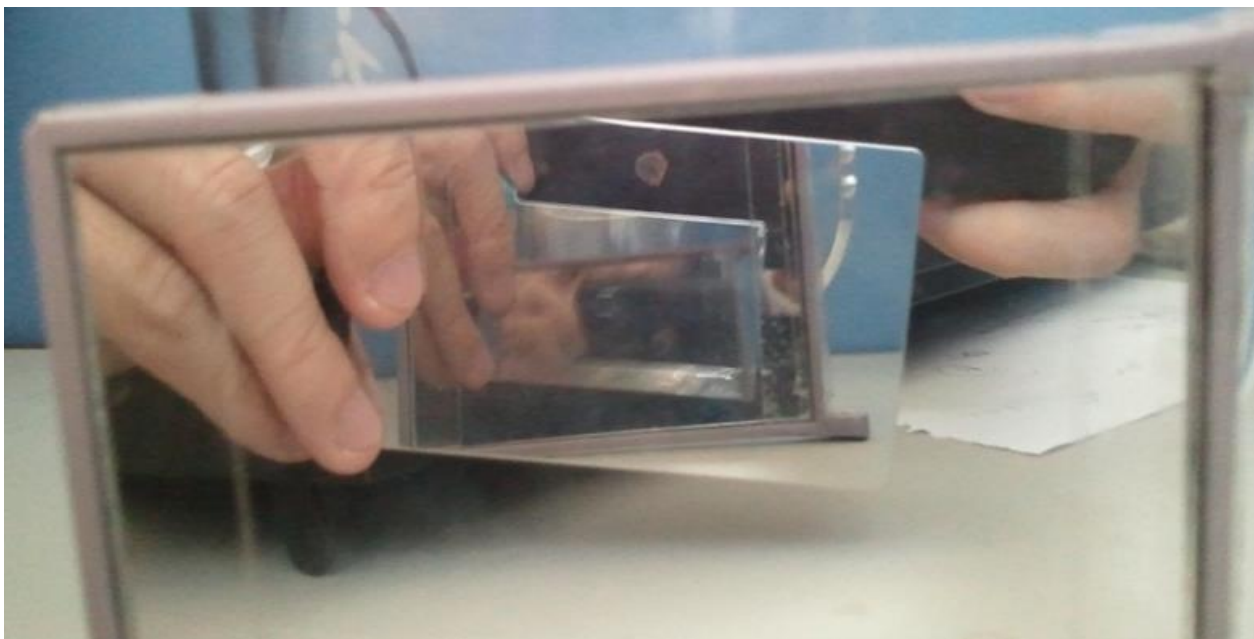


# 函数的递归调用

•→从前有座山，山里有座庙，里面有两个和尚，一个大和尚，一个小和尚，一天大和尚对小和尚说，我给你讲个故事：~~从前有座山.....~~

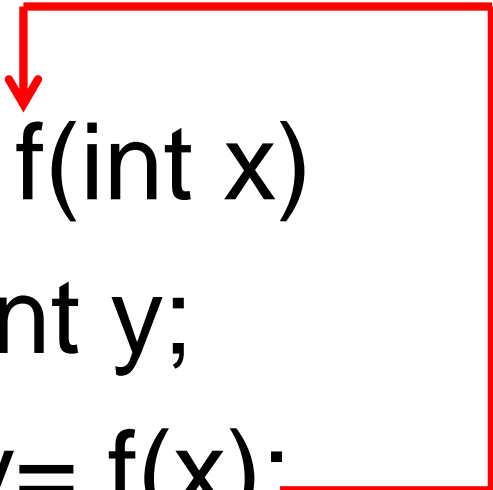
- 两面镜子相互对照



# 函数的递归调用

- 在调用一个函数的过程中又出现直接或间接地调用函数本身的情况，称为函数的递归调用。
- 两种形式的递归
  - 直接递归：f1中调用f1
  - 间接递归：f1中调用f2、f2再调用f1

