Самостоятельная работа. Реализация основных рекурсивных алгоритмов

Задание. Создать библиотеку(модуль) рекурсивных функций на основе псевдокода их описаний..

Написать программу тестирующую все функции этой библиотеки.

* Создать фрактальные изображения (дополнительно). Рекурсия в изображении.

1. Факториал числа

```
алг цел Факториал(арг цел н)   :алгоритм-функция
нач
если н=1
       то знач: =1                          значение функции
иначе                                       рекурсивный вызов функции Факториал
       знач := н * Факториал(н-1)
все
кон

алг Расчет факториала                  основной алгоритм

нач цел  н
  вывод "Задайте число н"
   ввод  н
  вывод "н ! =", Факториал(н)
кон
```

2. Натуральная степень числа

```
алг вещ Степень(арг  вещ а, цел н)
если  н=0
        то  знач:=1
иначе
        знач: = а * Степень(а, н-1)
все

кон

алг  Возведение в степень        основной алгоритм
нач
  вещ а, цел н
  вывел "Задайте число а"
  ввод а
  вывод "Задайте показатель степени н"
  ввод н
  вывод "Число а в степени н равно",
 Степень (а. н )
кон
```

3. Член прогрессии

а) арифметической:

```
алг  вещ Член арифм(арг вещ а1, д, цел н)
нач
  если н=1
    то знач: =а1
  иначе
     знач: =Член арифм (а1,д,н-1) + д
  все
кон
```

б) геометрической:

```
алг вещ Член геом(арг вещ а1, к, цел н)
нач
  если н=1
      то знач:=а1
  иначе
    знач := Член геом(а1,к,н-1)*к
  все
кон
```

в) основной алгоритм:

```
алг Определение члена н прогрессии
нач
 вещ а1,д, к цел н
 вывод "Задайте первый член прогрессии"
 ввод а1
 вывод "Задайте разность арифметической прогрессии"
 ввод д
 вывод "Задайте знаменатель геометрической прогрессии"
 ввод к
 вывод  "Задайте номер члена прогрессии"
 ввод н
 вывод "Арифм =", Член арифм(а, д, н)
 вывод "Геом. = ",  Член геом(а1,к,н)

кон
```

4. Сумма членов прогрессии

а) арифметической:

```
алг вещ Сумма арифм (арг вещ а1, д, цел  н)
нач
  если н=1
    то знач:=а1
  иначе
     знач: =Сумма арифм(а1, д, н-1) + Член арифм(а1, д, н)
  все

кон
```

б) геометрической:

```
алг вел Сумма геом(арг вещ а1, к,   цел  н)
нач
если н=1
то знач:=а1;
иначе
знач:=Сумма геом(а1,к,н-1)+Член геом(а1,к,н)

кон
```

в) основной алгоритм:

```
алг Определение суммы членов прогрессии
нач
вещ а1,д,к.цел н
вывод "Задайте первый член прогрессии"
ввод а1
вывод "Задайте разность арифметической прогрессии"
ввод д
вывод Задайте знаменатель геометрической прогрессии"
ввод к
вывод "Задайте число членов прогрессии"
ввод н
вывод "Сумма арифм.=", Сумма арифм(а1, д, н)
вывод ."Сумма геом.=",Сумма геом(а1, к, н)
```

```
кон
```

Примечание. При расчете суммы членов прогрессии используются функции, описанные в п. 4.

5. Последовательность Фибоначчи

```
алг цел Фиб(цел н)              функция для расчета н-го члена
нач
  если н=1 или н=2
        то знач:=1
  иначе
        знач :=Фиб (н-1)+Фиб (н-2)
   все
кон
```

Основной алгоритм:

```
алг Нахождение члена последовательности Фибоначчи
нач
 цел н
 вывод "Задайте номер члена последовательности"
 ввод н
 вывод "Этот член последовательности =", Фиб( н )
кон
```

6. Алгоритм поиска минимального элемента таблицы (массива)

Приведем сначала функцию, вычисляющую минимум из двух чисел.

```
алг цел Мин(арг цел a, b)
нач
    если a>b
        то знач: = b
    иначе
         знач:=a
    все
кон
```

Теперь напишем функцию, которая находит минимум среди первых н элементов таблицы т[1:нтаб]. При этом, если н больше двух, то будем считать результатом минимум из двух чисел т[н] и минимального числа из первых (н-1) элементов таблицы (рекурсивный вызов).

```
алг цел Минимум(арг цел н, цел таб т[1:нтаб])
нач
цел i
  если н=2                                  !н - номер последнего элемента
                                            !в рассматриваемой части таблицы
        то знач: = Мин(т[н], т[1] )
  иначе
        знач:=Мин(т[н],  Минимум(н-1, т))  !рекурсивный вызов функции Минимум
  все
кон
```

Чтобы найти минимум всех элементов таблицы, надо обратиться к функции Минимум, указав в качестве первого аргумента длину таблицы нтаб.

