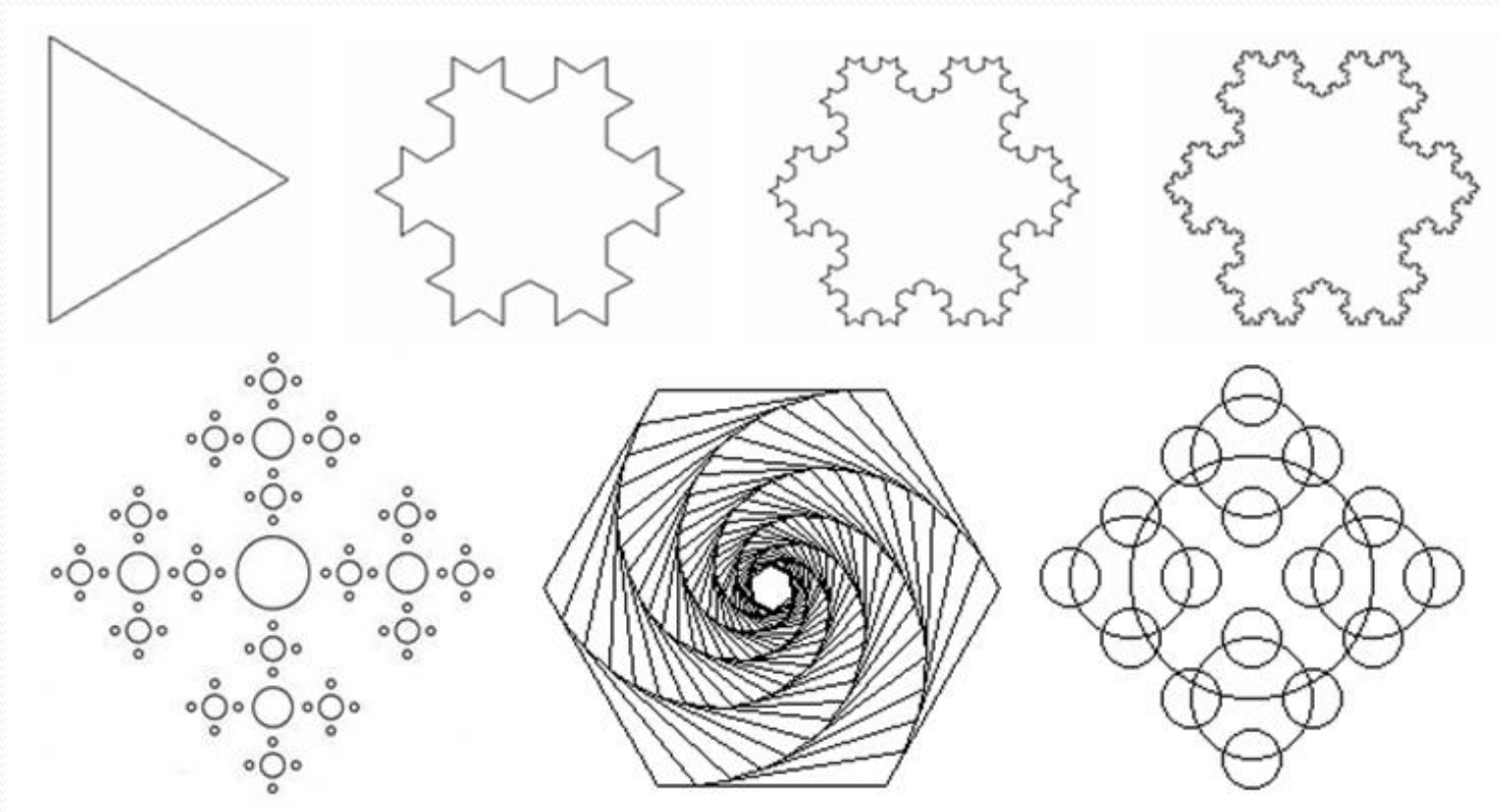


Рекурсия и рекурсивные алгоритмы

В окружающем нас мире часто можно встретить объекты, обладающие **самоподобием**. То есть часть большого объекта в чем-то сходна с самим объектом.



