第二章　栈和队列（1）

- 栈和队列也是线性表，只是操作受限的线性表。
- 由于操作上的限制，使他们的行为不同于一般的线性表，而有自己的特点。
  - 栈的定义
  - 栈的顺序存储与实现---顺序栈
  - 栈的链式存储与实现---链式栈
  - 队列的定义
  - 队列的顺序存储与实现
  - 队列的链式存储与实现
  - 栈与队列的应用

2.1 栈（Stack)

1、逻辑定义及特点

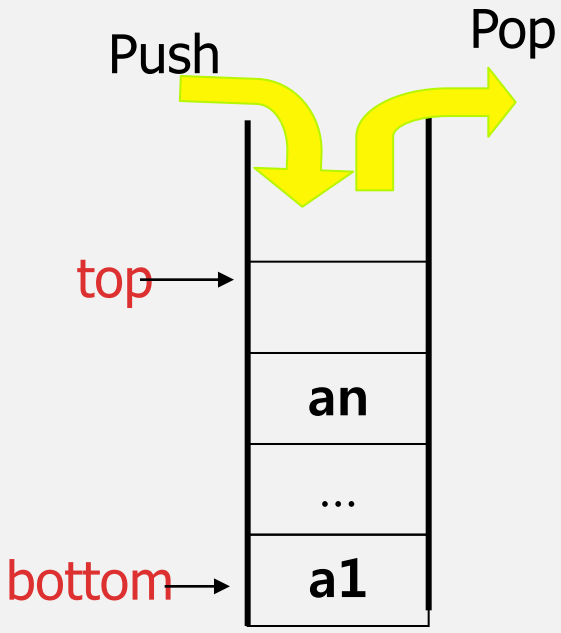
– 定义：只能在一端进行插入和删除操作的线性表。**(操作限制)**

- 允许插入和删除的一端称为**栈顶**(top)
- 另一端称为**栈底**(bottom)

- 特点：

  后进先出LIFO（Last In First Out）

  先进后出FILO（First In Last Out）



栈（Stack)

