Рекурсивное программирование

Recursio (лат.) - возврвщение

Рекурсия – это метод, сводящий общую задачу к некоторым задачам более узкого, простого типа

Рекурсивное программирование

Рекурсивный алгоритм – это алгоритм, который в процессе своей работы обращается сам к себе.

Рекурсивное программирование

Суть заключается в том, что при каждом вызове создается новая копия со своими переменными, но как только она заканчивает свою работу, то память, занятая этими локальными переменными, освобождается, а полученные результаты передаются в точку вызова.

Рекурсия … Ее можно:

- **послушать**… «*У попа была собака,*

  Он ее любил. …
- **посмотреть**… встаньте между двумя зеркалами и смотрите…*(не детали гардероба, нет. Отражение себя, там …)*
- **нарисовать**… нарисуйте себя, где на полотне вы рисуете себя, где на полотне…

Это примеры «бесконечной» рекурсии…

Пример: Задача вычисления факториала натурального числа с рекурсивной формулировкой

$$N! = \begin{cases} 1, \text{ при } N = 1 \\ N * (N - 1)!, \text{ при } N > 1 \end{cases}$$

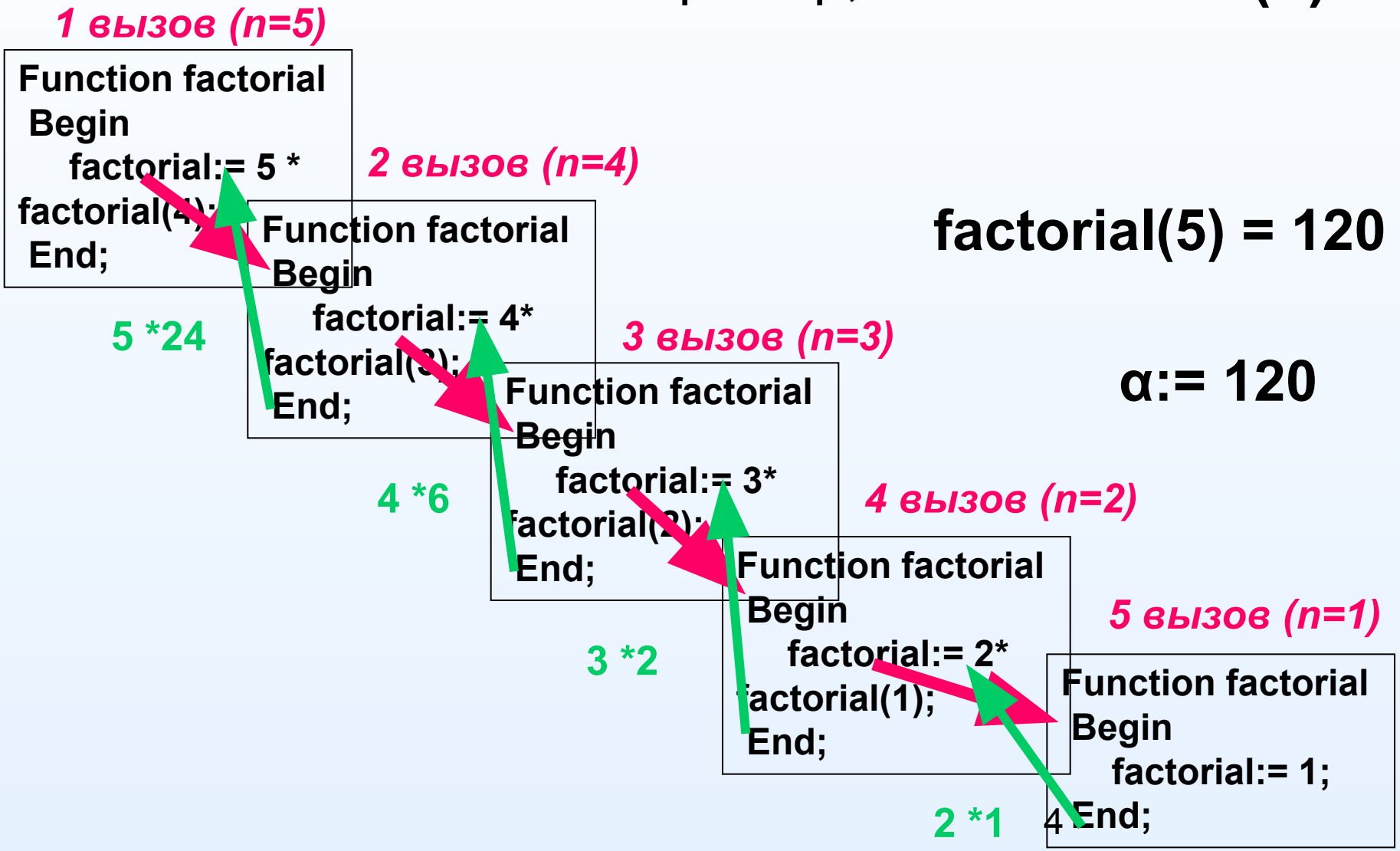
При использовании рекурсии необходимо обеспечить выход из подпрограммы в нужный момент, т.к. ***алгоритм должен быть конечным*** (одно из свойств).

```
Function factorial(n: integer): longint;
 Begin
   If n = 1 then factorial:=1
   else factorial:= n * factorial (n – 1);
 End;
```