Общий случай проявления рекурсивности может быть сформулирован как **наличие циклических взаимных обращений в определении объекта**, которые в итоге замыкаются на сам объект.

Под рекурсией понимают **прием последовательного сведения решения некоторой задачи к решению совокупности "более простых" задач такого же класса и получению на этой основе решения исходной задачи.**

Рекурсия в широком смысле – это определение объекта посредством ссылки на себя.

Рекурсия - свойство алгоритмической системы на промежуточных этапах своего функционирования создавать другие системы, включая идентичные себе самой, и использовать результаты их функционирования в дальнейшей работе. При достаточно широкой трактовке понятия алгоритмической системы концепция рекурсивности отражает основные формы развития материи и является одним из важнейших методов познания.

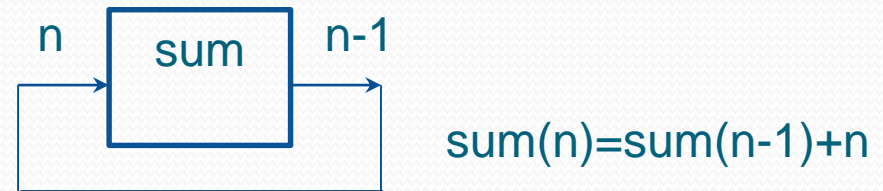
Построим последовательность чисел следующим образом: возьмем целое число $i > 1$. Следующий член последовательности равен $i/2$, если i четное, и $3i+1$, если i нечетное. Если $i=1$, то последовательность останавливается.

- Program Arsac;
 Var first: word;
 Procedure posledov (i: word);
 Begin
 Writeln (i);
 If i=1 then exit;
 If odd(i) then posledov(3*i+1) else posledov(i div 2);
 End;
 Begin
 Write (' введите первое значение '); readln (first);
 Posledov (first);
 Readln ;
 End.

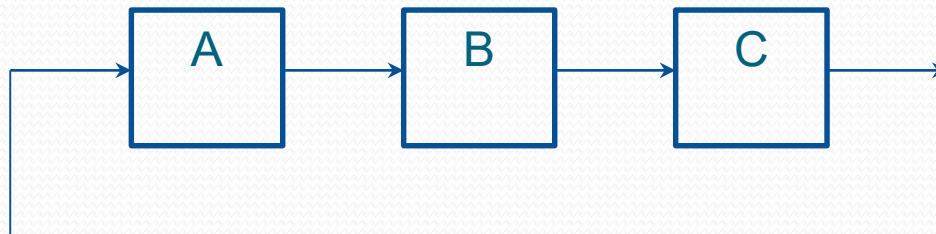
Рекурсия в программировании – это пошаговое разбиение задачи на подзадачи, подобные исходной.

Рекурсивный алгоритм – это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма

Прямая рекурсия - обращение функции к самой себе предполагает, что в теле функции содержится вызов этой же функции, но с другим набором фактических параметров



Косвенная рекурсия - функция содержит вызовы других функций из своего тела. При этом одна или несколько из вызываемых функций на определенном этапе обращаются к исходной функции с измененным набором входных параметров.



Типы рекурсии

- *Линейная рекурсия* - Если исполнение подпрограммы приводит только к одному вызову этой же самой подпрограммы, то такая рекурсия называется *линейной*.
- *Ветвящаяся рекурсия* - Если каждый экземпляр подпрограммы может вызвать себя несколько раз, то рекурсия называется *нелинейной* или *"ветвящейся"*.



```
procedure rec ( a:byte);  
begin  
  If a > 0 then rec(a-1);  
end;  
begin  
  rec(3);  
end.
```



```
function rec( a:byte):integer;  
begin  
  if a > 3 then  
    rec = rec (a-1) * rec(a-2);  
end;  
begin  
  writeln(rec(6) );  
end;
```