- 直接递归：f1中调用f1

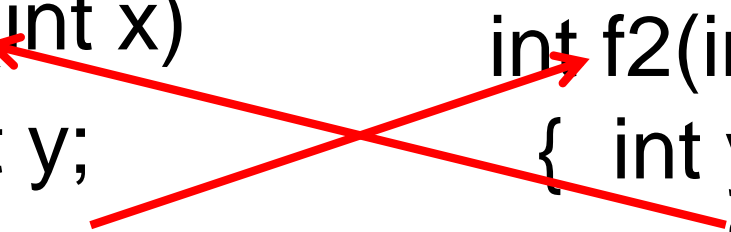


```
int f(int x)
{ int y;
  y= f(x);
  return y;
}
```

- 间接递归：f1中调用f2、f2再调用f1

```
int f1(int x)
{
    int y;
    y = f2(x);
    return y;
}

int f2(int x)
{
    int y;
    y = f1(x);
    return y;
}
```



- 递归函数的编写与使用应注意：递归函数一定要有“出口”，否则无法停止。

例 8. 7 有 5 个人坐在一起，问第 5 个人多少岁？他说比第 4 个人大 2 岁。问第 4 个人岁数，他说比第 3 个人大 2 岁。问第 3 个人，又说比第 2 个人大 2 岁。问第 2 个人，说比第 1 个人大 2 岁。最后问第 1 个人，他说是 10 岁。请问第 5 个人多大。

$$\text{age}(5) = \text{age}(4) + 2$$

$$\text{age}(4) = \text{age}(3) + 2$$

$$\text{age}(3) = \text{age}(2) + 2$$

$$\text{age}(2) = \text{age}(1) + 2$$

$$\text{age}(1) = 10$$

可以用数学公式表述如下：

$$\text{age}(n) = 10 \quad (n = 1)$$

$$\text{age}(n - 1) + 2 \quad (n > 1)$$

# 递归与数学归纳法的类比

- 递归与数学归纳法的类比
- 用数学归纳法证明:  
$$1 + 3 + 5 \dots + (2n - 1) = n^2$$
- 先设定 一个初始值当 $n=1$ 时，证明成立  
假设 $n=k$ 时等式成立，推出 $n=k+1$ 成立



# 递归的解题思路

- 先设定一个递归出口，一般是边界情况
- 将问题化为与原问题**类似**的子问题，子问题的参数类型、个数以及函数的返回值与原问题一致，只是在问题的规模上比原问题小。
- 假设子问题可以解决，利用子问题去解决原问题。

# 递归例题

## P174 例题8.8 求阶乘 $n!$

- 1、递归出口，边界情况  $n=0$ 或 $1$ 时  $n!=1$
- 2、转化为与原问题类似的子问题  
 $n! = n \times (n-1)!$   
其中的子问题是  $(n-1)!$
- 3、利用子问题去解决原问题：  
 $n! = n \times (n-1)!$

# 递归例题

```
float fac(int n)
{
    float f;
    if(n<0)
    {
        printf("n<0,dataerror!");
    }
    else if(n==0||n==1) f=1;
        else f=fac(n-1)*n;
    return(f);
}
```

