Рекурсивный вызов подпрограмм

К функции можно обращаться из тела самой этой функции (вызвать саму себя). Данный вид планирования решения задачи или алгоритма программы называется рекурсией.

Большинство современных языков высокого уровня поддерживают рекурсивный алгоритм (recursive algorithm) – способ организации вычислений, при котором функция (подпрограмма) вызывает сама себя с другим аргументом. При этом функция, как элемент структуры языка программирования, возвращающая вычисленное значение по своему имени, содержит внутри себя обращение к самой себе с тем же самым именем, но с другими аргументами. Эта возможность позволяет напрямую реализовывать вычисление рекурсивно определенных объектов. Следует отметить, что любой рекурсивный алгоритм может быть реализован итерационно, причем итерационные решения предпочтительны, итерационные решения работают быстрее рекурсивных.

Числа Фибоначчи. Золотое сечение

Рассмотрим в качестве примера рекурсивного алгоритма программу для вычисления последовательности чисел Фибоначчи. В этой последовательности при $n>1$ каждый следующий элемент есть сумма двух предыдущих:

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \tag{7.1}$$

С помощью рекурсивного алгоритма программа для вычисления чисел Фибоначчи может быть записана в виде:

```
program Fbn;
var N: integer;
function Fib(n:integer):integer;
begin
if (n>1)    then    Fib:=Fib(n −1)+Fib(n − 2);
if (n=0) or (n=1) then Fib:=1;
end;
        BEGIN
        N:=3;
        writeln(' F(',N,')= ',Fib(N));
        END.
```

Чтобы понять, как конкретно работает рекурсивный алгоритм этого примера, удобно ввести в подпрограмму Fib операторы вывода

writeln('in…) на входе в подпрограмму, и writeln('out…) на выходе из неё.

<table>
<tr><th>Код программы</th><th>Результат</th></tr>
<tr><td>
<pre>
var N,F: integer;
function Fib(n:integer):integer;
var y:integer;
begin
writeln('in (N=',n,') ');
if (n>1) then
          y:= Fib(n−1)+Fib(n−2);
if (n=0) or (n=1) then y:=1;
Fib:=y;
writeln('out   ',' F(',n,')= ',y);
end;

BEGIN
N:=3;   F:=Fib(N);
writeln(' F(',N,')= ',F);
END.
</pre>
</td><td>
<pre>
in (N=3)
          in (N=1)
          out F(1) =1
          in (N=2)
                    in (N=0)
                    out F(0) = 1
                    in (N=1)
                    out   F(1) = 1
          out F(2) = 2
out F(3) = 3

           F(3) = 3
</pre>
</td></tr>
</table>

В графе «результат» каждая строчка дополнительно сдвинута вправо, чтобы отобразить глубину рекурсии на каждом этапе.

На первом шаге мы входим в подпрограмму с параметром N=3. Поскольку аргумент подпрограммы Fib(N) больше 1, она вызывает очередного исполнителя для вычисления нужных значений Fib(1), Fib(2). Получается серия вызовов, которые выстраиваются в ***дерево рекурсивных вызовов***. При вычислении значения $F(3)$ будут вызваны процедуры вычисления $F(1)$ и $F(2)$. В свою очередь, для вычисления последнего потребуется вычисление $F(0)$ и $F(1)$. Можно заметить, что $F(1)$ вычисляется два раза.

Если рассмотреть вычисление $F(n)$ при больших n, то повторных вычислений будет очень много. Это и есть основной недостаток рекурсии — повторные вычисления одних и тех же значений. Кроме того, дерево рекурсивных вызовов может оказаться бесконечным и компьютер «зависнет». Для того чтобы рекурсивный алгоритм заканчивал свою работу, необходимо, чтобы дерево рекурсивных вызовов при любых входных данных обрывалось и было конечным. В данном примере дерево рекурсивных вызовов обрывается на $F(1)$ и $F(0)$, при вычислении которых подпрограмма подставляет конкретные значения.

Следует отметить, что числа Фибоначчи можно вычислять и по более эффективному алгоритму (без рекурсии) или просто воспользоваться прямой формулой.