Сущность рекурсии

Пример рекурсивной процедуры:

```
procedure Rec(a: integer);
```

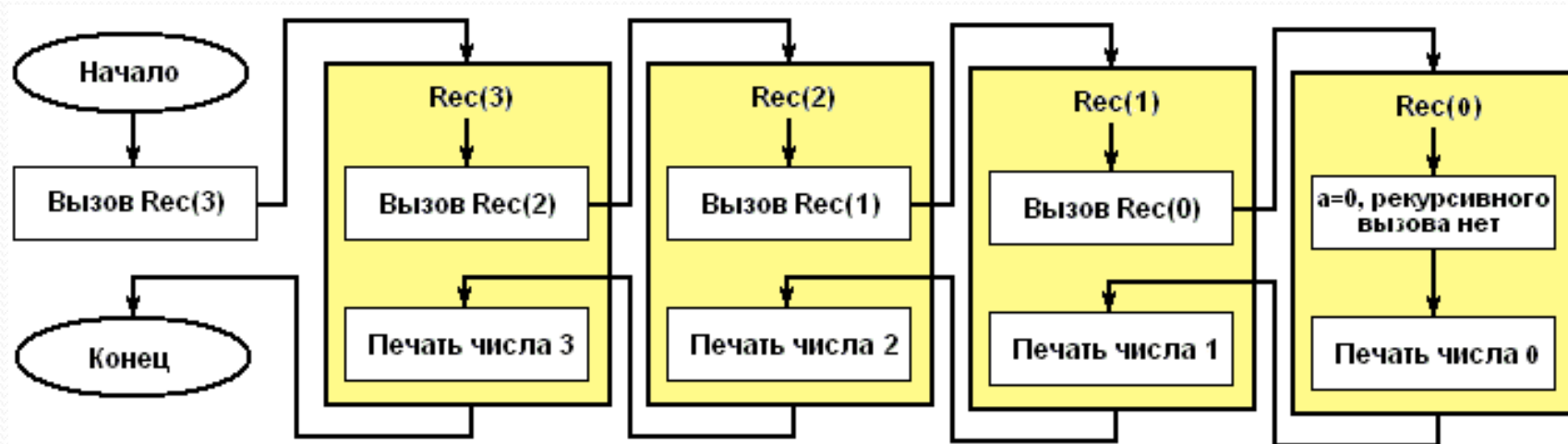
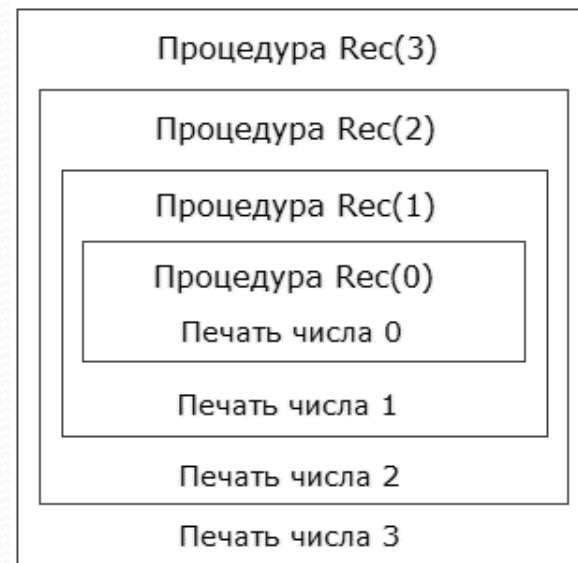
```
begin
```

```
  if a>0 then
```

```
    Rec(a-1);
```

```
    writeln(a);
```

```
end;
```



Количество одновременно выполняемых процедур называют **глубиной рекурсии**

Функция называется рекурсивной, если в своем теле она содержит обращение к самой себе с измененным набором параметров. При этом количество обращений конечно, так как в итоге решение сводится к базовому случаю, когда ответ очевиден.

Пример 1.

В арифметической прогрессии найдите a_n , если известны $a_1 = -2.5$, $d=0.4$, не используя формулу n -го члена прогрессии.

По определению арифметической прогрессии, $a_n = a_{n-1} + d$, при этом $a_{n-1} = a_{n-2} + d$, $a_{n-2} = a_{n-3} + d, \dots a_2 = a_1 + d$.

