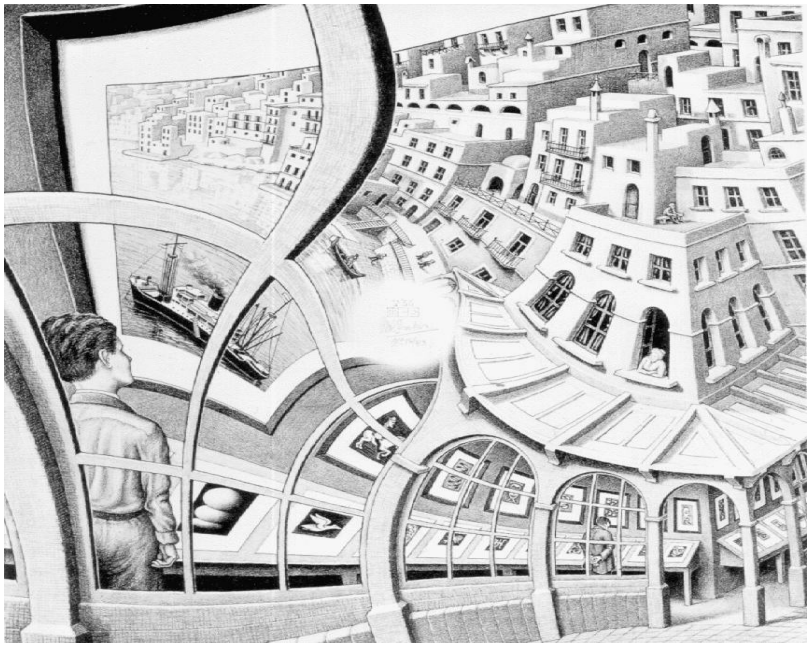


Рекурсия

1

Понятие рекурсии (1)

- Объект называется *рекурсивным*, если он содержит сам себя или определен с помощью самого себя.
- Рекурсия встречается не только в математике, но и в обыденной жизни.
- Каждый сталкивается с рекурсией, когда стоит с зеркальцем перед большим зеркалом.
- Рекурсия встречается обычно и в природе: деревья имеют рекурсивное строение (ветки образуются из других веток), реки образуются из впадающих в них рек.
- Клетки делятся рекурсивно.
- Продолжение жизни связано с рекурсивным процессом.
- Молекулы ДНК и вирусы размножаются, копируя себя, живые существа имеют потомство, которое в свою очередь, тоже имеет потомство и т. д.
- Рекурсия распространена и в языке, и в поведении так же, как и в способах рассуждения и познания.
- Рекурсия в языке, например, может быть в структуре или в содержании:

«Петя сказал, что Вася сказал, что...»

«Знаю, что знаю, но не помню»

«Сделать; заставить сделать; заставить, чтобы заставили сделать;...»

«Замени *x* этим предложением»

«Запомни и передай это сообщение»

2

Понятие рекурсии (2)

Литературным примером может служить «рассказ в рассказе», как–то:

- известное стихотворение «У попа была собака,...»,
- роман Яна Потоцкого «Рукопись, найденная в Сарагосе»,
- рассказ Хулио Кортасара «Непрерывность парков».

Музыкальные формы и действия также могут быть рекурсивными во многих отношениях (например, канон или фуга, в котором мелодия сопровождается той же мелодией с задержкой, и другие).

Целенаправленное поведение и решение проблем так же являются рекурсивными процессами.

3

Понятие рекурсии (3)

Рекурсия является особенно мощным средством в математических определениях. Известны примеры рекурсивных определений натуральных чисел, древовидных структур и некоторых функций:

- *Натуральные числа*:

1) 1 есть натуральное число;

2) целое число, следующее за натуральным, есть натуральное число.

- *Древовидные структуры*:

1) атомарный объект есть дерево;

2) если t_1 и t_2 – деревья, то

есть дерево (нарисованное сверху вниз).

