第三章　栈和队列

一、栈

（一）概述

1．栈：只允许在一端进行插入和删除操作的线性表。

2．栈顶：允许进行添加或删除元素的一端

3．栈底：不允许进行添加或删除元素的一端

4．空栈：没有元素的栈

5．入栈（进栈/压栈）（*push*）：栈的插入运算

6．出栈（退栈/弹出）（*pop*）：栈的删除运算

7．栈的特点：后进先出（LIFO）

在栈中插入和删除的示例

（二）栈的顺序存储结构

1．栈类型的定义

```
#define  STACKMAX 100
typedef struct{
  int top;
  int data[STACKMAX];
}SqStack;
```

2．栈的初始化

top 值为下一个进栈元素在数组中的下标值，入栈操作是先压栈再移动栈顶指针，出栈操作是先移动栈顶指针再取出元素

1）栈空：top= =0(当 top= =-1，top 指向栈顶元素）

2）栈满：top= =STACKMAX（栈空为-1 时，栈满为 STACKMAX-1）

3．进栈与出栈顺序

若进栈序列为a、b、c，则全部可能出栈的序列为（’→’表示进，’←’表示出）：

①a→ a← b→ b← c→ c← 即abc

②a→ b→ b← c→ c← a← 即bca

③a→ a← b→ c→ c← b← 即acb

④a→ b→ b← a← c→ c← 即bac

⑤a→ b→ c→ c← b← a← 即cba

（三）栈的链式存储结构

1．链栈：用线性链表表示的栈

2．栈顶指针：链栈的表头指针

3．栈的变化

1）栈空：top==NULL(带头结点：top->next=NULL）

2）非空：top指向链表的第一个结点，栈底结点的指针域为空

- 无栈满情况

4．结点类型

```
typedef struct Linknode
{int data;
 struct Linknode *next;}*LiStack;
```

（四）栈的应用

1.表达式求值（★重点★）

1）算术表达式的中缀表示：运算符放在两个操作数中间的算术表达式

例：a+b*c+d/e*f

2）中缀表示在计算机处理中的的问题

受括号、优先级和结合律的限制，不一定按运算符出现的先后顺序执行，完成运算需对表达式进行多遍扫描，浪费时间。

3）算术表达式的后缀表示（逆波兰表示法：运算符在操作数后面）

例：a+b→ ab+ a*b→ ab*

算术表达式的前缀表示（运算符在操作数前面）

例：a+b→ +ab a*b→ *ab

4）中缀表达式转后缀表达式

①确定中缀中各个运算符的运算顺序。（左优先原则：只要左边的运算符能先计算，优先左边的）

②选择执行顺序最高的运算符，按照[左操作数 右操作数 运算符]的方式组成一个新的操作数。

③若还有运算符没处理，则继续第二步

【例】a+b*c-d/(e*f)→ a+b*c-d/ef*→ a+bc*-d/ef*→a+bc*-def*/→ abc*+-def*/→ abc*+def*/-

5）后缀表达式的求值

从左→右扫描，遇到运算符将该运算符前面两个数取出运算，合体为一个操作数，结果带回后缀表达式参与后面的运算，直到扫描到最后一个运算符并计算得最终结果

例：345/6*+2-→ 3（4/5）6*+2-→ 3（（4/5）*6）+2-→（3+（（4/5）*6））2-→（3+（（4/5）*6））-2=5.8

6）中缀表达式转前缀表达式

①确定中缀中各个运算符的运算顺序。（右优先原则：只要右边的运算符能先计算，优先右边的）

②选择执行顺序最高的运算符，按照[运算符 左操作数 右操作数]的方式组成一个新的操作数。

③若还有运算符没处理，则继续第二步

7）前缀表达式的求值

从右→左扫描，遇到运算符将该运算符后面两个数取出运算，合体为一个操作数，结果带回前缀表达式参与后面的运算，直到扫描到最后一个运算符并计算得最终结果

例：+3-*/4562→ +3-*（4/5）62→ +3-（（4/5）*6）2→+3→ +3（（（4/5）*6）-2）=3+（（（4/5）*6）-2）=5.8

二、队列

（一）概述

1．队列：只允许在表的一端进行插入操作而在另一端进行删除操作的线性表

2．队尾：允许进行插入操作的一端

3．队首：允许进行删除操作的一端

4．特点：先进先出（FIFO）

（二）队列的顺序存储结构

1.顺序队列的变化

front 指向队头元素，rear 指向队尾元素的下一个位置（不同教材对 front，rear 的定义可能不同），入队操作是先插入元素，再队尾指针加 1；出队操作是先取出元素，再队首指针加 1。

空：front==rear==0

2.假溢出

1）假满：当 rear== QUEUEMAX 时，数组前端有空闲单元，出现假满

2）解决方案：

①一个元素出队时，将所有元素依次向队首方向移动一个位置，修改头尾指针，使空闲单元留在后面，此方案花费时间较多

②出队时队列不移动，当出现假溢出时，将所有元素依次向队首方向移动，此方案也要花费较多时间

③把队列看作是一个首尾衔接的循环队列，这是最有效的解决方案

（三）循环队列

1．特点：存储队列元素的表从逻辑上视为一个环

1）空队：front==rear

2）队满：front==rear

3．问题：当队列空和满时条件相同，无法判定队列是空还是满

4．最好的解决方案

设置一个空闲单元，则可用单元为 QUEUEMAX-1 个，当判别队列是否满时，确定 rear 的下一个单元的位置是否为 front 所指的单元

队满条件：(rear+1)% QUEUEMAX==front

队空条件：rear==front

队列长度：（rear+ QUEUEMAX-front）%QUEUEMAX

5．循环队列指针的移动

插入或删除元素时，指针按顺时针方向进 1

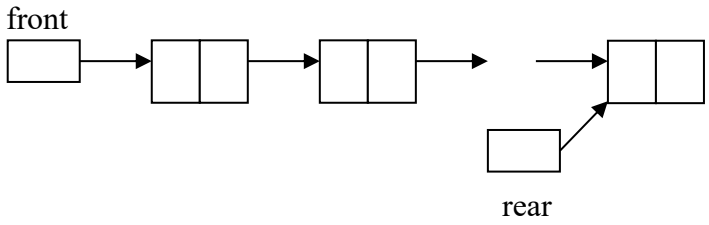
队首指针进 1：front=(front+1)% QUEUEMAX

队尾指针进 1：rear=(rear+1)% QUEUEMAX

（四）队列的链式存储结构

1．链队：用线性链表表示的队列

2．结构形式



3．操作：在尾结点后插入，在首结点处删除

4．结点类型：

```
typedef struct node
{char data;
 struct node *next;}QNODE;
```

5．链队的指针：QNODE *front, *rear;

6．队列的变化

1）队空：front==rear==NULL

2）非空：rear->next==NULL