栈顶（Top)——栈中可插入、删除数据一端。

栈底（Bottom)——栈中不可插删入数据一端。

插入：进栈、压栈、入栈

删除：退栈、弹出、出栈

栈的基本操作：

1、创建一个空栈；

2、进栈

3、出栈

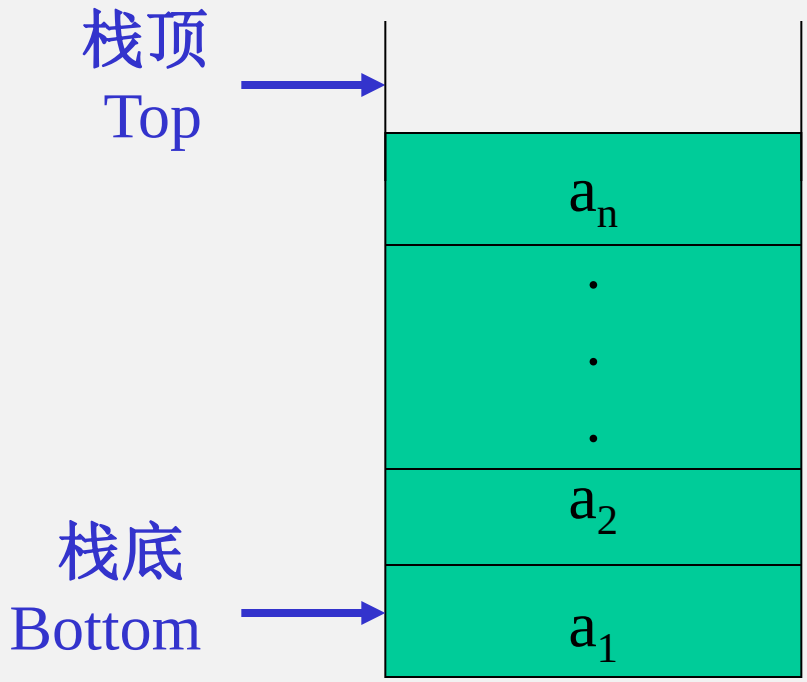
4、取栈顶元素

5、判断栈空、栈满、清空栈

栈的存储实现方式：

1、基于数组的栈----顺序栈

2、基于链表的栈----链式栈



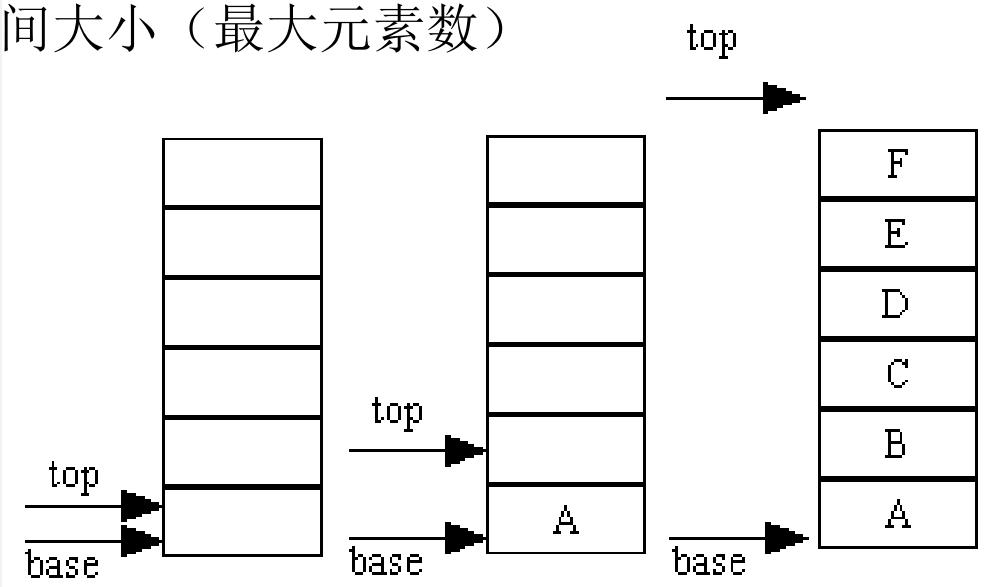
栈的顺序存储实现(顺序栈)

2、栈的顺序存储实现

（1）顺序栈定义（顺序存储结构上的C语言实现）

- **栈顶、栈底和栈大小要指示出来**

```
Typedef struct {
    ElemType *Base;  //栈底指针，也就是栈区指针
    int   Top;       // 栈顶位置，Top总是指向下一个待存位置
    int StackSize;   //已分配的空间大小（最大元素数）
} SeqStack;
SeqStack  SStack;  //定义栈变量
```



Top指向栈顶元素的下一个位置

栈的顺序存储实现(顺序栈)

（2）、创建一个空栈（栈的初始化）

```
bool  Init_SeqStack(SeqStack &S, int Ssize=DefaultSize) // 栈变量，栈区大小
{   // 动态分配栈的顺序存储空间
    S.Base = ( ElemType *)malloc( Ssize*sizeof(ElemType) );
    if (S.Base == NULL) return FALSE;       //栈区分配失败
    S.StackSize = Ssize;                    // 填写栈区大小
    S.Top =  0;                             // 填写栈顶初值
    return TURE;
}
```

调用方式： Init_SeqStack(SStack, 50)　其中：SStack为栈变量

算法复杂度为： O(1)

（3）、判断空栈

```
bool  Empty_SeqStack(SeqStack *S)           // 栈变量指针
{   if (S->Top == 0) return TURE;  //栈空，返回真值
    else  return  FALSE;                    // 栈非空，返回假
}
```

调用方式： Empty_SeqStack (&SStack) ;　　算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（4）、判断栈满

```
bool  Full_SeqStack(SeqStack *S)  // 栈变量指针
{   if (S->Top == S->StackSize)  return TURE;        //栈满，返回真值
    else  return  FALSE;                  // 栈未满，返回假
}
```

调用方式： Full_SeqStack (&SStack) ;　　算法复杂度为： O(1)

（5）、返回栈大小

```
int  Size_SeqStack(SeqStack *S)    // 栈变量指针
{   return  S->Top;          //返回栈中现有的元素个数
}
```

调用方式： Size_SeqStack (&SStack) ;　　算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（6）、取栈顶元素值（取栈顶---GetTop操作）

```
bool  GetTop_SeqStack(SeqStack *S, ElemType &x) // 栈变量指针，元素变量
{
    if ( Empty_SeqStack(S) ) return FALSE;  //返回取栈顶失败
    else {
            x = S->Base[S->Top-1] ;  //非空栈，取栈顶值
            return TURE;                    //返回取栈顶成功
    }
}
```

调用方式： GetTop_SeqStack (&SStack, y)　其中：SStack为栈变量，y为栈元素变量

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（7）、进栈（压入栈---Push操作）

```
bool  Push_SeqStack(SeqStack *S, ElemType x) // 栈变量指针，待存栈元素
{
    if ( Full_SeqStack(S) ) return FALSE;      //栈已满，入栈操作失败
    else {
            S->Base[S->Top] = x ;       //在栈顶空位存入元素
            S->Top++;                   // 调整栈顶位置
            return TURE;                //返回入栈成功
    }
}
```

