第三章 栈和队列

1、回文是指正读反读均相同的字符序列，如“abba”和“abdba”均是回文，但“good”不是回文。试写一个算法判定给定的字符向量是否为回文。(提示：将一半字符入栈)

[题目分析]

将字符串前一半入栈，然后，栈中元素和字符串后一半进行比较。即将第一个出栈元素和后一半串中第一个字符比较，若相等，则再出栈一个元素与后一个字符比较，……，直至栈空，结论为字符序列是回文。在出栈元素与串中字符比较不等时，结论字符序列不是回文。

[算法描述]

```
//以下为顺序栈的存储结构定义
#define StackSize 100 //假定预分配的栈空间最多为 100 个元素
typedef char DataType;//假定栈元素的数据类型为字符
typedef struct{
   DataType data[StackSize];
   int top;
}SeqStack;

int IsHuiwen( char *t)
  {//判断 t 字符向量是否为回文，若是，返回 1，否则返回 0
     SeqStack s;
     int i , len;
     char temp;
     InitStack( &s);
     len=strlen(t); //求向量长度
     for ( i=0; i<len/2; i++)//将一半字符入栈
        Push( &s, t[i]);
     while( !EmptyStack( &s))
        {// 每弹出一个字符与相应字符比较
          temp=Pop (&s);
          if( temp!=S[i])  return 0 ;// 不等则返回 0
          else i++;
```

```
        }
    return 1 ; //  比较完毕均相等则返回  1
}
```

2、设从键盘输入一整数的序列：a1, a2, a3，…，an，试编写算法实现：用栈结构存储输入的整数，当 ai≠-1 时，将 ai 进栈；当 ai=-1 时，输出栈顶整数并出栈。算法应对异常情况（入栈满等）给出相应的信息。

[算法描述]

```
#define maxsize  栈空间容量

void InOutS(int s[maxsize])

//s 是元素为整数的栈，本算法进行入栈和退栈操作。

{int top=0;                    //top 为栈顶指针，定义 top=0 时为栈空。

 for(i=1; i<=n; i++)      //n 个整数序列作处理。

    {cin>>x);      //从键盘读入整数序列。

      if(x!=-1)                 // 读入的整数不等于-1 时入栈。

      {if(top==maxsize-1){cout<<“栈满”<<endl;exit(0);}

else s[++top]=x; //x 入栈。

}

      else     //读入的整数等于-1 时退栈。

      {if(top==0){ cout<<“栈空”<<endl;exit(0);}

else cout<<“出栈元素是”<< s[top--]<<endl;}

}

}//算法结束。
```

3、假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素站点(注意不设头指针)，试编写相应的置空队、判队空 、入队和出队等算法。

[题目分析]

置空队就是建立一个头节点，并把头尾指针都指向头节点，头节点是不存放数据的；判队空就是当头指针等于尾指针时，队空；入队时，将新的节点插入到链队列的尾部，同时将尾指针指向这个节点；出队时，删除的是队头节点，要注意队列的长度大于 1 还是等于 1 的情况，这个时候要注意尾指针的修改，如果等于 1，则要删除尾指针指向的节点。

[算法描述]

```
//先定义链队结构:
typedef struct queuenode
{Datatype data;
struct queuenode *next;
}QueueNode; //以上是结点类型的定义
typedef struct
{queuenode *rear;
}LinkQueue; //只设一个指向队尾元素的指针
```

(1)置空队

```
void InitQueue( LinkQueue *Q)
{ //置空队：就是使头结点成为队尾元素
   QueueNode *s;
Q->rear = Q->rear->next;//将队尾指针指向头结点
while (Q->rear!=Q->rear->next)//当队列非空，将队中元素逐个出队
{s=Q->rear->next;
Q->rear->next=s->next;
delete s;
   }//回收结点空间
}
```