- *Функция факториал n!* для неотрицательных целых чисел:

1) 0!=1;

2) если $n>0$, то $n!=n\,(n-1)!$.

Очевидно, что мощность рекурсии связана с тем, что она позволяет определить бесконечное множество объектов с помощью конечного высказывания.

4

Понятие рекурсии (4)

Рекурсия в программировании – один из важнейших принципов построения подпрограмм.

Если процедура *P* содержит явное обращение к самой себе, то она называется *прямо рекурсивной*; (P ⟶ P)

если *P* содержит обращение к процедуре *Q*, которая содержит (прямо или косвенно) обращение к *P*, то *P* называется *косвенно рекурсивной*. (P ⟶ Q ⟶ P)

Поэтому использование рекурсии не всегда сразу видно из текста программы.

Пример 1. Подпрограмма для вычисления факториала неотрицательного числа:

```
function fac(n:integer{n>=0}):integer;
    begin
     if n=0 then fac:=1
     else fac:=n*fac(n-1)
    end
```

5

Понятие рекурсии (5)

Рекурсивную функцию можно заменить рекурсивной процедурой. Для данного случая определим процедуру:

```
procedure f(var y:integer;n:integer);
var x:integer;
    begin
     if n=0 then y:=1
     else begin f(x,n-1);y:=n*x end
    end
```

Первый параметр в процедуре *f* в результате работы получает значение равное *n*!.

С процедурой принято связывать некоторое множество локальных объектов, т. е. переменных, констант, типов и процедур, которые определены локально в этой процедуре, а вне ее не существуют или не имеют смысла.

Каждый раз, когда такая процедура рекурсивно вызывается, для нее создается новое множество локальных переменных. Хотя они имеют те же имена, что и соответствующие элементы множества локальных переменных, созданного при предыдущем обращении к этой же процедуре, их значения различны.

6

Примеры рекурсии (1)

Пример 2. Подпрограмма для вычисления наибольшего общего делителя двух положительных целых чисел:

(алгоритм Евклида)

```
function gcd(m,n:integer);integer;
    begin
      if m=n then gdc:=m else
      if m<n then gdc:=gdc(n-m,m)
      else gdc:=gdc(m-n,n)
    end
```

Как работает эта функция?

gcd(12,18) = gcd(18-12,12) = gcd(6,12) = gcd(12-6,6) = gcd(6,6) = 6

или

gcd(1000,1) = gcd(999,1) = gcd(998,1) = … gcd(2,1) = gcd(1,1) = 1

7

Примеры рекурсии (2)

Другая версия алгоритма Евклида:

```
function gcd(m,n:integer);integer;
    begin
      if (m=0) or (n=0) then gdc:=n+m
                        else gdc:=gdc(n, m mod n)
    end
```

Как работает эта функция?

gcd(12,18) = gcd(18,12) = gcd(12,6) = gcd(6,0) = 6

или

gcd(18,12) = gcd(12,6) = gcd(6,0) = 6

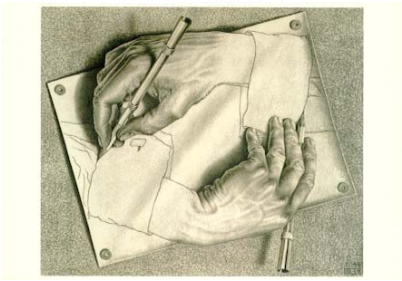
или

gcd(1000,1) = gcd(1,0) = 1

8

Примеры рекурсии (3)

Идею косвенной рекурсии хорошо иллюстрирует гравюра Мориса Эшера.



Косвенная рекурсия может быть задана в системе подпрограмм, которые определяются «бок о бок» и взаимно опираются друг на друга.

