第8次实验

1. 知识梳理：堆栈

栈是限制在表的一端进行插入和删除操作的线性表。允许进行插入和删除操作的一端称为栈顶，另一端称为栈底。堆栈的示意图如图1所示。堆栈的主要操作有：创建空栈、进栈、出栈、读栈顶元素。

进栈

出栈

图1 栈示意图

a1

a3

a2

top

顺序栈利用一组连续的存储的存储单元存放栈中的数据元素。例如下面结构创建了顺序栈。变量data并没有象顺序表中数组的定义那样，给定一个固定长度，而仅仅是给了一个指针。这样就可以根据空间的需要，在使用顺序栈之前进行数组的初始化。

typedef struct Sqstack{

ElemType data[Max]; //存储元素, 最大可分配空间Max，以元素为单位

int top; //栈顶指针，存储元素的下标

}SqStack;

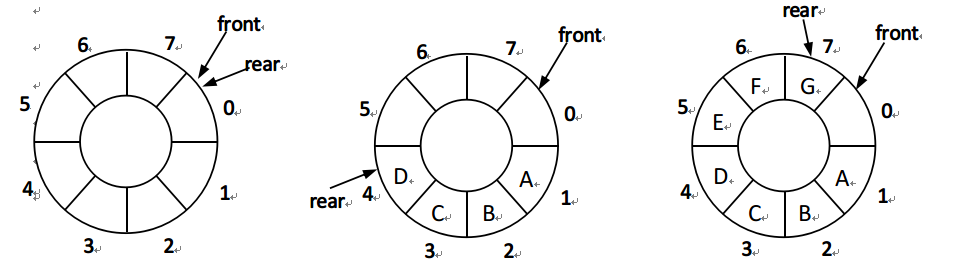
一般将数组的0下标作为栈底，将栈顶元素的下标存储在栈顶指针top中，它随着元素进栈出栈而变化。top为-1表示空栈，top等于stacksize-1则表示栈满。如果要将栈置为空栈，只要将top设为-1即可。

1. 知识梳理：队列

队列是只能在表的一端进行插入、在另一端进行删除操作的线性表。允许删除元素的一端称为队头，允许插入元素的一端称为队尾。



图2 一般队列示意图



(a)循环队列空 (b)非空循环队列 (c)循环队列满

图3 循环队列示意图

顺序存储的队列也可用一维数组来实现，front和rear指针分别是队头和队尾元素的下标值。顺序结构中，采用数学方法将rear或front指针从数组空间的最大下标位置移到最小下标位置，形成循环队列，如图3。front所知位置留空，为了编程方便。

循环队列可采用结构体类型描述如下：

const int MAX=100; // 队列最大容量

struct SqQueue{

int data[MAX]; // 存放元素的数组（这里是int型元素）

int front; // 队头指针

int rear; // 队尾指针

};

SqQueue Q; // 定义队列q

Q.front=Q.rear=0; // 指针初始化

上述循环队列的元素**入队列**操作为：

**Q.rear = (Q.rear+1)%MAX；**

**再将新元素在rear指示位置加入；**

元素**出队列**时操作为：

**Q.front = (Q.front+1)% MAX；**

**再将下标为front的元素取出；**

同时，队空条件为：**Q.front = Q.rear**，

队满条件为：**(Q.rear+1) % MAX = Q.front**

* 实验1 后缀表达式求值

四则运算中缀表达式是我们熟悉的一般形式，例如a + b\*c + (d \* e + f) \* g。后缀表达式是操作符在两个操作数之后的表示方法。可以将上面中缀表达式转换成后缀表达式a b c \* + d e \* f  + g \* +。转换方式如下：

a + **b\*c** + (d \* e + f) \* g

↓

**a + bc\*** + (d \* e + f) \* g

↓

**abc\*+** + (**d \* e** + f) \* g

↓

**abc\*+** + (**de\* + f**) \* g

↓

**abc\*+** + **de\*f+** \* g

↓

**abc\*+** + **de\*f+ g \***

↓

**abc\*+ de\*f+ g \* +**

后缀表达式的求值规则为：从左到右扫描后缀表达式，如果遇到一个操作数，将其压入栈中，如果遇到一个操作符，则从栈中弹出两个操作数，计算结果，然后把结果入栈，直到遍历完后缀表达式，则计算完成，此时的栈顶元素即为计算结果。