与原问题相类似，参数个数以及参数类型都一致

# 递归框架

```
float fac(int n)
```

根据实际需要确定函数名称、参数个数，参数类型及函数返回值

```
{
```

```
float f;
```

根据实际需要确定是否需要变量

```
if(n==1)
```

用条件判断语句表示递归出口

```
{
```

```
    f=1;
```

递归出口，一般是边界情况

```
}
```

```
else
```

```
{ f = fac(n-1).....
```

递归调用，子问题的参数个数以及参数类型与原问题一致

```
}
```

```
return(f);
```

```
}
```

# 递归例题

## Fibonacci数列 P125

$$F(1)=1 \quad (n=1)$$

$$F(2)=1 \quad (n=2)$$

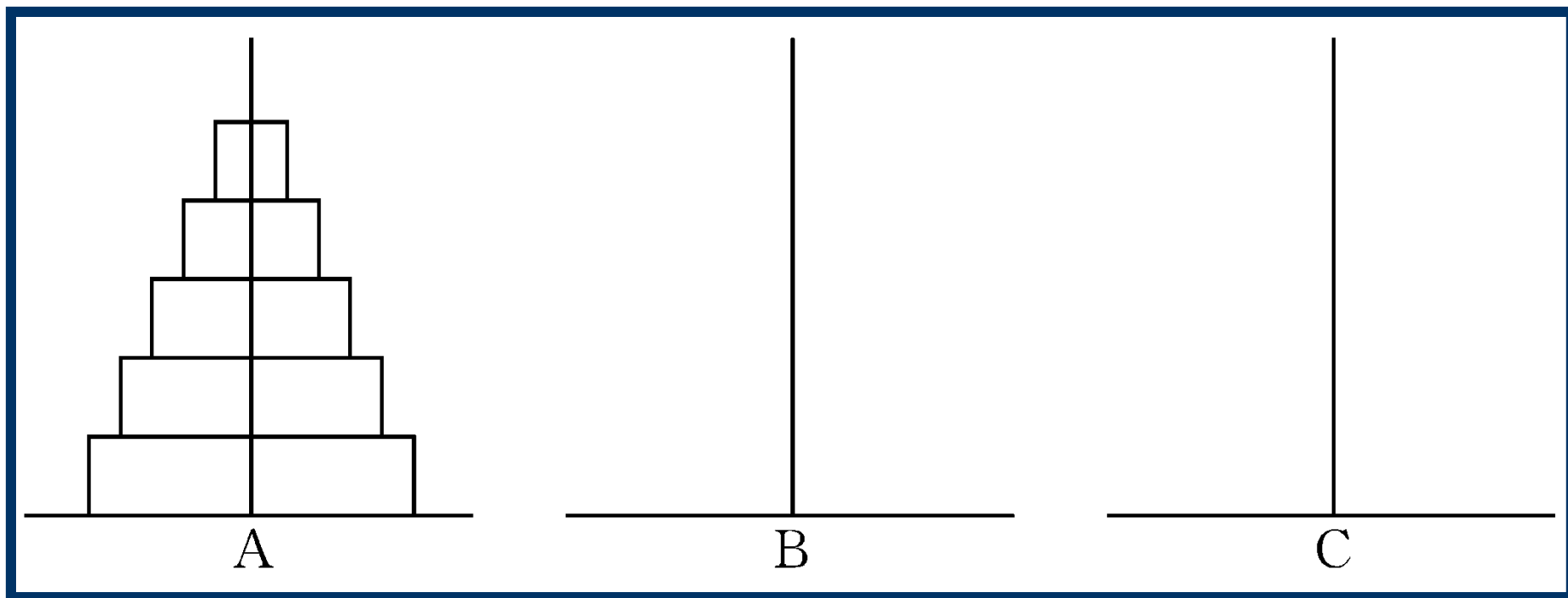
$$F(n)=F(n-1)+F(n-2) \quad (n \geq 3)$$

思考：

- 1、递归的出口条件是什么？
- 2、如何转化为类似的子问题？
- 3、函数的参数个数要几个？类型是什么？函数返回值是什么？

# 递归例题

## Hanoi问题



# Hanoi问题

由分析可知：将  $n$  个盘子从 A 座移到 C 座可以分为以下3个步骤：

- (1) 将 A 上  $n - 1$  个盘借助 C 座先移到 B 座上。
- (2) 把 A 座上剩下的一个盘移到 C 座上。
- (3) 将  $n - 1$  个盘从 B 座借助于 A 座移到 C 座上。