Основной алгоритм:

```
алг Поиск минимального элемента
нач
 цел таб[1:10], цел i
 нц для i от 1 до 10                    Заполнение таблицы
```

```
    т[i] := random(50)
вывод т                                    печать таблицы
вывод. "Значение миним. элемента =", Минимум(10,т)

кон
```

7. Алгоритм поиска индекса минимального элемента таблицы (массива)

Функция, определяющая индекс минимального из двух элементов т[перв] и т[посл] таблицы т, описывается следующим образом

```
алг цел Инд мин(арг цел посл, перв, цел таб т[1:нтаб])
нач
   если т[перв]>т[посл]
     то знач =посл
   иначе
       знач:=перв
   все
кон
```

Следующая функция находит индекс минимума первых н элементов массива т[1:нтаб] (по аналогии с п.6).

```
алг  цел Индекс минимума(арг  цел н, цел таб т[ 1:нтаб])
нач
цел i
  если н=2
       то знач: =Инд мин(н,1,т)
   иначе                                   рекурсивный вызов функции Индекс минимума
       знач: =Инд мин( н, Индекс минимума(н-1,т),т)
   все
кон
```

Как и в предыдущем случае, обратившись к функции Индекс минимума с первым аргументом — длиной таблицы нтаб, найдем индекс минимального элемента.

8. Алгоритм бинарного поиска элемента таблицы (массива)

```
Реализовать на основе описания рекурсивную версию.
Бинарный поиск может использоваться только для упорядоченного списка.
Имеем массив из n элементов.
Индексы в концах списка: low=0, high=n-1.
Алгоритм бинарного поиска:
1. Вычислить индекс серединного элемента массива
    mid=(low+high)/2.
2. Сравнить значение в серединном элементе с key.
    Если совпадение найдено, возвратить индекс mid для нахождения ключа.
    Если a[mid]<key, то проводить повторный поиск в правой половине
рассматриваемого списка.
    Если a[mid]>key, то проводить повторный поиск в левой половине
рассматриваемого списка.
3. Если искомый элемент не находится в списке, то возвратить индикатор сбоя.
```

Анализ рекурсивных алгоритмов

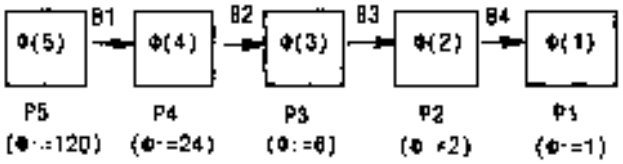
При изучении темы "Рекурсия", как показывает опыт, полезно проанализировать рекурсивные алгоритмы с точки зрения последовательности их выполнения. Под последовательностью выполнения рекурсивного алгоритма будем понимать последовательность вызовов алгоритма с различными значениями аргументов и очередность определения результатов. Такой анализ способствует лучшему пониманию механизма рекурсии, а также позволяет сделать вывод о нецелесообразности применения рекурсивных алгоритмов при решении некоторых задач (см. ниже).

Рассмотрим сначала функцию расчета факториала числа:

```
алг цел ф(арг цел н)
нач
если н=1
  то  знач:=1
  иначе
      знач:=н*ф(н-1)
все кон
```

Для этого алгоритма последовательность его выполнения достаточно очевидна и для случая н=5 представлена на рис 4.

Рис. 4. Последовательность действий при расчете Ф(5)



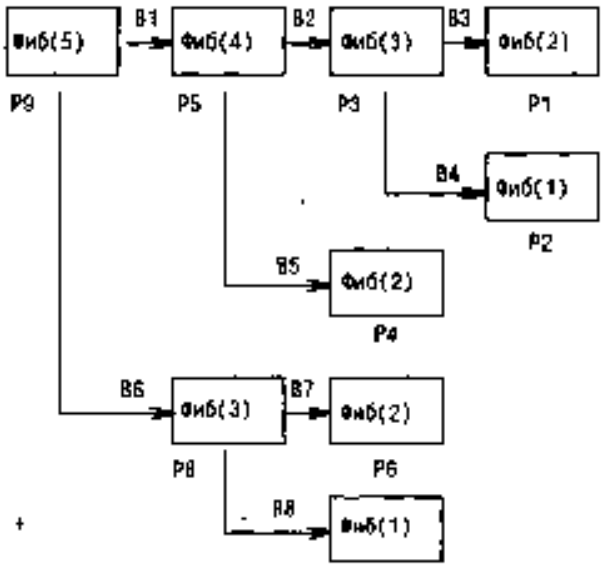
Стрелка с буквой В обозначает вызов функции (число при букве В указывает порядковый номер вызова), а числа при букве В указывают порядковый номер в последовательности расчетов (с присваиванием значений) функции Ф.

