Грацианова Татьяна Юрьевна
Преподаватель подготовительных курсов факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова.

Рекурсивные узоры

Мы посетили уже несколько занятий кружка по информатике (см. «Потенциал» №4 за 2013 г. и №4, №5 за 2014 г.), на которых ребята обсуждали, что такое рекурсия, и учились писать реурсивные процедуры и функции. Учитель рассказал им, что с помощью рекурсии можно не только вычислять значения функций и решать задачи ЕГЭ, но и рисовать. И вот сегодня они будут писать программы, которые рисуют на экране узоры при помощи рекурсивных алгоритмов. Берите компьютер и присоединяйтесь к ним – все рисунки, приведённые в статье, – «реальные», нарисованы программами. У вас получатся такие же.

– Давайте будем рисовать наши узоры при помощи PascalABC.NET. Изучение графики не является обязательным, поэтому сначала посмотрим, как писать программу, которая будет строить простейшие изображения. Вот эта программа должна в центре экрана нарисовать квадрат со стороной 400 и вписанный в него круг (центр круга в центре экрана, а радиус – 200). Проверьте, правильно ли у вас всё работает, и мы на примере этой программы рассмотрим простейшие графические процедуры PascalABC (в других Паскаль-компиляторах и в языке СИ тоже есть похожие процедуры, многие ничем не отличаются от нижеприведённых).

```
Program PROVERKA;
uses GraphABC;
Procedure Ug4(x1, y1, x2, y2, x3, y3, x4, y4:Integer);
Begin
 Line(x1, y1, x2, y2);
 Line(x2, y2, x3, y3);
 Line(x3, y3, x4, y4);
 Line(x4, y4, x1, y1)
End;
var x0, y0, x1,y1,x2, y2, x3, y3, x4, y4, L:integer;
begin
x0:=WindowWidth div 2;
y0:=WindowHeight div 2;
```

```
L:=200;
X1:=X0-L;Y1:=Y0+L;
X2:=x1; Y2:=Y0-L;
x3:=X0+L; Y3:=Y2;
X4:=X3; Y4:=Y1;
Ug4(x1,y1,x2,y2,x3,y3,x4,y4);
circle(x0,y0,L)
end.
```

В начале программы написаны «волшебные» слова **uses** GraphABC. Это наш «пропуск» в мир графики. После написания этой строчки мы можем использовать специальные процедуры, которые рисуют на экране линии, окружности, точки, меняют цвет изображения и фона. Мы их все изучать не будем, воспользуемся только малой частью. Мы написали процедуру Ug4, которая строит четырехугольник с вершинами, координаты которых задаются её восемью параметрами (ведь у каждой точки две координаты). Для построения четырёхугольника надо соединить между собой его вершины, провести 4 отрезка (линии). Это делает встроенная процедура LINE. У неё 4 параметра – координаты начала и конца отрезка. Мы строим замкнутый четырёхугольник, поэтому первая точка первой линии совпадает с последней точкой четвёртой линии.

Посмотрим теперь на основную программу. Во-первых, обратите внимание – все переменные целые. Мы будем использовать эти переменные для хранения координат, координаты на экране – целые числа. А каков диапазон этих чисел? Координаты на экране могут быть только положительными. Центр координат находится в левом верхнем углу экрана. Ось OX расположена привычным нам образом, направлена слева направо, а вот ось OY направлена сверху вниз. Максимальные значения координат можно узнать с помощью функций `WindowWidth` и `WindowHeight`. Советую вам всегда использовать эти функции, а не какие-то числа. Во-первых, числа надо помнить (или где-то записывать), а во-вторых, эти значения могут быть разными для разных компиляторов и настроек Паскаля. Мы в нашей программе с помощью этих функций определяем координаты центра экрана. Обратите внимание – деление целочисленное.

Теперь определим координаты вершин четырёхугольника так, чтобы он располагался в центре экрана и был квадратом со стороной 2*L. Вызываем написанную ранее процедуру, получаем искомый квадрат.

– Но ведь в Паскале есть встроенная процедура для рисования прямоугольника. Зачем же мы свою писали?

– Во-первых, чтобы потренироваться, посмотреть, как рисовать отрезки с заданными координатами. Во-вторых, встроенная процедура рисует прямоугольник со сторонами, параллельными сторонам экрана. По-другому расположенный прямоугольник она нарисовать не может. А наша процедура – может!

И, наконец, последняя строка нашей программы – процедура рисования окружности. Именно так она везде называется, но название это неточное, процедура рисует круг (закрашивает то, что ограничено окружностью, цветом фона – в нашем случае белым). Чтобы убедиться в этом, давайте нарисуем кружочек в 2 раза меньшего радиуса с центром на правой стороне квадрата: `circle(x0+L,y0,L div 2)`.