Золотое сечение. Прямоугольник с отношением большей стороны к меньшей равной числу Φ, где Φ = 1.618…, соответствует так называемой «золотой пропорции» или «золотому сечению». Число Φ удовлетворяет уравнению $\Phi(\Phi-1)=1$. Рассмотрим рекурсивную процедуру построения последовательности «прямоугольников золотого сечения», по алгоритму, который очевиден из рисунка 5.1.

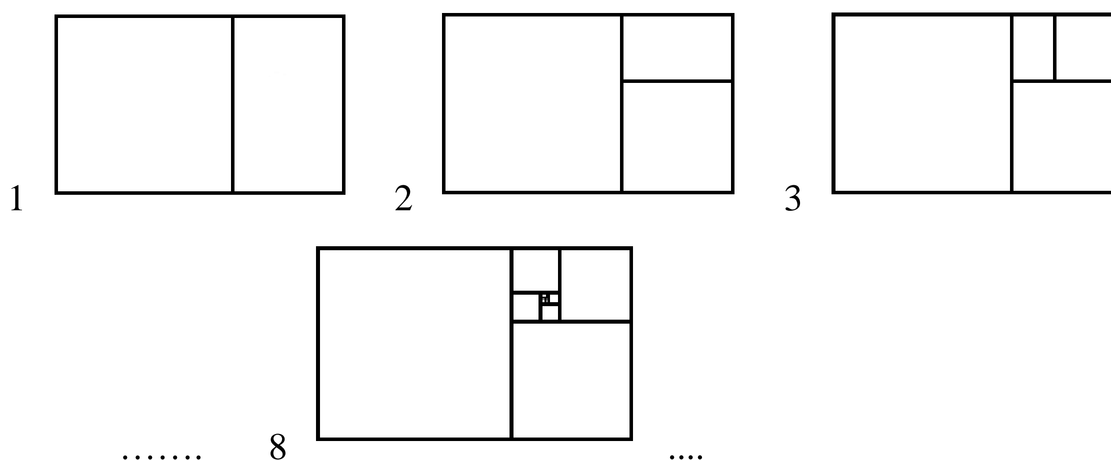


Рис. 5.1. Построение последовательности прямоугольников золотого сечения.

На каждом шаге внутри текущего прямоугольника строится квадрат, при этом оставшаяся часть фигуры будет опять прямоугольником золотого сечения, стороны которого в Φ раз меньше сторон исходного. Обозначения в программе соответствуют рисунку 5.2.

```
const Fi=1.618;              //  параметр золотого сечения
var  x1,y1, A,B: integer;
procedure Gold(x1,y1,nx,ny,N: integer; a:real);
var   tx, ty, x2, y2, x3, y3:integer;
begin
if N<1 then EXIT;
x2:=x1+nx*round(a);          y2:=y1+ny*round(a);
x3:=x1+nx*round(a*Fi);       y3:=y1+ny*round(a*Fi);
tx:=ny; ty:= –nx;  //поворот направляющего вектора на 90 градусов
line(x2, y2,  x2+tx*round(a), y2+ty*round(a));
Gold(x3,y3,tx,ty,N −1, a*(Fi −1));
end;
BEGIN
x1:=50; y1:=200; A:=100; B:=round(Fi*A);
```

```
line(x1,y1,x1+B,y1); line(x1,y1−A,x1+B,y1−A);     //построение
line(x1,y1,x1,y1−A); line(x1+B,y1,x1+B,y1−A);     //прямоугольника
Gold(x1,y1,1,0,8,A);
END.
```

