**第3章栈和队列作业**

**一、填空题**

**1. 向量(线性表)、栈和队列都是 线性 结构，可以在向量的 任何 位置插入和删除元素；对于栈只能在 栈顶 插入和删除元素；对于队列只能在 队尾 插入和 队首 删除元素。**

**2. 栈是一种特殊的线性表，允许插入和删除运算的一端称为 栈顶 。不允许插入和删除运算的一端称为 栈底 。**

**3. 队列 是被限定为只能在表的一端进行插入运算，在表的另一端进行删除运算的线性表。**

**4. 在具有n个单元的循环队列中，队满时共有 n-1 个元素。**

**5. 向栈中压入元素的操作是先 移动栈顶指针 ，后 存入该元素 。（现存后移动）**

**6. 从循环队列中删除一个元素时，其操作是 先 移动队首指针 ，后 取出元素 。**

**7. 带表头结点的空循环双向链表的长度等于 0 。**

**二、判断正误（判断下列概念的正确性，并作出简要的说明。）**

**（ 错 ）1. 线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型。**

线性表只是逻辑结构，不管是顺序存储还是链式都可以复杂类型

**（ 错 ）2. 在表结构中最常用的是线性表，栈和队列不太常用。**

栈和队列是特殊的线性表，且经常被函数调用

**（ 对 ）3. 栈是一种对所有插入、删除操作限于在表的一端进行的线性表，是一种后进先出型结构。**

定义

**（ 对 ）4. 对于不同的使用者，一个表结构既可以是栈，也可以是队列，也可以是线性表。**

都是线性结构

**（ 错 ）5. 栈和链表是两种不同的数据结构。**

栈是逻辑结构，是线性表的一种，可以用链表实现，链表是存储结构

**（ 错 ）6. 栈和队列是一种非线性数据结构。**

都是特殊的线性表，是线性结构

**（ 对 ）7. 栈和队列的存储方式既可是顺序方式，也可是链接方式。**

可以顺序存储也可以链表

**（ 对 ）8. 两个栈共享一片连续内存空间时，为提高内存利用率，减少溢出机会，应把**两个栈的栈底分别设在这片内存空间的两端。

**（ 错 ）9. 队是一种插入与删除操作分别在表的两端进行的线性表，是一种先进后出型结构。**

队先进先出

**（ 错 ）10. 一个栈的输入序列是12345，则栈的输出序列不可能是12345。**

1入栈出栈2入栈出栈3入栈出栈4入栈出栈5入栈出栈就是12345

**三、单项选择题**

**（ b ）1. 栈中元素的进出原则是**

**Ａ．先进先出 Ｂ．后进先出 Ｃ．栈空则进 Ｄ．栈满则出**

定义**（ c ）2. 若已知一个栈的入栈序列是1，2，3，…，n，其输出序列为p1，p2，p3，…，pn，若p1=n，则pi为**

**Ａ．i Ｂ．n=i Ｃ．n-i+1 Ｄ．不确定**

因为n已经出栈了，而由题可知1-n-1已经按顺序全部在栈中了，所以出栈顺序只有n,n-1,...2,1一种，所以pi是n-i+1

**（ b ）3. 判定一个栈ST（最多元素为m0）为空的条件是**

**Ａ．ST->top<>0 Ｂ．ST->top=0 Ｃ．ST->top<>m0 Ｄ．ST->top=m0**

top指向栈顶元素存储的下一个存储单元的位置

**（ a ）4. 判定一个队列QU（最多元素为m0）为满队列的条件是**

**Ａ．QU->rear － QU->front = = m0**

**Ｂ．QU->rear － QU->front －1= = m0**

**Ｃ．QU->front = = QU->rear**

**Ｄ．QU->front = = QU->rear+1**

普通队列队满条件是元素个数为m0，约定满队时队首指针与队尾指针相差1,不用减1

**四、简答题**

**1、简述队列和堆栈这两种数据类型的相同点和差异处**

同：都是线性结构，都是逻辑结构的概念。都可以用顺序存储或链表存储；栈和队列是两种特殊的线性表，受限的线性表，只是对插入、删除运算加以限制。