9

Примеры рекурсии (4)

Пример 3. Пара подпрограмм (*iseven*, *isodd*) для определения того, четно или нечетно данное натуральное число *n*.

```
function iseven(n:integer):boolean;
    {is iven - быть четным}
    begin
      if n=1 then iseven:=false else iseven:=isodd(n-1)
    end;

function isodd(n:integer):boolean;
    {is odd - быть нечетным}
    begin
      if n=1 then isodd:=true else isodd:=iseven(n-1)
    end;
```

iseven(7) = isodd(6) = iseven(5) = isodd(4) = iseven(3) = isodd(2) = iseven(1) = false

10

Примеры рекурсии (5)

В качестве еще одного примера покажем, как можно рекурсивно определить возведение в степень (причем алгоритм получается более эффективный, чем с использованием *n*-кратного умножения).

Алгоритм:
$x^0 = 1$,
$x^{2n} = (x \times x)^n$,
$x^{2n+1} = x \times (x \times x)^n$.

Получаем естественную программу:

```
function power(x:real; n:integer):real;
{предполагаем, что n>0}
begin
  if n=0 then power:=1 else
  if n mod 2 = 0 then power:= power(x*x, n div 2)
  else power := x * power(x*x, n div 2)
end;
```

$2^{20} = (2\times2)^{10} = 4^{10} = (4\times4)^5 = 16^5 = 16\times(16\times16)^2 = 16\times256^2 =$
$16\times(256\times256)^1 = 16\times65536^1 = 16\times65536\times(65536\times65536)^0 = 16\times65536 \times1 =$
$= 1048576$

11

Примеры рекурсии (6)

Перевернуть строку: f(‘123456’) = ‘654321’

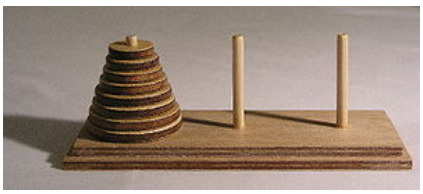
```
function f(s:string):string;
begin
   if s='' then f:=s
          else f:= f(copy(s,2,length(s)-1))+s[1]
end
```

12

Ханойские башни (1)

В конце 19 столетия в Европе появилась игра под названием «Ханойские башни». Популярности игры содействовали рассказы о том, что этой игрой заняты служители храма брахманов, и что завершение игры будет означать конец света.

Предметы, которыми пользовались монахи, будто бы состояли из медной платформы с укрепленными на ней тремя алмазными иглами с насажанными 64 золотыми дисками. Цель игры - перенести башню с левой иглы на правую, причем за один раз можно переносить только одно кольцо, кроме того, запрещается помещать большее кольцо над меньшим.



13

Ханойские башни (2)

Предположим, что иглы пронумерованы числами 1, 2 и 3 и что монахам надо перенести 64 диска с иглы 1 на иглу 3. Обозначим поставленную задачу таким образом:

перенести башню(64,1,3)

Наша цель будет заключаться в разработке алгоритма, который подскажет монахам последовательность корректных перемещений, в результате чего задача будет решена.

К простому решению приводит догадка о том, что основное внимание нужно обратить на нижний диск иглы 1, а не на верхний. Задача «перенести башню(64,1,3)» после этого оказывается эквивалентной следующей последовательности подзадач:

перенести башню(63,1,2)*;*

перенести диск с иглы 1 на иглу 3;

перенести башню(63,2,3)*)*

14

Ханойские башни (3)

Это маленький, но значительный шаг к решению. Маленький, поскольку нам все еще надо дважды переносить по 63 диска. Значительный, потому что мы можем повторить подобный анализ столько раз, сколько будет необходимо. Например, задача

перенести башню(63,1,2)

может быть выражена как

перенести башню(62,1,3);

перенести диск с иглы 1 на иглу 2;

перенести башню(62,3,2)

15

Ханойские башни (4)

Для построения общего алгоритма нам надо указывать, какой иглой можно пользоваться как временным хранилищем. Это можно сделать, расширив нашу запись так, чтобы

перенести башню(*n*,*a*,*b*,*c*)

значило бы

перенести n дисков с иглы a на иглу b, используя иглу c для временной башни

Мы можем утверждать, что задача

перенести башню(*n*,*a*,*b*,*c*)

может быть решена за три шага:

перенести башню(*n*–1,*a*,*c*,*b*);

перенести диск с a на b;

перенести башню(*n*–1,*c*,*b*,*a*).