Таким образом, нахождение a_n для номера n сводится к решению аналогичной задачи, но только для номера $n-1$, что в свою очередь сводится к решению для номера $n-2$, и так далее, пока не будет достигнут номер **1** (значение a_1 дано по условию задачи).

Function arifm (n:integer,a ,d: real): real;

begin

if $n < 1$ then exit; // для неположительных номеров

if $n = 1$ then arifm:=a // базовый случай: $n=1$

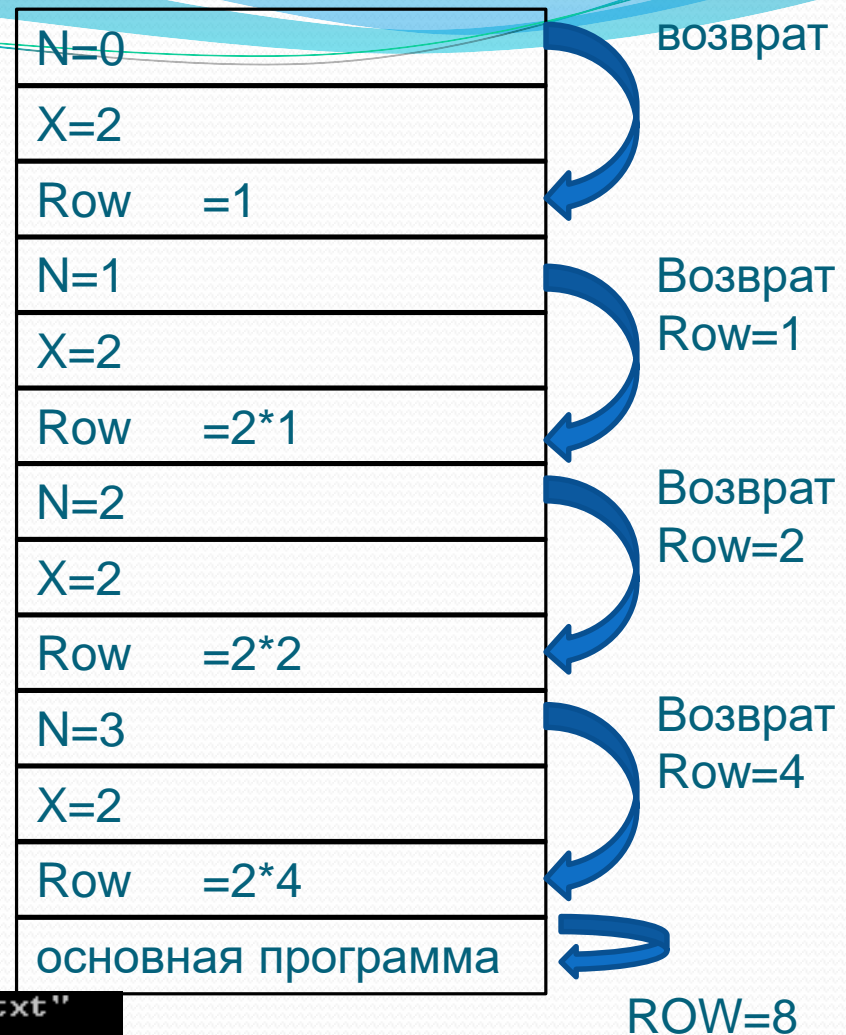
else arifm:=arifm(n-1,a,d)+d; // общий случай

end;

Рекурсия изнутри

```
program Rec_2_2;  
// целая степень целого положительного  
  числа  
var i: integer;  
function Pow (x: byte; n: integer):integer;  
begin  
  writeln('прямой ход', ' x=',x,' n=',n);  
  if n=0 then  
  begin  
    writeln('завершение рекурс. вызова');  
    Pow:=1  
  end  
  else  
    Pow:=x*Pow(x,n-1);  
  writeln('обратный ход', ' pow=',pow,' n=',n);  
end;  
begin  
  writeln('Pow(2,3)=' ,Pow(2,3))  
end
```

```
Running "h:\fpc\2.6.0\bin\i386-win32\кис.exe 1.txt"  
Pow(2,3)=прямой ход x=2 n=3  
прямой ход x=2 n=2  
прямой ход x=2 n=1  
прямой ход x=2 n=0  
завершение рекурсивного вызова  
обратный ход pow=1 n=0  
обратный ход pow=2 n=1  
обратный ход pow=4 n=2  
обратный ход pow=8 n=3  
8
```



