水善利万物而不争，

处众人之所恶，故几于道

```
typedef  struct {

  ElemType  *elem;  //待分配存储空间
  int  front;  //头指针
  int  rear;  //尾指针
  } SeqQueue ;
typedef int status;
InitQueue(SeqQueue *sq);
InQueue(SeqQueue *sq,ElemType e);
OutQueue(SeqQueue *sq,ElemType *e);
status IsFull(SeqQueue sq);
status  IsEmpty(SeqQueue sq);
```

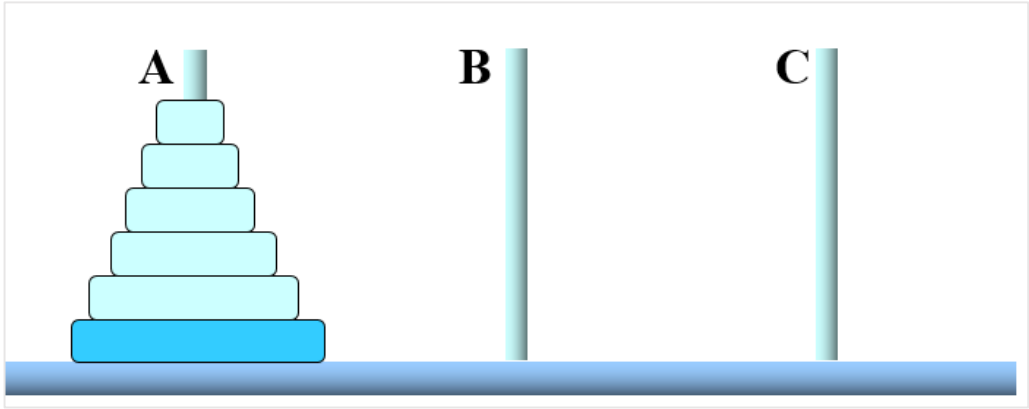
利用该数据结构，通过增加操作控制来实现银行排队问题模拟

```
/*模拟银行排队*/
0:  开始办理          InitQueue()
1:请您拿号            InQueue()
2:  *顾客到1号窗口办理  OutQueue()
3:  *顾客到2号窗口办理  OutQueue()
4:下班          ClearQueue()
```

3.6 栈与递归

回顾我们学过的递归函数

- ✓阶层
- ✓费波纳切数列 → （数学递推）
- ✓汉诺塔 （问题递推）
- ✓单链表 （结构递归）



3. 6. 1递归算法

调用自身的方法称为递归方法

（1）数学递推

例1. 求n!

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

```
int  fact(int  n){
 if (n<=1)   return 1;
  else {
        f= n*fact(n-1);
        printf (f);
   }
}
```

数学函数是递归定义的，可用递归方法

（2）问题递推---汉诺塔问题

有一些问题，如果不用递归，几乎无法求解？

史上智者公推的智力谜题之一——汉诺塔问题

- 来源于印度传说的一个故事。Benares标志着世界中心的神圣庙宇中，放置着三根金刚柱，在一根柱子上摞着64片黄金圆盘，最大的盘子在最下边，从下到上，盘子越来越小。上帝命令婆罗门把圆盘按大小顺序摆放在另一根柱子上。并规定：小圆盘上不能放大圆盘，一次只能移动一个圆盘。