Этих знаний нам вполне хватит, чтобы построить наш первый рекурсивный узор. Алгоритм построения будет такой. Рисуем квадрат. Находим середины всех сторон. Рисуем четырёхугольник с вершинами в найденных точках (он тоже будет квадратом, но наша программа этого не знает, она будет рисовать четырёхугольник по заданным координатам). С получившимся квадратом делаем то же самое: находим середины сторон, на них строим новый квадрат. И так до тех пор, пока сторона квадрата не станет слишком маленькой.

```
Program kvadrat;
uses GraphABC;
Function Mid(A,B:Integer):Integer;
Begin
  Mid:=(A+B)div 2;
End;
Procedure Uzor(x1, y1, x2, y2, x3, y3, x4, y4:Integer);
Begin
 Line(x1, y1, x2, y2);
 Line(x2, y2, x3, y3);
 Line(x3, y3, x4, y4);
 Line(x4, y4, x1, y1);
 if (abs(y1-y2)>15) or(abs(x1-x2)>15)
      then Uzor(mid(x1,x2), mid(y1,y2),
                mid(x2,x3), mid(y2,y3),
                mid(x3,x4), mid(y3,y4),
                mid(x4,x1), mid(y4, y1))
end;
var x0, y0, x1,y1,x2, y2, x3, y3, x4, y4, L:integer;
begin
x0:=WindowWidth div 2;
y0:=WindowHeight div 2;
l:=200;
X1:=X0-L;Y1:=Y0+L;
X2:=x1; Y2:=Y0-L;
x3:=X0+L; Y3:=Y2;
X4:=X3; Y4:=Y1;
Uzor(x1, y1, x2, y2, x3, y3, x4, y4);
end.
```

Если всё правильно, должен получиться следующий рисунок (см. рис. 1).

Как видите, программа легко делается из предыдущей путём внесения некоторых дополнений.

Основная программа совсем не изменилась. Она определяет координаты первого, самого большого квадрата. Название процедуры изменилось, но параметры у неё те же. Изменилось тело процедуры.

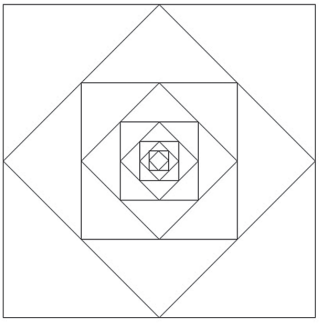

Рис. 1

Сначала она, как и прежде, строит четырёхугольник с заданными координатами. А потом – это и есть наше добавление, которое позволяет получить узор, именно в нём и содержится рекурсия – процедура определяет, не слишком ли мала сторона квадрата и, если это не так, вызывает сама себя, но уже с другими параметрами – вершинами нового четырёхугольника теперь будут середины сторон предыдущего. Для этого мы должны уметь находить середину отрезка (среднее арифметическое соответствующих координат). Функция MID упростит нам эту задачу.

Итак – база (в данном случае уместнее сказать «окончание») рекурсии – это маленькая сторона квадрата, а рекурсивный шаг – вызов процедуры с другими параметрами.

– А почему такое сложное условие окончания рекурсии? И иксы проверяются, и игреки? У нас ведь квадрат, все стороны одинаковые, можно было бы какую-то одну проверить.

– Дело в том, что мы здесь не проверяем длину стороны квадрата. Можно было бы её проверять, но для этого её надо посчитать. Если квадрат «ровный» (стороны параллельны сторонам экрана), длина стороны вычисляется простым вычитанием «игреков» или «иксов», если «косой» – придётся вычислять (например, по теореме Пифагора). Поэтому мы здесь берём просто разность координат – нам ведь не важно, какой величины будет последний квадрат, нужно, чтобы рекурсия закончилась, а не пыталась рисовать и рисовать квадраты всё меньшего и меньшего размера. Однако и с «ровным» квадратом всё не так просто. Ведь у его соседних вершин координаты X или Y равны. А так как «начальная» точка рисования квадрата у нас всё время разная, чтобы не «угадывать», какие координаты у данного квадрата будут равными, какова длина его стороны, поставлено такое сложное условие.

Вы можете изменить начальную длину стороны квадрата (переменная L), условие окончания рисования, можете вообще вначале нарисовать не квадрат, а произвольный четырехугольник. Будут получаться разные рисунки.

Надо сказать, что этот узор мы с вами нарисовали «для затравки». Он не отражает всех возможностей рекурсивных алгоритмов – точно такой же узор можно очень легко сделать без рекурсии, с помощью цикла.

Рассмотрим более сложные алгоритмы. Создадим узор таким образом (рис. 2): построим квадрат – стороны параллельны сторонам экрана, центр в центре экрана, длина стороны задана. Теперь построим 5 таких же квадратов (стороны тоже параллельны сторонам экрана), но с меньшей длиной стороны и с центрами, расположенными так: у четырёх – на серединах сторон «главного» квадрата, а у пятого – в центре «главного» квадрата. Закончим рекурсию так же – когда квадрат станет слишком маленьким. Но здесь это будет проще, у нас будет параметр L – половина длины стороны квадрата.

