Использование циклов при построении изображения

Построение повторяющихся элементов изображения имеет смысл включать в операторы цикла. Операторы цикла условно можно разделить на циклы по условию и циклы по количеству повторений (циклы-счетчики).

Когда точно известно количество повторяющихся элементов изображения удобно использовать цикл **for**.

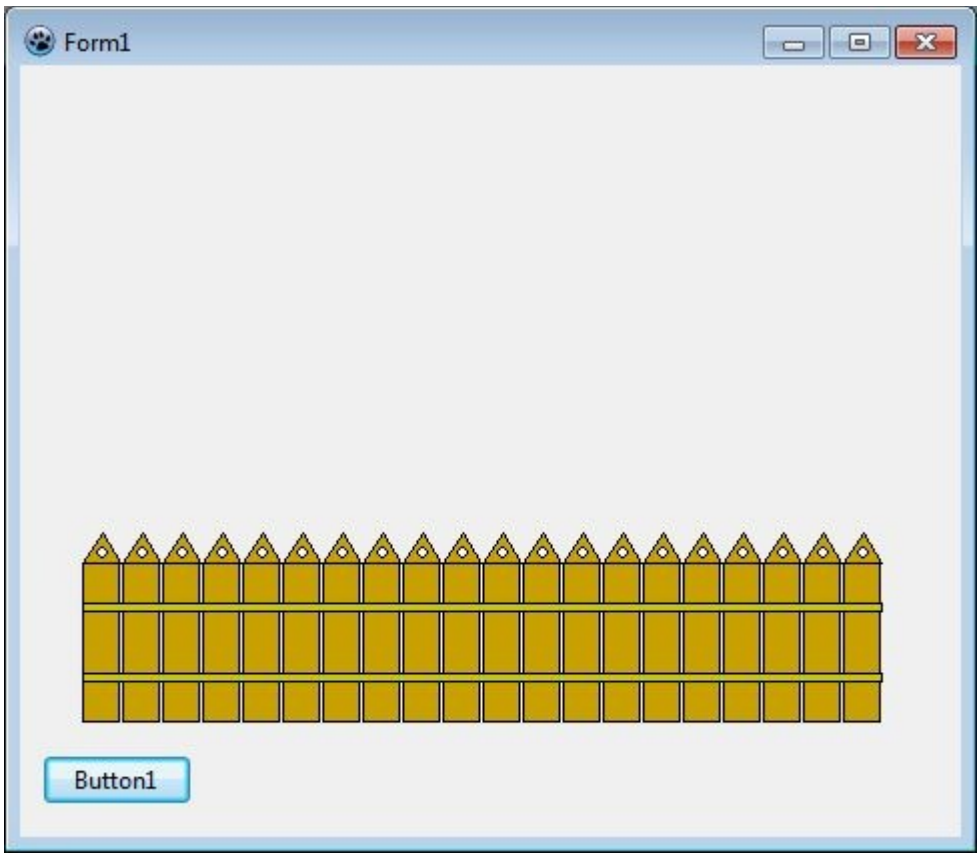
```
for счетчик:=значение to конечное_значение do
     тело_цикла;

for счетчик:=значение downto конечное_значение do
     тело_цикла;
```

При переходе к обработке оператора цикла **for** управляющей переменной *счетчик* присваивается заданное начальное значение. Затем в цикле выполняется исполнительный оператор (или составной оператор begin..end). Каждый раз при выполнении исполнительного оператора управляющая переменная увеличивается на 1 (для for...to) или уменьшается на 1 (для for...downto). Цикл завершается при достижении управляющей переменной своего конечного значения.

