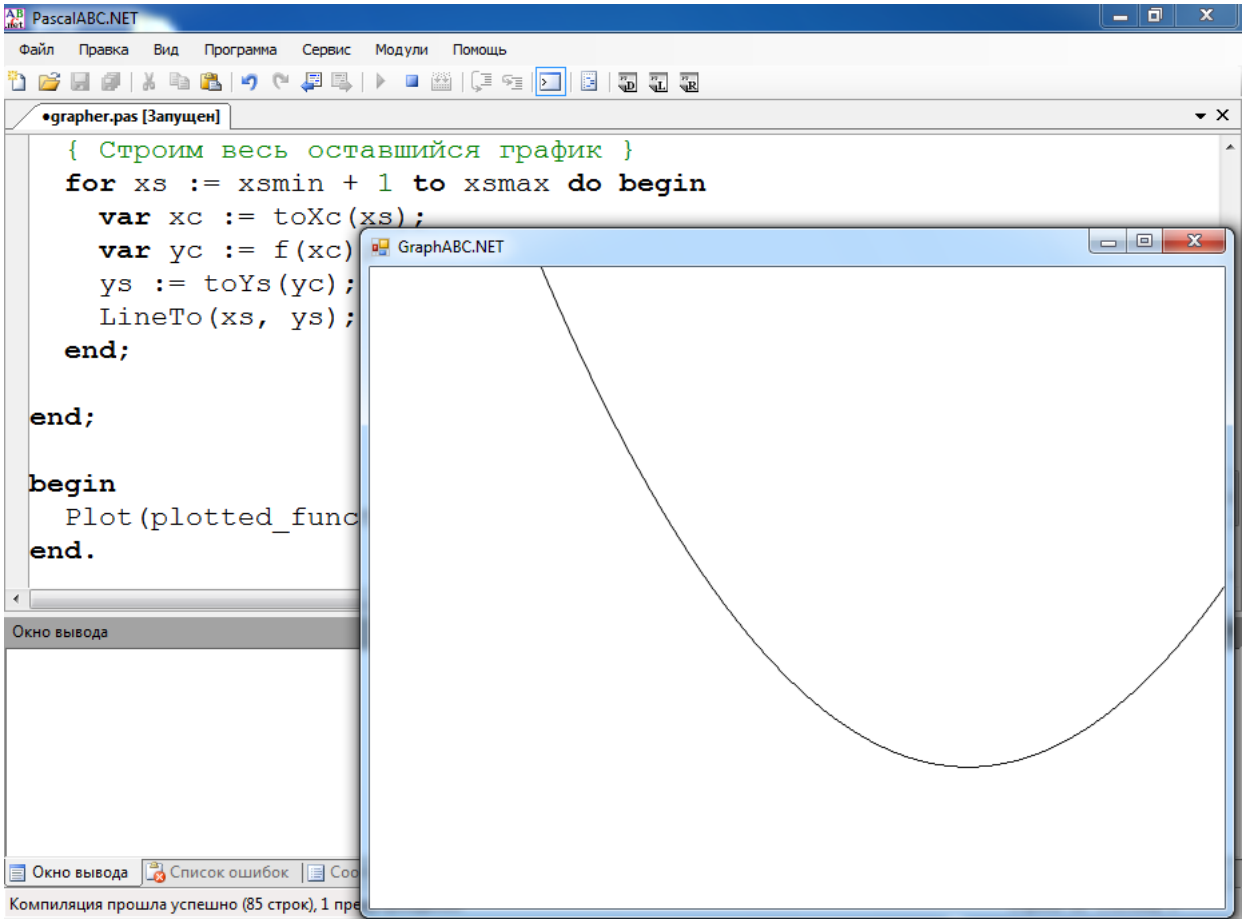

Рис. 7: Пример работы указанной программы

Пример работы данной программы приведен на рисунке 7. Следует отметить, что у приведенной программы есть несколько недостатков:

- На экране не отображаются координатные оси, что затрудняет оценку самого графика;

- На экране также отсутствуют какие-либо метки, позволяющие оценить масштаб графика;
- Процедура написана таким образом, что требует указания границ как по горизонтальной, так и по вертикальной оси; в результате часть графика не отображается на экране, а сам график занимает не всё пространство по вертикали.

Одним из заданий будет доработка (или полное переписывание) данной программы с целью устранения указанных недостатков.

2.3 Массивы

При построении графиков функций, как было отмечено в разделе 1.2, масштаб по вертикальной оси может быть вычислен, если известно минимальное и максимальное значение функции на отрезке. Одним из способов приближенного определения этих значений является вычисление значений функции на отрезке с заданным шагом. Этим шагом может быть, например, расстояние между двумя соседними точками экранной системы координат, спроецированное на декартову систему координат.