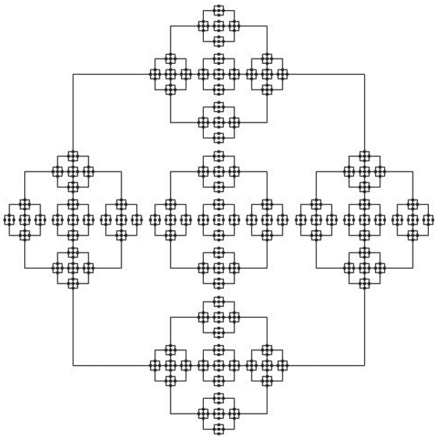

Рис. 2

Рисунок сильно отличается от предыдущего, а программа будет отличаться не очень. Нам также потребуется наше умение рисовать квадрат и находить середину его стороны. Входными параметрами для узора будут координаты середины очередного квадрата и длина его стороны (точнее, этот параметр равен половине длины).

Процедура для рисования узора сначала вычисляет координаты сторон квадрата (как мы это уже делали – по заданному центру и полудлине стороны), строит квадрат с получившимися координатами углов. Потом уменьшает длину стороны (в нашем случае – в 3 раза) и, если она ещё ненулевая, рекурсивно обращается сама к себе 5 раз – с разными координатами центра узора, как это описано в алгоритме. Здесь нам уже будут видны достоинства рекурсивного алгоритма – без рекурсии такое проделать сложно, так как количество «обрабатываемых» квадратов будет все время увеличиваться. Вот соответствующая программа:

```
Program kvadrat1;
uses GraphABC;
Function Mid(A,B:Integer):Integer;
Begin
  Mid:=(A+B)div 2;
End;
Procedure Uzor (x0,y0, L:Integer);
Var x1,x2,x3,x4,y1,y2,y3,y4:Integer;
Begin
 x1:=x0-L;y1:=y0+L;
 x2:=x1; y2:=y0-L;
 x3:=x0+L; y3:=y2;
 x4:=x3; y4:=y1;
 Line(x1, y1, x2, y2);
 Line(x2, y2, x3, y3);
 Line(x3, y3, x4, y4);
 Line(x4, y4, x1, y1);
 L:=L div 3;
 If L> 0 Then Begin   Uzor(x0,y0,L);
                      Uzor(mid(x1,x2), mid(y1,y2),L);
                      Uzor(mid(x2,x3), mid(y2,y3),L);
                      Uzor(mid(x3,x4), mid(y3,y4),L);
                      Uzor(mid(x4,x2), mid(y4,y1),L)
              End
End;
var x0, y0, x1,y1, L:integer;
begin
x0:=WindowWidth div 2;
y0:=WindowHeight div 2;
L:=160;
Uzor(x0,y0,L);
end.
```

Вы можете изменять параметры и получать разные рисунки. Можно взять другое начальное значение стороны квадрата и при этом каким-либо другим образом уменьшать *L* (а можно и увеличивать, но для этого надо начать с маленького квадрата и грамотно определить, когда остановиться). Можно делать не пять рекурсивных обращений, а, например, только те, которые рисуют квадраты на правой и левой стороне.

– А можно, наоборот, больше сделать? Нарисовать еще квадратики с центрами в углах «главного» квадрата?

– Конечно, попробуйте. Давайте посмотрим, что при этом получится.

– Вот я сделала, что длина стороны уменьшается в 3 раза и происходит 9 рекурсивных вызовов – добавила ещё квадраты в вершинах (рис. 3). Только совсем не видно, что узор получается таким образом. Кажется, будто я сначала нарисовала ряды больших квадратиков, потом ряды квадратиков поменьше...

– Да, так часто бывает, когда деталей в узоре много и они слишком мелкие.

– А у меня совсем что-то странное получилось (рис. 4): как будто совсем другая программа. Ромб, а внутри круги какие-то.

– А, может, у тебя и есть другая программа? Ты много изменил?

– Да нет. Я решил, что длина стороны слишком быстро уменьшается, и поставил `L:=L-L` **`div`** `3`. Ну и в самом начале сделал L:=100. При этом у меня чуть ли не все закрасилось, и я сделал условие окончания L>10. Откуда круги? Компьютер их сам зачем-то нарисовал?

– Нет, никаких кругов компьютер, конечно же, сам не рисовал. Из-за того, что длина стороны уменьшается медленно, квадратов в центре получается очень много, они рисуются друг на друге. Заполняют весь центр рисунка – видите, там все черно. Рисунок у нас симметричный, вот в центре из кусочков многочисленных квадратов и образуется черный круг.

– А ромб откуда?

– Опять же, от того, что сторона квадрата достаточно большая, а мы пририсовываем новые квадраты по центрам сторон, рисунок и растёт вправо и влево, вверх и вниз. Первоначальный квадрат закрывается вновь нарисованными квадратиками.

