Лабораторная работа №2

Тема: Двумерное векторное изображение

цель: Написать программу вывода на экран векторных рисунков с использованием преобразований переноса, масштабирования и поворота.

ПЛАН

1. Плоская векторное изображение из отрезков.

2. Дополнение вывода рисунка увеличением.

3. Дополнение вывода рисунка вращением.

1. Плоская векторное изображение из отрезков

Для конкретизации изложения материала определимся целью создания изображения собачку, который изображен на следующем рисунке:

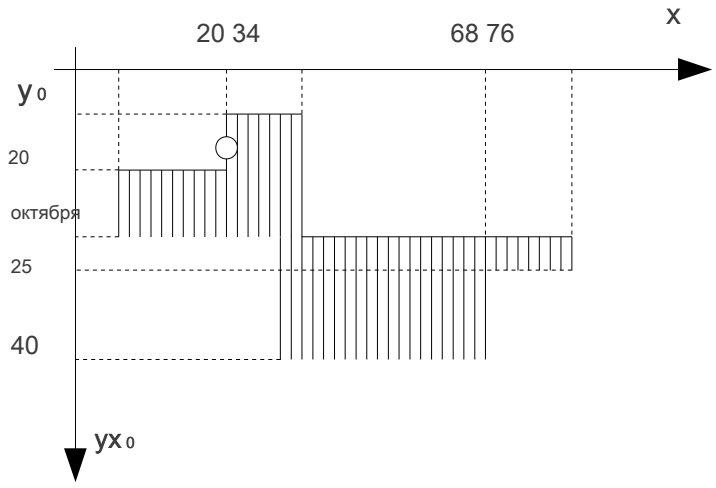

Рис. 1 - Векторное изображение собаки

Код для рисования такого щенка иметь вид:

```
    Image1.Canvas.Pen.Color = clBlack; // Рисуем
мордочку
```

```
for i: = 0 to 7 do begin
    Image1.Canvas.Line (i * 2,10, i * 2,20) end; //
Председатель:

for i: = 0 to 7 do begin
    Image1.Canvas.Line (14 + i * 2,0,14 + i * 2,20) end; // Тулуп

for i: = 0 to 20 do begin
    Image1.Canvas.Line (24 + i * 2,20,24 + i * 2,40) end; // Хвост

for i: = 0 to 8 do begin
    Image1.Canvas.Line (62 + i * 2,20,62 + i * 2,25) end; // Глаз

Image1.Canvas.Brush.Color = clWhite;
Image1.Canvas.Ellipse (12,3,17,8)
```

Однако, нам нужно рисовать собачку в произвольном месте, которое задается начальной точкой (x; y). Для этого достаточно сместить рисунок на заданную величину:

```
// Рисуем мордочку
    Image1.Canvas.Line (x + i * 2, y + 10 x + i * 2, y + 20) // Председатель:

    Image1.Canvas.Line (x + 14 + i * 2, y, x + 14 + i * 2, y + 20) // Тулуп

    Image1.Canvas.Line (x + 24 + i * 2, y + 20 x + 24 + i * 2, y + 40); // Хвост

    Image1.Canvas.Line (x + 62 + i * 2, y + 20 x + 62 + i * 2, y + 25); // Глаз

    Image1.Canvas.Ellipse (x + 12 y + 3, x + 17 y + 8);
```

Теперь код можно оформить отдельной процедурой, которая будет принимать значение смещения (x, y: Integer):

```
procedure TForm1.DrawPesik (x, y: Integer); var i: Integer; begin


// Рисуем щенка
. . .
end;
```

Следующим шагом создадим процедуру, которая будет вызываться при нажатой кнопки и, после очистки зоны рисования белым прямоугольником, можем рисовать собачек. Результат показан на рис. 2.

```
procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) DrawPesik (100,100) DrawPesik
   (150,150) DrawPesik (200,200) DrawPesik (250,100) DrawPesik (300,150) end;
```