思考：

- 1、函数要几个参数？参数的类型是什么？
- 2、函数的返回值是什么？
- 3、递归调用的出口是什么？
- 4、如何将原问题转化为子问题？

# 递归例题

- (1) 将 A 上  $n - 1$  个盘借助 C 座先移到 B 座上。
- (2) 把 A 座上剩下的一个盘移到 C 座上。
- (3) 将  $n - 1$  个盘从 B 座借助于 A 座移到 C 座上。

```
void hanoi(int n,char one,char two,char three)  
{  
    if(n==1) move(one,three);  
    else  
        {  
            hanoi(n-1,one,three,two);  
            move(one,three);  
            hanoi(n-1,two,one,three);  
        }  
}
```



- 母牛数列问题：若一头小母牛，从出生起第四个年头开始每年生一头母牛，按此规律，求第n年时有多少头母牛。

月份	上月的母牛数量	新生的母牛数量	母牛的总数量
1			1
2	1	0	$1 + 0 = 1$
3	1	0	$1 + 0 = 1$
4	1	1	$1 + 1 = 2$
5	2	1	$2 + 1 = 3$
6	3	1	$3 + 1 = 4$
7	4	2	$4 + 2 = 6$
8	6	3	$6 + 3 = 9$
9	9	4	$9 + 4 = 13$
.....			

- 编程求解：若一头小母牛，从出生起第四个年头开始每年生一头母牛，按次规律，第n年时有多少头母牛？

年头 **1 2 3 4 5 6 7 8...**

牛数 **1 1 1 2 3 4 6 9...**

```
int cattle(int n)           cattle(n)=cattle(n-1)+ cattle(n-3);
{
    if(n<=0)
        return 0;
    if(n<=3)
        return 1;
    return cattle(n-1)+ cattle(n-3);
}
```