例如：后缀表达式 6 5 2 3 + 8 \* + 3 + \* ，即 6\*(5+(2+3)\*8+3)。 利用栈求值过程：

依次读取后缀表达式的内容，按照下面步骤处理

1. 遇到数字就入栈，6 5 2 3 入栈 [6,5,2,3]
2. 遇到 +，出栈两个数，计算 2+3=5, 再次将5入栈 [6,5,5]
3. 遇到数字8 则入栈 [6,5,5,8]
4. 遇到 \*，出栈两个数，计算 5\*8=40, 再次将40入栈 [6,5,40]
5. 遇到 +，出栈两个数，计算 5+40=45, 再次将45入栈 [6,45]
6. 遇到数字3 则入栈 [6,45,3]
7. 遇到 +，出栈两个数，计算 45+3=48, 再次将48入栈 [6,48]
8. 遇到 \*，出栈两个数，计算 6\*48=288, 再次将288入栈 [288]

处理完毕，288就是解。

* **实验要求：**

读入一个后缀表达式，仅含有单个正整数和+、\*运算。中间用空格分割，以#号表示结束。输出计算表达式结果。例如：

6 5 2 3 + 8 \* + 3 + \* #

Result = 288

**[代码提示]**

....

struct **SqStack**{

int d[100]; //the number of integer <100

int top=-1; //栈顶指针，存储栈顶元素的下标

};

//入栈

void Push(SqStack \*s, int x)

{

if(栈顶指针 < 栈容量-1) { // 注意：s->top, s->d[..]

栈顶数值加 1

给栈顶数据赋值x

}else 输出"栈满"信息

}

//出栈

int Pop(SqStack \*s)

{

if(栈顶数值大于-1) {

将栈顶数据赋给变量e

栈顶数值减 1

return e;

}else {

输出"栈空"信息

return -9999; // -9999 is a number express empty

}

}

int **main**()

{

SqStack T; //定义一个堆栈

char c;

cin>>c;

while(c != '#')

{

if(c 在 '0' 和 '9' 之间)

{

c转换为数字k';

Push(&T,k); // 入栈

}else{

出栈一个数到 e1;

再出栈一个数到 e2;

if(c=='+') 将 e2+e1 入栈 ;

如果c是'\*' 将 e2\*e1 入栈 ;

}

再次读取字符到c;

}

出栈一个数到 r;

cout<<"result="<<r<<endl;

return 0;

}

* 实验2——队列实验
* **实验要求：**

用**char**数组建立顺序队列，实现并验证下面算法

1）入队

【算法】

void EnQueue(SqQueue \*Q , char x)

{

if(队尾加一取模MAX等于队头)

输出"队列已满"的信息；

else {

队尾后移一位； // 队尾=队尾加1取模MAX

队尾的数据设置为x；

}

}

2）出队

【算法】

char DeQueue(SqQueue \*Q )

{

if(队尾等于队头) {

输出"队列已空"信息；

返回 ’#’； // 假定返回 # 代表无数据

} else {

队头后移一位； // 加一取模MAX

返回队头数据；

}

}

3）运行下面测试主函数

// 主函数

int main()

{

SqQueue \*queue;

queue = new SqQueue; //创建队列

queue->rear=queue->front=0;

EnQueue(queue, 'A');

EnQueue(queue, 'B');

EnQueue(queue, 'C');

//一个元素出队,并显示

printf("出队：%c\n", (char)DeQueue(queue));

//再入队3个后打印队列

EnQueue(queue, 'D');

EnQueue(queue, 'E');

EnQueue(queue, 'F');

//所有元素依次出队直到队空

while (queue->front != queue->rear)

printf("出队：%c\n", (char)DeQueue(queue));

return 0;

}

【运行结果】

出队：A

出队：B

出队：C

出队：D

出队：E

出队：F