Занятие 15. Рекурсия

Рекурсивным называют любой объект, частично определяемый через такой же объект. Рекурсивной называется подпрограмма, которая вызывает саму себя. Такая рекурсия называется прямой. Существует ещё косвенная рекурсия, когда две или более подпрограммы вызывают друг друга. Далее мы будем рассматривать только прямую рекурсию.

Классическим примером рекурсивной подпрограммы является вычисление факториала. Факториалом натурального числа n (обозначается $n!$) называется произведение первых n натуральных чисел. Дополнительно полагается, что 0!=1.

$$n! = \begin{cases} 1 \cdot 2 \cdot \ldots \cdot n \text{ при } n > 0 \\ 1 \text{ при } n = 0 \end{cases}$$

Это же определение можно переписать рекурсивно, определив факториал через факториал от меньшего числа

$$n! = \begin{cases} n \cdot (n-1)! \text{ при } n \geq 1 \\ 1 \text{ при } n = 0 \end{cases}$$

Таким образом, чтобы вычислить 3!, надо вычислить 2!, чтобы вычислить 2!, надо вычислить 1!, чтобы вычислить 1!, надо вычислить 0!, а вот 0! вычисляется непосредственно.

Программа вычисления факториала может выглядеть следующим образом:

```
{$mode delphi}
Program Sample1;

//Рекурсивная функция
Function Fact(n : Int64) : Int64;
Begin
  If n=0 //Условие выхода из рекурсии
  Then Fact:=1 //Нерекурсивная ветвь
  Else Fact:=n*Fact(n-1) //Рекурсия
End;

Var n : Int64;
Begin
  Write('n=');
  Read(n);
  WriteLn(n,'!=',Fact(n))
End.
```

Ясно, что если не предусмотреть в подпрограмме условие выхода из рекурсии, т.е. случай, в котором подпрограмма саму себя не вызовет, то процесс вычислений не завершится успехом. При каждом вызове рекурсивной подпрограммы в специальной области памяти, в стеке, выделяется место для хранения локальных переменных и параметров этой подпрограммы[1]. При возврате из подпрограммы память, выделенная для этой копии подпрограммы, освобождается. При отсутствии условия выхода из рекурсии мы получим не зависшую программу, находящуюся в бесконечных вычислениях, а аварийное завершение этой программы из-за переполнения стека. В системах программирования для операционной системы MS-DOS, например Turbo Pascal, размер стека мал. В этом случае можно организовать лишь рекурсивные вычисления с небольшим числом вложенных вызовов подпрограммы, и использования рекурсии избегали. Однако в Delphi и FreePascal размер стека ограничен только размером доступной памяти системы.

Таким образом, для построения рекурсивной подпрограммы надо свести решаемую задачу к самой себе, но при других значениях параметров, а также предусмотреть условие выхода из рекурсии.

[1] А также для хранения адреса возврата из подпрограммы

Покажем это на примерах.

Задача 1. Сумма цифр числа

Найти сумму цифр заданного натурального числа m ($m \le 10^{18}$).

Решение.

Для однозначного m сумма цифр числа равна самому m. Для остальных же m легко свести задачу к вычислению суммы цифр числа, полученного зачёркиванием у исходного последней цифры.

$$summ(m) = \begin{cases} m, & m < 10 \\ summ(m\ div\ 10) + m\ mod\ 10, & m \ge 10 \end{cases}$$

Всё! Более при разработке алгоритма ни о чём думать не нужно. Разбираться, как это будет работать, на первых порах не стоит. Только запутаемся. Пишем программу. Учитываем, что m может быть велико, а вот сумма его цифр не превышает $9 \cdot 18 < 255$.

```
Program SummDigit;

Function summ(m : Int64) : Byte;
Begin
  If m<10 Then summ := m
  Else summ := summ(m div 10) + m mod 10
End;

Var a : Int64;
Begin
  WriteLn('Введите натуральное число ');
  Read(a);
  WriteLn('Сумма цифр числа ',a,' равна ',summ(a))
End.
```

Задача 2. Быстрое возведение в степень

Вычислить x^n, где x — вещественное число, n — целое неотрицательное. Будем считать, что x и n одновременно нулю не равны.

Решение.