- 有N阶台阶，上楼可以一步上一阶，也可以一次上二阶。编一个程序，计算共有多少种不同的走法。

递归的形式： $s[n]=s[n-1]+s[n-2]$

基本式子： $s[1]=1;s[2]=2$

# 八皇后问题

- 八皇后问题：假设将八个皇后放到国际象棋棋盘上，使其两两之间无法相互攻击。共有几种摆法？

基础知识：

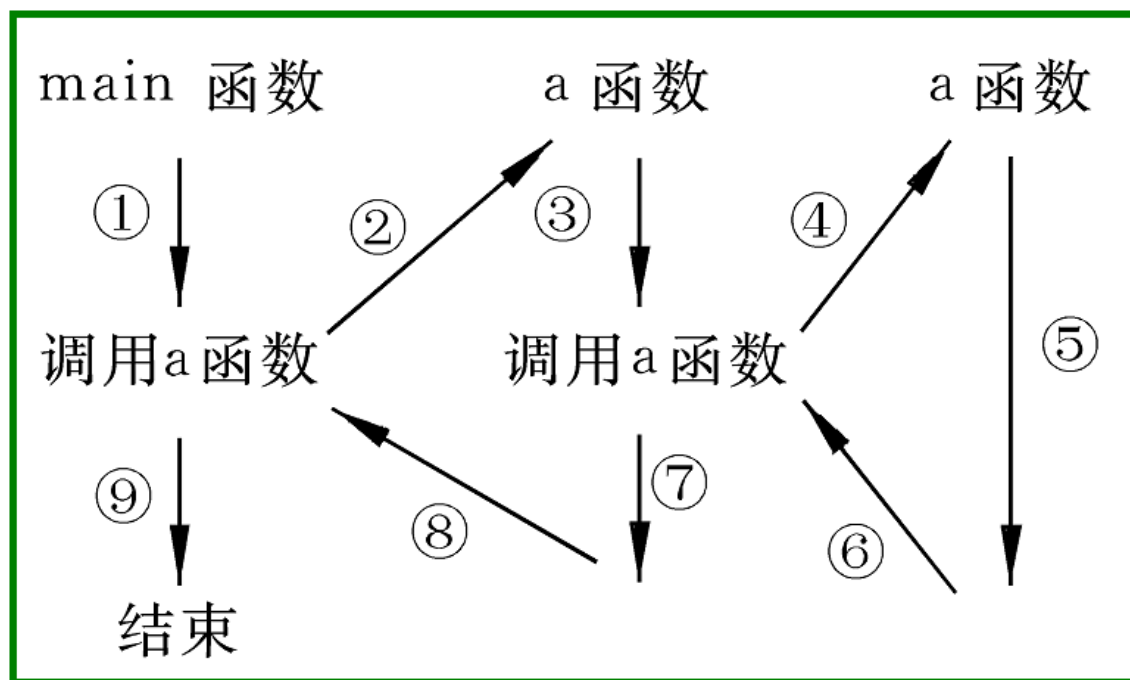
- 国际象棋里，棋盘为8X8格。
- 皇后每步可以沿直线、斜线 走任意格。

# 用递归、回溯实现

- <https://www.cnblogs.com/cnnnnnnn/p/8506883.html>
- [https://blog.csdn.net/qq\\_42552533/article/details/86684045](https://blog.csdn.net/qq_42552533/article/details/86684045)

# 递归的实际过程

递归属于函数的嵌套调用。调用函数时需要保存调用的一些现场参数等等，因此函数的调用需要占用内存空间。当递归的次数很多时，消耗的内存空间是相当可观的。递归实际是以牺牲大量的存储空间为代价的。



# 递归的实际过程

基本上所有递归都可以由循环来代替，但是，有些时候用递归较好，有些时候用循环较好。由于递归涉及函数的调用，要消耗栈的空间，同时还得花时间建立函数调用过程记录，因此，通常优先使用循环的方法。

**注意：**对于递归程序，在递归语句之前的语句，在递归之前执行；在递归语句之后的语句，在递归返回之后执行。

- 内部变量只在所在块中有效，故称局部变量。
- 函数体是典型的块，所以在函数声明语句部分定义的变量也是局部变量。
- 形式参数只在函数中有效，也是局部变量。

```
#include <stdio.h>
int main()
```

```
4 integer numbers:12 45 -6 89
max=89
```

```
{ .....
  max=max4(a,b,c,d);
  .....
}
```

函数的调用过程

```
int max4(int a,int b,int c,int d)
{ int max2(int a,int b);
  return max2(max2(max2(a,b),c),d);
}
```

```
int max2(int a,int b)
{ return(a>b?a:b); }
```





# 递归的实际调用过程

```
#include <stdio.h>
```

```
int main()
```

```
{ int age(int n);
```

```
    printf("NO.5,age:%d\n",age(5));
```

```
    return 0;
```

```
}
```

```
int age(int n)
```

```
{ int c;
```

```
    if(n==1) c=10;
```