Текущий шаг	Следующий шаг

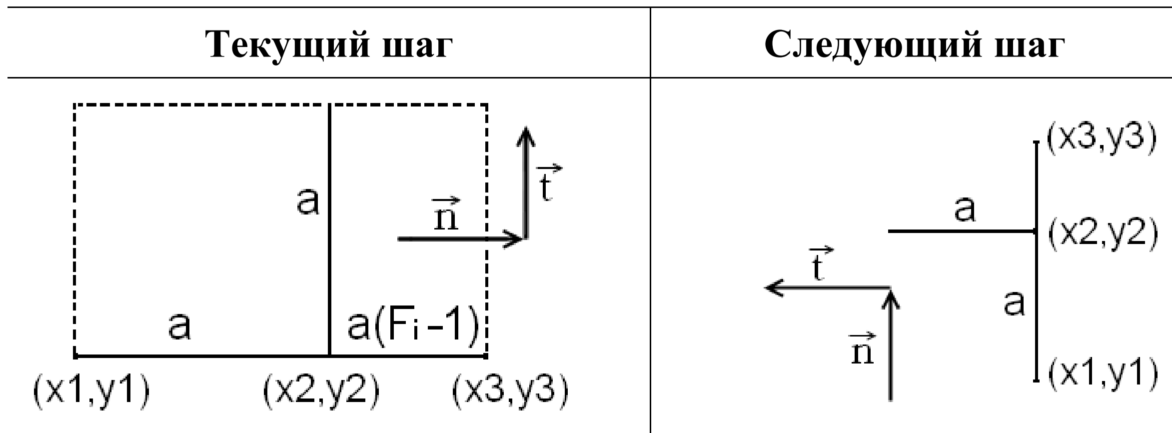


Рис. 5.2. Расшифровка обозначений в программе. Напомним, что ось Y на рисунках направлена вертикально вниз, а не вверх

На каждом шаге подпрограмма Gold строит одну линию, перпендикулярную большей стороне текущего прямоугольника (длиной a·Fi). Координаты двух «нижних» вершин прямоугольника – (x1,y1) и (x3,y3) соответственно. Данная линия разбивает этот прямоугольник на квадрат и уменьшенный «золотой прямоугольник». Координаты начала линии – (x2,y2). Введем два единичных взаимно перпендикулярных вектора $\vec{n}$ и $\vec{t}$, вектор $\vec{n}$ параллелен большей стороне прямоугольника (см. рис 5.2). Тогда конец линии получатся прибавлением вектора ($a\vec{t}$) к её началу. В программе это соответствует строке line(x2,y2, x2+tx*round(a), y2+ty*round(a)).

После выполнения очередного шага происходит рекурсивный вызов подпрограммы Gold с подстановкой x3,y3 вместо x1,y1. и вектора $\vec{t}$ вместо $\vec{n}$. При этом происходит поворот пары векторов $\vec{n}$ и $\vec{t}$ на 90° против часовой стрелки. Например, если на каком-то шаге $\vec{n}$ =(1, 0), $\vec{t}$ =(0, −1), то на следующем шаге $\vec{n}$ =(0, −1) и $\vec{t}$ =(−1, 0) и т.д.

Опережающее описание. Существуют и более сложные схемы рекурсии, например: функция F1 вызывает функцию F2, а та в свою очередь вызывает F1. При этом оказывается, что первая процедура должна вызывать еще не объявленную процедуру. Чтобы это было возможно, требуется использовать опережающее описание.

```
procedure F2(…); {Опережающее описание второй процедуры}
procedure F1(…); {Полное описание процедуры F1}
begin
```

```
…………………
F2(…);
end;
procedure F2(…); {Полное описание процедуры F2}
begin
…………………
F1(…);
end;
```

Опережающее описание процедуры F2 позволяет вызывать ее из процедуры F1.

Построение самоподобных (фрактальных) структур

Отличительной особенностью фракталов является наличие специфических геометрических свойств (самоподобие, самоаффинность и т. д.). Проще говоря, если изменить масштаб изображения фракталов определенное количество раз, новое изображение будет таким же, или статистически эквивалентно исходному, причем новое изображение при повторении этого преобразования ведет себя так же.

На рисунке изображены примеры типичных регулярных фракталов. Фрактал «дерево» получается путем последовательного разветвления всех веток на две новые ветки меньшего размера, начиная с центрального ствола. Треугольник Серпинского получается из равностороннего треугольника, если соединить линиями середины его сторон, а затем повторять этот алгоритм для всех получающихся малых треугольников, и так до бесконечности.