Мы уже решали эту задачу с использованием циклического алгоритма (см. занятие 5). Теперь получим рекурсивное решение. Пусть $f(x,n) = x^n$. Тогда

$$f(x,n) = \begin{cases} x \cdot f(x,n-1) \text{при нечётном } n \\ f\left(x^2, \frac{n}{2}\right) \text{ при чётном } n \\ 1 \text{ при } n = 0 \end{cases}$$

В этом рекурсивном определении первая строка применяется для сведения нечётного показателя степени к чётному. Во второй строке сформулирована основная идея сокращения вычислений: вместо возведения x в чётную степень n быстрее возвести x^2 в степень $\frac{n}{2}$. Постепенно уменьшаясь при выполнении первых двух строк, n станет равным нулю. Тогда и произойдёт выход из рекурсии. Пишем программу.

```
Program Power;

Function QPow(x : Extended; n : Integer) : Extended;
Begin
  If odd(n) Then
    QPow := x*QPow(x,n-1)
  Else If n=0 Then
         QPow:=1
```

```
        Else QPow := QPow(x*x,n div 2)
End;

Var x,y : Extended;
    n   : Integer;
Begin
  Write('x=');
  Read(x);
  Write('n=');
  Read(n);
  y:=QPow(x,n);
  WriteLn(x:0:5,'^',n,'=',y:0:5)
End.
```

Вспомните, что функция odd(n) возвращает значение true, если n — нечётное число.

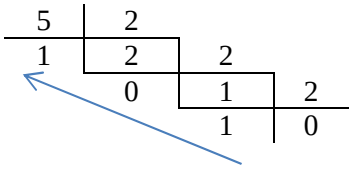
Теперь посмотрим пример рекурсивной процедуры.

Задача 3. Переход к другой системе счисления

Дано натуральное число *n*. Вывести на экран его запись в двоичной системе счисления.

Решение.

Вспомним алгоритм перевода записи натурального числа в двоичную систему счисления на примере числа 5_{10}. Будем делить это число и все получаемые частные на 2_{10}, пока не получим в частном ноль. Потом все остатки запишем в обратном порядке.



$5_{10}=101_2$

Можно заметить, что для того, чтобы перевести в двоичную систему число 5_{10}, мы должны перевести в двоичную систему число 5 div 2 и приписать в конец число 5 mod 2. Это даёт возможность построить рекурсивный алгоритм. Условие выхода из рекурсии — если число меньше двух, то его запись в двоичной системе счисления будет иметь тот же вид, что и в десятичной.

```
{$mode delphi}
Program Notation;
Const BASE=2; //Основание системы счисления

Procedure Numeration(n : Int64);
Begin
  If n >=BASE Then
    Numeration(n div BASE);
  Write(n mod BASE)
End;

Var n : Int64;
Begin
  Write('Введите натуральное число ');
  Read(n);
  Numeration(n);
  WriteLn //Чтобы опустить курсор на новую строку
End.
```

Основание системы счисления в этой программе задано в виде константы BASE. Её можно сделать равной любому значению $2 \leq p \leq 9$, и программа будет давать запись числа в -ичной системе счисления.

Задача 4. Ханойские башни

Даны три стержня: A, B и C. На стержень A нанизаны N колец, причем кольца отличаются размером и лежат меньшее на большем. Задача состоит в том, чтобы перенести пирамиду из N колец за наименьшее число ходов на стержень B. За один раз разрешается переносить только одно кольцо со стержня на другой стержень, причём нельзя класть большее кольцо на меньшее.

Решение

Если $N = 1$, то решение задачи очевидно: надо просто перенести один стержень. В противном случае переносим пирамиду из верхних $N - 1$ кольца со стержня A на промежуточный стержень C, нижнее кольцо переносим со стержня A на стержень B, а потом со стержня C переносим всю находящуюся там пирамиду на стержень B. Этими рассуждениями мы свели задачу переноса пирамиды из N колец к задачам переноса пирамиды из $N - 1$ кольца.