Область памяти, предназначенная для хранения всех промежуточных значений локальных переменных при каждом следующем рекурсивном обращении, образует **рекурсивный стек**.

- Для каждого текущего обращения формируется локальный слой данных стека (при этом совпадающие идентификаторы разных слоев стека независимы друг от друга и не отождествляются).
- Завершение вычислений происходит посредством восстановления значений данных каждого слоя в порядке, обратном рекурсивным обращениям.

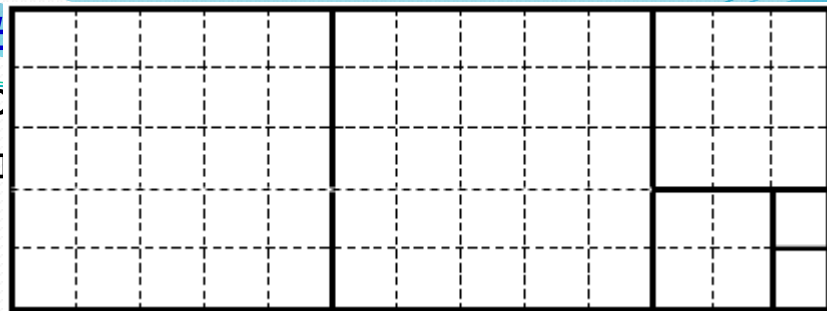
Количество рекурсивных обращений ограничено размером области памяти, выделяемой под программный код.

Для решения задач рекурсивными методами разрабатывают следующие этапы, образующие **рекурсивную триаду**:

- ❑ **параметризация** – выделяют параметры, которые используются для описания условия задачи, а затем в решении;
- ❑ **база рекурсии** – определяют тривиальный случай, при котором решение очевидно, то есть не требуется обращение функции к себе;
- ❑ **декомпозиция** – выражают общий случай через более простые подзадачи с измененными параметрами.

Задача о разрезании прямоугольника

Дан прямоугольник, стороны которого выражены натуральными числами m и n . Разрежьте его на минимальное число квадратов. Найдите число получившихся квадратов.



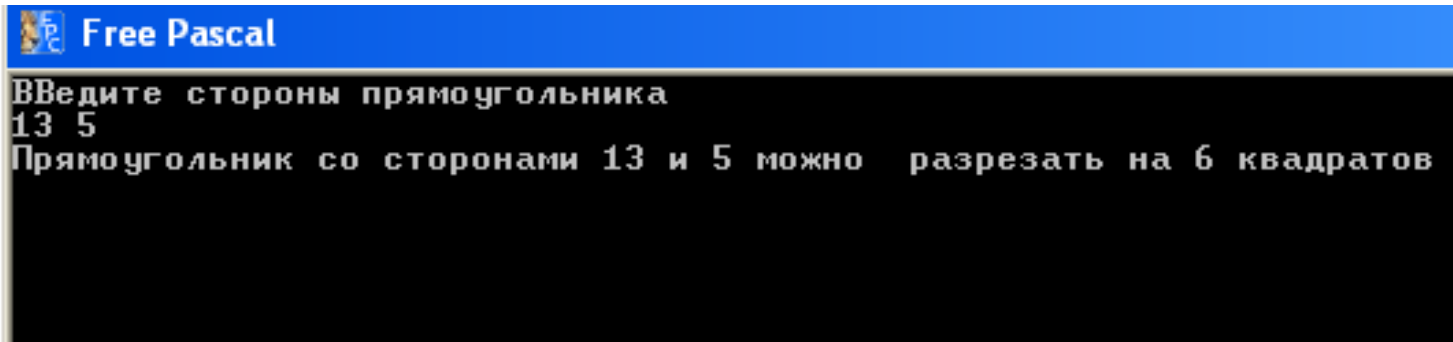
Параметризация: m , n – натуральные числа, соответствующие размерам прямоугольника.