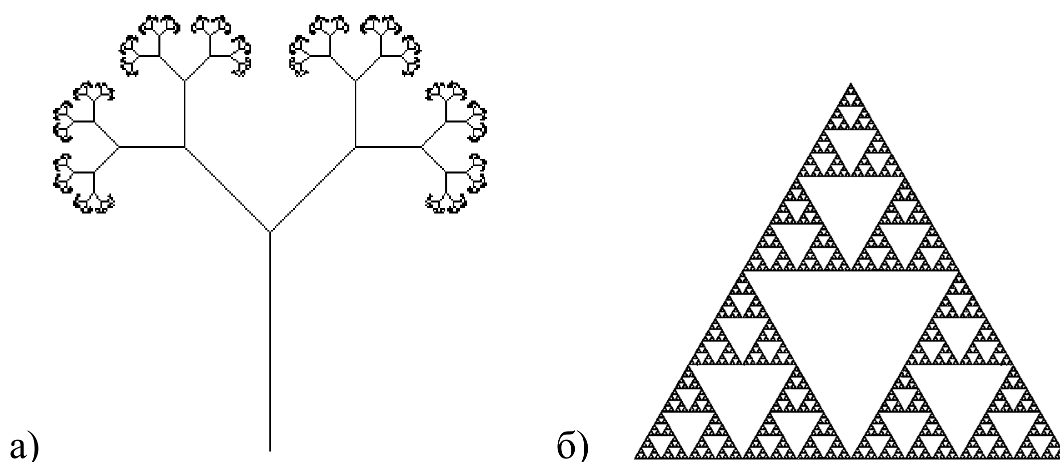


Рис. 5.3. Фракталы: а) бинарное дерево. б) треугольник Серпинского

При этом любая ветка дерева является уменьшенной копией всего дерева, аналогично, любой внутренний треугольник повторяет структуру полного треугольника.

Ниже приводятся коды программ для рисования этих фракталов. Обе программы имеют общую структуру – рекурсивная подпрограмма

рисует базовый геометрический объект по заданным параметрам, а затем обращается сама к себе с измененными значениями этих параметров.

Рекурсивная процедура построения дерева должна рисовать одну линию (ствол до первого разветвления), а затем вызывать сама себя для рисования двух поддеревьев. Поддеревья отличаются от содержащего их дерева координатами начальной точки x1,y1, углами поворота f1, f2, уменьшенной длиной ствола L и количеством содержащихся в них разветвлений n (на одно меньше). Все эти отличия следует сделать параметрами рекурсивной процедуры.

Дерево на рисунке 5.3 получено для следующих значений параметров f1=pi/4; f2=pi/4; q=0.55; и при первом вызове подпрограммы вида Tree(300,400,pi/2,100,12) Если углы поворота f1, f2 разные, дерево будет несимметричным.

Дерево

```
program tree_2;
uses graphabc;
const f1=pi/4; f2=pi/4; q=0.58;
procedure
TREE(x1,y1,F,L:real; n:byte);
var
x2, y2: real;
begin
x2 := x1 + L*cos(F);
y2 := y1 − L*sin(F);
MoveTo(round(x1), round(y1));
LineTo(round(x2), round(y2));
if n > 1 then
begin
TREE(x2, y2, F+f1, q*L, n−1);
TREE(x2, y2, F−f2, q*L, n−1);
end;
end;

BEGIN
Tree(300,400,pi/2,100,12);
END.
```

Треугольник Серпинского

```
Uses graphabc;
Var x1,y1,x2,y2,x3,y3,L:word;
procedure
TR(x1,y1,x2,y2,x3,y3, N:word);
Var x12,y12,x23,y23,x31,y31:word;
Begin
If N=0 then EXIT;
        x12:=(x1+x2) div 2;
        y12:=(y1+y2) div 2;
        x23:=(x2+x3) div 2;
        y23:=(y2+y3) div 2;
        x31:=(x3+x1) div 2;
        y31:=(y3+y1) div 2;
Line(x31,y31,x12,y12);
LineTo(x23,y23);
LineTo(x31,y31);
TR(x1,y1,x12,y12,x31,y31, N-1);
TR(x12,y12,x2,y2,x23,y23, N-1);
TR(x31,y31,x23,y23,x3,y3, N-1);
end;
BEGIN
L:=500;
x1:=100; y1:=450;
x2:=x1+L div 2;
y2:=y1-round(L*sqrt(3)/2);
x3:=x1+L; y3:=y1;
Line(x1,y1,x2,y2);
```

```
LineTo(x3,y3);
LineTo(x1,y1);
TR(x1,y1,x2,y2,x3,y3,8);
END.
```