Однако вычисление значений некоторых функций может быть весьма вычислительно затратно (например, при построении графиков функций вида $f(x) = \int_0^x g(t)\,dt$). В таком случае разумно сократить количество вызовов функции, и вычислять её для каждой точки лишь один раз. При этом значения этой функции будут требоваться как минимум дважды: при вычислении масштаба и непосредственно при построении графика. Для того, чтобы избежать повторных вычислений, можно использовать **массивы** для хранения значений функции.

Массивами будем называть **непрерывную, типизированную область памяти**. В массиве подряд записаны значения указанного **типа** (отсюта - типизированность массива; типом массива будем называть тип хранимых элементов). За счёт **непрерывности** доступ к каждому элементу осуществляется за постоянное время. Для указания, к какому именно элементу нужно обратиться, используется **индекс** - целое число, задающее порядковый номер элемента в массиве. Индексация в массивах сквозная, то есть между элементами с индексом 3 и 5 обязательно должен быть элемент с индексом 4. Традиционно в языке программирования Паскаль индексация могла начинаться с любого целого числа (обычно - с единицы), однако в PascalABC.Net рекомендуется использовать нумерацию с нуля.

Существуют **статические** и **динамические** массивы. Различие в них состоит в том, что размер **статических** массивов известен заранее и неизменен, в то время как **динамические** массивы способны изменять свой размер, но требуют задания своего размера во время работы программы. В текущей реализации PascalABC.Net динамические массивы являются предпочтительными, поскольку реализация статических массивов использует, на самом деле, динамические массивы, размер которых задаётся непосредственно перед их применением, а индекс просто пересчитывается. Статические массивы в PascalABC.Net работают несколько медленнее динамических, при этом передача их как параметров функций затруднительно.

Объявление массивов происходит следующим образом:

```
1  var
2    { Статический массив: размер указывается при  }
3    { объявлении, тип - после ключевого слова  of }
4    { Индексация: первый элемент будет иметь индекс }
5    { -5, последний - 12. Всего в массиве будет     }
6    { 12 - (-5) + 1 = 18 элементов                  }
7    static_array: array[-5..12] of Real;
8    { Динамический массив: размер не указывается, }
9    { указывается только тип хранимых элементов   }
10   dynamic_array: array of Real;
11   { Допускается указывать в качестве типа хранимых }
12   { элементов массивы; в этом случае говорят о     }
13   { многомерных массивах                           }
14   multidimensional_array: array of array of Integer;
15 begin
16   { Перед использованием динамического массива     }
17   { следует задать его размер с помощью процедуры  }
18   { SetLength                                      }
19   SetLength(dynamic_array, 10);
20   { При использовании многомерных массивов следует }
21   { указывать размер "внешнего" массива, после чего}
```

```
22    { в цикле задавать размеры "внутренних" массивов.}
23    SetLength(multidimensional_array, 5);
24    { В языке PascalABC.Net массивы "знают" свой      }
25    { размер; его можно получить из свойства Length   }
26    { При этом последний элемент будет иметь индекс   }
27    { имя_массива.Length - 1 (поскольку индексация    }
28    { начинается с нуля, а количество элементов равно}
29    { значению свойства Length данного массива)       }
30    for var i := 0 to multidimensional_array.Length do begin
31      { В качестве длины массива можно указывать любое }
32      { неотрицательное целочисленное выражение         }
33      SetLength(multidimensional_array[i], i + 5);
34    end;
35    writeln(multidimensional_array.Length);
36    for var i := 0 to multidimensional_array.Length - 1 do begin
37      writeln(multidimensional_array[i].Length);
38    end;
39  end.
```

Доступ к элементам массива осуществляется с помощью квадратных скобок: dynamic_array[i]. Элемент, полученный с помощью квадратных скобок, возможно использовать как обычную переменную:

```
1  { Запись в массив с клавиатуры }
2  readln(static_array[3]);
3  { Запись в массив присвоением  }
4  static_array[0] := static_array[-3] + static_array[3];
5  { Вывод элемента на экран      }
6  writeln(static_array[0]);
7  { Использование выражения в качестве индекса }
8  var n := 5;
9  for var i := 0 to n - 1 do begin
10   dynamic_array[n - i] := static_array[i] + static_array[i - 1];
11 end;
```

Рассмотрим использование массива в процедуре Plot.