Пример использования оператора цикла **for** при построении забора из 20 элементов.

```
procedure TForm1.Button1Click(Sender: TObject);
Var maxX, maxY : Integer;
    i : Integer;
begin
  maxX := PaintBox1.Width;
  maxY := PaintBox1.Height;
  // Количество повторений
  for i := 1 to 20 do  begin
    // Определение цвета досок забора
    PaintBox1.Canvas.Brush.Color := RGBToColor(200,160,0);
    // Отрисовка досок забора
    PaintBox1.Canvas.Rectangle(i*20, maxY, 19 + i*20, maxY-80);
    // Верхняя шапка забора
    PaintBox1.Canvas.Polygon([Point(i*20, maxY-80),
                              Point(10 + i*20, maxY-95), Point(19 + i*20, maxY-80)]);
    // Круглые отверстия в шапке
    PaintBox1.Canvas.Brush.Color := clWhite;
    PaintBox1.Canvas.Ellipse(7 + i*20, maxY-82, 13 + i*20, maxY-88);
  end;
  // Поперечные перекладины
  PaintBox1.Canvas.Brush.Color := RGBToColor(200,200,0);
  PaintBox1.Canvas.Rectangle(0, maxY-20, 620, maxY-25);
  PaintBox1.Canvas.Rectangle(0, maxY-55, 620, maxY-60);
end;
```



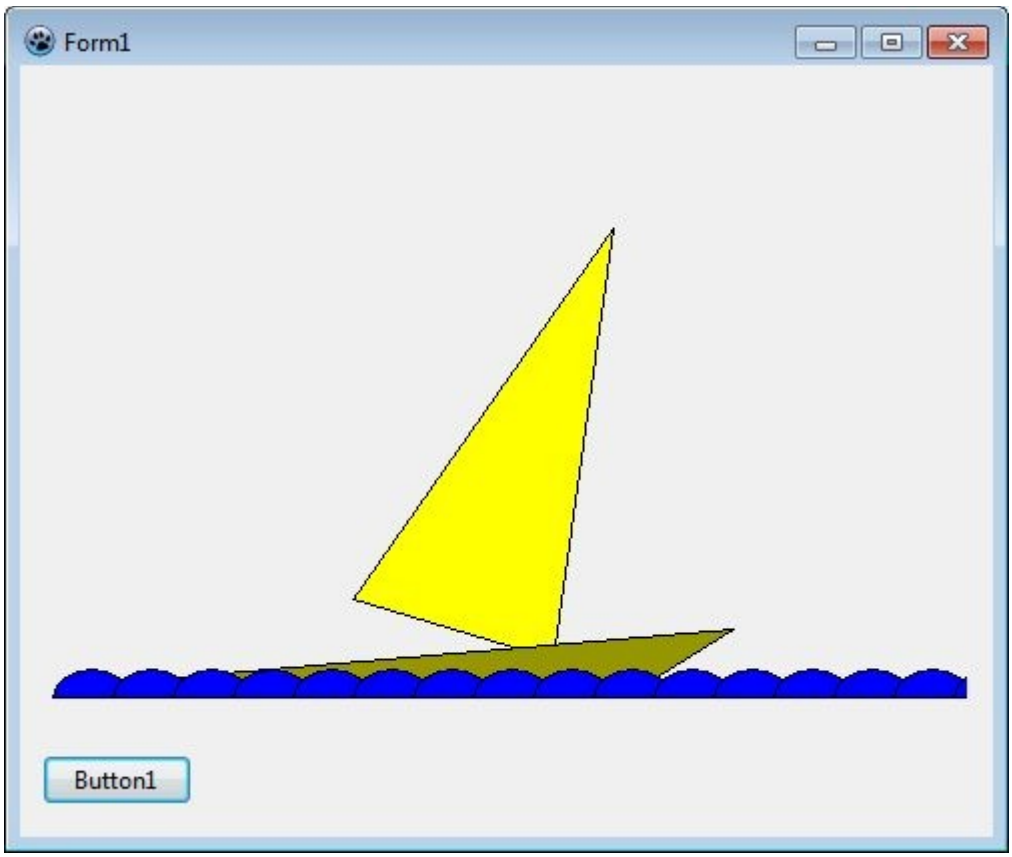
Если количество элементов не известно, то можно воспользоваться циклом по условию. Например, строить элементы изображения пока не будет достигнут край экрана (или поля для рисования). Различают циклы с предусловием и с постусловием. Цикл с предусловием

```
while expression do statement;
```

При выполнении этого оператора вначале вычисляется значение логического выражения expression. Если это значение истинно, выполняется оператор statement, затем значение выражения проверяется вновь и т. д., до тех пор, пока выражение не примет значение «ложь». Если выражение принимает значение «ложь» при первой же проверке, то оператор statement не выполняется вообще.