Найдем 5!

Первый вызов этой функции будет из основной программы.

Например, **α:= factorial(5)**



factorial(5) = 120

α:= 120

Задание

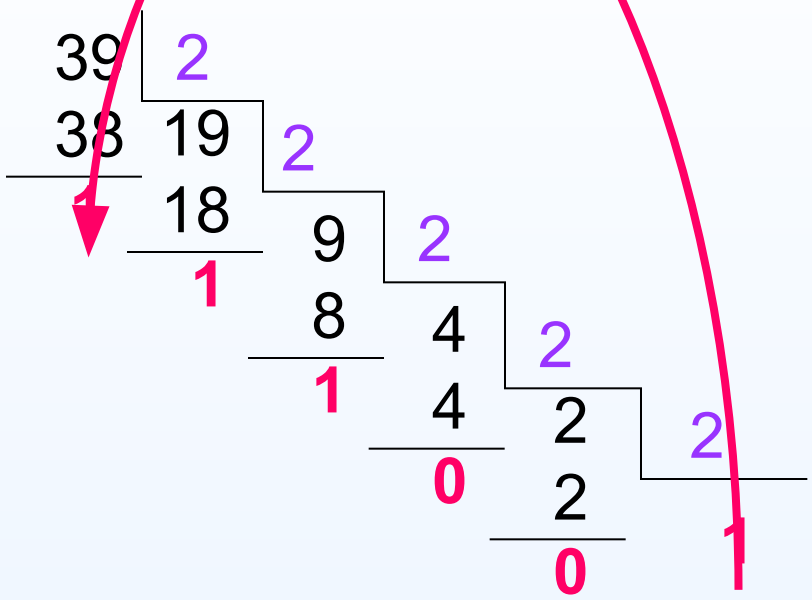
1. Напишите рекурсивную программу определения суммы первых ***n*** натуральных чисел:

   $S_n = S_{n-1} + n;$

   $S_1 = 1.$

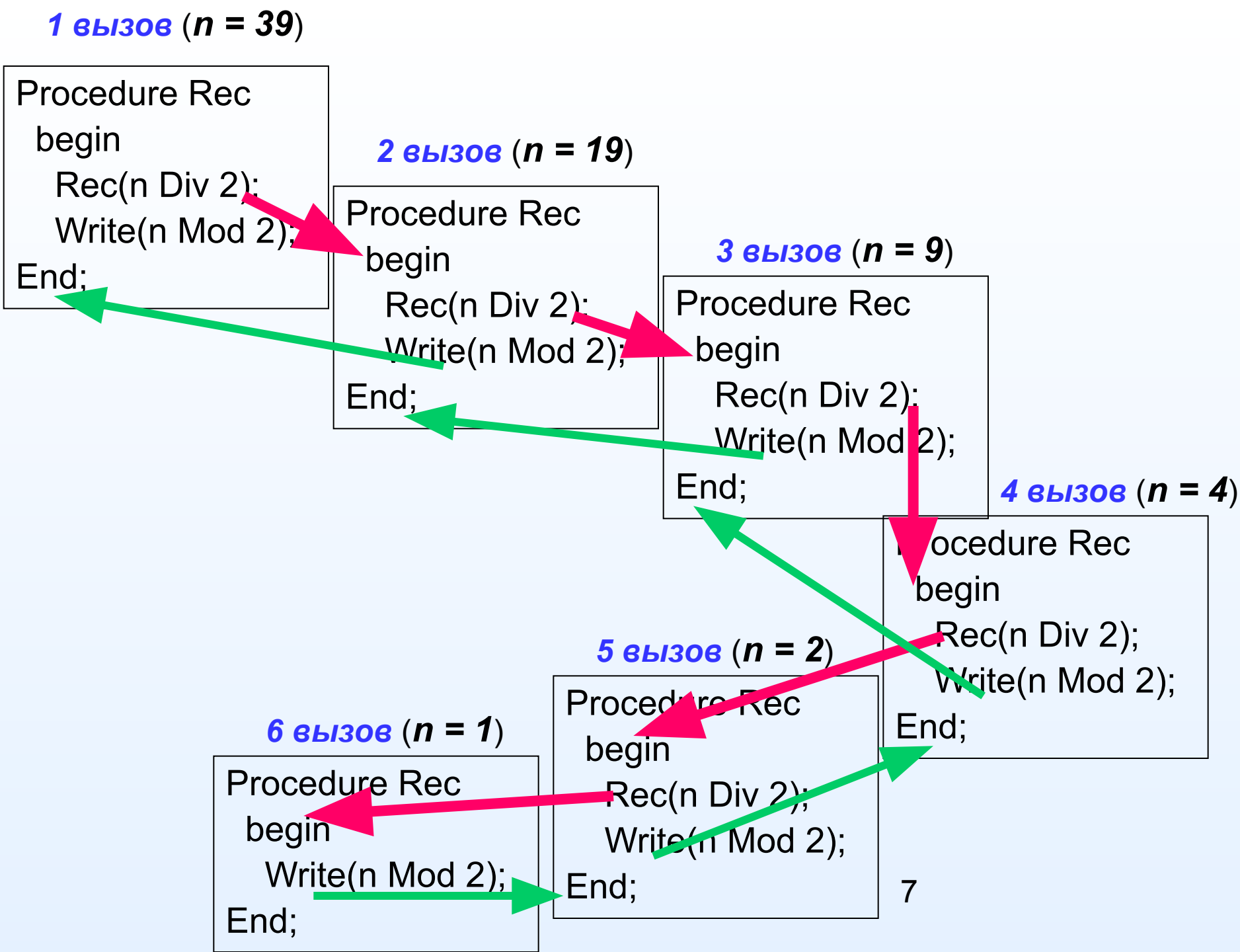
2. Составить рекурсивную программу ввода с клавиатуры последовательность чисел (окончание ввода - 0) и вывода ее на экран в обратном порядке.

Пример: *перевод натурального числа из десятичной системы счисления в двоичную.*



$39_{10} = 100111_2$

```
Procedure Rec(n: integer);
  begin
     If n > 1 then Rec(n Div 2);
     Write(n Mod 2);
End;
```


1 вызов (n = 39)
Procedure Rec
begin
Rec(n Div 2);
Write(n Mod 2);
End;
2 вызов (n = 19)
Procedure Rec
begin
Rec(n Div 2);
Write(n Mod 2);
End;
3 вызов (n = 9)
Procedure Rec
begin
Rec(n Div 2);
Write(n Mod 2);
End;
4 вызов (n = 4)
begin
Rec(n Div 2);
Write(n Mod 2);
End;
5 вызов (n = 2)
begin
Rec(n Div 2);
Write(n Mod 2);
End;
6 вызов (n = 1)
Procedure Rec
begin
Write(n Mod 2);
End;

Задание

1. Написать процедуру перевода числа из десятичной системы в ***N*** - ю, при условии, что $2 \le N \ge 16$ и его значение вводится с клавиатуры.

Каким будет условие завершения входа в рекурсию?

Напишите программу:

Найти первые N чисел Фибоначчи. Каждое число равно сумме двух предыдущих чисел при условии, что первые два равны 1 (1, 1, 2, 3, 5, 8, 13, 21, …).