调用方式： Push_SeqStack (&SStack, 85)　其中：SStack为栈变量

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（8）、出栈（弹出栈---Pop操作）

```
bool  Pop_SeqStack(SeqStack *S, ElemType &x) // 栈变量指针，弹出的栈元素
{
    if ( Empty_SeqStack(S) ) return FALSE;  //栈空，出栈操作失败
    else {
            x = S->Base[S->Top-1];    //取出栈顶元素
            S->Top--;                 // 调整栈顶位置
            return TURE;              //返回出栈成功
    }
}
```

调用方式： Pop_SeqStack (&SStack, y) 其中:SStack为栈变量,y为元素值变量

算法复杂度为： O(1)

栈的顺序存储实现(顺序栈)

（9）、清空栈（清除现有元素，回复空栈）

```
bool  Clear_SeqStack(SeqStack *S)          // 栈变量指针
{   S->Top = 0;                 //设置栈顶为0
    return  TURE;               // 返回清空成功，其实是不会失败的
}
```

调用方式： Clear_SeqStack (&SStack) ;　　算法复杂度为： O(1)

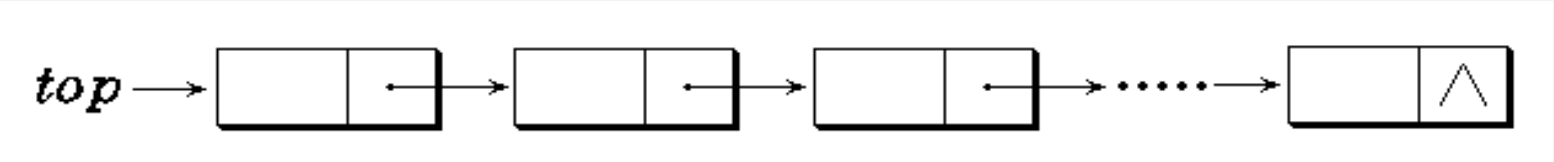
（10）、删除栈（删除元素，并释放栈空间）

```
bool  Delete_SeqStack(SeqStack *S)          // 栈变量指针
{   S->Top = 0;                 //清除元素
    free(S->Base);              //释放栈区存储空间
    return  TURE;               //返回成功
}
```

调用方式： Delete_SeqStack (&SStack) ;　　算法复杂度为： O(1)

栈的链式存储实现(链式栈)

3、栈的链式存储实现（一般采用单链表存储）

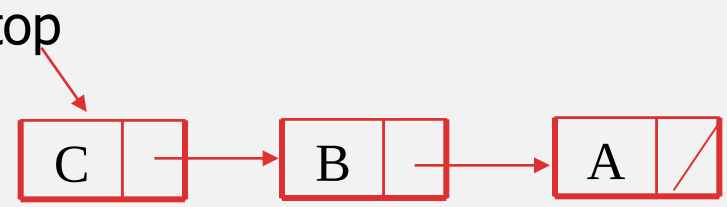


-仅需指示栈顶的指针

-空间分配灵活,充分利用空间

（1）链式栈定义（链表存储结构上的实现，不带头结点）

```
Typedef struct {
    Node *Top;          //栈顶指针，也就是栈链表的头指针; Node单链表结点
    int StackSize;      //当前栈中元素个数
} LinkStack;
LinkStack  LStack;  //定义栈变量
```



栈的链式存储实现(链式栈)

（2）、创建一个空栈（栈的初始化）

```
bool  Init_LinkStack(LinkStack *S ) // 链式栈变量
{
    S->Top = NULL;                  // 栈顶指针置为空
    S->StackSize =  0;              // 填写栈计数初值
    return TURE;
}
```

调用方式： Init_LinkStack(&LStack)　其中：LStack为链式的栈变量

算法复杂度为： O(1)

（3）、判断空栈

```
bool  Empty_LinkStack(LinkStack *S)        // 栈变量
{   if (S->Top == NULL) return TURE;       //栈空，返回真值
    else  return  FALSE;                   // 栈非空，返回假
}
```

调用方式： Empty_LinkStack (&LStack) ;　　算法复杂度为： O(1)

栈的链式存储实现(链式栈)

（4）、返回栈大小

```
int  Size_LinkStack(LinkStack *S)  // 栈变量指针
{    return  S->StackSize;   //返回栈中现有的元素个数
}
```

调用方式： Size_LinkStack (&LStack) ;　　**算法复杂度为**： O(1)