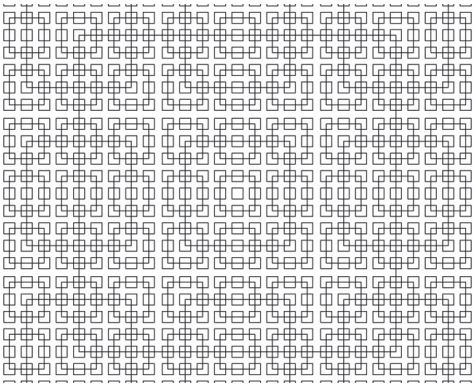

Рис. 3

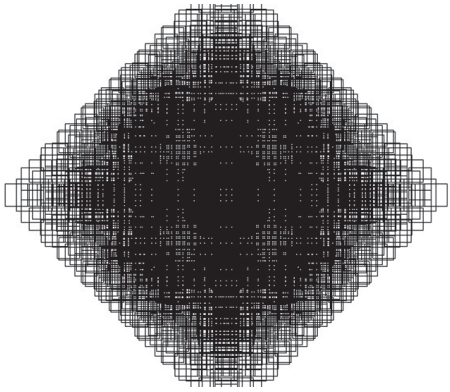

Рис. 4

Поэкспериментируем теперь с кругом. Проделаем примерно то же самое. Нарисуем в центре экрана круг. Затем нарисуем 4 круга меньшего радиуса (например, в 2 раза) с центрами, вычисленными так же,

как мы вычисляли вершины квадрата. А потом проделаем то же самое с каждым из нарисованных 4 кругов – и так далее (рис. 5). Закончим рекурсию, когда круги станут слишком маленькими (то есть будем проверять их радиус). Опять можно заметить, что на рисунке не очень-то заметны большие окружности, и угадать алгоритм, по которому производились построения, глядя на рисунок, довольно трудно.

Программа практически не отличается от предыдущей – вместо квадрата (четырёх отрезков) рисуем окружность.

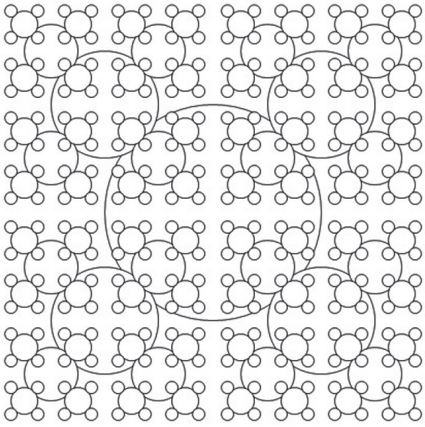

Рис. 5

```
Program krug;
uses GraphABC;
Procedure UzorKrug(x0,y0,l:Integer);
var x1,x2,x3,x4,y1,y2,y3,y4:Integer;
Begin
 circle(x0,y0,L);
 X1:=X0-L;Y1:=Y0+L;
 X2:=x1; Y2:=Y0-L;
 x3:=X0+L; Y3:=Y2;
 X4:=X3; Y4:=Y1;
 if L>10  then Begin UzorKrug (x1,y1,L div 2);
                     UzorKrug (x2,y2,L div 2);
                     UzorKrug (x3,y3,L div 2 );
                     UzorKrug (x4,y4,L div 2);

                End
End;
var x0, y0, L:integer;
Begin
x0:=WindowWidth div 2;
y0:=WindowHeight div 2;
L:=105;
UzorKrug(x0, y0,l);
End.
```

Тоже можно изменять параметры и получать самые разные рисунки. Попробуйте.

– Вот посмотрите, какая у меня чешуя получилась (рис. 6). Я сделала L:=L-15 и условие окончания L>5.

– А мне не нравится, что мы рисуем круги, а не окружности. По-моему, было бы красивее, если бы были видны все окружности, если бы большие не «заклеивались» маленькими. Можно это сделать?

– Можно вместо окружности (процедура circle) нарисовать дугу (процедура arc). У этой процедуры добавляются ещё два параметра: углы начала и конца дуги. Чтобы получилась окружность, их надо поставить 0 и 360.

– Вот, я так сделала (рис. 7). Только пришлось поставить L:=L-20 и L>20, иначе весь экран чёрным закрашивался, так много кружочков рисовалось. Получилось кружево.

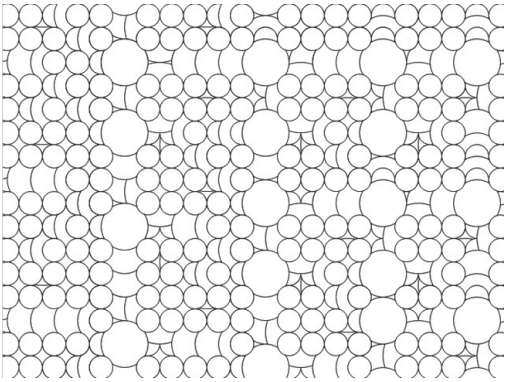

Рис. 6

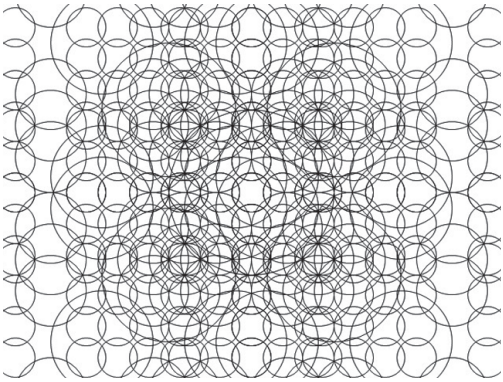

Рис. 7

– А у меня что-то непонятное. Нарисовались 4 кружочка, поморгали, а потом было написано, что программу надо закрыть...