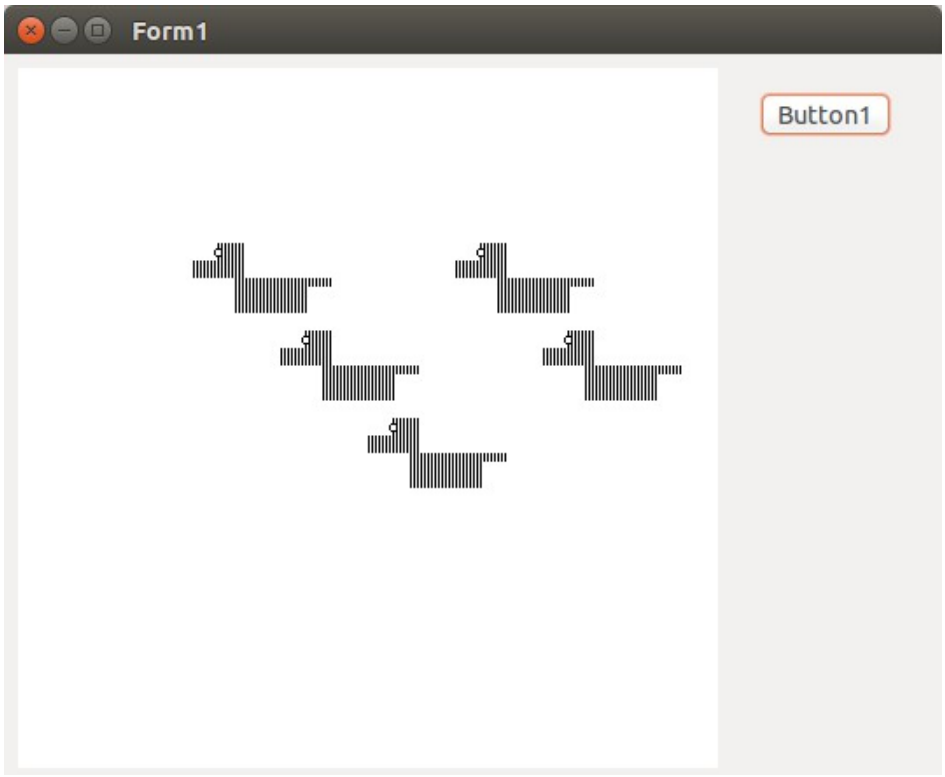

Рис. 2 - Размножение изображения собак

Не совсем удобно использовать рисование линий указанным образом. К координат, определяющих линии нужно добавлять x и y, это делает запись довольно громоздким. В дальнейшем, при добавлении других преобразований, такие записи будут осложнены еще сильнее. Изменим запись рисования собачку и сравним с предыдущими записями:

```
procedure TForm1.DrawPesik; var i: Integer;
begin

  Image1.Canvas.Pen.Color = clBlack;
  for i: = 0 to 7 do begin MLine (i * 2,10, i * 2,20) end; for i: = 0 to 7 do begin MLine (14 + i *
 2,0,14 + i * 2,20) end; for i: = 0 to 20 do begin MLine (24 + i * 2,20,24 + i * 2,40) end; for i: = 0 to
 8 do begin MLine (62 + i * 2,20,62 + i * 2,25) end; Image1.Canvas.Brush.Color = clWhite;
 MEllipse (12,3,17,8) end;
```

Такой код является более привычным и является идентичным рисованию единичного объекта. Смещение собачке сейчас происходит в середине собственной процедуры рисования линии, код которой показан

в следующем листинге:

```
procedure TForm1.MLine (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Line (x1 + Tra.x, y1 + Tra.y,
                                    x2 + Tra.x, y2 + Tra.y)
end;
```