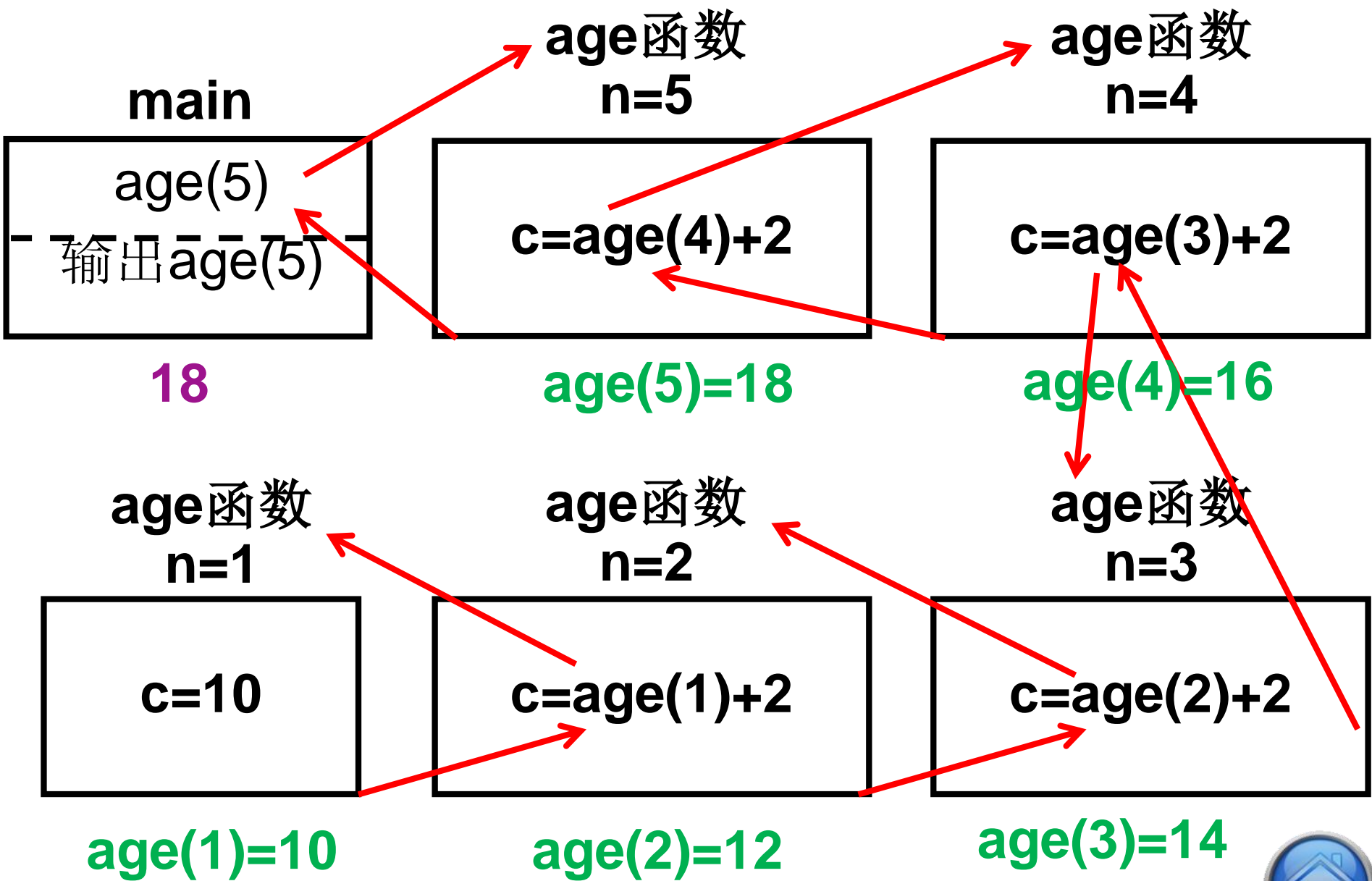
```
    else c=age(n-1)+2;
```

```
    return(c);
```

```
}
```