(2)判队空

```
int EmptyQueue( LinkQueue *Q)
{ //判队空。当头结点的 next 指针指向自己时为空队
   return Q->rear->next->next==Q->rear->next;
}
```

(3)入队

```
void EnQueue( LinkQueue *Q, Datatype x)
```

```
{ //入队。也就是在尾结点处插入元素
QueueNode *p=new QueueNode;//申请新结点
p->data=x; p->next=Q->rear->next;//初始化新结点并链入
Q-rear->next=p;
Q->rear=p;//将尾指针移至新结点
}
```

(4)出队

```
Datatype DeQueue( LinkQueue *Q)
{//出队,把头结点之后的元素摘下
Datatype t;
QueueNode *p;
if(EmptyQueue( Q ))
Error("Queue underflow");
p=Q->rear->next->next; //p 指向将要摘下的结点
x=p->data; //保存结点中数据
if (p==Q->rear)
{//当队列中只有一个结点时，p 结点出队后，要将队尾指针指向头结点
   Q->rear = Q->rear->next;
Q->rear->next=p->next;
}
else
Q->rear->next->next=p->next;//摘下结点 p
delete p;//释放被删结点
return x;
}
```

4、假设以数组 Q[m]存放循环队列中的元素，同时设置一个标志 tag，以 tag == 0 和 tag == 1 来区别在队头指针(front)和队尾指针(rear)相等时，队列状态为“空”还是“满”。试编写与此结构相应的插入(enqueue)和删除(dlqueue)算法

[算法描述]

(1)初始化

```
SeQueue QueueInit(SeQueue Q)
{//初始化队列
```

```
Q.front=Q.rear=0; Q.tag=0;
return Q;
}
(2)入队
SeQueue QueueIn(SeQueue Q,int e)
{//入队列
if((Q.tag==1) && (Q.rear==Q.front)) cout<<"队列已满"<<endl;
else
{Q.rear=(Q.rear+1) % m;
Q.data[Q.rear]=e;
if(Q.tag==0) Q.tag=1; //队列已不空
}
return Q;
}
(3)出队
ElemType QueueOut(SeQueue Q)
{//出队列
if(Q.tag==0) { cout<<"队列为空"<<endl; exit(0);}
else
{Q.front=(Q.front+1) % m;
e=Q.data[Q.front];
if(Q.front==Q.rear) Q.tag=0; //空队列
}
return(e);
}
```

5、如果允许在循环队列的两端都可以进行插入和删除操作。要求：

① 写出循环队列的类型定义；

② 写出“从队尾删除”和“从队头插入”的算法。

[题目分析] 用一维数组 v[0..M-1]实现循环队列，其中 M 是队列长度。设队头指针 front 和队尾指针 rear，约定 front 指向队头元素的前一位置，rear 指向队尾元素。定义 front=rear 时为队空，(rear+1)%m=front 为队满。约定队头端入队向下标小的方向发展，队尾端入队向下标大的方向发展。

[算法描述]

①

```
#define M   队列可能达到的最大长度
typedef struct
{elemtp data[M];
int front,rear;
}cycqueue;
```

②

```
elemtp delqueue ( cycqueue Q)
      //Q 是如上定义的循环队列，本算法实现从队尾删除，若删除成功，返回被删除元素，否则给出出错信息。
{if (Q.front==Q.rear)   { cout<<"队列空"<<endl; exit(0);}
Q.rear=(Q.rear-1+M)%M;                  //修改队尾指针。
return(Q.data[(Q.rear+1+M)%M]); //返回出队元素。
}//从队尾删除算法结束

void enqueue (cycqueue Q, elemtp x)
// Q 是顺序存储的循环队列，本算法实现“从队头插入”元素 x。
{if (Q.rear==(Q.front-1+M)%M)   { cout<<"队满"<<endl; exit(0);)
  Q.data[Q.front]=x;              //x  入队列
Q.front=(Q.front-1+M)%M;   //修改队头指针。
}//  结束从队头插入算法。
```