Теперь начальную точку рисования любого объекта можно задать с помощью глобального объекта, описание которого показано ниже:

```
TTransformer = class
    public
        x, y: Integer;
        procedure SetXY (dx, dy: Integer); end;

procedure TTransformer.SetXY (dx, dy: Integer); begin x = dx; y = dy; end;
```

Также меняется и код вывода на экран и изображений собак:

```
procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) Tra.SetXY (100,100) DrawPesik;
   Tra.SetXY (150,150) DrawPesik; Tra.SetXY (200,200) DrawPesik; Tra.SetXY (250,100)
   DrawPesik; Tra.SetXY (300,150) DrawPesik; end;
```

Эти изменения делают код вызова рисования самой собаки несколько более обременительным, код изменения положения оторвано от кода рисования, но это с лихвой компенсировалось в других процедурах и, как увидим дальше, выигрыш будет только увеличиваться с добавлением функциональности. Запустив измененную программу мы не увидим изменений, что означает адекватность изменений координат что одним и другим методом. Окончательный код рисования собак приведены в приложении А.

2. Дополнение вывода рисунка увеличением

Следующим шагом является добавление коэффициентов масштабирования отдельно по каждой из координат. Тогда для рисования нужно перечислять координаты по таким соотношениями:

$x_{new} = x \cdot m_x + d_x$, $y_{new} = y \cdot m_y + d_y$, где m обозначает коэффициент масштабирования, d - смещение рисунке от начала координат. Нужно также учесть особенности языка программирования - PASCAL запрещает до целочисленных переменных записывать действительные числа непосредственно, для этого нужно использовать функцию округления. Такие расчеты удобно оформить отдельными функциями класса преобразования:

```
procedure TTransformer.SetMXY (masX, masY: Double) begin mx = masX; my =
masY; end;

function TTransformer.GetX (oldX: Integer): Integer; begin GetX = Round (mx *
oldX + x) end;

function TTransformer.GetY (oldY: Integer): Integer; begin GetY = Round (my *
oldY + y) end;
```

Согласно изменениям, нужно вывести вычисления координат по рисованию линии и эллипсу:

```
procedure TForm1.MLine (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Line (Tra.GetX (x1), Tra.GetY (y1),
                                    Tra.GetX (x2), Tra.GetY (y2))
end;

procedure TForm1.MEllipse (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Ellipse (Tra.GetX (x1), Tra.GetY (y1),
                                    Tra.GetX (x2), Tra.GetY (y2))
end;
```