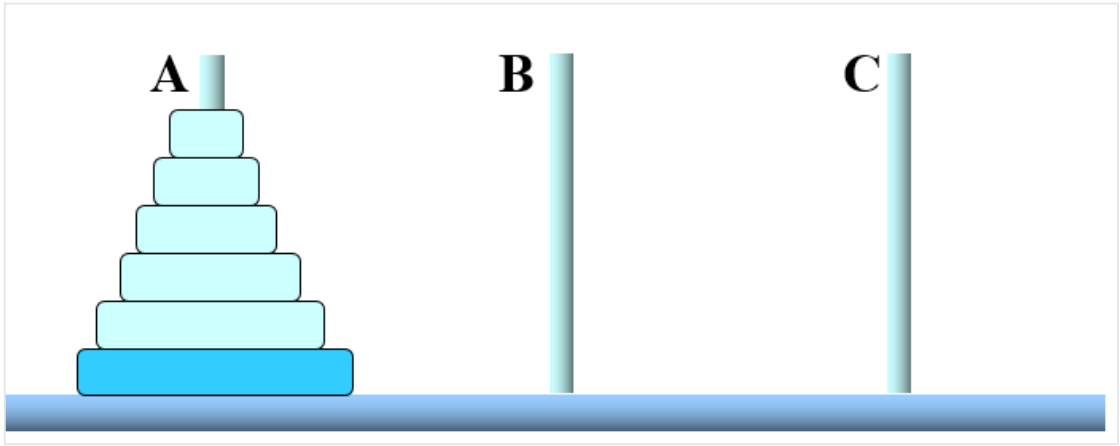
移动圆片的次数　18446744073709551615次！

例3. 汉诺塔问题

Move（8，‘A’，‘B’，‘C’）

Hanoi塔问题，将a塔上的n个盘子通过b塔移到c塔上。

需要考虑：盘子数
开始放置的塔座
目标塔座
临时放置塔座



规则：

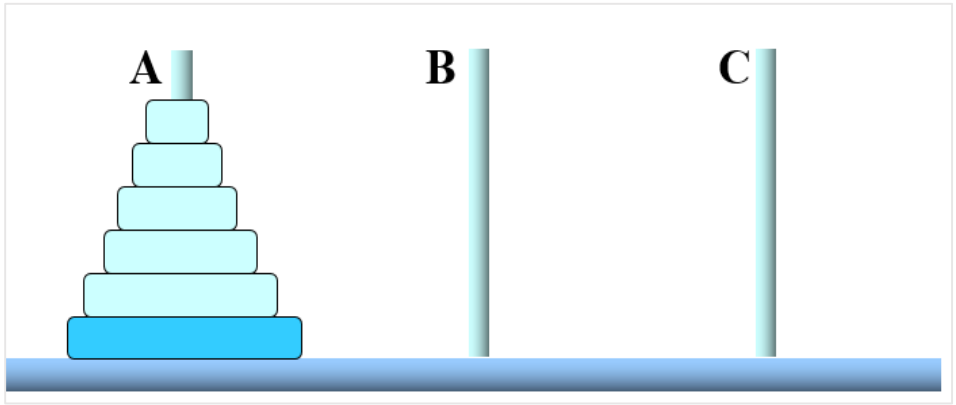
（1）每次只能移动一个

（2）盘子只许在三座塔上存放

（3）不许大盘压小盘

递归策略

分治法：将问题化为规模更小的子问题，确保子问题的问题性质和原问题一样。最小的问题可以直接求解。



将 n 个盘分成两个子集(1至n-1和 n)，从而产生下列3个子问题：

1） 将1至n-1号盘从A塔移动至B塔；

2） 将n号盘从 A 塔移动至 C 塔；

3） 将1至n-1号盘从B塔移动至C塔；

1） 将1至n-1号盘从A塔移动至B塔；

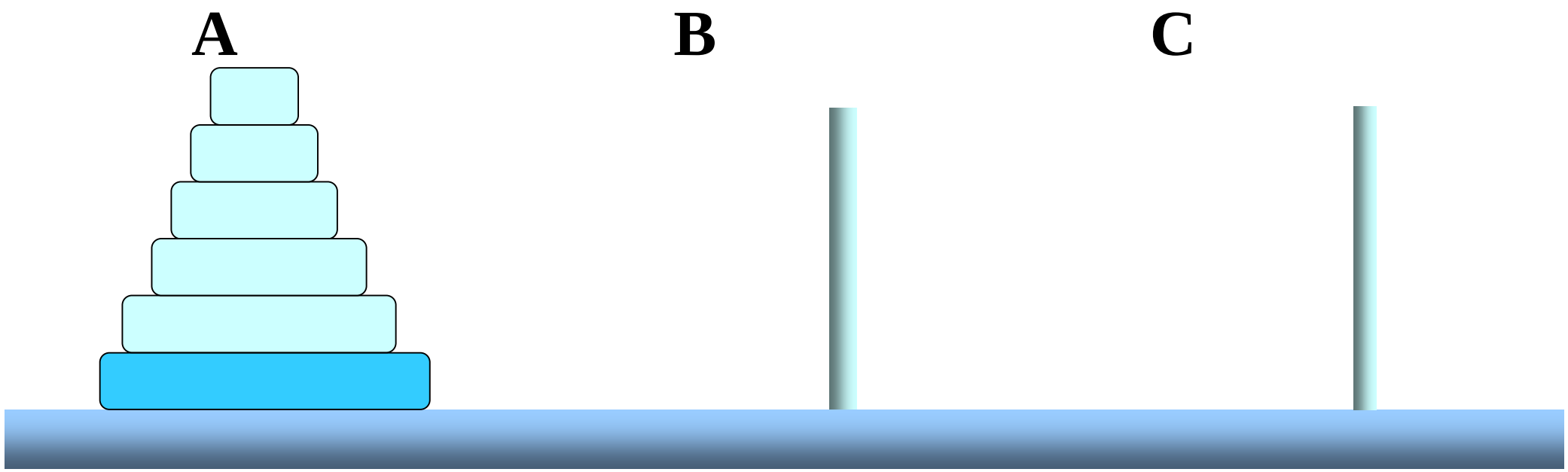
2） 将n号盘从 A 塔移动至 C 塔；

3） 将1至n-1号盘从B塔移动至C塔；

递归方法：Hanoi(n-1, A, C, B)

move(A，n，C)；

递归方法：Hanoi(n-1, B, A, C)



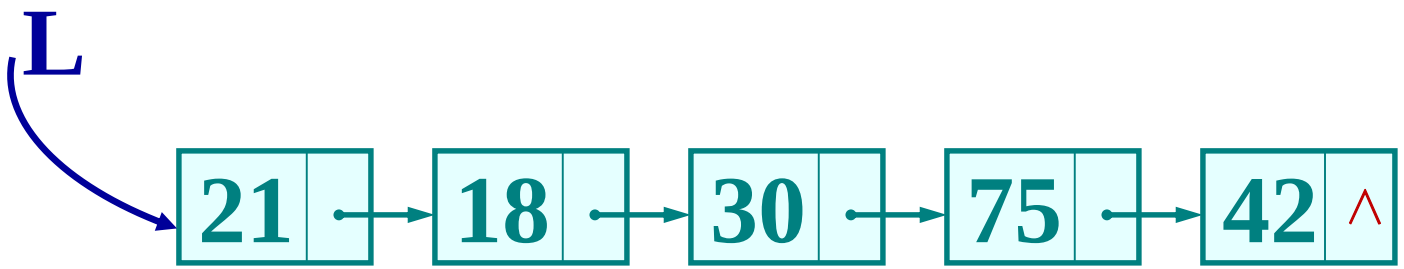
```
void hanoi(int n, char x, char y, char z)
 { if (n== 1)
     move(x, 1, z); //x塔上的1号盘移到Z塔,
  else {
    hanoi(n-1, x, z, y);
    move(x, n, z);  //x塔上的n号盘移到Z塔,
    hanoi(n-1, y, x, z) ;
     }
} // hanoi
```