Для алгоритма определения n-го члена ряда Фибоначчи (алгоритм 5)

```
алг цел Фиб(адп пел н)

нач
 если  н=1 или н=2
    то   знач:=1
    иначе
         знач := Фиб(н-1)+Фиб(н-2)
все кон
```

соответствующая схема усложняется и для случая нахождения значения Фиб(5) приведена на рис. 5:



Р7

Рис. Последовательность действий при расчете Фиб(5)

На рис. 5 видно, например, что порядковый номер расчета значения функции Фиб(4) в общей последовательности расчетов — 5, так как до этого необходимо определить значения Фиб(2), Фиб(1), Фиб(3) и Фиб(2).

Из схемы видно также, что в рассматриваемом случае значения функции Фиб(1),Фиб(2) и Фиб(3) определяются дважды. При нахождении члена последовательности с большим номером число повторных вычислений значительно увеличивается. В результате, например, при определении значения Фиб(17) компьютер выполнит свыше тысячи, значения Фиб(31) — свыше миллиона, а значения Фиб(45) — свыше миллиарда [3] операций сложения. В то же время при использовании нерекурсивного – итерационного - алгоритма

```
алг цел Фибоначчи(арг цел н)

 нач цел а, в,с, i
  если н<=2  то  знач:=1
   иначе
      а:=1; в:=1;
      нц для i от 3 до н
        с:=а
        а:=а+в
         в:=с
      кц
        знач:=а
все кон
```

для вычисления 45-го члена последовательности Фибоначчи потребуются всего 43 операции сложения.

Это позволяет сделать вывод о неэффективности использования рекурсии для решения рассматриваемой задачи. Аналогичный вывод можно сделать в отношении ряда других задач.

Задание. Определить результат выполнения следующих рекурсивных алгоритмов при н=5. Нарисовать схему вызовов.

```
алг Алгоритм1!(арг цел н) нач
если н>0 то
   вывод н
   Алгоритм1(н-1)
 все кон

алг Алгоритм2(арг цел н) нач
если н>0
    Алгоритм2(н-1)
    вывод н
все
кон

алг АлгоритмЗ(арг цел н) нач
если н>0
      вывод н
      АлгоритмЗ(н-1)
      вывод н
все кон
```

Графика в C++Builder

1. Канва — холст для рисования. Канва и пиксели

Многие компоненты в C++Builder имеют свойство Canvas (канва, холст), представляющее собой область компонента, на которой можно рисовать или отображать готовые изображения. Это свойство имеют формы, графические компоненты Image, PaintBox, Bitmap и многие другие. Канва содержит свойства и методы, существенно упрощающие графику C++Builder. Все сложные взаимодействия с системой спрятаны для

пользователя, так что рисовать в C++Builder может человек, совершенно не искушенный в машинной графике.

Каждая точка канвы имеет координаты X и Y. Система координат канвы, как и везде, имеет началом левый верхний угол канвы. Координата X возрастает при перемещении слева направо, а координата Y — при перемещении сверху вниз.

С координатами вы уже имели дело многократно, но пока вас не очень интересовало, что стоит за ними, в каких единицах они измеряются. Координаты измеряются в пикселях. Пиксель — это наименьший элемент поверхности рисунка, с которым можно манипулировать. Важнейшее свойство пикселя — его цвет. Для описания цвета используется тип **TColor**. С цветом вы встречаетесь практически в каждом компоненте и знаете, что в C++Builder определено множество констант типа **TColor**. Одни из них непосредственно определяют цвета (например clBlue — синий), другие определяют цвета элементов окон, которые могут меняться в зависимости от выбранной пользователем палитры цветов Windows (например, clBtnFace — цвет поверхности кнопок). Полный перечень этих констант с пояснениями см. в справке C++Builder.

Но для графики иногда этих предопределенных констант не хватает. Вам могут понадобиться такие оттенки, которых нет в стандартных палитрах. В этом случае можно задавать цвет 4-байтовым шестнадцатеричным числом, три младших разряда которого представляют собой интенсивности синего, зеленого и красного цвета соответственно. Например, значение $OOFFOOOO соответствует чистому синему цвету, $OOOOFFOO — чистому зеленому, $OOOOOOFF — чистому красному. $00000000 — черный цвет, $OOFFFFFF — белый.