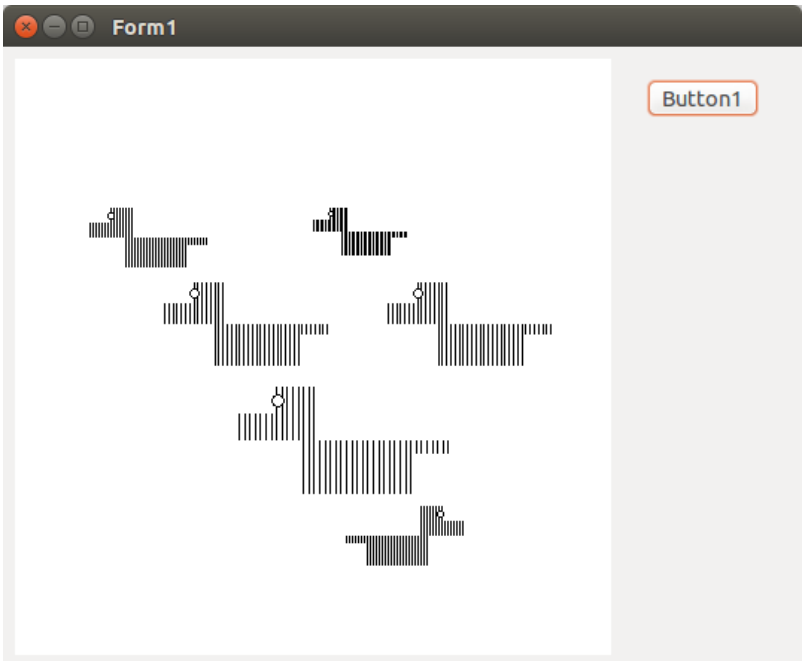

Рис. 3 - Масштабируемые собаки

Благодаря введению дополнительного класса, который отвечает за преобразование координат, можно выполнить рисования собак разного размера:

```
procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) Tra.SetXY (50,100) Tra.SetMXY
    (1.0,1.0) DrawPesik;

    Tra.SetXY (100,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (150,220) Tra.SetMXY (1.8,1.8) DrawPesik;

    Tra.SetXY (200,100) Tra.SetMXY (0.8,0.8) DrawPesik;

    Tra.SetXY (250,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (300,300) Tra.SetMXY (-1.0,1.0) DrawPesik; end;
```

В качестве бонуса добросовестного выполнения математических правил, мы получили возможность

рисовать собак не только разного размера, но и их отражение: для нижней собаки использовано масштабный коэффициент по горизонтали 1, что и привело к ее прорисовке в обратном направлении.

Полный листинг варианта программного кода с увеличением векторной собачки приведены в приложении Б.

3. Дополнение вывода рисунка вращением

Последним шагом в преобразовании двумерных координат является вращение изображения на определенный угол. Для этого нужно найти формулы перехода от одной системы координат к другой при известном углу вращения. Для этого изобразим эти две системы координат, обращены друг относительно друга:

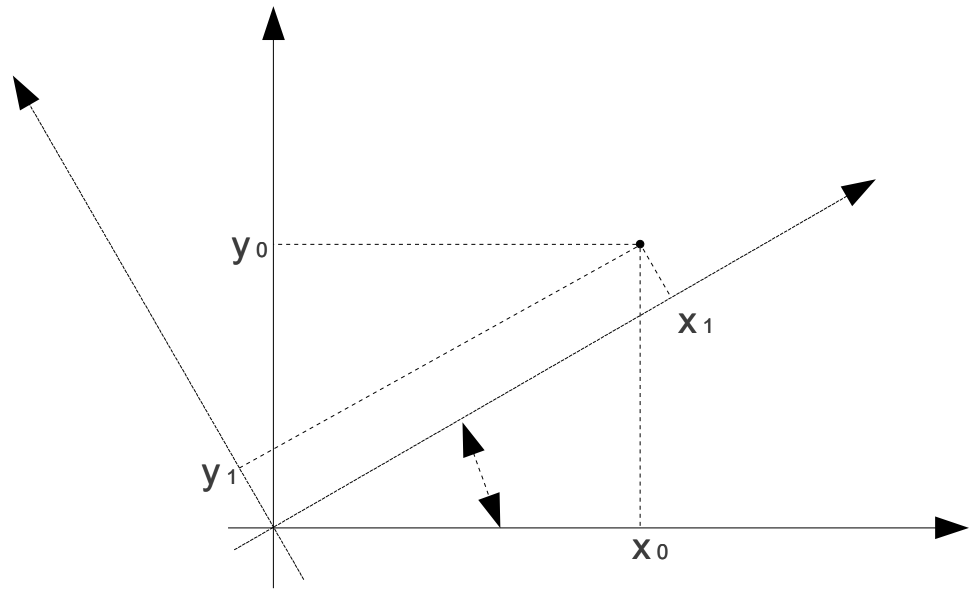

Рис. 4 - поворот координатных осей

Угол поворота обозначены как α = 30 °, при этом на рис. 4 видно, что координаты (x_0; y_0) переходят к координатам (x_1, y_1). Решать задачу поиска формул для нахождения (x_1, y_1) из известных (x_0; y_0) можно несколькими способами. На этот раз можно воспользоваться аналитической геометрией.

Определим уравнение прямой, которая будет отвечать возвращенной оси OX. Мы знаем, что коэффициент наклона прямой $y = kx + c$ имеет зависимость $k = \tan(\alpha)$, или $k = \sin(\alpha) / \cos(\alpha)$.

