3.3 栈与递归的实现

1、递归

一个对象如果部分地由它自身来定义（或描述），则称其为递归。例如：

- 阶乘函数：

$$
Fact(n)=\begin{cases} 1 & n=0 \\ n*Fact(n-1) & n>=1 \end{cases}
$$

- 二阶Fibonacci数列：

$$
Fib(n)=\begin{cases} 0 & n=0 \\ 1 & n=1 \\ Fib(n-1)+Fib(n-2) & n>1 \end{cases}
$$

2、递归函数

一个直接调用自己或通过一系列的调用语句间接地调用自己的函数，称作递归函数。

当在一个函数的运行期间调用另一个函数时，在运行该被调用函数之前，系统需先完成三项任务：

- 将所有的实在参数、返回地址等信息传递给被调用函数保存；
- 为被调用函数的局部变量分配存储区；
- 将控制转移到被调用函数的入口。

从被调用函数**返回**调用函数**之前**，应该完成下列三项任务：

- 保存被调函数的计算结果；
- 释放被调函数的数据区；
- 依照被调函数保存的返回地址将控制转移到调用函数。

多个函数嵌套调用的规则是：

后调用先返回 ！

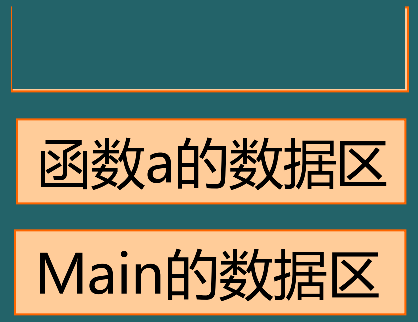
此时的内存管理实行 **“栈式管理”**

例如：

```
void main( ){
    …
    a( );
    …
}//main
```

```
void a( ){
    …
    b( );
    …
}// a
```

```
void b( ){
    …
}// b
```



递归函数执行的过程可视为**同一函数**进行嵌套调用。

递归工作栈： 递归过程执行过程中占用的数据区。

递归工作记录： 每一层的递归参数合成一个记录。

当前活动记录： 栈顶记录指示当前层的执行情况。

当前环境指针： 递归工作栈的栈顶指针。

递归是程序设计中一个强有力的工具。那么，如何设计递归过程呢？可根据以下几种情况考虑：

一、问题的定义是递归的。

例如，数学中的阶乘函数、二阶Fibonacci函数等。

n阶阶乘的递归函数如下：

```
int Fact(int n)
{
    if(n==0)   return 1;
    else   return n*Fact(n-1);
}
```

递归出口

递归表达式

求二阶Fibonacci数列的递归函数如下：

```
int Fib(int n) {
   if(n==0)      return 0;
   else  if(n==1)    return 1;
         else    return Fib(n-1)+Fib(n-2);
}
```

递归出口

递归表达式

二、有的数据结构，如二叉树、广义表等，由于结构本身固有的递归特性，则它们的操作可递归的描述。

三、有些问题，虽然问题本身没有明显的递归结构，但用递归求解比迭代求解更简单。如八皇后问题、Hanoi塔问题等。

例3-2（n阶Hanoi塔问题）假设有三个分别命名为X、Y和Z的塔座，在X上插有n个直径大小各不相同、依小到大编号为1，2，...n的圆盘（如图3.5所示）。现要求将X轴上的n个圆盘移至Z上并仍按同样的顺序叠排，**圆盘移动时必须遵循下列规则：**

图3.5 3阶Hanoi塔问题

1、每次只能移动一个圆盘；

2、圆盘可以插在X、Y和Z中的任一塔座上；

3、任何时候都不能将一个较大的圆盘压在较小的圆盘之上。

如何实现移动圆盘的操作呢？当n=1时，问题比较简单，只需将编号为1的圆盘从塔座X直接移至塔座Z上即可；当n>1时，该如何考虑呢？

1、将压在编号为n 的圆盘之上的**n-1**个圆盘从塔座**X**（依照上述法则）移至塔座**Y**上；

2、将编号为**n** 的圆盘从塔座**X**移至塔座**Z**上；

3、再将塔座**Y**上的**n-1**个圆盘（依照上述法则）移至塔座**Z**上；

其中，将n-1个圆盘从一个塔座移至另一塔座的问题是一个和原问题相同的问题，只是问题的规模小1

```
void hanoi (int n, char x, char y, char z) {
     // 将塔座x上按直径由小到大且至上而下编号为1至n
   // 的n个圆盘按规则搬到塔座z上，y可用作辅助塔座。
   //搬动操作move(x, n, z)可定义为(c是初值为0的全局
     //变量，对搬动计数)：printf( “%i.Move disk %i from
     // %c  to %c\n”, ++c, n, x, z);
{
    if (n==1)
        move(x, 1, z);              // 将编号为 1 的圆盘从x移到z
    else {
        hanoi(n-1, x, z, y);      // 将x上编号为 1 至n-1的
                                   //圆盘移到y, z作辅助塔
        move(x, n, z);             // 将编号为n的圆盘从x移到z
        hanoi(n-1, y, x, z);      // 将y上编号为 1 至n-1的
                                   //圆盘移到z, x作辅助塔
            }
}
```

算法 3.5

```
void hanoi (int n, char x, char y, char z) {
1  if (n==1)
2   move(x, 1, z);
3  else {
4    hanoi(n-1, x, z, y);
5    move(x, n, z);
6    hanoi(n-1, y, x, z);
7  }
8 }
```

假设主函数的返回地址为0，以递归函数的语句行号为返回地址。

返址	n	x	y	z
5	2	a	c	b
0	3	a	b	c

补充：给定集合A={$a_1,a_2, \dots ,a_{n-1},a_n$}，设计算法求A的所有排列。

先看集合A={a,b,c}的所有排列。

abc，acb，bac，bca，cab，cba 共六个 n!个。

规律？

以集合中任一元素为第一个元素，后跟其余剩余元素的所有排列。

所以：集合A={$a_1,a_2, \dots ,a_{n-1},a_n$}的所有排列就是：

a_1+集合{$a_2, \dots ,a_{n-1},a_n$}的所有排列

并 a_2+集合{$a_1,a_3 \dots ,,a_n$}的所有排列

……

并 a_n+集合{$a_1,a_2 \dots ,,a_{n-1}$}的所有排列

```
void arrange(int A[], int k, int n)//求A的前k-1个元素
{                  //不变，第k到第n个元素变化的所有排
 if(k == n)      //列的算法
   {
       for (int i = 0; i <= n; i++)
           printf("%d", A[i]);
       printf("\n");
    }
  else
    {
      for (i = k; i <= n; i++)
        {
           Exchange(A[k],A[i]);
           arrange(A,k+1,n);
           Exchange(A[k],A[i]);
        }
    }
}
```

```
Exchange(int a, int b, int temp)
{
  temp = a;
  a = b;
  b = temp;
}
```