Рекурсивная постановка данной задачи:

$$\Phi(n) = \begin{cases} 1, \text{ если } n = 1 \text{ или } n = 2; \\ \Phi(n-1) + \Phi(n-2), \text{ при } n > 2 \end{cases}$$

```
Program chisla_Fibonachi;
var i,n:integer;
  function fib(nf:integer):longint;
    begin
    if (nf=1) or (nf=2) then fib:=1
    else fib:=fib(nf-1)+fib(nf-2);
    end;
  begin
  readln(n);
  for i:=1 to n do
     writeln(fib(i));
  end.
```

Данная программа раскрывает ***недостатки рекурсии***:

с увеличением ***n*** глубина рекурсии многократно (*во сколько?)* увеличивается и выполняемая программа потребует много памяти, что может привести к переполнению ***стека***.

Если при решении есть выбор между итерацией и рекурсией, то предпочтительным выбором является ***итерация***.

Итерация

- ***повторное выполнение некоторых действий до тех пор, пока не будет удовлетворяться некоторое условие.***

Большинство алгоритмов можно реализовать двумя способами:

Итерацией:

```
function fibon(nf:integer):longint;
begin
   f1:=1; f2:=1;
   for i:=3 to n do
      begin f:=f1+f2;
               f1:=f2; f2:=f;
      end;
  fibon:=f;
end;
```

Рекурсией:

```
function fib(nf:integer):longint;
   begin
   if (nf=1) or (nf=2) then fib:=1
   else fib:=fib(nf-1)+fib(nf-2);
   end;
```

Выводит цифры целого положительного числа в обратном порядке

```
Program Perevertish;
Var n: longint;
Procedure reverse(n: longint);
  begin
    write (n mod 10);
    if  n div 10 <> 0 then reverse (n div 10)
  end;
Begin
  writeln('Введите натуральное число <=', 2 147 483 647);
  readln(n);
  reverse(n);
  writeln;  readln
End.
```

Измените процедуру. Сформируйте число-«перевертыш». Выдайте сообщение, является ли введенное число палиндромом

Число-полиндром

- число, которое имеет тот же вид при прочтении его справа налево. Например: 121, 1230321, 99 и т.п.

Решение задач

1. Даны первый член и разность арифметической прогрессии. Написать рекурсивную функцию для нахождения:

   а) n-го члена прогрессии;

   б) суммы n первых членов арифметической прогрессии.

2. Даны первый член и знаменатель геометрической прогрессии. Написать рекурсивную функцию для нахождения:

   а) ее n-го члена;

   б) суммы n первых членов прогрессии.

3. Написать рекурсивную функцию для вычисления:

   а) суммы цифр натурального числа;

   б) количества цифр натурального числа.

4. Написать рекурсивную процедуру для вывода на экран цифр натурального числа в обратном порядке.

5. Написать рекурсивную функцию, определяющую является ли заданное натуральное число простым.

```
Procedure Picture1(x,y,r,r1,n:integer);
Var x1,y1:integer; i:integer;
Begin
  if n > 0 then {“заглушка”}
    begin
      circle(x,y,r);r1:=trunc(r*k2); {рисование окружности}
      r1:=trunc(r*k2)          {вычисление радиуса орбиты}
      For i:=1 to 4 do
          begin
            x1:=trunc(x+r1*cos(pi/2*i);   { координаты центра }
            y1:=trunc(y+r1*sin(pi/2*i);    { i-ой окружности }
            Picture1(x1,y1,trunc(r*k1),r1,n-1);
          end;
    end;
 end;
```

```
Uses Graph;
Var x,y,n,r,r1,cd,gm:integer; k1,k2:real;
Procedure Picture1(x,y,r,r1,n:integer);
…
End;
Begin
  Writeln('Введите количество уровней n'); Readln(n);
  x:=600 Div 2; y:=400 Div 2;
  Writeln('Введите радиус первой окружности r');
readln(r);
  k1:=0.3; k2:=2;
  Cd:=detect; gm:=1;
Initgraph(cd,gm,'c:\tp7\bin');
Picture1(x,y,r,r1,n);
Readln;
Closegraph;
End.
```