写递归算法

- ✓递归调用
- ✓出口条件

```
main(){
 int n;
 scanf("%d", &n);
hanoi(n, ' a', ' b', ' c'); }
```

（3）结构递归



输出

```
void PrintList(LinkList L){
  if (L == NULL)return;
   else{
      printf(L->data);
      PrintList(L->next);}}
```

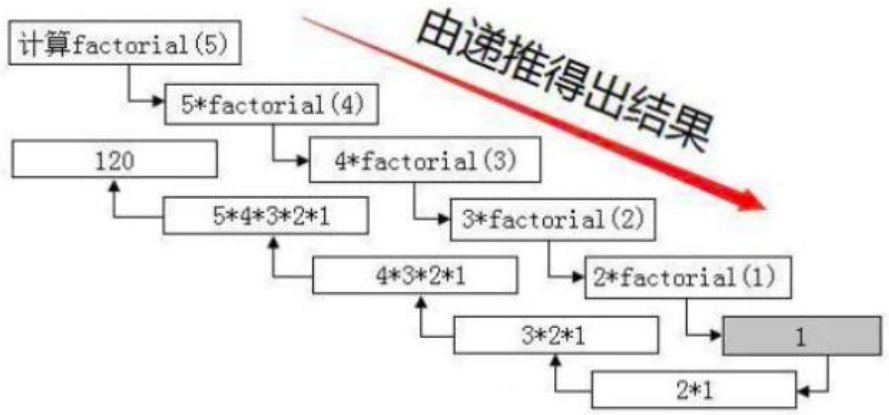
单链表L：由L所指的结点和L->next为头指针的单链表；

因此，原问题可分解为L->next的问题和L所指结点的问题

3.6.2 递归执行过程与输出

例1. 求n!

$$n! = \begin{cases} 1 & (n<1) \\ n*(n-1)! & (n\geqslant 1) \end{cases}$$




```
int  fact(int  n){
  if (n<=1)   return 1;
   else {
         f= n*fact(n-1);
         printf (f);
    }
}
```

