一、引入

堆栈存储结构的用途：存储计算机认识的运算式

二、堆栈的抽象数据类型描述

生成空堆栈

判断堆栈是否为空

判断堆栈是否满了

插入数据：push 入栈

删除数据： pop 出栈

特点：后入先出 LIFO

三、堆栈结构实现

3.1顺序存储结构

用一维数组和一个记录栈顶的整型变量

（1）入栈

```
void push (stack ptrs ,elementtype  item)
{
if(ptrs->top=maxsize-1)表示栈满
else ptrs->data[++ptrs->top]=item;
}
```

(2)出栈

```
elementtype pop(stack ptrs)1
{
判断栈是不是空栈
else return ptrs->data[ptrs->top--]
}
```

3.2链式存储实现

从链表的头节点处进行出栈入栈操作

头节点没有数据域，只起到辅助头节点的下一节点出栈入栈

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

(1) 堆栈初始化（建立空栈）
(2) 判断堆栈S是否为空

（1）建立空栈，返回头节点

（2）判断堆栈是否为空 s->next==null

(3)入栈

```
stack push(elementtype item,stack s)
{
stack node=(stack)malloc(sizeof(struct snode));
node->data=item;
node->next=s->next;
s->next=node;}
```

(4)出栈

```
elementtype pop(elementtype item,stack s)
{
```

注：先判断是否是空栈

```
if(isempty(s))...
else
```

```
stack p=s->next;
s->next=p->next;
elementtype topelm=p->data;
free p;
return topelm;

}
```

四、堆应用：前中后缀表达式相互转换

手算：

中缀转前缀：右优先，从右往左搜索，按照先加减后乘除有括号先算括号里面的原则找第一个能生效的运算符

前缀转中缀：从右往左搜索，碰到一个运算符就取其前两个操作数进行运算，并得到一个新操作数放在原位置

中缀转后缀：左优先，从左往右搜索，按照先加减后乘除有括号先算括号里面的原则找第一个能生效的运算符

后缀转中缀：从左往右搜索，碰到一个运算符就取其前两个操作数进行运算并加上括号，并得到一个新操作数放在原位置，同时也计算出了表达式的结果

机算：

后缀转中缀：从左往右扫描，遇到操作数进栈，遇到运算符弹出两个操作数进行机算，并且先弹出的操作数在运算符右侧

前缀转中缀:从右往左扫描,遇到操作数进栈，遇到运算符弹出两个操作数进行机算，并且先弹出的操作数在运算符左侧

中缀转后缀：设运算符栈，从左往右搜索，当遇到操作数就输出，遇到运算符时依次弹出栈内优先级比他高的运算符，这个过程中遇到栈空或者碰到左括号就停止，然后将该运算符入栈，遇到左括号入栈，遇到右括号一直出栈直到弹出左括号（左括号不输出）

中缀转前缀：

用栈实现中缀表达式的计算

同时进行中缀转后缀和后缀计算

设运算符栈和操作数栈，遇到操作数就入栈，遇到操作符就按中缀转后缀的操作，并且弹出一个运算符时要依次从操作数栈弹出两个数进行运算，先弹出的放在右边，后弹出的放在左边