2.2.1什么是堆栈

后缀表达式求值策略：从左往右扫描，逐个处理运算数和运算符号

1. 遇到运算数怎么办；如何“记住”目前还不参与运算的数
2. 遇到运算符号怎么办；对应的运算数是什么

启示：需要有种存储方法，能顺序存储运算数，并在需要的时候倒序取出

$$T(N) = O(N)$$

堆栈的抽象数据类型描述

堆栈（stack）：具有一定操作约束的线性表

只在一端（栈顶，Top）做插入、删除

插入数据：入栈（Push）

删除数据：出栈（Pop）

后入先出：Last In First Out(LIFO)

类型名称：堆栈（Stack）

数据对象集：一个有0或多个元素的有穷线性表

操作集：长度为MaxSize的堆栈 $S \in Stack$，堆栈元素 $item \in ElementType$

1. `Stack CreateStack(int MaxSize)`:生成空堆栈，其最大长度为MaxSize
2. `int IsFull(Stack S, int MaxSize)`:判断堆栈S是否已满
3. `void Push(Stack S, ElementType item)`:将元素压入堆栈
4. `int IsEmpty(Stack S)`:判断堆栈是否为空
5. `ElementType Pop(Stack S)`:删除并返回栈顶元素

push和pop可以交替进行

2.2.2堆栈的顺序存储实现

栈的顺序存储实现

栈的顺序存储结构通常由一个一维数组和一个记录栈顶元素位置的变量组成

```c
#define MaxSize
typedef struct SNode *Stack;
struct SNode
{
    ElmentType Data[MaxSize];
    int Top;
};

/* （1）入栈 */
void Push(Stack PtrS,ElementType item)
{
    if(PtrS->Top==MaxSize-1)
    {
        printf("栈堆满");
        return;
    }
    else
    {
        PtrS->Data[++(PtrS->Top)]=item;//++p是先将指针p自加1，再取出p所指向的值；p++是先取出p指向的值，再将p加1
        return;
    }
}

/* （2）出栈 */
ElementType Pop(Stack PtrS)
{
    if(PtrS->Top==-1)
    {
        printf("堆栈空");
        return ERROR;//ERROR为ElementType的特殊值，标志错误
    }
    else
    {
        return (PtrS->Data[(PtrS->Top)--]);
    }
}
```

例：使用一个数组实现两个堆栈，要求最大地利用数组空间，是数组只要由空间入栈操作就可以成功。

【分析】一种比较聪明的方法是这两个栈分别从数组的**两头开始向中间增长**；当两个栈的**栈顶指针**相遇时，标识两个栈都满了。

```
#define MaxSize
struct DStack
{
    ElementType Data[MaxSize];
    int Top1;
    int Top2;
}S;
S.Top1=-1;
S.Top2=MaxSize;

void Push(struct DStack *PtrS,ElementType item,int Tag)
{
    if(PtrS->Top2-PtrS->Top1==1)
    {
        printf("堆栈满");
        return ;
    }
    if(Tag==1)
    {
        PtrS->Data[++(PtrS->Top1)]=item;
    }
    else
    {
        PtrS->Data[--(PtrS->Top2)]=item;
    }
}

ElementType Pop(struct DStack *PtrS,int Tag)
{
    if(Tag==1)
    {
        if(PtrS->Top1==-1)
        {
            printf("堆栈1空");
            return NULL;
        }
        else
            return PtrS->Data[(PtrS->Top1)--]
    }
    else
    {
        if(PtrS->Top2==MaxSize)
        {
            printf("堆栈2空");
            return NULL;
        }
        else
            return PtrS->Data[(PtrS->Top2)++]
    }
}
```

2.2.3堆栈的链式存储实现

堆栈的链式存储实现

栈的链式存储结构实际上就是一个**单链表**，叫做**链栈**。插入和删除操作只能在链栈的栈顶进行。**栈顶指针Top应该在链表的哪一头？**

单向链表的Top只能是头结点

如果将尾结点当作是Top，会出现删除后无法找到最后一个结点之前的结点（由于是单向链表）

堆栈的链式存储实现

```c
typedef struct SNode *Stack;
struct SNode
{
    ElementType Data;
    struct SNode *Next;
};

Stack CreateStack()
{/* 构建一个堆栈的头结点，返回指针 */
    Stack S;
    S=(Stack)malloc(sizeof(struct SNode));
    S->Next=NULL;
    return S;
}

int IsEmpty(Stack S)
{/* 判断堆栈S是否为空，若为空则返回整数1，否则返回0 */
    return (S->Next==NULL);
}

/* 1.堆栈初始化（建立空栈） */
/* 2.判断堆栈S是否为空 */

void Push(ElementType item,Stack S)
{/* 用链表实现堆栈的时候，不需要考虑堆栈是否满 */
    struct SNode *TmpCell;
    TmpCell=(struct SNode *)malloc(sizeof(struct SNode));
    TmpCell->Element=item;
    TmpCell->Next=S->Next;
    S->Next=TmpCell;
}

ElementType Pop(Stack S)
{
    struct SNode *FirstCell;
    ElementType TopElem;
    if(IsEmpty(S))
    {
        printf("堆栈空");
        return NULL;
    }
    else
    {
        FirstCell=S->Next;
        S->Next=FirstCell->Next;
        TopElem=FirstCell->Element;
        free(FirstCell);
        return TopElem;
    }
}
```

2.2.4堆栈的应用：表达式求值

堆栈应用：表达式求值

回忆：应用堆栈实现后缀表达式求值的基本过程：

从左到右读入**后缀表达式的各项**（运算符或运算数）

1. 运算数：入栈；
2. 运算符：从堆栈中弹出适当数量的运算数，计算结果并入栈；
3. 最后，堆栈顶上的元素就是表达式的结果值。

中缀表达式求值

基本策略：将中缀表达式转换为后缀表达式，然后再求值。

1. 运算数相对顺序不变
2. 运算符号顺序发生改变
    1. 需要存储“等待中”的运算符号
    2. 要将当前运算符号与“等待中”的最后一个运算符号比较
3. “等待中”的优先级如果更高，则可以开始计算。

堆栈！

括号怎么办？

左括号在堆栈外时：优先级最高

左括号在堆栈内时：优先级最低

- 碰到右括号的时候，把堆栈中的操作一个一个抛出来，直到遇到左括号为止。
- 优先级相同时，先抛出堆栈内的，再将操作入栈。
- 结束后也抛出。

中缀表达式如何转换为后缀表达式 从头到尾读取**中缀表达式的每个对象**，对不同对象按不同情况处理

1. **运算数**：直接输出
2. **左括号**：压入堆栈
3. **右括号**：将**栈顶的运算符**弹出并**输出**，**直到遇到左括号**（出栈，不输出）PS：括号是不输出的，即后缀表达式中不应该出现括号。
4. **运算符**
    1. 若**优先级大于栈顶运算符**，则将其压栈
    2. 若**优先级小于等于栈顶运算符**，先将**栈顶运算符弹出并输出**（等于时，要先弹出）；再比较新的栈顶运算符，直到该运算符大于栈顶运算符优先级为止，然后将该**运算符压栈。**
5. 若各对象**处理完毕**，则把堆栈中存留的**运算符一并输出。**

堆栈的其他应用：

- 函数调用及递归实现
- 深度优先搜索
- 回溯算法