Этот алгоритм не имеет смысла для случая $n<1$,поэтому добавляем правило:

ничего не делать, если n<1.

16

Ханойские башни (5)

Процедуру для действия «перенести диск» запишем в следующем виде:

```
procedure Disk(from{откуда},where{куда}:integer);
    begin
      writeln(from,'->',where)
    end
```

Рекурсивную процедуру для действия «перенести башню» получаем в виде:

```
procedure Tower(n,from,where,work:integer);
    begin
      if n>0 then begin
                    Tower(n-1,from,work,where);
                    Disk(from,where);
                    Tower(n-1,work,where,from)
                  end
    end
```

17

Ханойские башни (6)

Пусть главная программа содержит операторы:

read(n); Tower(n,1,3,2)

Тогда будем иметь

Ввод :

3

Вывод :

1→ 3
1→ 2
3 →2
1→ 3
2→ 1
2→ 3
1→ 3

Математический анализ алгоритма показывает, что время, нужное программе для переноса башни, состоящей из n дисков пропорционально 2 в степени *n*. Современные вычислительные машины, решая задачу Ханойские башни, могут вычислить (но не напечатать) одно перемещение за одну миллионную долю секунды. Даже при такой скорости для расчета переноса башни из 64 дисков им потребуется около миллиона лет.

18

Сортировка Хоара (1)

Рекурсия дает лучший из известных до сего времени метод сортировки массивов. Он обладает столь блестящими характеристиками, что его изобретатель К. Хоар окрестил его быстрой сортировкой.

Пусть дан массив целых чисел:

var a:array[1..n] of integer;

Попробуем рассмотреть следующий алгоритм:

Пусть x – элемент массива, расположенный в середине (или почти в середине),

просмотрим массив, двигаясь слева направо, пока не найдем элемент a[i]>x, а затем просмотрим его справа налево, пока не найдем элемент a[j] ≤ x.

Теперь поменяем местами эти два элемента и продолжим процесс «просмотра с обменом», пока два просмотра не встретятся где-то в середине массива.

В результате массив разделится на две части: левую – с элементами меньшими или равными *x*, и правую – с элементами большими *x*.

5 1 7 3 2 5 x=3

2 1 7 3 5 5

2 1 3 7 5 5 => 2 1 3 и 7 5 5

Разделив массив, нужно сделать то же самое с обеими полученными частями, затем с частями этих частей и т. д., пока каждая часть не будет содержать только один элемент.

19

Сортировка Хоара (2)

```
Const n = …;  Type index=1..n;  Var a:array[index] of integer;

procedure sort(l,r:index);
      var i,j:index;
          w,x:integer;
       begin
       i:=l;j:=r;
       x:=a[(l+r) div 2];   { x – 'средний' элемент}
       repeat
             while a[i]<x do i:=i+1;
             while x<a[j] do j:=j-1;
             if i<=j then
                    begin
                    w:=a[i];a[i]:=a[j];a[j]:=w;
                    i:=i+1;j:=j-1
                   end
       until i>j;
       if l<j then sort(l,j);
       if i<r then sort(i,r)
 end;{sort}
```

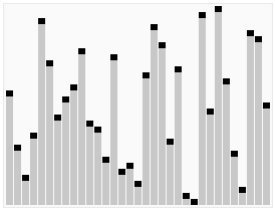
20

Сортировка Хоара (3)

Вызов сортировки:

Sort(1,n);

Анализ алгоритма показывает, что общее число сравнений и обменов пропорционально $n \log n$.



21