Размер массива примем равным количеству точек на горизонтальной оси экранной системы координат: оно равно $x_s^{max} - x_s^{min} + 1$. В каждом элементе массива будем хранить значение функции в данной точке. При этом следует помнить, что при вычислении очередной точки следует переводить координату по горизонтальной оси из экранной системы координат в декартову.

В коде это будет выглядеть так:

```
1  { Листинг основан на предыдущем, в котором }
2  { была описана процедура Plot. Отметим     }
3  { основные изменения:                      }
4
5  { Процедура построения графика функции f }
6  { Параметры:                             }
7  {   f: функция одной переменной, график  }
8  {      которой требуется построить       }
9  {   xmin, xmax: границы области построения}
10 {      графика функции по горизонтальной  }
11 {      оси                                }
12 { Заметим, что процедура больше не принимает }
13 { параметров ymin, ymax                      }
14 procedure Plot(f: function(x: Real): Real;
15                 xmin, xmax: Real);
16 var
17   { Масштабы по осям }
18   mx, my: Real;
19   { Начало координат ДСК }
20   x0s, y0s: Real;
21
22   { Вспомогательные функции для перевода }
```

```
23    { в различные системы координат описаны}
24    { идентично предыдущему листингу        }
25
26  var
27    { Массив для хранения значений функции }
28    values: array of Real;
29
30  begin
31    { Задаём границы в экранных координатах }
32    var xsmin := 0; var xsmax := window.Width - 1;
33    var ysmin := 0; var ysmax := window.Height - 1;
34
35    { Пересчитываем масштаб }
36    mx := (xsmax - xsmin) / (xmax - xmin);
37    { ВАЖНО: Надо пересчитать X-координату  }
38    { начала координат ДСК как можно раньше!}
39    x0s := xsmin - mx * xmin;
40    { Находим минимум и максимум функции }
41    var ymin, ymax: Real;
42    { Массив values - динамический, устанавливаем }
43    { его длину по указанному выражению           }
44    SetLength(values, xsmax - xsmin + 1);
45    ymin := f(xmin); ymax := f(xmin);
46    for var i := xsmin to xsmax - 1 do begin
47      values[i - xsmin] := f(toXc(i));
48      { Минимум и максимум функции находим }
49      { одновременно с заполнением массива }                                  }
50      if ymin > values[i - xsmin] then
51        ymin := values[i - xsmin];
52      if ymax < values[i - xsmin] then
53        ymax := values[i - xsmin];
54    end;
55
56    { Наконец, масштаб по оси Y находим как прежде }
57    my := (ysmax - ysmin) / (ymax - ymin);
58
59    { Находим начало координат ДСК }
60    y0s := ysmin + my * ymax;
61
62    { Находим координаты первой точки }
63    { Заметим, что теперь мы не используем функцию, }
64    { а пользуемся лишь значениями из массива       }
65    var xs, ys: Integer;
66    xs := toXs(xmin);
67    ys := toYs(values[xs - xsmin]);
68    MoveTo(xs, ys);
69    { Строим весь оставшийся график }
70    for xs := xsmin + 1 to xsmax do begin
71      var yc := values[xs - xsmin];
72      ys := toYs(yc);
73      LineTo(xs, ys);
74    end;
75  end;
```

Результат работы такой программы более привлекателен: график функции будет отмасштабирован таким образом, чтобы занимать всё вертикальное пространство окна. Пример работы программы показан на рисунке 8

2.4 Построение осей и проверка корректности

Заметим, что отрезки осей в декартовой системе координат имеют координаты: $\left(x_c^{min}, 0\right)$ и $(x_c^{max}, 0)$ для горизонтальной оси, $\left(0, y_c^{min}\right)$ и $(0, y_c^{max})$ для вертикальной. Построение осей заключается в