如何读递归算法？

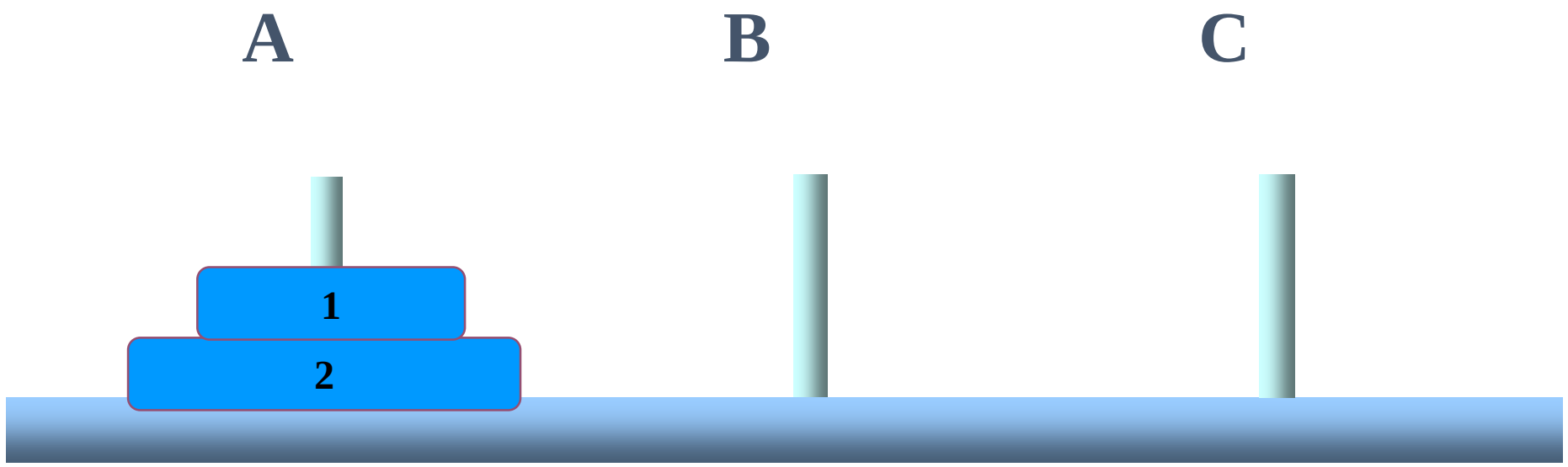
看 A 柱上有2只盘子hanoi(2, A, B, C)的情况：

hanoi(1，A，C，B)；　　move(A，1，B)；

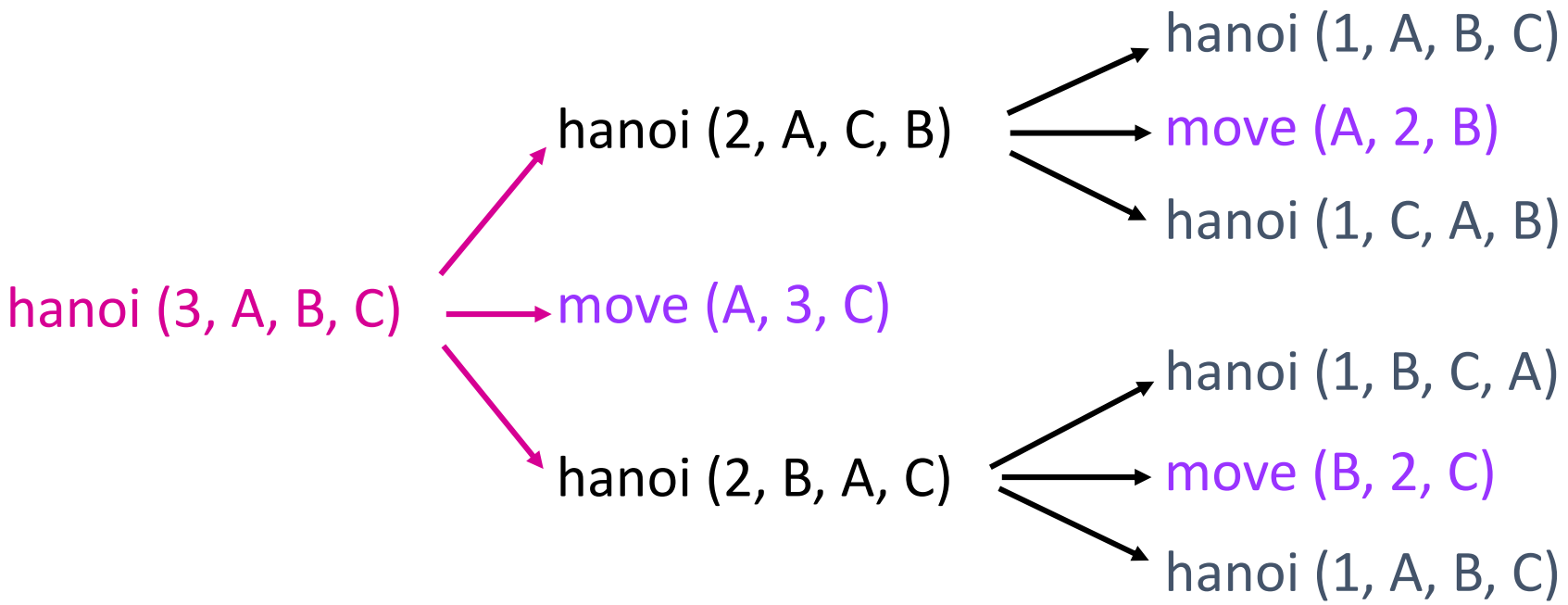
move(A，2，C)；　　move(A，2，C)；

hanoi(1，B，A，C)；　　move(B，1，C)；

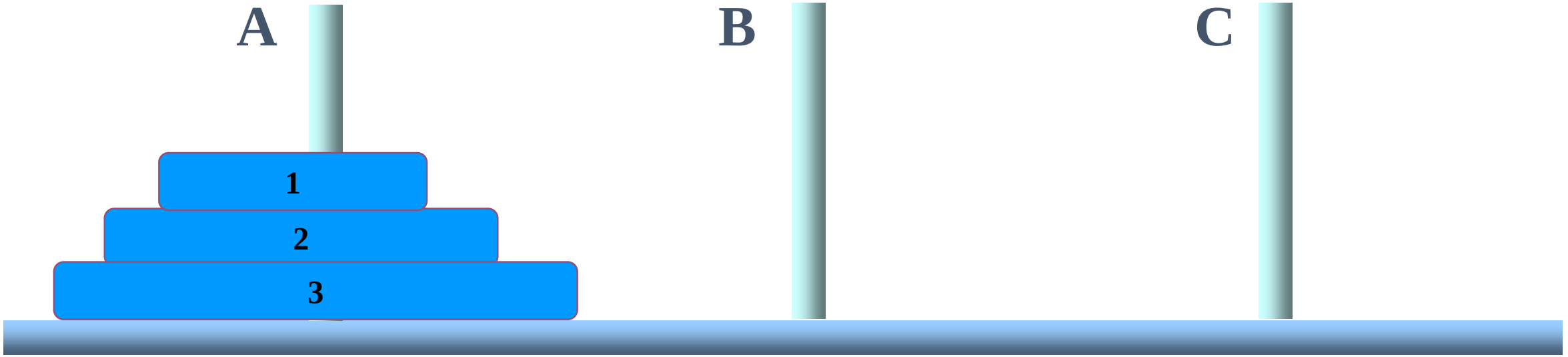
```
void hanoi(int n，char x，char y，char z)
{ if (n== 1)
    move(x，1，z)；//x塔上的1号盘移到Z塔，
  else {
    hanoi(n-1，x，z，y);
    move(x，n，z)；  //x塔上的n号盘移到Z塔，
    hanoi(n-1，y，x，z) ;
    }
} // hanoi
```