База рекурсии: для $m=n$ число получившихся квадратов равно 1, так как данный прямоугольник уже является квадратом.

Декомпозиция: если $m \neq n$, то отрезем от прямоугольника наибольший по площади квадрат с натуральными сторонами. Длина стороны такого квадрата равна наименьшей из сторон прямоугольника. После того, как квадрат будет отрезан, размеры прямоугольника станут следующие: большая сторона уменьшится на длину стороны квадрата, а меньшая не изменится. Число искомых квадратов будет вычисляться как число квадратов, на которые будет разрезан полученный прямоугольник, плюс один (отрезанный квадрат). К получившемуся прямоугольнику применим аналогичные рассуждения: проверим на соответствие базе или перейдем к декомпозиции.

```
Var a,b,k:byte;
Function kv(m,n:byte):byte;
begin
  if m=n then kv:=1 else //база рекурсии
  if m>n then kv:=1+kv(m-n,n) //декомпозиция для m>n
  else kv:= 1+kv(m,n-m); //декомпозиция для m<n
end;

begin
writeln('Введите стороны прямоугольника');
  readln(a,b);
  k = kv(a,b);
writeln('Прямоугольник со сторонами 'a' и 'b,' можно разрезать на 'k,'
квадратов');
end.
```

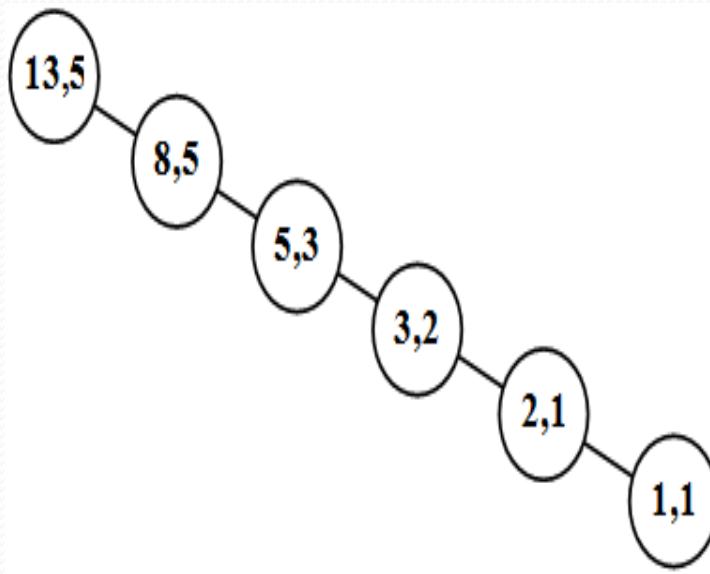


```
Free Pascal
Введите стороны прямоугольника
13 5
Прямоугольник со сторонами 13 и 5 можно разрезать на 6 квадратов
```

Для оценки трудоемкости рекурсивных алгоритмов строится **полное дерево рекурсии**.

Глубина рекурсивных вызовов – наибольшее одновременное количество рекурсивных обращений функции, определяющее максимальное количество слоев рекурсивного стека, в котором осуществляется хранение отложенных вычислений.

Объем рекурсии – количество вершин полного рекурсивного дерева без единицы



Имитация работы цикла с помощью рекурсии

```
procedure LoopImitation(i, n: integer);  
begin  
  writeln('Hello N ', i);  
  if i<=n then  
    LoopImitation(i+1, n);  
end;
```

```
Hello N 1  
Hello N 2  
...  
Hello N 10
```