2. Рисование по пикселям

Рисовать на канве можно разными способами. Первый вариант — рисование по пикселям. Для этого используется свойство канвы **Pixels**. Это свойство представляет собой двумерный массив **Canvas->Pixels[int X][int Y]**, который отвечает за цвета канвы. Например, **Canvas->Pixels[10][20]** соответствует цвету пикселя, 10-го слева и 20-го сверху. С массивом пикселей можно обращаться как с любым свойством: изменять цвет, задавая пикселю новое значение, или определять его цвет по хранящемуся в нем значению. Например, **Canvas->Pixels[10][20] = clBlackin** — это задание пикселю черного цвета.

Давайте попробуем нарисовать график некоторой функции F(X) на канве компонента Imagel, если известен диапазон ее изменения Ymax и Ymin и диапазон изменения аргумента Xmin и Xmax. Это можно сделать такой программой:

```
float X,Y;        // координаты функции
int PX,PY;        // координаты пикселей
for (PX = 0; PX <= Imagel->Width; PX++)
//X — координата, соответствующая пикселю с координатой PX
X = Xmin + PX*(Xmax — Xmin) / Image1->Width;
Y = F(X);
//PY — координата пикселя, соответствующая координате У
PY = Imagel->Height - (Y - Ymin)*Imagel->Height/(Ymax-Ymin);
 //Устанавливается черный цвет выбранного пикселя
Imagel->Canvas->Pixels[PX][PY] = clBlack;
```

В этом коде вводятся переменные X и Y, являющиеся значениями аргумента и функции, а также переменные PX и PY, являющиеся координатами пикселей, соответствующими X и Y. Сама процедура состоит из цикла по всем значениям горизонтальной координаты пикселей PX компонента Imagel. Сначала выбранное значение PX пересчитывается в соответствующее значение X. Затем производится вызов функции и определяется ее значение Y. Это значение пересчитывается в вертикальную координату пикселя PY. И в заключение цвет пикселя с координатами (PX, PY) устанавливается черным.

Попробуйте создать соответствующее приложение и посмотреть, как оно работает. Пусть для простоты мы будем ориентироваться на функцию sin(X), для которой Xmin=0, Xmax=4p (2 периода в радианах), Ymin=-l, Ymax=l.

Начните новый проект, поместите на него компонент **Image** и кнопку с надписью «Нарисовать», в обработчик события **OnClick** которой запишите код, аналогичный приведенному выше, но конкретизирующий функцию:

```
#define Pi 3.14159
float X,Y;                  // координаты функции
int PX,PY;                   // координаты пикселей
for (PX = 0; PX <=   Image1->Width; PX++)
//X — координата, соответствующая пикселю с координатой PX
X = px * 4 * Pi / Image1->Width;
Y = sin(X);
//PY — координата пикселя, соответствующая координате У
PY = Image1->Height - (Y+1) * Imagel->Height / 2;
 //Устанавливается верный цвет выбранного пикселя
Image1->Canvas->Pixels[PX][PY] = clBlack;
```

Откомпилируйте ваш проект, сохраните его и выполните.

2. *Рисование с помощью пера Pen*

У канвы имеется свойство Pen — перо. Это объект, в свою очередь имеющий ряд свойств. Одно из них уже известное вам свойство **Color** — цвет, которым наносится рисунок. Второе свойство -- **Width** (ширина линии). Ширина задается в пикселях. По умолчанию ширина равна 1.