– Что же ты там такого натворил?

– Да ничего особенного. Мне не нравится, что кружки очень скоро становятся слишком маленькими, почти точечками, вот я и решил не изменять L.

– Ну, тогда все понятно – ты не написал выход из рекурсии, у нас ведь он был, когда L маленьким становится, а у тебя не становится.

– Нет, выход я написал вот такой:

```
if (x1>10) and (y2>10) and (x3<WindowWidth) and (y1<WindowHeight)
```

Я подумал, что наши кружочки заполняют весь экран и можно остановиться, когда их центры будут слишком близко к краям экрана. Но у меня картинка даже и не думает «добегать» до краев экрана! Смотрите, какие вначале 4 кружочка мелькают (рис. 8).

– Давайте разбираться, что происходит. Для этого вернёмся к нашему первому варианту программы Krug. Чтобы понять, как она работает, после вызова процедуры circle поставим Readln. Теперь после рисования одного круга программа будет останавливаться и ждать нажатия клавиши ввода. Только после этого она нарисует следующий круг. Вот какая картинка получится после нескольких шагов (рис. 9).

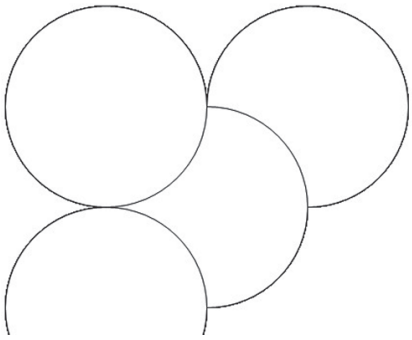

Рис. 8

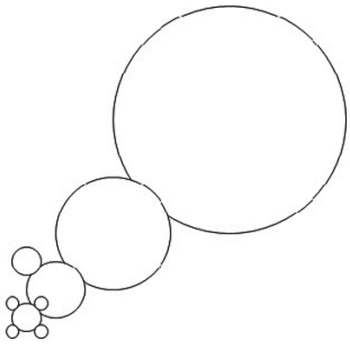

Рис. 9

После рисования первого (самого большого) круга работает первый вызов рекурсивной процедуры, и рисуется круг поменьше – слева снизу, потом круг ещё поменьше (опять слева снизу, так как работает опять первый из рекурсивных вызовов). Когда слева снизу рисуется самый маленький круг, оказывается, что радиус следующего уже будет меньше 10, ветвь рекурсии прерывается, работает второй, третий и четвёртый рекурсивный вызовы (рисуются три остальные маленькие круга). Ветвь «влево вниз» закончилась, теперь для третьего по величине кружка рисуется левый верхний сосед. То есть программа сначала «зарисовывает» всё, что возможно, влево вниз, потом влево вверх потом вправо вверх и, наконец, вправо вниз – в таком порядке у нас стоят рекурсивные вызовы.

Теперь посмотрим на программу, которая зациклилась. Нарисован центральный круг (на рисунке он самый правый верхний). К нему пририсовывается один слева снизу, затем другой. Центр последнего оказался слишком близко к краю экрана, поэтому рекурсивная ветвь заканчивается, влево вниз идти некуда. Значит надо рисовать остальные круги, которые стоят вокруг второго (по порядку появления на рисунке) круга. Рисуется левый верхний круг. Так получается, что вокруг него рисовать нечего. Надо продолжать рисовать круги вокруг второго. Но справа сверху от второго уже есть круг – первый. Программа этого не знает, рисует его ещё раз. А что она должна делать дальше? Рисовать круги для этого (первого и одновременно четвёртого) круга. Нарисовать она может только то же самое. Вот и мелькают круги на экране (потому что то один, то другой становятся «верхними»), программа не останавливается, стек переполняется... Помните, мы на одном из первых занятий по рекурсии употребляли термин «порочный круг». Вот, рисуя круги, его и получили. Нельзя писать программу так, чтобы она приходила на то же место, с которого начала – зациклится.

– А как же быть? Как исправить?

– Просто исправить, заменив в программе несколько символов, здесь не получится, надо изменять алгоритм. Можно запоминать центры уже нарисованных кругов и не рисовать их по второму разу. Можно смотреть, не нарисован ли на заданном месте круг. В конце концов, можно не пользоваться рекурсией. Рекурсивные алгоритмы удобны не для всяких узоров.

– Пока это обсуждали, я вот какую красоту сделала (рис. 10)! Угадайте, из чего это получилось?

– Ромбики? Квадратики?