При написании рекурсивной подпрограммы учтём, что количество ходов может быть большим и на экране не обозримым. Поэтому выводить будем в файл. Перенос кольца со стержня A на стержень B будем обозначать $A \to B$.

```
{$mode delphi}
//Ханойские башни
Program Hanoi;
Var f : Text;

Procedure Move(N : Byte; otkuda, kuda, cherez : char);
Begin
  If N=1 Then
    WriteLn(f,otkuda,'-->',kuda)
  Else Begin
         Move(N-1,otkuda,cherez,kuda);
         WriteLn(f,otkuda,'-->',kuda);
         Move(N-1,cherez,kuda,otkuda)
       End
End;

Var n : Byte;
Begin
  Assign(f,'Hanoi.in');
  Reset(f);
  Read(f,n);
  Close(f);

  Assign(f,'Hanoi.out');
  Rewrite(f);
  Move(n,'A','B','C');
  Close(f)
End.
```

Количество колец вводится из файла Hanoi.in. Последовательность ходов выводится в файл Hanoi.out. В этой программе файловая переменная объявлена глобальной во избежание ненужного расходования памяти при передаче дополнительного параметра. Если вы захотите передавать файловую переменную в качестве параметра процедуры, не забудьте объявить этот параметр с модификатором var, то есть как параметр-переменную.

Задача 5. Самый длинный связный кусок

Пусть дана двоичная матрица, т.е. матрица, элементы которой равны 0 или 1. Связным куском этой матрицы назовём такое подмножество её элементов, равных 1, в котором от одного элемента до другого можно перейти, перемещаясь по матрице на каждом шаге на один элемент вверх, вниз, влево или вправо. Количество элементов в связном куске назовём его длиной. (На примере в таблице 1 связные куски выделены различными цветами. Их длины равны 1, 4, 5, 5.)

В заданной двоичной матрице a найти длину наибольшего связного куска.

Таблица 1

1	1	0	1	0
0	1	0	0	1
1	1	0	1	1
0	0	1	0	1
1	1	1	0	1

Формат входных данных. В первой строке входного файла input.txt содержатся числа M и N. Следующие M строк по N чисел 0 или 1 содержат описания строк матрицы. Числа M и N не превосходят 30.

Формат выходных данных. В файл output.txt вывести единственное число — длину самого длинного связного куска.

Пример. Для входного файла, соответствующего таблице 1, ответом будет число 5 (длина жёлтого или зелёного связных кусков).

Решение.

Пусть у нас есть функция Count(row,col), вычисляющая длину связного куска, содержащего элемент матрицы, находящийся в строке row и колонке col. Если указанный элемент равен нулю, функция возвращает ноль. Тогда задача решается очень просто (см. рис. 1). Достаточно для каждого элемента матрицы вычислить длину связного куска, в который входит этот элемент, и найти максимальную из вычисленных величин.

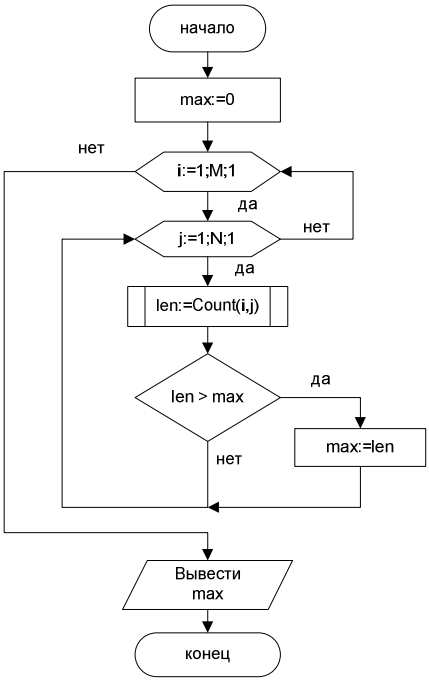


Рисунок 1

Приступим к разработке функции Count.

Длина связного куска, содержащего элемент $a[i,j]$, равна нулю, если $a[i,j]=0$. В противном случае длина связного куска равна 1 + длина связного куска сверху + длина связного куска снизу + длина связного куска слева + длина связного куска справа. Чтобы разделить один связный кусок на элемент $a[i,j]$ и четыре соседних связных куска (возможно, нулевой длины), перед вычислением длин соседей выполним присваивание $a[i,j]:=0$.

Таким образом, функция Count получается рекурсивной.

Чтобы каждый элемент матрицы a, равный 1, имел соседей со всех сторон, окружим исходную матрицу «забором» из нулей (см. таблицу 2). «Забор» показан серым цветом.