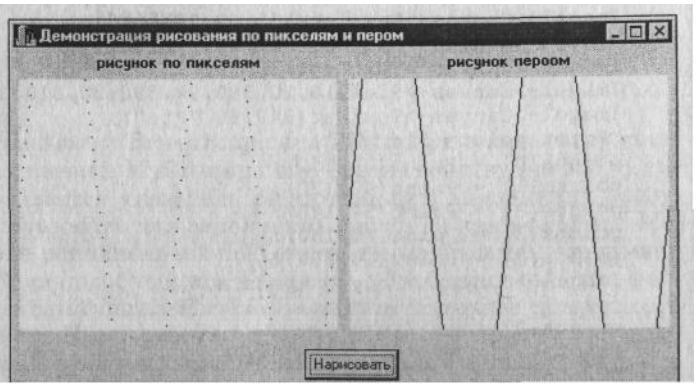
Свойство **Style** определяет вид линии. Это свойство может принимать следующие значения:
Solid Сплошная линия
Dash Штриховая линия
Dot Пунктирная линия
DashDot Штрих-пунктирная линия
DashDotDot Линия, чередующая штрих и два пунктира
Clear Отсутствие линии
InsideFrame Сплошная линия, но при **Width >** 1 допускающая цвета, отличные от палитры Windows
Все стили со штрихами и пунктирами доступны только при **Width** = 1. В противном случае линии этих стилей рисуются как сплошные.
Стиль **psInsideFrame** — единственный, который допускает произвольные цвета.
Цвет линии при остальных стилях округляется до ближайшего из палитры Windows.
У канвы имеется свойство **PenPos.** Это свойство определяет в координатах канвы текущую позицию пера. Перемещение пера без прорисовки линии, т.е. изменение **PenPos,** производится методом канвы **MoveTo(X,Y).** Здесь **(X,Y)** — координаты точки, в которую перемещается перо. Эта текущая точка становится исходной, от которой методом **LineTo(X,Y)** можно провести линию в точку с координатами **(X,Y).** При этом текущая точка перемещается в конечную точку линии и следующий вызов **LineTo** будет проводить точку из этой новой текущей точки.
Давайте попробуем нарисовать пером график синуса из предыдущего примера. Откройте прежний проект, добавьте на него еще один компонент **Image** и разместите компоненты так, как показано на рис. Размеры обоих компонентов **Image** должны быть абсолютно одинаковы, так как на этом для экономии размера и вашего труда основана программа, которую мы напишем. Сделать размеры компонентов абсолютно одинаковыми легко, выделив их оба и воспользовавшись командой всплывающего меню **Size.**
Затем в уже написанном вами обработчике щелчка на кнопке добавьте перед телом цикла оператор
```
Image2->Canvas->MoveTo(0,Image2->Height / 2);
```
который переводит перо в начало координат второго графика — на левый край канвы в середину ее высоты. А в конце цикла добавьте оператор
```
Image2->Canvas->LineTo(PX, PY) ;
```
который рисует на втором графике линию, соединяющую соседние точки. Иначе говоря, теперь ваш код должен иметь вид:
```
#define Pi 3.14159
float X,Y;              // координаты функции-
int PX,PY;              // координаты пикселей
Image2->Canvas->MoveTo(0,Image2->Height /2);
for (PX = 0; PX <= Imagel->Width; PX++)
{
                        //X — координата, соответствующая пикселю с координатой PX
X = PX * 4 * Pi;        // Imagel->Width;
Y = sin (X) ;           //PY — координата пикселя, соответствующая координате У
PY = Imagel->Height - (Y+l) * Imagel->Height /2;
Imagel->Canvas->Pixels[PX][PY] = clBlack; //Устанавливается черный цвет выбранного пикселя
Image2->Canvas->LineTo(PX, PY) ; }//Проводится линия на втором графике
```
Для экономии кода мы воспользовались тем, что оба графика у нас абсолютно одинакового размера и, следовательно, пересчет координат достаточно провести для одного из них, а потом воспользоваться этими координатами для рисования обоих графиков.
Откомпилируйте приложение и выполните его. Вы получите результат, представленный на рис. 5.8. Легко видеть, что качество двух одинаковых графиков сильно различается. В левом на крутых участках сплошной линии нет — она распадается на отдельные точки — пиксели. А правый график весь сплошной. Это показывает, что при прочих равных условиях рисовать лучше не по пикселям, а пером.
Отметим еще одно ценное свойство компонента **Image** и его канвы. Вы можете задавать координаты пикселей, выходящие за пределы размеров канвы, и ничего страшного при этом не случится. Это позволяет не заботиться о том, какая часть рисунка попадает в рамку **Image,** а какая нет. Вы можете легко проверить это, увеличив, например, вдвое размах вашей синусоиды. Для этого достаточно изменить оператор, задающий значение Y, на следующий:

```
Y = 2 * sin(X);
```

Вы получите результат, показанный на рис. 5.10.

Изобразилась только та часть рисунка, которая помещается в рамку канвы. Это позволяет легко осуществлять приложения, в которых пользователю предоставляется возможность увеличивать и просматривать в деталях какие-то фрагменты графиков.

Рис. 5.10
Изображение при размерах изображения, превышающих размер канвы

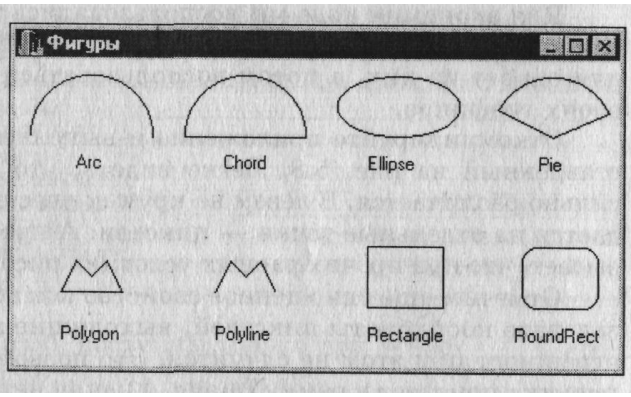


Перо может рисовать не только прямые линии, но и фигуры. Ниже перечислены некоторые из методов канвы, использующие перо для рисования фигур:

Arc Рисует дугу окружности или эллипса
Chord Рисует замкнутую фигуру, ограниченную дугой окружности или эллипса и хордой
Ellipse Рисует окружность или эллипс
Pie Рисует сектор окружности или эллипса
Polygon Рисует замкнутую фигуру с кусочно-линейной границей
Polyline Рисует кусочно-линейную кривую
Rectangle Рисует прямоугольник
RoundRect Рисует прямоугольник со скругленными углами

Примеры фигур приведены на рис. 5.11. –

Рис. 5.11
Примеры фигур, нарисованных пером



Ниже приведен текст процедуры, которая рисовала фигуры, показанные на рис. 5.11. Этот текст поможет вам понять методы, осуществляющие рисование фигур.

```
Imagel->Canvas->Font->Style =fsBold;
Imagel->Canvas->Arc(10,10, 90, 90, 90, 50, 10, 50) ;
Imagel->Canvas->TextOut(40,60, "Arc");
Imagel->Canvas->Chord(110,10,190,90,190,50,110,50);
Imagel->Canvas->TextOut(135,60,"Chord");
Imagel->Canvas->Ellipse(210,10,290,50);
Imagel->Canvas->TextOut(230,60,"Ellipse");
Imagel->Canvas->Pie(310,10,390,90,390,30,310,30);
Imagel->Canvas->TextOut(340,60,"Pie");
TPoint points [5];
points [0] = Point (30,150);
points[1] = Point (40,130);
points[2] = Point(50,140);
points [3] = Point(60,130);
points [4] = Point (70,150);
Imagel->Canvas->Polygon(points,4);
Imagel->Canvas->TextOut(30,170,"Polygon");
points[0].x += 100; points[1].x += 100; points[2].x += 100;
points[3].x += 100; points[4].x += 100;
Imagel->Canvas->Polyline(points,4);
Imagel->Canvas->TextOut(130,170,"Polyline") ;
 Imagel->Canvas->Rectangle(230,120,280,160) ;
 Imagel->Canvas->TextOut(230,170,"Rectangle");
```

```
Imagel->Canvas->RoundRect(330,120,380,160, 20, 20) ;
Imagel->Canvas->TextOut(325,170,"RoundRect");
```

Проглядите этот текст, вы можете не только просмотреть текст, но и поменять параметры методов, чтобы лучше понять, как они работают.

Для вывода текста на канву в приведенном примере использован метод **TextOut,** синтаксис которого:

```
void    fastcall  TextOut (int  X,  int  Y,  const  System::AnsiString  Text)
```

Имеется еще несколько методов вывода текста, которые применяются. Все методы вывода текста используют свойство канвы **Font** — шрифт,в частности, в приведенном примере с помощью этого свойства установлен жирный шрифт надписей.

Рекурсия для изображения

Демонстрация изображения (рис. 1): на нем представлена центральная планета П с рядом спутников, у каждого из которых свои спутники, у тех — свои и т.д. Очевидно также, что каждый спутник может рассматриваться как планета с соответствующими спутниками.

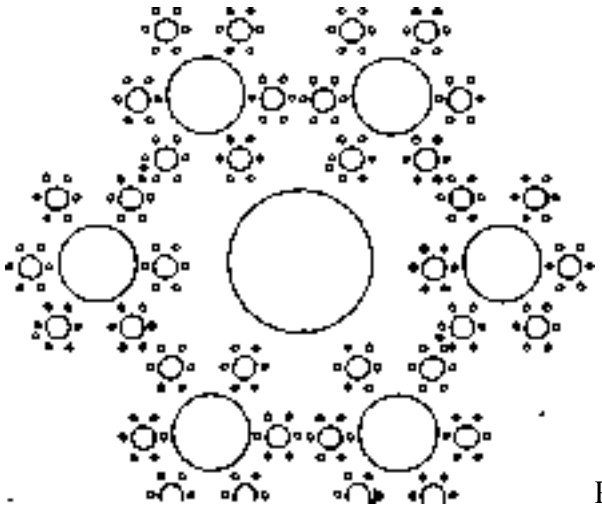


Рис.1

Поэтому если составить алгоритм, с помощью которого можно изобразить на экране некоторую окружность-планету с рядом окружностей-спутников, а для рисования каждого из спутников использовать этот же алгоритм (естественно, с другими параметрами координатами, радиусами и др.), то можно получить рассматриваемое изображение. Таким образом, мы приходим к мысли использовать в некотором алгоритме его же самого в качестве вспомогательного.