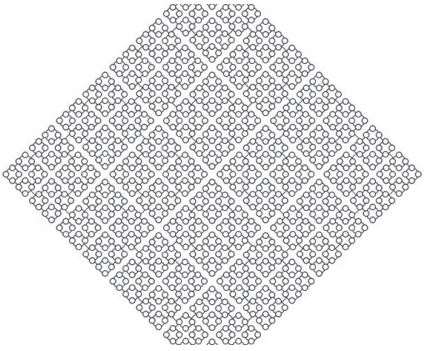

Рис. 10

– Нет! Это всё та же программа с окружностями. Только я сделала, чтобы новые кружочки – 4 штуки – рисовались с центрами на первоначальной окружности: в самой верхней, самой нижней, самой левой и в самой правой. Вот как у меня центры окружностей задаются:

```
X1:=X0-L; Y1:=Y0-L;
X2:=X0+L; Y2:=Y0+L;
```

Двух координат вполне достаточно, для определения координат центров новых окружностей нужны ещё координаты центра первоначальной.

– А можно теперь объединить программы и все кружочки нарисовать?

– Конечно, можно. Но давайте лучше напишем универсальную программу, пользуясь которой, можно будет, изменяя параметры, получать разные картинки.

Для этого нам придётся вспомнить, как рисуются вписанные в окружность правильные многоугольники.

– Ну вот, опять геометрия! Я сюда рисовать пришёл.

– Но мы ведь на самом деле не рисуем, а чертим. Нужно уметь вычислять координаты точек, тем более считать ничего не придётся – это сделает компьютер, надо только правильные формулы вспомнить.

Формул немного, но есть сложные, с синусами-косинусами. Кто еще эти понятия не проходил, просто поверьте, что координаты вычисляются именно так, доказывать мы здесь ничего не будем.

Итак, любой правильный N-угольник можно вписать в окружность. Если из центра окружности провести отрезки (радиусы) к вершинам N-угольника, получим N одинаковых треугольников. У этих треугольников угол с вершиной в центре окружности равен 360 °/N – полный оборот 360°, а мы его разделили на N частей. В Паскале в некоторых формулах используются не градусы, а радианы (кто пока не знает, что это такое, опять же просто поверьте на слово). В радианах величина такого угла выражается 2*Pi/N (число «пи» сразу обозначим, как это принято в Паскале). Теперь вспомним, что координаты любой точки на окружности можно вычислить по формуле

$$X = X_0 + \cos(F),$$
$$Y = Y_0 + \sin(F),$$

где X_0, Y_0 – координаты центра окружности, F – угол (между осью OX и радиусом, проведённым к точке).

По этим формулам мы можем вычислять координаты вершин правильного N-угольника. Начальное значение угла F задаем любое (оно будет влиять на то, каким образом «повёрнут» многоугольник), а затем, для расчёта координат следующей точки, увеличиваем F на величину угла.

Сам многоугольник рисовать не будем. Будем рисовать окружности с центрами в вершинах воображаемого правильного N-угольника. Рисуется окружность, затем N меньших окружностей, центры которых равномерно расположены по первой окружности. На каждой из них рисуется по N ещё меньших окружностей. Остановка такая же, как всегда – когда размер станет слишком маленьким. Приведём здесь не полную программу, а только процедуру.

```
Procedure MnUg(x0,y0,l, n:Integer);
var x, y, i: Integer;
    F:Real;
Begin
 arc(x0,y0,L, 0, 360);
 if L>60 then Begin
       F:=0;
       For I:=1 to N do
       Begin X:=round(X0+L* cos(F));Y:=round(Y0+L*sin (F));
             MnUg(x,y,L div 2, n );
             F:=F+2*Pi/n
       End
              End
End;
```

У нашей процедуры 4 параметра – центр окружности, радиус и количество окружностей, которые будут «висеть» на «главной». Процедура рисует окружность с заданным центром, а потом в цикле рассчитывает координаты меньших окружностей и с полученными параметрами рекурсивно к себе обращается.

Помните, на прошлом занятии кто-то спросил, бывают ли программы, в которых одновременно встречается и цикл, и рекурсия?

– Да, и Вы сказали, что мы до таких программ ещё не доросли!

– Что же, как видите, прогресс налицо. Только не забудьте посмотреть на текст программы. Обратите внимание: рассчитанные по формулам значения координат округляются, так как на экране компьютера координаты могут быть только целыми.

Давайте теперь посмотрим, какие картинки можно получить, меняя параметры этой программы.

– Вот у меня кружочки в хороводе (рис. 11)! Я взяла начальное L:=200, N:=18 и конечное L>40.

– А у меня (рис. 12) числа 150, 5 и 8 – пятиугольник и получился.

– Да, если все числа те же, только вместо 5 взять 3, получится треугольник (рис. 13).