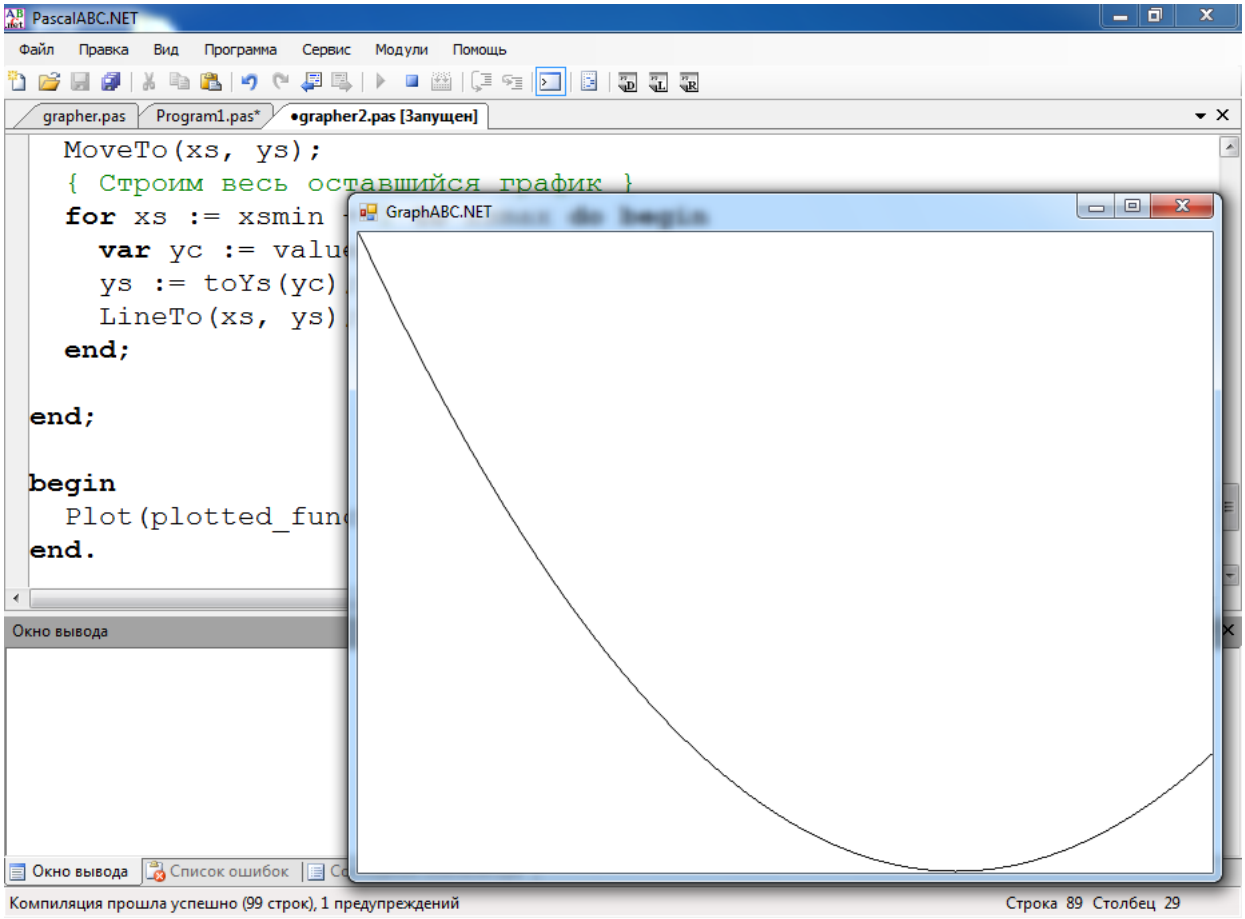

Рис. 8: Пример работы программы с автоматическим масштабированием

проведении таких линий и проецировании их в экранную систему координат.

На концах осей следует построить стрелки направления. Для вертикальной оси можно отступить от верхней точки оси на 15 пикселей вниз и на 5 пикселей влево и вправо. Для горизонтальной оси - от правой точки оси на 15 пикселей влево и на 5 пикселей вверх и вниз.

На оси также следует оставить «масштабный штрих» - штрих в точке, соответствующей координате $(1, 0)$ на горизонтальной оси и $(0, 1)$ на вертикальной. Штрих должен отстоять от оси на 5 пикселей в обе стороны.

Координатные оси следует изображать тонкими (с толщиной линии, равной 1 пикселю), в то время как сам график следует изображать более широкой линией (толщина 2 пикселя). Изменение толщины линии можно производить с помощью изменения свойства Width объекта Pen:

```
1 Pen.Width := 1;
2 { Линии будут отображены с толщиной 1 }
3 Line(0, 0, 200, 200);
4 Line(250, 150, 50, 0);
5 Pen.Width := 2;
6 { Линии будут отображены с толщиной 2 }
7 Line(0, 200, 200, 0);
8 Line(150, 250, 0, 50);
```

Пример работы программы при построении графика функции приведен на рисунке 9.

Для самопроверки можно использовать построенные вручную или в математических пакетах графики функций.