异：栈限制是仅允许在表的一端进行插入和删除运算,LIFO,多用于子程序调用

队列限制是仅允许在表的一端进行插入,而在表的另一端进行删除,FIFO，多用于作业排队。

**2、写出以下程序段的输出结果（队列中的元素类型QElemType为char）**

**void main()**

**{**

**Queue Q;**

**InitQueue(Q);**

**char x= 'e'；y= 'c';**

**EnQueue(Q，'h');**

**EnQueue(Q，'r');**

**EnQueue(Q，y);**

**DeQueue(Q，x);**

**EnQueue(Q， x);**

**DeQueue(Q， x);**

**EnQueue(Q， 'a');**

**While(!QueueEmpty(Q))**

**{**

**DeQueue(Q，y);**

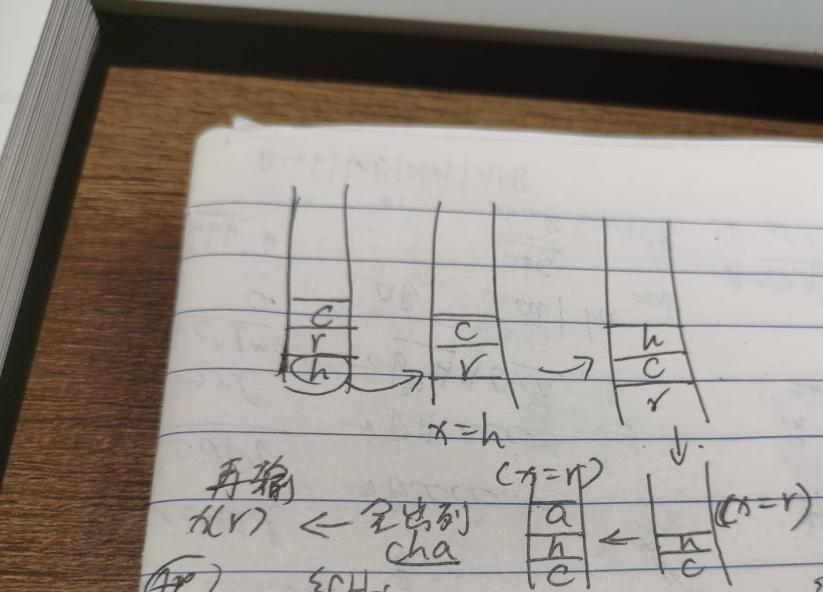
**Printf(y);**

**}**

**Prinntf(x);**

**}**

结果：char

****

**3、简述以下算法的功能（栈和队列的元素类型均为int）**

**void algo3(Queue &Q)**

**{**

**Stack S;**

**int d;**

**InitStack(S);**

**while(!QueueEmpty(Q))**

**{**

**DeQueue(Q,d);**//先将队列的值全部放入栈中

**Push(S,d);**

**}**

**while(!StackEmpty(S))**

**{**

**Pop(S, d);**//因为栈LIFO所以队列中数组逆置

**EnQueue(Q, d);**

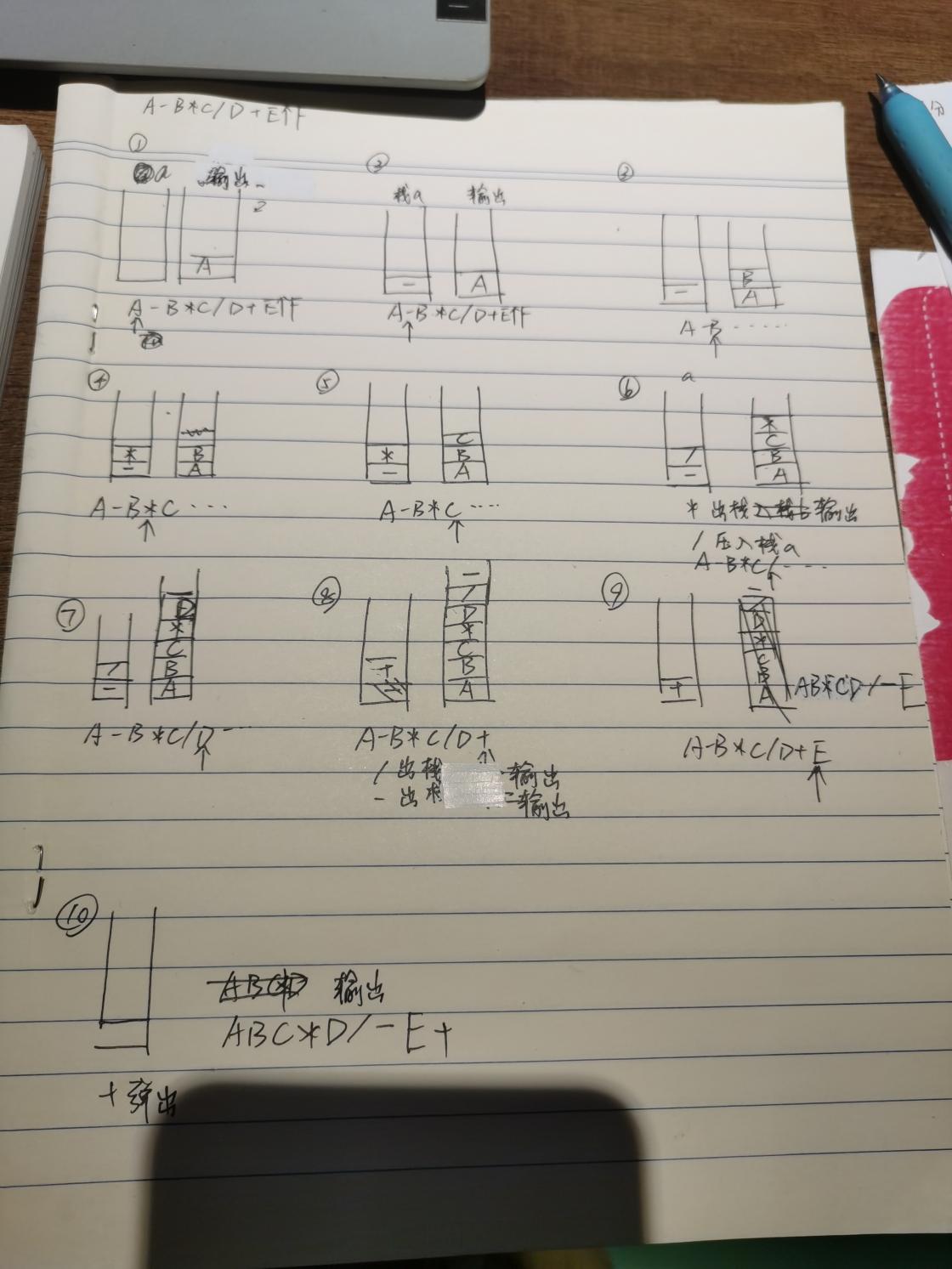
**}**

**}**

用栈逆置队列

4、**按照四则运算加、减、乘、除和幂运算（↑）优先关系的惯例，画出将下列算术表达式变为后缀表达式栈及输出串的过程：**

**A－B＊Ｃ／Ｄ＋Ｅ↑Ｆ**

****

**#include<cstdio>**

**#include<cstdlib>**

**#include<cstring>**

**#include<iostream>**

**#include<cmath>**

**#define MAXSIZE 10086**

**using namespace std;**

**typedef struct**

**{**

**char data[MAXSIZE];**

**int top;//栈顶指针**

**}sqstack;**

**sqstack s1;**

**int k;//测试组数**

**int len;**

**int i;**

**char inf[100086];**

**//判断优先级 (+-<\*/)**

**int pri(char a)**

**{**

**if(a=='+'||a=='-')**

**{**

**return 1;**

**}**

**else if(a=='/'||a=='\*')**

**{**

**return 2;**

**}**

**}**

**//判断是否为操作数**

**int op(char a)**

**{**

**if(a=='+'||a=='-'||a=='/'||a=='\*')**

**{**

**//cout<<a<<endl;**

**return 1;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**//初始化栈**

**void inits(sqstack &s)**

**{**

**s.top=-1;**

**}**

**//判断栈是否为空**

**bool emptys(sqstack &s)**

**{**