Пример использования оператора цикла **while** при построении морских волн.

```
procedure TForm1.Button1Click(Sender: TObject);
Var maxX, maxY : Integer;
    x : Integer;
begin
  maxX := PaintBox1.Width;
  maxY := PaintBox1.Height;
  // Построение паруса
  PaintBox1.Canvas.Brush.Color := clYellow;
  PaintBox1.Canvas.Polygon([Point(250,maxY-35), Point(280, maxY-250), Point(150, maxY-65)]);
  // Построение корпуса лодки
  PaintBox1.Canvas.Brush.Color := RGBToColor(150, 150, 0);
  PaintBox1.Canvas.Polygon([Point(50,maxY-25),
                            Point(300,maxY-25), Point(340,maxY-50)]);
  // Построение волн в виде полуокружностей.
  PaintBox1.Canvas.Brush.Color := clBlue;
  x:=0;
  while x < maxX do begin
    PaintBox1.Canvas.Pie(0 + x, maxY, 40 + x, maxY-30,
                         40 + x, maxY-15, 0 + x, maxY-15 );
    x := x+30;
  end;
end;
```



Процедуры и функции при построении изображений

Понятие процедуры, функции и метода класса

Важной составной частью программирования в Object Pascal является использование подпрограмм - специальным образом оформленных и логически законченных блоков инструкций. Подпрограмму можно вызывать любое число раз из других мест программы, или из других подпрограмм. Таким образом, использование подпрограмм позволяет сделать исходный код более стройным и наглядным.

Подпрограммы в Pascal делят на процедуры и функции. Процедуры - это такие подпрограммы, которые выполняют предназначенное действие и возвращают выполнение в точку вызова. Функции в целом аналогичны процедурам, за тем исключением, что они еще и возвращают результат своего выполнения.

Процедуры и функции, объявленные внутри класса называют методами класса.

Процедуры и функции могут иметь свой собственный набор переменных, объявленных внутри нее и называемых локальными переменными. По завершению работы значения таких переменных теряются.

```
procedure Stars (N : Integer); // N - параметр, определяющий количество звезд
Var
   i : Integer; // Локальная переменная, используемая в операторе цикла
begin
 for i:=1 to N do
  Form1.PaintBox1.Canvas.Pixels[Random(PaintBox1.Width),
Random(PaintBox1.Height)]:=clRed;
end;
```

Обмен значениями между различными процедурами возможен через параметры, глобальные

переменные и возвращаемые в функциях значения.

Переменные объявленные в разделе **Var** модуля являются глобальными для всех процедур и классов модуля и доступны в любой процедуре, функции и методе класса данного модуля. Однако, работа с такими переменными напрямую внутри процедур считается "не красивым" и усложняет отлов ошибок. Поэтому удобно использовать передачу параметров и возвращаемые значения.

Передача параметров

Pascal позволяет передавать параметры в функции и процедуры либо по значению, либо по ссылке. Передаваемый параметр может иметь любой встроенный или пользовательский тип либо являться открытым массивом. Параметр также может быть константой, если его значение в процедуре или функции не меняется.

Передача параметров по значению

Этот режим передачи параметров применяется по умолчанию. Если параметр передается по значению, создается локальная копия данной переменной, которая и предоставляется для обработки в процедуру или функцию. Посмотрите на следующий пример:

```
procedure Test(s: string);
```

При вызове указанной процедуры будет создана копия передаваемой ей в качестве параметра строки s, с которой и будет работать процедура Test. При этом все внесенные в строку изменения никак не отразятся на исходной переменной s.

Этот способ передачи является у большинства самым излюбленным, но в тоже время является и самым не практичным, т.к. для выполнения метода выделяется дополнительная память для создания точной копией передаваемой переменой. Для решения этой проблемы следует использовать один из способов описанных ниже.