Так как новая координатная ось проходит через начало координат, свободный коэффициент $c = 0$.

В целом уравнение прямой будет иметь вид $y = x \cdot \sin(\alpha) / \cos(\alpha) \Rightarrow x \cdot \sin(\alpha) - y \cdot \cos(\alpha) = 0$.

тогда координатой y_1 будет расстояние от точки (x_1, y_1) к прямой, совпадающей новой осью:

$$y_1 = x_0 \cdot \sin(\alpha) - y_0 \cdot \cos(\alpha).$$

Аналогично, вертикальная ось будет коэффициент наклона $k = -ctan(\alpha)$, или $k = -\cos(\alpha) / \sin(\alpha)$. Этому коэффициента наклона соответствует уравнение прямой $y = -x \cdot \cos(\alpha) / \sin(\alpha) \Rightarrow x \cdot \cos(\alpha) + y \cdot \sin(\alpha) = 0$. тогда координатой x_1 будет расстояние от точки $(x_0; y_0)$ к прямой, совпадающей новой вертикальной осью:

$$x_1 = x_0 \cdot \cos(\alpha) + y_0 \cdot \sin(\alpha).$$

Окончательно, для нахождения координат точки $(x_0; y_0)$ в системе координат, повернута на угол α (x_1, y_1), нужно воспользоваться выражениями:

$$\begin{cases} x_1 = x_0 \cdot \cos(\alpha) + y_0 \cdot \sin(\alpha), \\ y_1 = x_0 \cdot \sin(\alpha) - y_0 \cdot \cos(\alpha). \end{cases}$$

Комбинация поворота с перемещением на вектор $(dx; dy)$ дает полную формулу перемещение точки с ее поворотом:

$$\begin{cases} x_1 = x_0 \cdot \cos(\alpha) + y_0 \cdot \sin(\alpha) + dx, \\ y_1 = x_0 \cdot \sin(\alpha) - y_0 \cdot \cos(\alpha) + dy. \end{cases}$$

Для учета вращения в программном коде нужно необходимо модифицировать трансформер добавлением угла поворота, функции получения новых координат должны принимать уже не одну координату а две:

```
TTransformer = class
    public
        x, y: Integer;
        mx, my: Double; alpha:
        Double;
        procedure SetXY (dx, dy: Integer); procedure SetMXY (masX,
        masY: Double) procedure SetAlpha (a: Double)

        function GetX (oldX, oldY: Integer): Integer; function GetY (oldX, oldY:
        Integer): Integer;
```

```
    end;
function TTransformer.GetX (oldX, oldY: Integer): Integer; begin

    GetX = Round (mx * oldX * cos (alpha) - my * oldY * sin (alpha) + x) end;

function TTransformer.GetY (oldX, oldY: Integer): Integer; begin

    GetY = Round (mx * oldX * sin (alpha) + my * oldY * cos (alpha) + y) end;
```

Соответственно и рисования происходит несколько иначе:

```
procedure TForm1.MLine (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Line (Tra.GetX (x1, y1),
                                        Tra.GetY (x1, y1), Tra.GetX
                                        (x2, y2), Tra.GetY (x1, y2))

end;
procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) Tra.SetAlpha (0.0)

    Tra.SetXY (50,100) Tra.SetMXY (1.0,1.0) DrawPesik;

    Tra.SetXY (100,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (150,220) Tra.SetMXY (1.8,1.8) DrawPesik;

    Tra.SetXY (200,100) Tra.SetMXY (0.8,0.8) DrawPesik;

    Tra.SetAlpha (PI / 4.0); // Здесь изменено угол Tra.SetXY
    (250,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (300,350) Tra.SetMXY (-1.0,1.0)
```

```
DrawPesik; end;
```

Теперь собака может менять не только свой размер и направление, но и вращаться на произвольный угол. Результат рисования можно увидеть на следующем малышу:

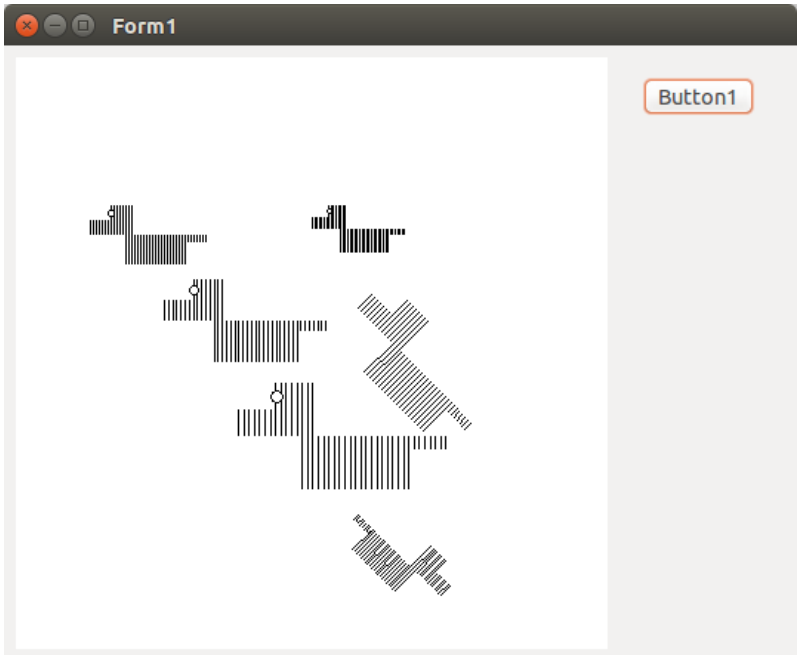

Рис. 5 - комбинация перемещения масштабирования и вращения

Приложение А. Собачки.

```
unit Unit1;
{$ Mode objfpc} {$ H +} interface
uses

    Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, ExtCtrls, StdCtrls;



type
  {TForm1}
  TTransformer = class
    public
      x, y: Integer;
      procedure SetXY (dx, dy: Integer); end;



  TForm1 = class (TForm)
    Button1: TButton; Image1:
    TImage;
    procedure Button1Click (Sender: TObject); procedure FormCreate
   (Sender: TObject); procedure FormDestroy (Sender: TObject);
   private

    {Private declarations} public

    {Public declarations} Tra:
    TTransformer;
    procedure MLine (x1, y1, x2, y2: Integer); procedure MEllipse (x1,
  y1, x2, y2: Integer); procedure DrawPesik; end;



var
```

```
    Form1: TForm1;

implementation

{$ R * .lfm}
{TForm1}
procedure TForm1.DrawPesik; var i: Integer;
begin

    Image1.Canvas.Pen.Color = clBlack;
    for i: = 0 to 7 do begin Mline (i * 2,10, i * 2,20) end; for i: = 0 to 7 do begin MLine (14 + i * 2,0,14
    + i * 2,20) end; for i: = 0 to 20 do begin MLine (24 + i * 2,20,24 + i * 2,40) end; for i: = 0 to 8 do
    begin MLine (62 + i * 2,20,62 + i * 2,25) end; Image1.Canvas.Brush.Color = clWhite; MEllipse
    (12,3,17,8) end;



procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) Tra.SetXY (100,100) DrawPesik;
    Tra.SetXY (150,150) DrawPesik; Tra.SetXY (200,200) DrawPesik; Tra.SetXY (250,100)
    DrawPesik; Tra.SetXY (300,150) DrawPesik; end;



procedure TForm1.FormCreate (Sender: TObject); begin Tra =
TTransformer.Create; end;

procedure TForm1.FormDestroy (Sender: TObject); begin Tra.Destroy; end;
```

```
procedure TForm1.MLine (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Line (x1 + Tra.x,
                                        y1 + Tra.y, x2 +
                                        Tra.x, y2 + Tra.y)

end;

procedure TForm1.MEllipse (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Ellipse (x1 + Tra.x,
                                          y1 + Tra.y, x2 +
                                          Tra.x, y2 + Tra.y)

end;

procedure TTransformer.SetXY (dx, dy: Integer); begin x = dx; y = dy; end;


end.
```