看 A 柱上有3只盘子hanoi(3, A, B, C)的情况：

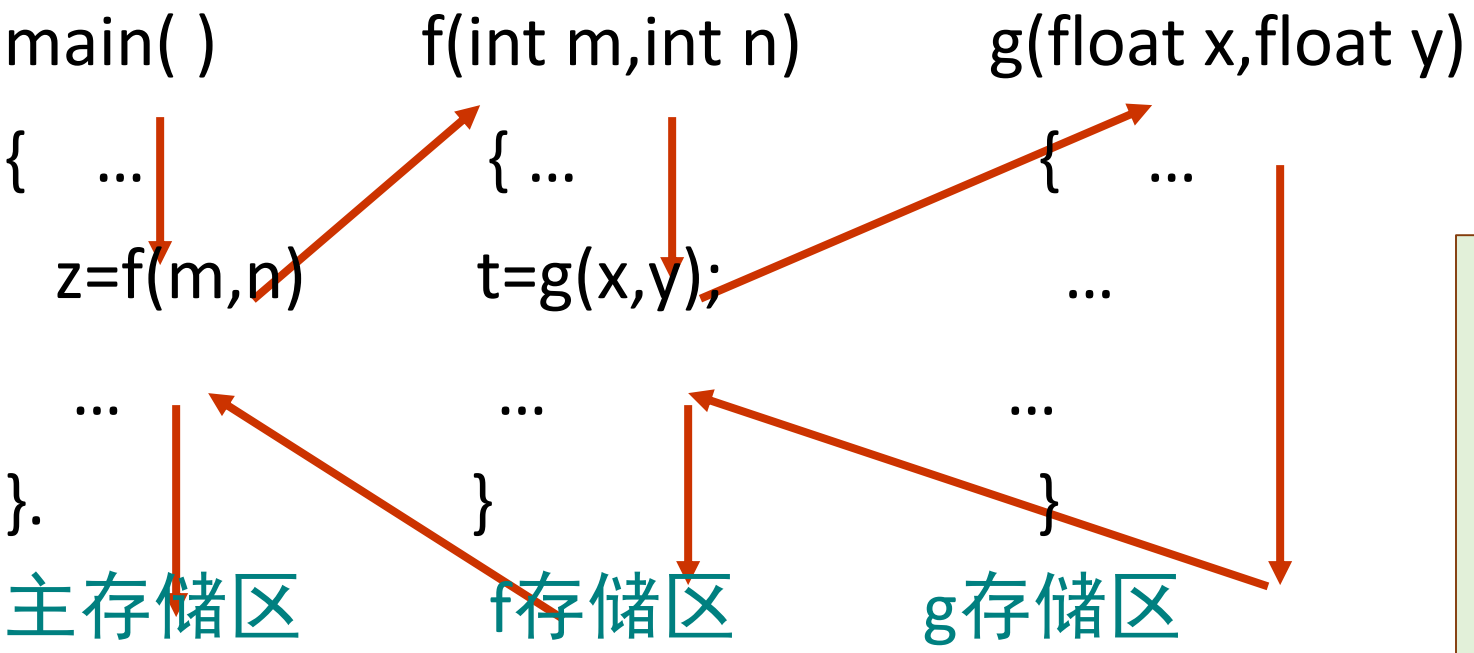


困难！



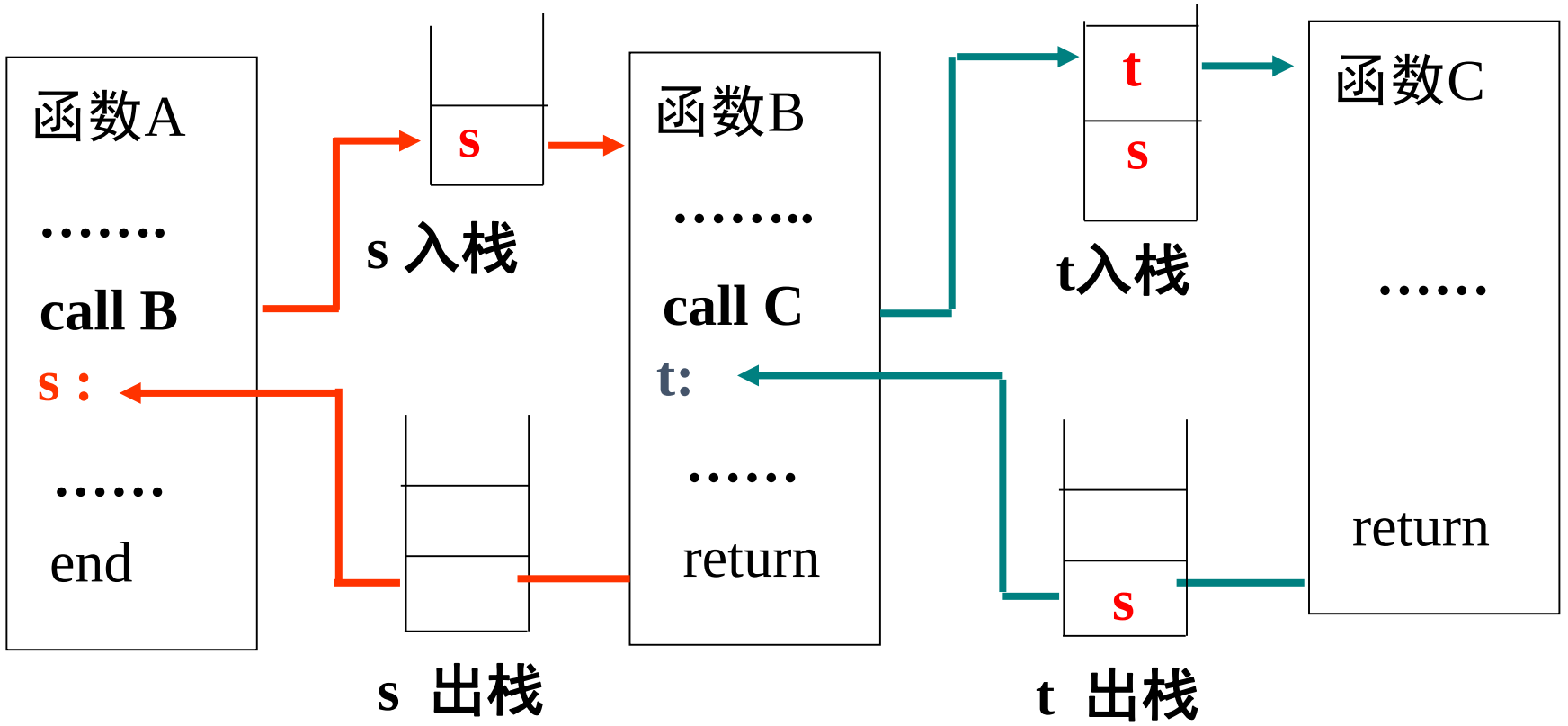
3. 6. 3 栈与函数调用

函数的(嵌套)调用



多个函数嵌套调用的规则是:**后调用的函数先返回**；此时的内存管理实行**“栈式管理”**

（1）保存地址和参数的工作栈



（2）如何实现函数调用