Примечание: в случае если в процедуру передается переменная объекта, то в данном случае произойдет передача по ссылке (даже если это не указано явно).

Передача параметров по ссылке

Pascal позволяет также передавать параметры в функции или процедуры по ссылке — такие параметры называются параметрами-переменными. Передача параметра по ссылке означает, что функция или процедура сможет изменить полученные значения параметров. Для передачи параметров по ссылке используется ключевое слово **var**, помещаемое в список параметров вызываемой процедуры или функции.

```
procedure ChangeMe(var x: longint);
 begin
 x := 2; // Параметр x изменен вызванной процедурой
 end;
```

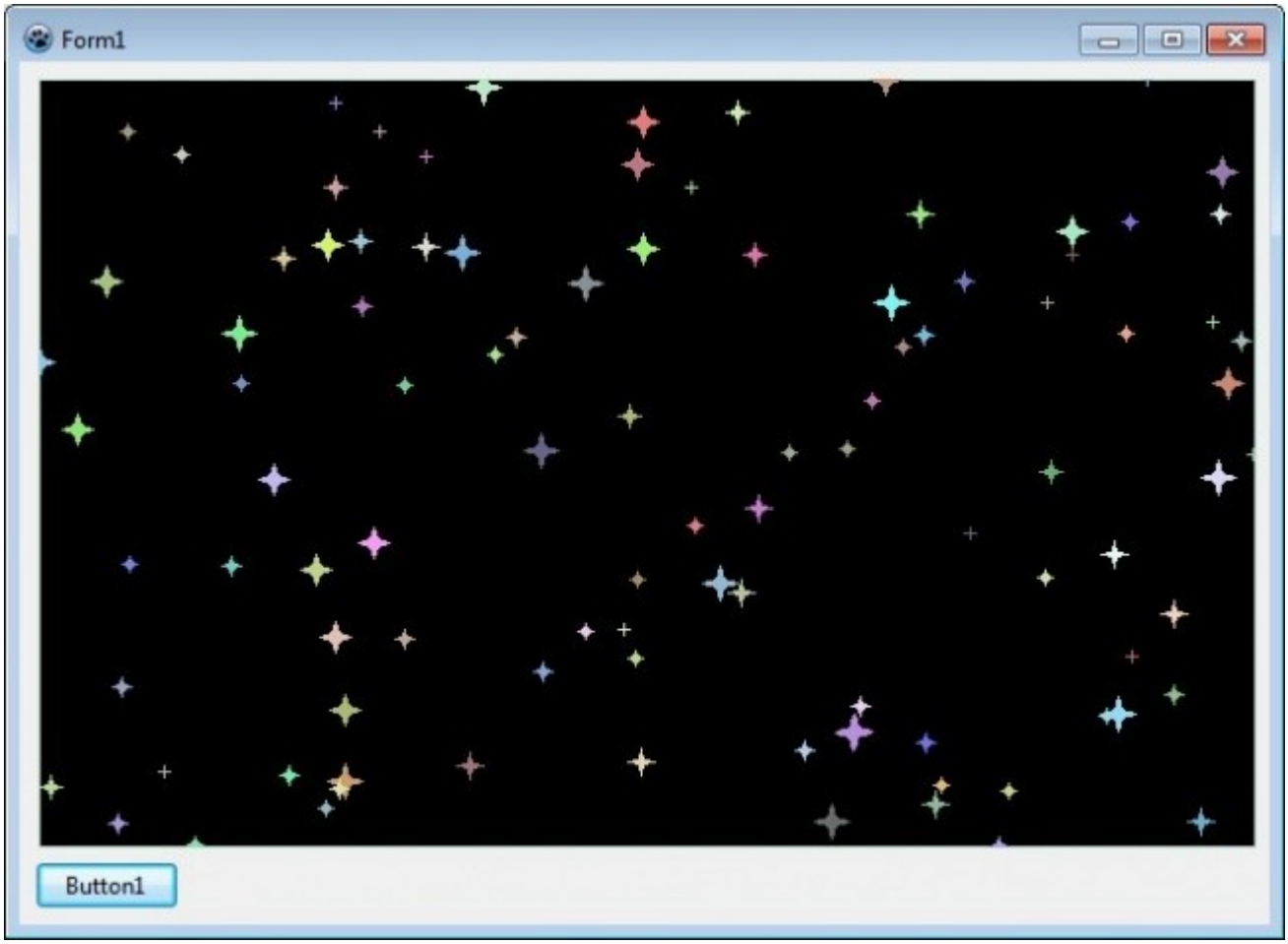
Вместо создания копии переменной **x**, ключевое слово **var** требует передачи адреса самой переменной **x**, что позволяет процедуре непосредственно изменять ее значение.