Приложение Б. Масштабируемые собачки.

```
unit Unit1;

{$ Mode objfpc} {$ H +}

interface

uses
        Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, ExtCtrls,

    StdCtrls;

type

    {TForm1}

    TTransformer = class
        public
            x, y: Integer;
            mx, my: Double;
            procedure SetXY (dx, dy: Integer); procedure SetMXY (masX,
          masY: Double) function GetX (oldX: Integer): Integer; function
          GetY (oldY: Integer): Integer; end;



    TForm1 = class (TForm)
        Button1: TButton; Image1:
        TImage;
        procedure Button1Click (Sender: TObject); procedure FormCreate
      (Sender: TObject); procedure FormDestroy (Sender: TObject);
      private
```

```
        {Private declarations} public

        {Public declarations} Tra:
        TTransformer;
        procedure MLine (x1, y1, x2, y2: Integer); procedure MEllipse (x1,
      y1, x2, y2: Integer); procedure DrawPesik; end;



var
    Form1: TForm1;

implementation

{$ R * .lfm}

{TForm1}

procedure TForm1.DrawPesik; var i: Integer;
begin

    Image1.Canvas.Pen.Color = clBlack; for i: = 0 to 7 do
    begin
        MLine (i * 2,10, i * 2,20) end;

    for i: = 0 to 7 do begin
        MLine (14 + i * 2,0,14 + i * 2,20) end;

    for i: = 0 to 20 do begin
        MLine (24 + i * 2,20,24 + i * 2,40) end;

    for i: = 0 to 8 do begin
        MLine (62 + i * 2,20,62 + i * 2,25) end;
```

```
    Image1.Canvas.Brush.Color = clWhite; MEllipse (12,3,17,8)
  end;


procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) Tra.SetXY (50,100) Tra.SetMXY
    (1.0,1.0) DrawPesik;

    Tra.SetXY (100,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (150,220) Tra.SetMXY (1.8,1.8) DrawPesik;

    Tra.SetXY (200,100) Tra.SetMXY (0.8,0.8) DrawPesik;

    Tra.SetXY (250,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (300,300) Tra.SetMXY (-1.0,1.0) DrawPesik; end;



procedure TForm1.FormCreate (Sender: TObject); begin

    Tra = TTransformer.Create; end;


procedure TForm1.FormDestroy (Sender: TObject); begin

    Tra.Destroy; end;


procedure TForm1.MLine (x1, y1, x2, y2: Integer); begin
```

```
    Image1.Canvas.Line (Tra.GetX (x1),
                                    Tra.GetY (y1), Tra.GetX
                                    (x2), Tra.GetY (y2))

end;

procedure TForm1.MEllipse (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Ellipse (Tra.GetX (x1),
                                    Tra.GetY (y1), Tra.GetX
                                    (x2), Tra.GetY (y2))

end;

{TTransformer}

procedure TTransformer.SetXY (dx, dy: Integer); begin

    x = dx; y = dy; end;


procedure TTransformer.SetMXY (masX, masY: Double) begin

    mx = masX; my = masY; end;


function TTransformer.GetX (oldX: Integer): Integer; begin

    GetX = Round (mx * oldX + x) end;


function TTransformer.GetY (oldY: Integer): Integer; begin

    GetY = Round (my * oldY + y)
```

```
end;


end.
```