在高级语言编制的程序中，主调函数与被调用函数之间的信息交换必须通过栈来进行。当一个函数在运行期间调用另一个函数时，在运行该被调用函数之前，需先完成3件事：

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

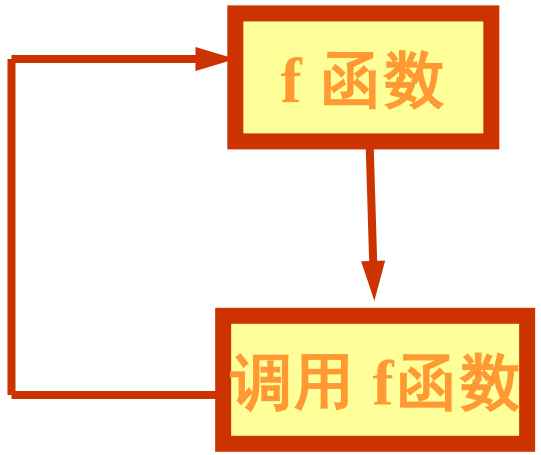
从被调用函数**返回**调用函数**之前**，应该完成：

1. **保存**被调函数的**计算结果**；
2. **释放**被调函数中局部变量的**存储区**
3. 依照被调函数保存的返回地址将**控制转移**到调用函数。