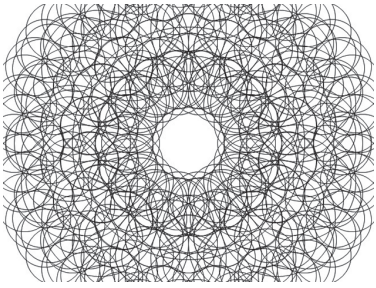

Рис. 11

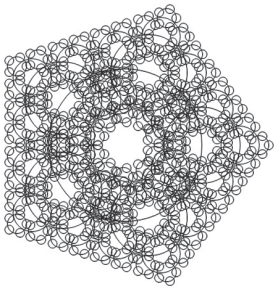

Рис. 12

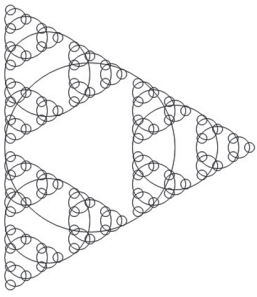

Рис. 13

– А теперь давайте рисовать не окружности, а линии – радиусы окружностей к вершинам воображаемого правильного N-угольника. Это сделать очень просто. Мы знаем координаты центра и координаты вершин. Из программы надо убрать arc, а в цикл после определения координат точек вставить рисование отрезка `line(x0,y0,x,y)`. Для N=18, 5 и 3 получаются рис. 14, 15 и 16.

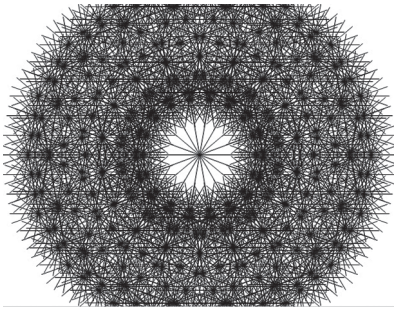

Рис. 14

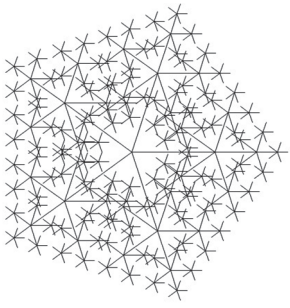

Рис. 15

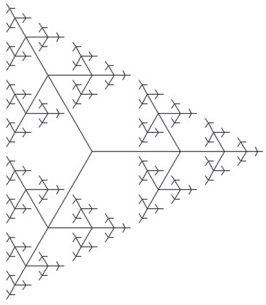

Рис. 16

Имея такой «конструктор», вы можете построить множество различных узоров. Попробуйте нарисовать многоугольник, а потом с цен-

трами в его вершинах – многоугольники поменьше и так далее. Можно маленькие многоугольники рисовать в углах «главного». В общем, включайте фантазию.

А напоследок мне бы хотелось разобрать с вами, как может выглядеть процедура закрашивания.

– А в Паскале есть такая процедура?

– Конечно, есть. Но эта процедура встроенная, мы можем видеть результат её работы, но не знаем, как она работает, какой алгоритм использует. Вот давайте и напишем свою (заметьте, я вовсе не утверждаю, что наша процедура будет использовать тот же алгоритм, что и стандартная).

Сначала подготовим картинку для закрашивания. Как «объяснить» процедуре, что надо закрашивать? Можно нарисовать контур (обязательно замкнутый, чтобы краска не «пролилась») и задать точку внутри этого контура. Из неё будет «литься» краска и «расползаться» по области, ограниченной контуром. Нарисуем несколько разноцветных фигур, чтобы область была поинтереснее (мы используем здесь некоторые новые графические процедуры, но, я думаю, все будет понятно, в программе есть пояснения к ним). Чтобы краска не «вылилась» за контур, сделаем его потолще, в Паскале есть такая возможность – увеличить толщину пера для рисования.

Основная программа будет выглядеть так:

```
begin
    SetPenWidth(4); {увеличение толщины пера}
    SetPenColor(ClRed); {изменение цвета пера.
    Далее фигуры будут рисоваться красным цветом}
    rectangle(13,13,620,260);{прямоугольник}
    circle(240,215,8);{кружок}
    SetPenColor(ClBlack);{цвет пера изменен на черный.
    Далее фигуры будут рисоваться черным цветом}
    arc(100,220,160, 0, 360);{окружность}
    arc(400,240,170,0,360); {окружность}
    circle(260,150,50);{круг}
    {Здесь должен быть вызов процедуры закрашивания}
end.
```

Фрагмент этого изображения можно видеть на рис. 17. Будем закрашивать область пересечения двух больших окружностей, точку начала закраски определим чуть ниже маленького красного круга.

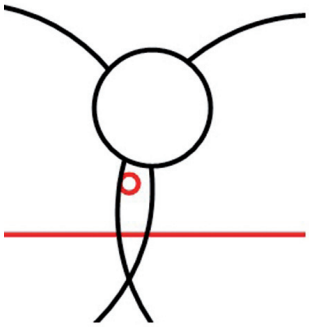

Рис. 17

А алгоритм закраски будет очень простой. Среди его параметров обязательно будут координаты точки и цвет закраски. Сначала определим цвет заданной точки – надо ли её красить? Ведь если цвет точки уже равен цвету закраски, её красить не надо. Могут быть и другие условия (мы сами их можем задавать), исходя из которых точку закрашивать не следует.