**if(s.top==-1)**

**return true;**

**else**

**return false;**

**}**

**//入栈**

**bool pushs(sqstack &s,char &x)**

**{**

**if(s.top==MAXSIZE-1)//栈满报错**

**{**

**return false;**

**}**

**s.top++;//指针先加**

**s.data[s.top]=x;//新元素入栈**

**return true;**

**}**

**//出栈**

**bool pops(sqstack &s,char &x)**

**{**

**if(s.top==-1)//栈空报错**

**{**

**return false;**

**}**

**x=s.data[s.top];//栈顶元素出栈**

**s.top--;//指针再减**

**return true;**

**}**

**//查栈顶元素**

**bool gets(sqstack &s,char &x)**

**{**

**if(s.top==-1)//栈空报错**

**{**

**return false;**

**}**

**x=s.data[s.top];//栈顶元素出栈**

**return true;**

**}**

**int main()**

**{**

**cin>>k;**

**while(k--)**

**{**

**cin>>len;**

**cin>>inf;**

**//cout<<inf[0]<<endl;**

**inits(s1);**

**//cout<<op(inf[1])<<endl;**

**for(int i=0;i<len;i++)**

**{**

**if(op(inf[i])==1)//非操作数**

**{**

**char x;**

**if(emptys(s1)==true) //栈为空直接入栈**

**{**

**pushs(s1,inf[i]);**

**//cout<<i<<inf[i]<<endl;**

**continue;**

**}**

**else**

**{**

**char x;**

**gets(s1,x);**

**//cout<<"x"<<x<<endl;**

**if(pri(inf[i])>=pri(x))//当前操作符优先级比栈中高**

**{**

**//cout<<x;**

**//cout<<"pri1"<<pri(inf[i])<<endl;**

**if(pri(inf[i])==pri(x))//相等时候因为从左到右先出栈**

**{**

**pops(s1,x);**

**cout<<x;**

**}**

**pushs(s1,inf[i]);**

**}**

**else**

**{**

**//cout<<"em"<<emptys(s1)<<endl;**

**//cout<<"pri1"<<pri(inf[i])<<endl;**

**while((pri(inf[i])<pri(x))&&(emptys(s1)!=1))**

**//站内元素出栈直到栈空或只有不比当前优先级小的操作符**

**{**

**pops(s1,x);**

**cout<<x;**

**}**

**pushs(s1,inf[i]);**

**}**

**}**

**}**

**else**

**{**

**cout<<inf[i];**

**}**

**}**

**while(emptys(s1)!=1)//栈中有残余元素出栈**

**{**

**char x;**

**pops(s1,x);**

**cout<<x;**

**}**

**cout<<endl;**

**}**

**return 0;**

**}**