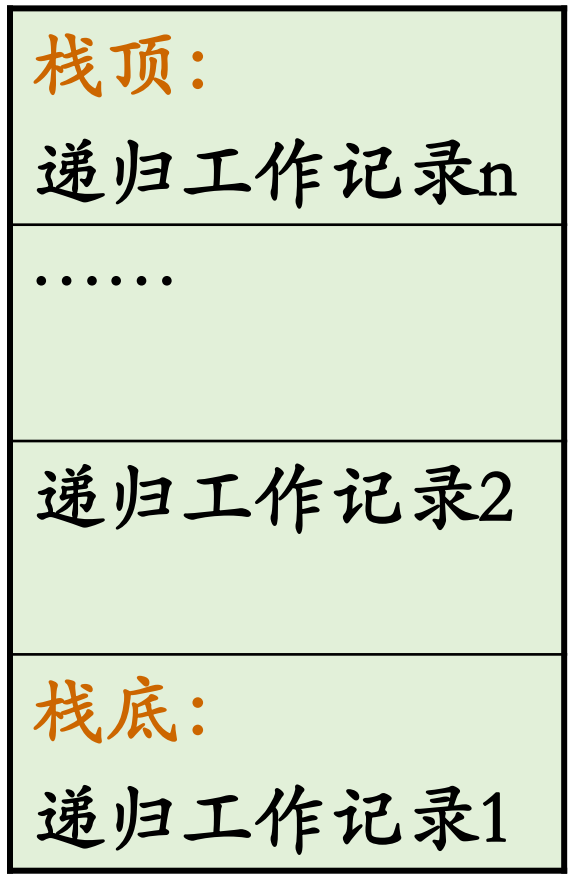
3.6.4 栈与递归函数

对于系统来说是如何进行递归调用的？

递归函数：是指在定义一个函数的过程中直接或间接地调用该函数本身



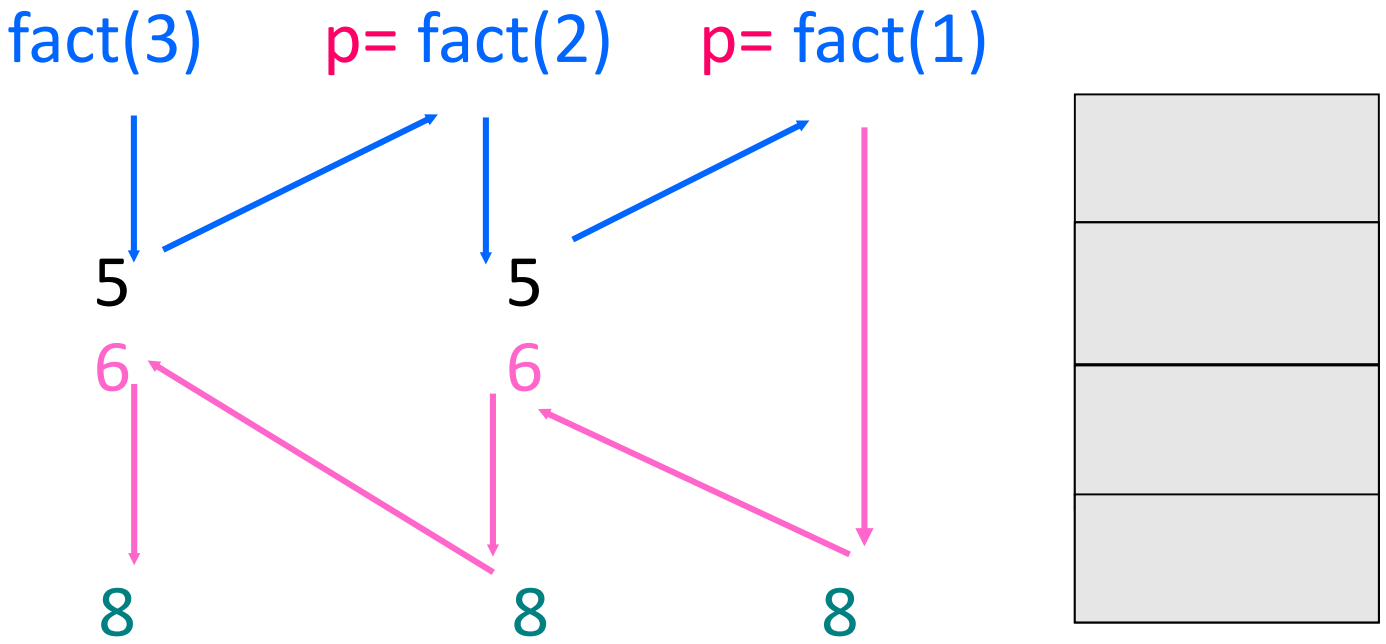
递归工作栈



递归工作记录的数据有：

1. 上一层函数调用的返回地址；
2. 局部变量（包括参数）表；

```
int  fact(int  n)
1:  {
2:  if (n≤1)
3:     retrun  1;
4:  else {
5:  p=  fact(n-1);
6:  return( n*p);
7:      }
8:  } //fact
```



看系统栈中的变化情况：(地址0表示调用该函数的主程序的返回地址)

```
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  } // hanoi
```

例如：调用方式：

hanoi (3, a, b, c) ;

1 a -> c

2 a -> b

1 c -> b

3 a -> c

```
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  } // hanoi
```

例如：调用方式：

```
hanoi (3, a, b, c) ;
```

1 a -> c

2 a -> b

1 c -> b

3 a -> c

1 b->a

```
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  } // hanoi
```