Для определения цвета точки есть встроенная функция `getpixel(x,y)`, здесь X и Y – координаты. Функция выдаёт ответ типа Color (это такой специальный зарезервированный тип в графическом модуле ABC-Паскаля), он содержит в себе три числа, обозначающих интенсивность красной, зеленой и синей составляющей исследуемого цвета.

– Это мы проходили! RGB называется.

– Правильно. Так как сравнивать мы можем числа по одному, придётся их «вытащить». Для этого воспользуемся функциями `getred`, `getgreen` и `getblue`, каждая из которых возвращает числовую величину соответствующей составляющей цвета. Итак, определяем, надо ли красить точку. Если надо – красим. Используем для этого процедуру `PutPixel` – она имеет 3 параметра (координаты и цвет) и окрашивает точку с заданными координатами в заданный цвет. А далее рекурсивно вызываем процедуру закраски для «окружения» нашей точки: точек, у которых координата X такая же, а Y на единичку больше и на единичку меньше, и точек, у которых такой же Y, а X на единичку больше и меньше.

Вот и вся процедура. Теперь о главном в ней – условии прекращения рекурсии. Оно может быть, например, таким: красим до тех пор, пока не встретим контур того же цвета, что и цвет закраски. В этом случае у процедуры будет 3 параметра (координаты и цвет), а основная программа будет выглядеть так:

```
Procedure Kraska(x,y:Integer;c:Color);
var c1:Color;
Begin
  c1:=getpixel(x,y);{определили три составляющие цвета}
{если хотя бы одна из составляющих цвета не совпадает с соот-
ветствующей составляющей другого цвета – цвета не равны}
  if (getred(c1)<>getred(c))or
     (getgreen(c1)<>getgreen(c))or
     (getblue(c1)<>getblue(c)) then Begin  PutPixel(x,y, c);
                                           Kraska(x+1,y,c);
                                           Kraska(x-1,y,c);
                                           Kraska(x,y+1,c);
                                           Kraska(x,y-1,c);
                                      End;

end;
```

В основной программе эта процедура вызывается следующим образом: `kraska(240,240,clBlack)`, здесь `clBlack` – цвет контура и цвет закраски (чёрный).

Посмотрите, как проработала наша процедура (рис. 18): она «не обратила внимания» на красные линии, закрасила их и остановилась, только когда встретила черные линии.

Давайте изменим условие закраски. Пусть процедура красит точку только в том случае, если она до этого была заданного цвета. У такой процедуры будет 4 параметра: координаты, цвет, при котором надо закрашивать, и цвет, в который надо закрашивать. Вызов может выглядеть так: `kraska(240,240,clWhite,clBlack)` – все встретившиеся «по пути» точки белого цвета перекрасим в чёрный цвет.

Условие остановки рекурсии в процедуре тогда будет выглядеть так:

```
if
(getred(cb)=getred(c1))
        and
(getgreen(cb)=getgreen(c1))
        and
(getblue(cb)=getblue(c1)).
```

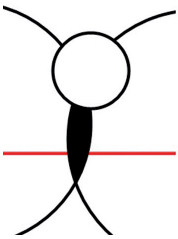

Рис. 18

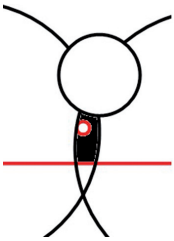

Рис. 19

Здесь cb – заданный цвет, который надо изменить, c1 – полученный цвет исследуемой точки. Для закраски точки необходимо полное совпадение цветов – все составляющие равны. Закрашивая ту же область этой процедурой (рис. 19), получим другой результат – окрашена меньшая область, процедура остановилась, обнаружив красные линии (они ведь не белые!).

Таким образом, задавая разные условия окончания рекурсии, вы можете создавать разные процедуры закраски.

Я надеюсь, вам понравилось такое несколько нестандартное использование компьютера, а главное, думаю, теперь вы подружитесь с рекурсивными процедурами и функциями и сможете создавать разные узоры. Экспериментируйте: мы сегодня рисовали только чёрно-белые картинки, использовали лишь линии и круги. Посмотрите, какие еще графические процедуры есть в Паскале, что можно нарисовать с их помощью.

Юмор Юмор Юмор Юмор Юмор Юмор

Из оды Нильсу Бору, написанной одним из его сотрудников

Наполните кубки, огонь разожгите,
Пойте во славу, забывши все споры!
В трубы трубите и лиры возьмите,
Одой восславьте героя Нильс Бора!

Кванты пред Вами склоняют колени,
А электроны летят по орбитам.
Каждый в своей, словно скованный пленник,
Рвёт свои цепи, готовит побег.

Бору поём мы торжественно славу!
Бор, наш учитель, ведёт нас вперёд.
Знаем и верим ему мы по праву,
Скоро с ума нас наука сведёт.

Мы всегда во всём за Вас,
Не ругайте только нас.
Мы Ваши теории все перечтём,
Даже... коль в них ничего не поймём.