（5）、取栈顶元素值（取栈顶---GetTop操作）

```
bool  GetTop_LinkStack(LinkStack *S, ElemType &x) //栈变量指针,栈元素变量
{    if ( Empty_LinkStack(S) ) return FALSE;          //返回取栈顶失败
     else  x = S->Top->data;                  //非空栈，取栈顶值
     return TURE;                             //返回取栈顶成功
}
```

调用方式： GetTop_LinkStack (&LStack, y)　其中：LStack为栈变量，y为栈元素变量

算法复杂度为： O(1)

栈的链式存储实现(链式栈)

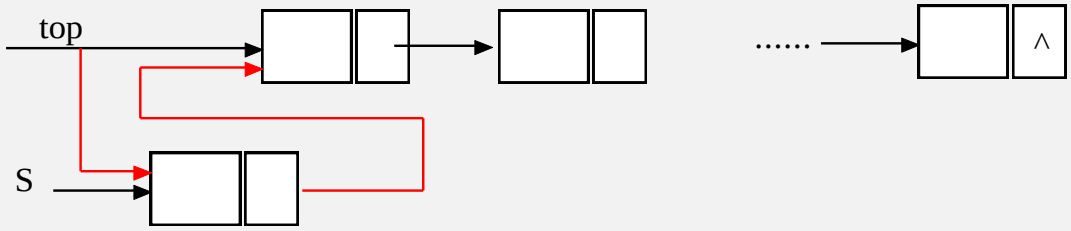
（6）、进栈（压入栈---Push操作）

```
bool  Push_LinkStack(LinkStack *S, ElemType x) // 栈变量指针，待存栈元素
{    Node *p;
     p = (Node *)malloc( sizeof(Node) );
     if ( !p ) return FALSE;   //申请分配结点失败，入栈操作失败
     p->data = x;              //在新节点存入元素
     p->next = S->Top;
     S->Top = p ;
     S->StackSize++;           // 调整栈大小
     return TURE;              //返回入栈成功
}
```

调用方式： Push_LinkStack (&LStack, 85)

其中：LStack为栈变量

算法复杂度为： O(1)



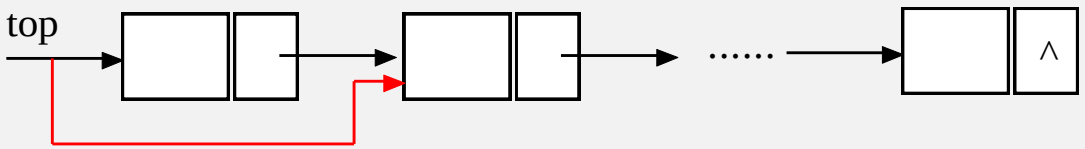
栈的链式存储实现(链式栈)

（7）、出栈（弹出栈---Pop操作）

```
bool  Pop_LinkStack(LinkStack *S, ElemType &x) // 栈变量，待删栈元素取值
{   Node *p;
    if ( !S->Top ) return FALSE;       //出栈操作失败
    p = S->Top;                  //将栈顶元素结点摘除
    S->Top = p->next;
    x = p->data;                  //取出栈顶元素
    free( p ) ;                   //释放栈顶元素结点
    S->StackSize--;              // 调整栈大小
    return TURE;                 //返回入栈成功
}
```

调用方式： Pop_LinkStack (&LStack, y)　其中：LStack为栈变量

算法复杂度为： O(1)



栈的链式存储实现(链式栈)

（8）、清空栈（清除现有元素，释放结点，回复空栈）

```
bool  Clear_LinkStack(LinkStack *S)          // 栈变量指针
{   Node *p, *q;
    p=S->Top;                  //让p指向栈顶
    while(p){                  //链式栈不空
          q = p;               //保存待删除结点
          p = p->next;         //p指针后移
          free(q);             //释放被删除结点空间
    }
    S->Top = NULL;             //设置栈顶指针为NULL
    S->StackSize = 0;          //栈计数归零
    return  TURE;              // 返回清空成功
}
```

调用方式： Clear_LinkStack (&LStack) ;

算法复杂度为： O(1)

4、栈的应用

a) **To transform an decimal(十进制) number to octonary(八进制) number.**

$$(N)_{10} = (M)_d = (b_{m-1}b_{m-2}\cdots b_1 b_0)_d$$

$$= b_{m-1}\times d^{m-1} + b_{m-2}\times d^{m-2} + \cdots + b_1\times d^1 + b_0\times d^0$$

$$= (b_{m-1}\times d^{m-2} + b_{m-2}\times d^{m-3} + \cdots + b_1)\times d + b_0$$

b) **To determine if the parentheses in expression are balanced and properly nested.**

c) **text edit:**