3 Задание

Требуется доработать приведенную процедуру (или написать свою) для вывода на экран осей и масштабных штрихов. С её помощью построить графики функций:

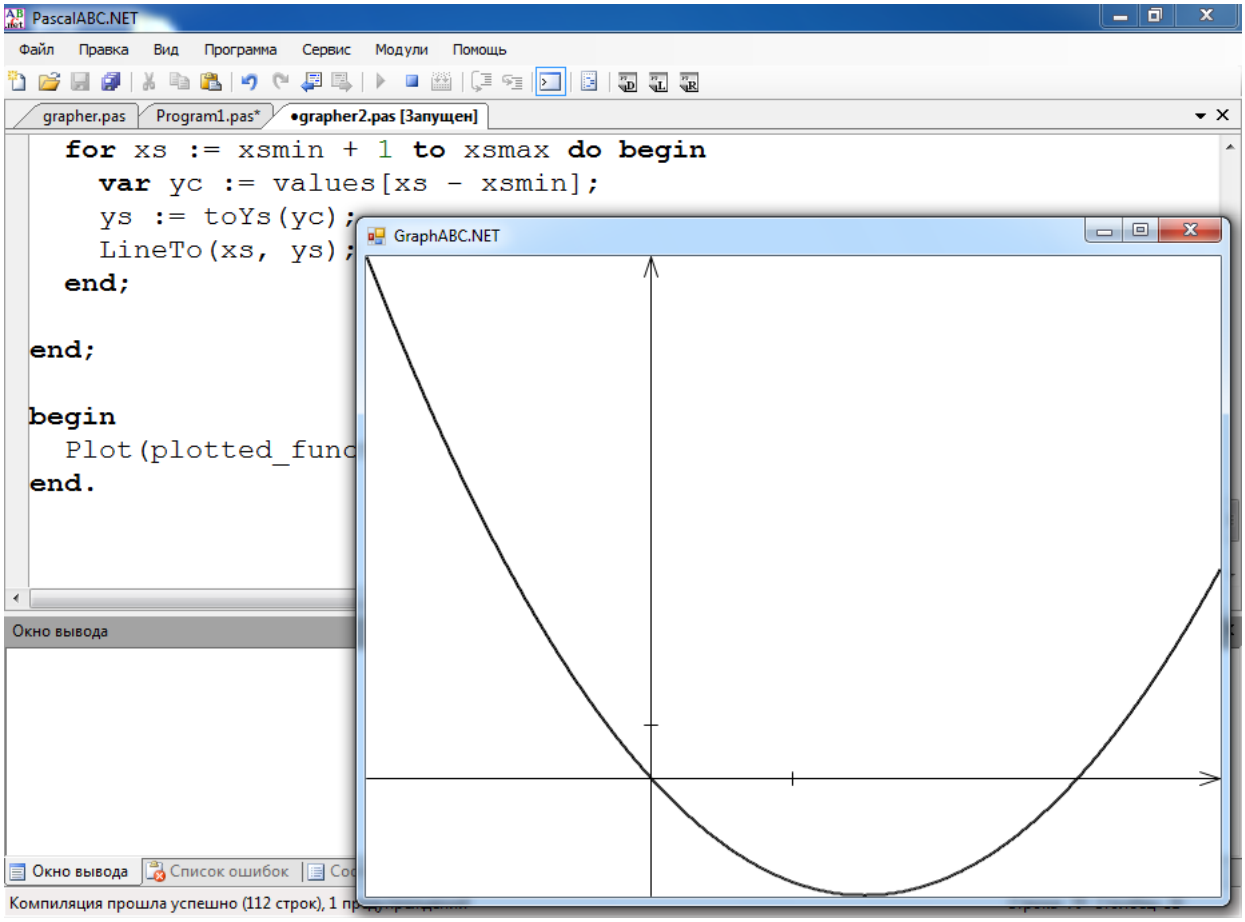

Рис. 9: Пример законченной программы

№	Функция	Границы
1	$y = x^2 - 3x$	$x \in [-3, 4]$
2	$y = \sin\left(x^2\right)$	$x \in [-1, 4]$
3	$y = x + \frac{1}{x}$	$x \in [0.25, 4], y \in [-1, 3]$
4	$y = \int_0^x \frac{tdx}{2\sqrt{9-t^2}}$	$x \in [0, 2], y \in [-1, 4]$

С помощью задания цвета пера (свойство Pen.Color) отобразить на одном графике две функции:

№	Функции	Границы
1	$y_1 = e^{-x^2}, y_2 = \int_{-100}^x e^{-t^2} dt$	$x \in [-3, 3], y \in [-0.25, 4]$
2	$y_1 = \sin x, y_2 = \int_{-2\pi}^x \sin t dt$	$x \in [-1, 7], y \in [-1.5, 1.5]$

Задание цвета производится командами

```
Pen.Color := clRed; { Цвет пера - красный }
Pen.Color := clBlue; { Цвет пера - синий }
Pen.Color := clBlack; { Цвет пера - чёрный }
```