Таблица 2

0	0	0	0	0	0	0
0	1	1	0	1	0	0
0	0	1	0	0	1	0
0	1	1	0	1	1	0
0	0	0	1	0	1	0
0	1	1	1	0	1	0
0	0	0	0	0	0	0

Наличие «забора» освобождает от необходимости проверки выхода за пределы массива при обработке матрицы a.

Пишем программу.

```
{$mode delphi}
Program Kusok;
Const MAXSIZE=30;

Type TIndex=0..MAXSIZE+1;
     TMatrix = array[TIndex,TIndex] of 0..1;

Var a : TMatrix;  //Глобальная матрица

//Вычисление длины связного куска, содержащего a[row,col]
Function Count(row,col : TIndex) : Integer;
Begin
  If a[row,col]=0 Then
    Count:=0
  Else Begin
         a[row,col]:=0; //Разрываем связный кусок
```

```
          Count:=1+Count(row-1,col)+Count(row+1,col)+
                  Count(row,col-1)+Count(row,col+1)
           //А может, здесь сделать a[row,col]:=1 ?
        End
End;

//Ввод данных
Procedure Input(out a : TMatrix; out M,N : TIndex);
Var f   : Text;
    i,j : TIndex;
Begin
  Assign(f,'input.txt');
  Reset(f);
  Read(f,M,N);

  For i:=1 To M Do
  For j:=1 To N Do
   Read(f,a[i,j]);
  Close(f);

  //Обносим матрицу "забором" сверху и снизу
  For i:=0 To N+1 Do
  Begin
    a[0,i]:=0;
    a[M+1,i]:=0
  End;
  //Теперь слева и справа
  For i:=1 To M Do
  Begin
    a[i,0]:=0;
    a[i,N+1]:=0
  End
End;

//Основная программа
Var max,len : Integer;
    i,j     : TIndex;
    M,N     : TIndex;
    f       : Text;
Begin
  Input(a,M,N);

  max:=0;
  For i:=1 To M Do
  For j:=1 To N Do
  Begin
    len:=Count(i,j);
    If len > max Then
      max:=len
  End;

  Assign(f,'output.txt');
  Rewrite(f);
  Write(f,max);
  Close(f)
End.
```

Может быть, вам показалось, что перед выходом из рекурсивной функции стоило восстановить разорванный связный кусок? Это можно сделать, но вот нужно ли? Если кусок не восстанавливать, то он исчезает, заполняясь нулями. Поэтому при повторных вычислениях длины этого куска, инициируемых вызовом len:=Count(i,j) при новых значениях i и j, сразу же возвращает нуль. Но нас это значение уже не интересует. Длину этого куска мы уже учли. Если бы кусок был восстановлен, вычисление его длины пришлось бы выполнять сначала. А это потеря времени.

По этой же причине, ради экономии времени, матрица обносилась забором, а не заполнялась нулями полностью перед вводом данных. Хотя для маленьких матриц это и не существенно.

Задачи для самостоятельного решения

1. Вычислите количество цифр натурального числа N, не превышающего двух миллиардов.
2. Дано натуральное M. Выведите на экран его цифры в обратном порядке. Например, для M=12345 выведите 54321. ($M \leq 10^{15}$)
3. Дана символьная строка, представляющая собой запись натурального числа в p-ичной системе счисления ($2 \leq p \leq 9$). Составьте программу перевода этого числа в десятичную систему счисления. Длина строки не превышает 18 цифр.
4. Описать рекурсивную логическую функцию *Simm(s,i,j)*, проверяющую, является ли симметричной часть строки *s*, начинающаяся *i*-м и заканчивающаяся *j*-м её элементами. Используя эту функцию, определить, является ли симметричной введённая строка.

Литература

1. Павловская Т.А. C/C++. Программирование на языке высокого уровня — СПб.: Питер, 2006.
2. Информатика. Задачник-практикум в 2 т. / Под ред. И.Г. Семакина, Е.К. Хеннера: Том 1. — М.: Лаборатория Базовых Знаний, 1999 г.
3. Ханойская башня. // http://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BD%D0%BE%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%B1%D0%B0%D1%88%D0%BD%D1%8F