Приложение В. Собаки с поворотом

```
unit Unit1;

{$ Mode objfpc} {$ H +}

interface

uses
        Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, ExtCtrls,

    StdCtrls;

type

    {TForm1}

    TTransformer = class
        public
            x, y: Integer;
            mx, my: Double; alpha:
            Double;
            procedure SetXY (dx, dy: Integer); procedure SetMXY (masX,
            masY: Double) procedure SetAlpha (a: Double)

            function GetX (oldX, oldY: Integer): Integer; function GetY (oldX, oldY:
        Integer): Integer; end;



    TForm1 = class (TForm)
        Button1: TButton; Image1:
        TImage;
        procedure Button1Click (Sender: TObject); procedure FormCreate
        (Sender: TObject);
```

```
        procedure FormDestroy (Sender: TObject); private

        {Private declarations} public

        {Public declarations} Tra:
        TTransformer;
        procedure MLine (x1, y1, x2, y2: Integer); procedure MEllipse (x1,
      y1, x2, y2: Integer); procedure DrawPesik; end;



var
    Form1: TForm1;

implementation

{$ R * .lfm}

{TForm1}

procedure TForm1.DrawPesik; var i: Integer;
begin

    Image1.Canvas.Pen.Color = clBlack; for i: = 0 to 7 do
    begin
        MLine (i * 2,10, i * 2,20) end;

    for i: = 0 to 7 do begin
        MLine (14 + i * 2,0,14 + i * 2,20) end;

    for i: = 0 to 20 do begin
        MLine (24 + i * 2,20,24 + i * 2,40) end;

    for i: = 0 to 8 do begin
```

```
        MLine (62 + i * 2,20,62 + i * 2,25) end;

    Image1.Canvas.Brush.Color = clWhite; MEllipse (12,3,17,8)
  end;


procedure TForm1.Button1Click (Sender: TObject); begin

    Image1.Canvas.Brush.Color = clWhite;
    Image1.Canvas.FillRect (0,0, Image1.Width, Image1.Height) Tra.SetAlpha (0.0)

    Tra.SetXY (50,100) Tra.SetMXY (1.0,1.0) DrawPesik;

    Tra.SetXY (100,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (150,220) Tra.SetMXY (1.8,1.8) DrawPesik;

    Tra.SetXY (200,100) Tra.SetMXY (0.8,0.8) DrawPesik;

    Tra.SetAlpha (PI / 4.0); // Здесь изменено угол Tra.SetXY
    (250,150) Tra.SetMXY (1.4,1.4) DrawPesik;

    Tra.SetXY (300,350) Tra.SetMXY (-1.0,1.0) DrawPesik; end;


procedure TForm1.FormCreate (Sender: TObject); begin

    Tra = TTransformer.Create; end;


procedure TForm1.FormDestroy (Sender: TObject); begin

    Tra.Destroy;
```

```
end;

procedure TForm1.MLine (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Line (Tra.GetX (x1, y1),
                                        Tra.GetY (x1, y1), Tra.GetX
                                        (x2, y2), Tra.GetY (x1, y2))

end;

procedure TForm1.MEllipse (x1, y1, x2, y2: Integer); begin

    Image1.Canvas.Ellipse (Tra.GetX (x1, y1),
                                        Tra.GetY (x1, y1), Tra.GetX
                                        (x2, y2), Tra.GetY (x2, y2))

end;

{TTransformer}
procedure TTransformer.SetXY (dx, dy: Integer); begin x = dx; y = dy; end;


procedure TTransformer.SetMXY (masX, masY: Double) begin mx = masX; my =
masY; end;

function TTransformer.GetX (oldX, oldY: Integer): Integer; begin

    GetX = Round (mx * oldX * cos (alpha) - my * oldY * sin (alpha) + x) end;


function TTransformer.GetY (oldX, oldY: Integer): Integer; begin

    GetY = Round (mx * oldX * sin (alpha) + my * oldY * cos (alpha) + y) end;
```

```
procedure TTransformer.SetAlpha (a: Double) begin

    alpha = a; end;



end.
```