Передача параметров констант

Если нет необходимости изменять передаваемые функции или процедуре данные, можно описать параметр как константу. Ключевое слово const не только защищает параметр от изменения, но и позволяет компилятору сгенерировать более оптимальный код передачи строк и записей. Вот пример объявления параметра-константы:

```
procedure Test(const s: string );
```

Пример использования процедуры Star для построения звездного неба



Процедура **Star** объявлена внутри класса **TForm1** и по сути является методом этого класса, позволяющим рисовать звезды на форме. При создании процедуры удобно использовать горячие клавиши (Ctrl-Shift+C), которые автоматически создадут тело процедуры внутри блока **implementation**. Для этого необходимо прописать заголовок процедуры в разделе **public** или **private** и нажать горячие клавиши Ctrl-Shift+C.

В процедуру передаются параметры x,y - координаты расположения звезды, Size - размер и Color - цвет отрисовки. Используя цикл задается 100 звезд разного цвета и размера.

```
unit Unit1;

interface

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

type
```

```
  { TForm1 }

  TForm1 = class(TForm)
    Button1: TButton;
    PaintBox1: TPaintBox;
    procedure Button1Click(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
    // Рисование звезды
    procedure Star(x,y: Integer; Size: Integer; Colour: TColor);
    // Получение случайного цвета. A - параметр, определяющий яркость
случайного цвета
    function GetRandomColor( A: Integer): TColor;
  end;

var
  Form1: TForm1;

implementation

{$R *.lfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);
// Локальные переменные i и c используются для задания количества и цвета звезд
Var i : Integer;
    c : TColor;
begin
  // Закрашиваем поле для рисования в черный цвет
  PaintBox1.Canvas.Brush.Color:=clBlack;
  PaintBox1.Canvas.Rectangle(0,0,PaintBox1.Width, PaintBox1.Height);
  // Вызываем процедуру Star 100 раз, тем самым рисуя 100 звезд
  for i := 1 to 100 do begin
    // Случайным образом формируем цвет звезды, с помощью функции
    c := GetRandomColor(100);
    // Вызываем процедуру отрисовки звезды с разными случайными параметрами
    Star( Random(PaintBox1.Width), Random(PaintBox1.Height), Random(7)+3, c);
  end;
end;

//Процедура прорисовки звезды с разными параметрами расположения, размера и
цвета
procedure TForm1.Star(x, y: Integer; Size: Integer; Colour: TColor);
begin
  // Задаем цвет в соответствии с полученным параметром
  PaintBox1.Canvas.Pen.Color:= Colour;
  PaintBox1.Canvas.Brush.Color:= Colour;
  // Рисуем звезду, используя переданные координаты и размер
  Paintbox1.Canvas.Polygon( [Point(x, y-size),
  Point(x-size div 4, y-size div 4), Point(x-size, y),
  Point(x-size div 4, y+size div 4), Point(x, y+size),
  Point(x+size div 4, y+size div 4), Point(x+size, y),
  Point(x+size div 4, y-size div 4), Point(x, y-size)]);
end;

function TForm1.GetRandomColor( A: Integer): TColor;
```

```
begin
 // Проверка на корректность задания параметра A (не больше 255)
 if A > 255 then A := 255;
 // Получение случайного цвета, в зависимости от значения A
 Result := RGBToColor(Random(256-A)+A, Random(256-A)+A, Random(256-A)+A);
end;

end.
```