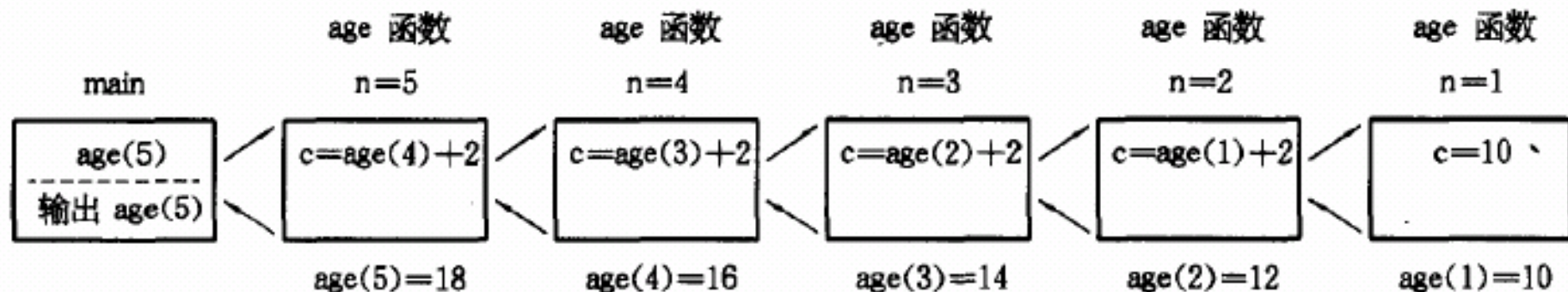
```
NO.5,age:18
```





# 递归的实际过程

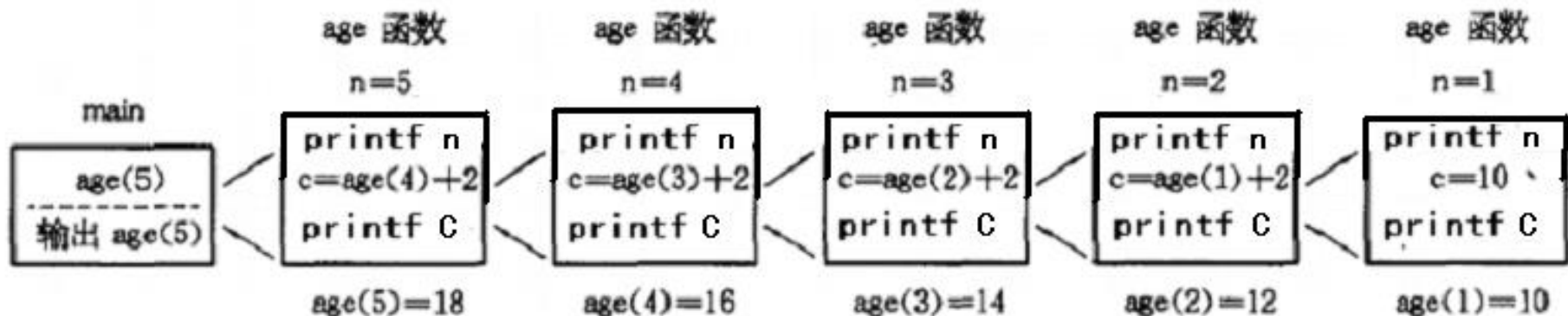
```
int age(int n)
{
    int c;
    if(n==1) c=10;
    else c=age(n-1)+2;
    return(c);
}
```



# 递归的实际过程

```
int age(int n)
{
    int c;
    printf("n=%d\n",n);
    if(n==1) c=10;
    else c=age(n-1)+2;
    printf("c=%d\n",c);
    return(c);
}
```

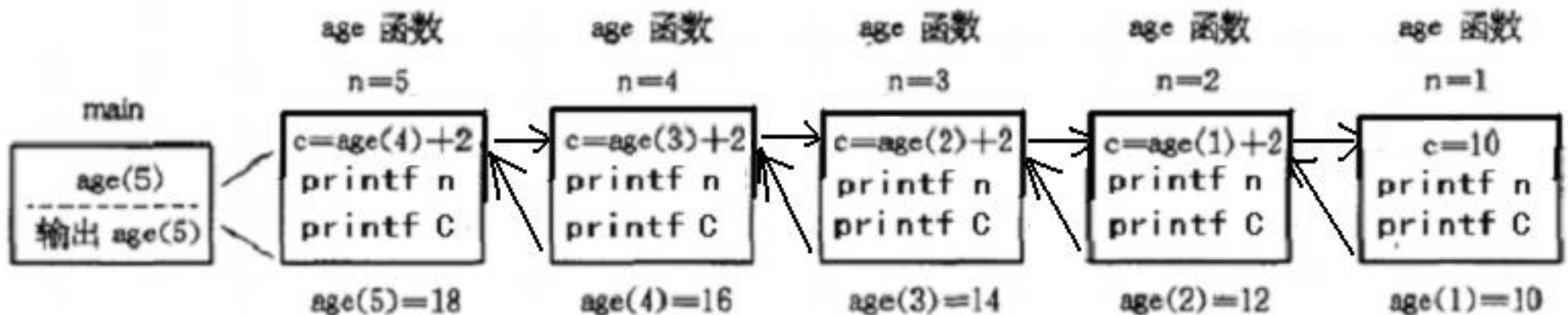
n=5  
n=4  
n=3  
n=2  
n=1  
c=10  
c=12  
c=14  
c=16  
c=18