Соответствующий алгоритм имеет вид:

```
алг Планета(арг цел x,  y, рад, нсп, вещ  к_орб, к_спут)
y - координаты планеты; рад - ее радиус
нсп- число спутников у каждой планеты
к_спут -отношение радиуса спутника к радиусу «своей» планеты
к_орб –то же для радиуса орбиты спутников.
нач
   цел i, x1, y1
вещ угол, рад_орб

  поз (x, y)                                  центр планеты
  окружность(рад)                             планета
  рад_орб = рад * к_орб                        радиус орбиты спутников
  угол =  6.28 / нсп                          угол между спутниками
  нц для  i от 1 до нсп                        для каждого спутника
     x1 = x + рад_орб * cos ( угол * i )        координаты центра i-того спутника
     y1 = y + рад_орб * sin  ( угол * i )
      вызываем алгоритм Планета с новыми аргументами
     Планета (x1, y1, цел( рад * к_спут), нсп, к_орб, к_спут)
  кц
кон
```

Ситуация, когда алгоритм вызывает себя в качестве вспомогательного, называется ***рекурсией*** (от латинского recursio—возвращение).

Как работает алгоритм Планета? Очевидно, что он будет выполняться бесконечно, рекурсивные вызовы никогда не кончатся.

Чтобы сделать рассмотренный алгоритм конечным, можно использовать в нем в качестве аргумента некоторую величину н, которая при каждом новом вызове алгоритма будет уменьшаться на 1, а в тело алгоритма следует включить условие, что его команды должны выполняться только при н>0. С учетом этого приведенный выше алгоритм должен быть модифинирован:

```
алг Планета(арг цел x, y, рад, н , нсп, вещ  к_орб, к_спут)
 нач

  …

если  н>0
  то
     поз(x,y)
     …

     Планета( x1,y1,  int(рад * к_спут),  н-1, нсп, к_орб, к_спут)

    ...

  все

кон
```

В заключение я представляю основной алгоритм, в котором можно задавать различные значения исходных величин:

```
алг Система планет
нач
   цел н, нсп, рад, x, y, вещ  к_орб, к_спут
   н = 7; нсп = 4
   x = максХ  / 2
   y = максУ / 2                                  планета помещается в центр экрана
  рад  = 120; к_орб = 1.6; к_спут = 0.5
  Планета(x, y, рад, н, нсп, к_орб, к_спут)
 кон
```

и демонстрирую его работу (кстати, с его помощью можно получить множество привлекательных рисунков). При этом, чтобы показать последовательность рисования окружностей при рекурсивных вызовах, я дополнительно включаю в алгоритм Планета перед командой «если» пустой цикл, обеспечивающий некоторую паузу перед выводом на экран каждой из окружностей.

2. Алгоритм, с помощью которого можно получить изображение, как на рис. 2

В треугольнике проводятся все средние линии. Тем самым он разбивается на 4 треугольника. К трем из них, примыкающим к вершинам первоначального треугольника, применяется тот же алгоритм.

Рис. 2

```
алг Треугольник(арг цел ха, уа, хв, ув, хс, ус, н)
нач
 цел хр, xq, xr, yp, yq, yr
   если н>0 то
     хр:  =(хв+хс)/2; yp:=(ув+ус)/2              координаты
     xq: =(ха+хс)/2; yq:=(уа+ус)/2               серединсторон треугольника
     xr: =(хв+ ха) /2; yr:=(ув+уа) /2
     поз(хр.yp)
     линия(xq, yq)                               разбиваем треугольник
     линия(xr, yr)
     линия(хр, yp)                               на4части
```

```
   Треугольник(ха, уа, xr, yr, xq, yq, н-1)
   Треугольник(хв, ув, хр, ур, xr, yr, н-1)
   Треугольник(хс, ус, xq, yq, хр, ур, н-1)
 все
кон
```

Основной алгоритм:

```
алг Множество треугольников
нач
цел ха,уа,хв,ув,хс,ус,н
видео(18)
хс:=0; ус:=0                          координаты вершин

хв:=максХ; ув:=максУ                  самого большого
ха:=0, уа:=максУ                      треугольника
поз(ха.уа)

линия(хв.ув)                          рисуем
линия(хс,ус)                          самый большой
линия(ха.уа)                          треугольник

Треугольник(ха,уа,хв,ув,хс,ус,6)
Кон
```