例如：调用方式：

```
hanoi (3, a, b, c) ;
```

1 a -> c

2 a -> b

1 c -> b

3 a -> c

1 b->a

```
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 } // hanoi
```

例如：调用方式：

```
hanoi (3, a, b, c) ;
```

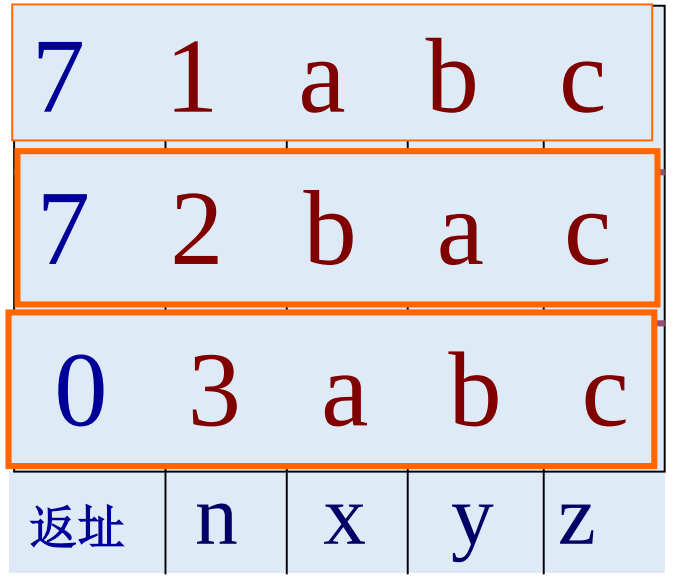
1 a -> c

2 a -> b

1 c -> b

3 a -> c

1 b->a



递归算法的特点

优点：

程序易于设计，程序结构简单精练；

缺点：

1） 有些递归算法较难理解，可读性差。

2） 程序运行速度慢，占较多的系统(栈)存储空间。

总结

写递归算法

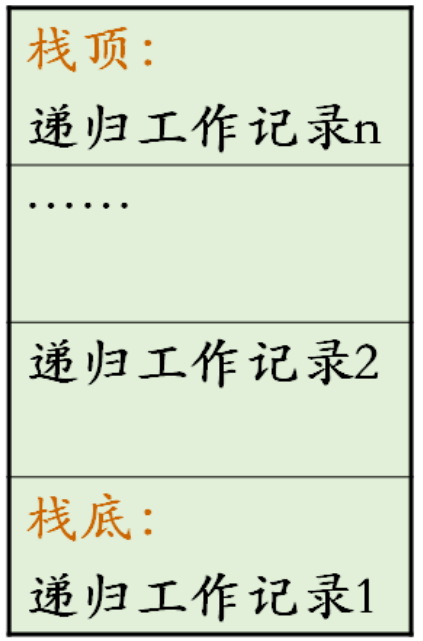
- ✓递归调用
- ✓出口条件

读递归程序

- ✓递归工作栈

```
int  fact(int  n){
 if (n<=1)   return 1;
  else {
        f= n* fact(n-1);
        printf (f);
   }
}
```

```
void hanoi(int  n，  char  x，  char y，  char z)
 { if (n== 1)
    move(x，  1，  z)；//x塔上的1号盘移到Z塔，
  else {
   hanoi(n-1，  x，  z，  y);
   move(x，  n，  z);   //x塔上的n号盘移到Z塔，
   hanoi(n-1，  y，  x，  z) ;
    }
} // hanoi
```



递归工作记录的数据有：

1. 上一层函数调用的返回地址；
2. 局部变量（包括参数）表；

练习

练1.读程序，描述功能

练2. 依次打印输出自然数1到n的递归函数

练3. 单链表的创建、求长度、正序输出、逆序输出

练1. 指出下列程序段的功能是什么

(1)

```
int i;arr[64];n=0;
while (!stackempty(s))
    arr[n++]=pop(s);
for(i=0;i<n;i++)
    push(s, arr[i]);
```

(2)

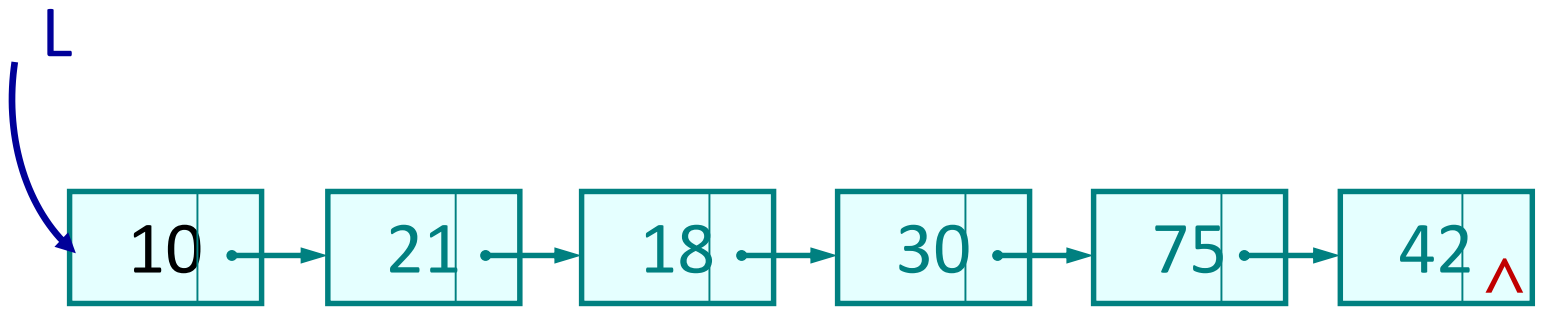
```
seqstack t; int i;
    initstack(t);
    while(! Stackempty(s))
        if((i=pop(s))!=m)
            push(t,i);
    While(! Stackempty(t)) {
        i=pop(t);
        push(s,i);}
```

练2. 依次打印输出自然数1到n的递归函数

```
void WRT(int n)
{
 if (n!=0)
  { WRT(n-1); printf("%d", n);}
}
```

练3. 单链表

写出单链表的创建，求长度的递归函数



创建

```
void create_List(LinkList &L, int n)
{
    if (n == 0)
    { L = NULL; return;}
    else{  //此时传进来的是第一个节点的指针
      L = new LNode;  //指针指向新生成的节点
      cin >> L->data;   //输入数据，将其压栈
     create_List(L->next, n - 1);
        //递归创建n个节点 }
 }
```

求表长

```
int count_LNode(LinkList L)
 if (L == NULL)
        return 0;
 else
     return
count_LNode(L->next) + 1};
```

输出

```
void PrintList(LinkList L){
 if (L == NULL)return;
  else{
      printf(L->data);
      PrintList(L->next);}}
```

逆序输出

```
void PrintLNode(LinkList L){
if (L == NULL)return;
else{
  PrintLNode(L->next);
  printf(L->data); }
}
```