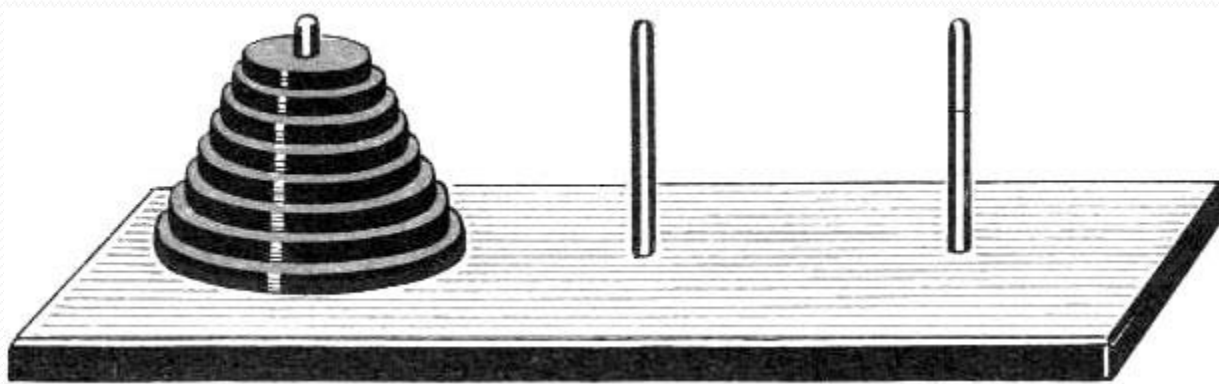
```
procedure LoopImitation2(i, n: integer);  
begin  
  if i<=n then  
    LoopImitation2(i+1, n);  
  writeln('Hello N ', i);  
end;
```

```
Hello N 10  
...  
Hello N 1
```

```
procedure LoopImitation3(i, n: integer);  
begin  
  writeln('Hello N ', i);  
  if i<=n then  
    LoopImitation3(i+1, n);  
  writeln('Hello N ', i);  
end;
```

```
Hello N 1  
...  
Hello N 10  
Hello N 10  
...  
Hello N 1
```


Ханойская башня



Напишем рекурсивную функцию, которая находит решение для произвольного числа колец.

Параметризация. Функция имеет четыре параметра:

- ❖ число переносимых колец,
- ❖ стрежень, на который первоначально нанизаны кольца
- ❖ стержень, на который требуется перенести кольца,
- ❖ стержень, который разрешено использовать в качестве вспомогательного.

База рекурсии. Перенос одного стержня.

Декомпозиция. Чтобы перенести n колец со стержня A на стержень B , используя стрежень C в качестве вспомогательного, можно поступить следующим образом:

- □ перенести $n-1$ кольцо со стержня A на C , используя стержень B в качестве вспомогательного стержня;
- □ перенести последнее кольцо со стержня A на стержень B ;
- □ перенести $n-1$ кольцо со стержня C на B , используя стержень A в качестве вспомогательного стержня.

//n – количество дисков

//a, b, c – номера штырьков. Перекладывание производится со штырька a,

//на штырек b при вспомогательном штырьке c.

procedure Hanoi(n, a, b, c: **integer**);

begin

if n > 1 **then**

begin

 Hanoi(n-1, a, c, b);

writeln(a, ' -> ', b);

 Hanoi(n-1, c, b, a);

end else

writeln(a, ' -> ', b);

end;

Краткие итоги

1. Свойством рекурсивности характеризуются объекты окружающего мира, обладающие самоподобием.
2. Рекурсия в широком смысле характеризуется определением объекта посредством ссылки на себя.
3. Рекурсивные функции содержат в своем теле обращение к самим себе с измененным набором параметров. При этом обращение к себе может быть организовано через цепочку взаимных обращений функций.
4. Решение задач рекурсивными способами проводится посредством разработки рекурсивной триады.

5. Целесообразность применения рекурсии в программировании обусловлена спецификой задач, в постановке которых явно или опосредовано указывается на возможность сведения задачи к подзадачам, аналогичным самой задаче.
6. Область памяти, предназначенная для хранения всех промежуточных значений локальных переменных при каждом следующем рекурсивном обращении, образует рекурсивный стек.
7. Рекурсивные методы решения задач нашли широкое применение в процедурном программировании.