1. Алгоритм, с помощью которого можно получить изображение, как на рис. 3

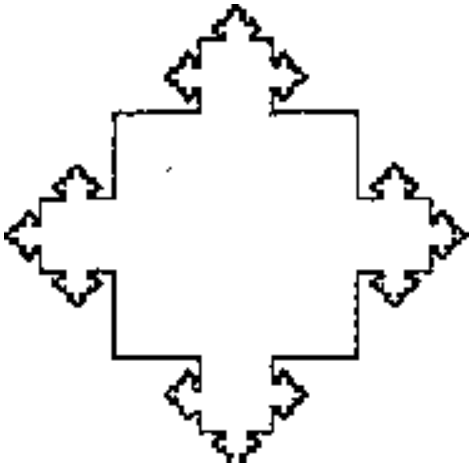


Рис. 3

На рис. 3 на каждой из сторон внутреннего (самого большого) квадрата нарисованы 3 стороны малого квадрата, на каждой из сторон которого также изображены 3 стороны еще меньшего квадрата и т.д. (напомним, что подобные фигуры называются **фракталами).**

Алгоритм, в котором выполняются соответствующие действия на некотором отрезке (с координатами концов ха, уа, xb, yb), может быть оформлен следующим образом:

```
алг Сторона(цел ха,уа,хо,уЬ,н, вещ к)

 нач

 цел   хр.ур.хр, уч.хг, уг,хз,уз,бх,бу

если н=0
   линия(хо,уЬ)
иначе
бх =0.5*(1-k)*(хо-ха)          коэф. уменьшения
бу:=0.5*(1- k)*(уЬ-уа)         размера квадратов
```

```
хр: =ха+бх; ур: =уа+бу       координаты
хз:=хо-бх;уз:=уо-бу
хц: =хр+(уз-ур);уч: =ур-(хз-хр)  изображаемой
хг.=хц+(х^хр);уг:=ур+(уз-ур)     на отрезке аЬ
Сторона(хр,ур,хр,ур, н-1, к )
Сторона(хр,ур,хг,уг,н-1, к )
Сторона(хг,уг,хз,уз,н-1, к )
линия(хо.уЬ)                  к концу отрезка
все

кон
```

Основной алгоритм:

```
алг Картинка
хс:=максХ/2, ус:=максУ/2                  половина длины стороны большого квадрата
в : =100
н:=5; к: =0.4,
поз(хс-в, ус-в)                           точка, с которой начинается  рисунок
Сторона(хс-в, ус-в, хс+в, ус-в, н, к)
Сторона(хс+в, ус-в, хс+в, ус+в, н, к)
Сторона(хс-в,ус+в,хс-в,ус-в, н, к)
кон
```

Косвенная рекурсия

Косвенной, или непрямой, рекурсией называется ситуация, когда алгоритм А вызывает себя в качестве вспомогательного не непосредственно, а через другой вспомогательный алгоритм Б.

Образно косвенную рекурсию можно описать так. Перед зеркалом 1 стоит зеркало 2. Что видно в зеркале 1 ? Зеркало 2, в котором отражается само зеркало 1. В последнем видно зеркало 2 и т.д.

В качестве примера алгоритма с косвенной рекурсией можно привести несколько измененный алгоритм для Планет:

```
алг   Система планет
нач  цел н, нсп, рад, х, у,
        вещ к_орб, к_спут
  видео(18)
  н:=5,  нсп : =4
  х:= максХ/2,  у:= максУ/2;  рад =120
  к_орб:=1.6, к_спут:=0.5
  Планета(х, у, рад, н, нсп, к_орб, к_спут )
 кон

алг Планета(арг  цел х, у, рад, цап н, нсп, вещ к_обр, к_спут)
нач
если н>0 то
            поз(х.у)                                    !центр планеты
            окружность(рад)                             !планета
            Спутники(х, у, рад.н, нсп, к_орб, к_спут)   !всп. алгоритм
 все
кон

алг Спутники(арг цел х, у, рад, н, нсп, вещ к_орб,  к_спут)

нач   цел i, х1, вещ угол, рад_орб
```

```
 рад_орб:=рад*к_орб

угол:=6.28/нсп
нц для i от 1 до нсп                    для каждого спутника
  x1=x+рад_орб * соз(угол*i)
  y1=y+рад_орб * соз(угол* i)
  !рекурсивный вызов алгоритма Планета
  Планета(x1, y1,  int(рад*к_спут), н-1, нсп, к_орб, к_спут)

кц

кон
```

Алгоритм Планета выводит на экран центральную окружность-планету, а для рисования ее спутников использует вспомогательный алгоритм Спутники. Последний рисует спутники с помощью рекурсивного вызова алгоритма Планета. На взгляд автора, в таком виде программа даже более логична и понятна, чем с "обычной" рекурсией.