# 递归的实际过程

```
int age(int n)
{
    int c;
    if(n==1) c=10;
    else c=age(n-1)+2;
    printf("n=%d\n",n);
    printf("c=%d\n",c);
    return(c);
}
```

n=1  
c=10  
n=2  
c=12  
n=3  
c=14  
n=4  
c=16  
n=5  
c=18



```
void f(int i,int j)  
{  
if(i<j)  
{  
    printf("i=%d\n",i);  
    f(i+1,j-1);  
    printf("j=%d\n",j);  
}  
}  
int main( )  
{  
    f(0,4);  
    return 0;  
}
```

**i=0**

**i=1**

**j=3**

**j=4**

```
#include <stdio.h>  
void rec(char c)  
{  
    printf("%c ", c);  
    if(c<'6') rec(c+2);  
    printf("%c ", c);  
}  
void main()  
{  
    rec('4');  
    printf("\n");  
}
```

4 6 6 4

```
void f(int i)  
{  
    putchar('+');  
    if (i)  
    {  
        f(i-1);  
        printf("%d",i);  
    }  
}  
int main()  
{  
    f(2);  
    return 0;  
}
```

+++12



```
#include <stdio.h>
```

```
void rec(char c)
```

```
{
```

```
    printf("%c ", c);
```

```
    if(c<'6') rec(c+=2);
```

```
    printf("%c ", c);
```

```
}
```

```
void main()
```

```
{
```

```
    rec('4');
```

```
    printf("\n");
```

```
}
```

4 6 6 6

```
#include "stdio.h"
void fun(int i)
{
    if (i>0)
    {
        fun(i-4);
        printf("%d,",i);
        fun(i-5);
    }
}
int main()
{
    fun(9);
    return 0;
}
```

1,5,9,4,

```
#include "stdio.h"
```

```
void fun(int i)
```

```
{
```

```
    if (i>0)
```

```
    {
```

```
        fun(i-=4);
```

```
        printf("%d,",i);
```

```
        fun(i-=2);
```

```
    }
```

```
}
```

```
int main()
```

```
{
```

```
    fun(9);
```

```
    return 0;
```

```
}
```

-3,1,5,-1,

```
void fun(int i)
{
    if (i>3)
    {
        fun(i-1);
        printf("%d ",i);
        fun(i-1);
    }
    else
    {
        printf("%d ",i);
    }
}

int main()
{
    fun(5);
    return 0;
}
```

3 4 3 5 3 4 3

```
void fun(int x)
{
    if(x/2>0) fun(x/2);
    printf("%d", x%2);
}

int main()
{
    fun(20);
    return 0;
}
```

10100

```
void f(int i,int j)  
{  
if(i<j)  
{  
    printf("i=%d\n",i);  
    f(++i,--j);  
    printf("j=%d\n",j);  
}  
}  
int main( )  
{  
    f(0,4);  
    return 0;  
}
```

**i=0**

**i=1**

**j=2**

**j=3**

```
void f(int i,int j)
{
    if(i+j<10)
    {
        printf("i=%d\n",i);
        f(++i,++j);
        printf("j=%d\n",j);
    }
}

int main( )
{
    f(0,4);
    return 0;
}
```

**i=0  
i=1  
i=2  
j=7  
j=6  
j=5**

```
void f(int i,int j)
{
  if(i+j<6)
  {
    printf("i=%d\n",i);
    f(++i, j++);
    printf("j=%d\n",j);
  }
}

int main( )
{
  f(0,4);
  return 0;
}
```

**i=0  
i=1  
j=5  
j=5**