Разберем более подробно рисование треугольника Серпинского. При вызове процедуры TR(…); в неё передаются координаты вершин треугольника, причем x_1,y_1 соответствуют левой нижней вершине, x_2,y_2 – центральной вершине, x_3,y_3 – правой нижней вершине. В первой части подпрограммы вычисляются координаты вершин центрального перевёрнутого треугольника, которые являются серединами сторон исходного, то есть,

$x_{12}:=(x_1+x_2)/2$

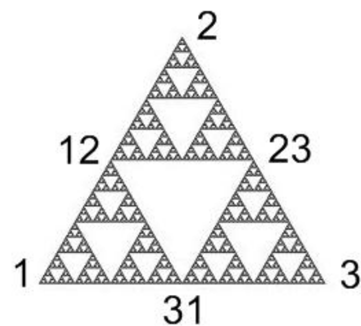
$y_{12}:=(y_1+y_2)/2$

$x_{23}:=(x_2+x_3)/2$

$y_{23}:=(y_2+y_3)/2$

$x_{31}:=(x_3+x_1)/2$

$y_{31}:=(y_3+y_1)/2;$



(в самой программе использован оператор деления целых чисел div).

После этого рисуются стороны этого треугольника. В результате образуются три новых треугольника, подобных исходному треугольнику (с центральной вершиной, направленных вверх).

На втором этапе три раза вызывается эта же процедура TR(...), параметрами которой являются вершины этих новых треугольников, таким образом, новые треугольники аналогичным образом делятся на подобные части и т.д. Очень важно не забыть уменьшить у внутренних вызовов подпрограммы параметр N (число шагов рекурсии) на единицу, чтобы дерево рекурсивных вызовов было конечным. То есть такого рода процедуры должны выглядеть примерно так:

```
procedure TR(…, N: word);
Begin
……………………
TR(..., N-1);
TR(..., N-1);
TR(..., N-1);
end;
```

Задания:

1. Используя программу для построения «золотых прямоугольников», построить «золотую спираль» (см. рис. 5.8). Углы всех квадратов нужно соединить «четвертинками» окружностей, так, чтобы получить непрерывную кривую (не путать её с логарифмической золотой спиралью).

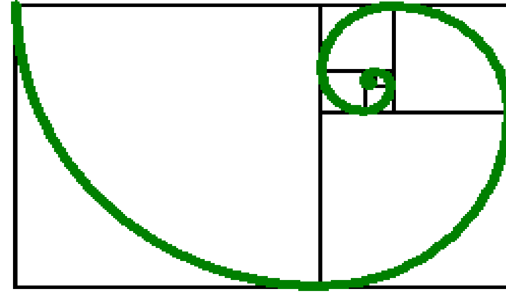


Рис. 5.4. «Золотая спираль»

2. Дополнить программу построения дерева так, чтобы получился «лист папоротника» (см. рис. 5.10).

Указание. Добавить в подпрограмму еще один рекурсивный вызов TREE(x2, y2,…, …, n−1); соответствующий центральной ветке.

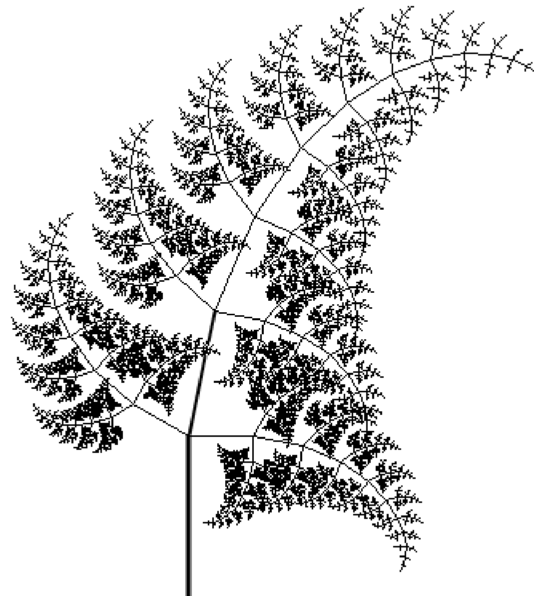


Рис. 5.5. «Лист папоротника»