第3章 栈和队列

3.1 练习题及参考答案

3.1.1 练习题

1. 单项选择题

(1) 栈的“先进后出”特性是指(　　)。

A. 最后进栈的元素总是最先出栈

B. 当同时进行进栈和出栈操作时,总是进栈优先

C. 每当有出栈操作时,总要先进行一次进栈操作

D. 每次出栈的元素总是最先进栈的元素

(2) 设一个栈的进栈序列是 a,b,c,d(即元素 $a\sim d$ 依次通过该栈),则借助该栈所得到的输出序列不可能是(　　)。

A. *abcd*　　B. *dcba*　　C. *acdb*　　D. *dabc*

(3) 一个栈的进栈序列是 a,b,c,d,e,则栈的不可能的输出序列是(　　)。

A. *edcba*　　B. *decba*　　C. *dceab*　　D. *abcde*

(4) 已知一个栈的进栈序列是 1,2,3,…,n,其输出序列的第一个元素是 i($1\leqslant i\leqslant n$),则第 j($1\leqslant j\leqslant n$)个出栈元素是(　　)。

A. i　　B. $n-i$　　C. $j-i+1$　　D. 不确定

(5) 设顺序栈 st 的栈顶指针 top 的初始值为 -1,栈空间大小为 MaxSize,则判定 st 栈为栈空的条件为(　　)。

A. st.top==-1　　B. st.top!=-1

C. st.top!=MaxSize　　D. st.top==MaxSize

(6) 设顺序栈 st 的栈顶指针 top 的初始值为 -1,栈空间大小为 MaxSize,则判定 st 栈为栈满的条件是(　　)。

A. st.top!=-1　　B. st.top==-1

C. st.top!=MaxSize-1　　D. st.top==MaxSize-1

(7) 当用一个数组 data[0..$n-1$]存放栈中元素时,栈底最好(　　)。

A. 设置在 data[0]处　　B. 设置在 data[$n-1$]处

C. 设置在 data[0]或 data[$n-1$]处　　D. 设置在 data 数组的任何位置

(8) 若一个栈用数组 data[1..n]存储,初始栈顶指针 top 为 0,则以下元素 x 进栈的正确操作是(　　)。

A. top++; data[top]=x;　　B. data[top]=x; top++;

C. top--; data[top]=x;　　D. data[top]=x; top--;

(9) 若一个栈用数组 data[1..n]存储,初始栈顶指针 top 为 n,则以下元素 x 进栈的正确操作是(　　)。

A. top++; data[top]=x;　　B. data[top]=x; top++;

C. top--; data[top]=x;　　D. data[top]=x; top--;

(10) 队列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

(11) 栈和队列的不同点是(　　)。

A. 都是线性表

B. 都不是线性表

C. 栈只能在一端进行插入/删除操作,而队列在不同端进行插入/删除操作

D. 没有不同点

(12) 元素 a、b、c、d 依次连续进入队列 qu 后,队头元素是(　①　),队尾元素是(　②　)。

A. a　　B. b　　C. c　　D. d

(13) 一个队列的进列序列为 1234,则队列可能的输出序列是(　　)。

A. 4321　　B. 1234　　C. 1432　　D. 3241

(14) 在循环队列中,元素的排列顺序(　　)。

A. 由元素进队的先后顺序确定　　B. 与元素值的大小有关

C. 与队头和队尾指针的取值有关　　D. 与队中数组大小有关

(15) 循环队列 qu(队头指针 front 指向队首元素的前一位置,队尾指针 rear 指向队尾元素的位置)的队满条件是(　　)。

A. (qu.rear+1)%MaxSize==(qu.front+1)%MaxSize

B. (qu.rear+1)%MaxSize==qu.front+1

C. (qu.rear+1)%MaxSize==qu.front

D. qu.rear==qu.front

(16) 循环队列 qu(队头指针 front 指向队首元素的前一位置,队尾指针 rear 指向队尾元素的位置)的队空条件是(　　)。

A. (qu.rear+1)%MaxSize==(qu.front+1)%MaxSize

B. (qu.rear+1)%MaxSize==qu.front+1

C. (qu.rear+1)%MaxSize==qu.front

D. qu.rear==qu.front

(17) 设循环队列中数组的下标是 0～$N-1$,其头尾指针分别为 f 和 r(队头指针 f 指向队首元素的前一位置,队尾指针 r 指向队尾元素的位置),则其元素个数为(　　)。

A. $r-f$　　B. $r-f-1$

C. $(r-f)\%N+1$　　D. $(r-f+N)\%N$

(18) 一个循环队列中用 data[0..$n-1$]数组保存队中元素,另设置一个队尾指针 rear 和一个记录队中实际元素个数的变量 count,则该队中最多可以存放的元素个数是(　　)。

A. $n-1$　　B. n

C. (rear+n) % n　　D. (n−rear) % n

(19) 设栈 S 和队列 Q 的初始状态为空,元素 e_1～e_6 依次通过栈 S,一个元素出栈后即进队列 Q,若 6 个元素出队的序列是 e_2,e_4,e_3,e_6,e_5,e_1,则栈 S 的容量至少应该是(　　)。

A. 5　　B. 4　　C. 3　　D. 2

(20) 与顺序队相比,链队的(　　)。

A. 优点是可以实现无限长队列

B. 优点是进队和出队时间性能更好

C. 缺点是不能进行顺序访问

D. 缺点是不能根据队首和队尾指针计算队的长度

2. 填空题

(1) 栈是一种特殊的线性表,允许插入和删除运算的一端称为(①)。不允许插入和删除运算的一端称为(②)。

(2) 若栈空间大小为 n,则最多的连续进栈操作的次数为( )。

(3) 一个栈的输入序列是 12345,输出序列为 12345,其进栈出栈的操作为( )。

(4) 有 5 个元素,其进栈次序为 a、b、c、d、e,在各种可能的出栈次序中,以元素 c、d 最先出栈(c 第一个且 d 第二个出栈)的次序有( )。

(5) 顺序栈用 data[0..$n-1$]存储数据,栈顶指针为 top,其初始值为 0,则元素 x 进栈的操作是( )。

(6) 顺序栈用 data[0..$n-1$]存储数据,栈顶指针为 top,其初始值为 0,则出栈元素 x 的操作是( )。

(7) ( )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。

(8) 设有数组 A[0..m]作为循环队列的存储空间,front 为队头指针(它指向队首元素的前一位置),rear 为队尾指针(它指向队尾元素的位置),则元素 x 执行进队的操作是( )。

(9) 设有数组 A[0..m]作为循环队列的存储空间,front 为队头指针(它指向队首元素的前一位置),rear 为队尾指针(它指向队尾元素的位置),则元素出队并保存到 x 中的操作是( )。

(10) 设循环队列的大小为 70,队头指针 front 指向队首元素的前一位置,队尾指针 rear 指向队尾元素位置。现经过一系列进队和出队操作后,有 front=20,rear=11,则队列中的元素个数是( )。

3. 简答题

(1) 简要说明线性表、栈与队的异同点。

(2) 当用一维数组实现顺序栈时,为什么一般将栈底设置在数组的一端,而不是设置在中间?

(3) 在以下几种存储结构中,哪个最适合用作链栈?

① 带头结点的单链表;

② 不带头结点的循环单链表;

③ 带头结点的双链表。

(4) 在循环队列中插入和删除元素时,是否需要移动队中元素?

(5) 顺序队的“假溢出”是怎样产生的?如何判断循环队列是空还是满?

4. 算法设计题

(1) 设计一个算法,利用一个顺序栈将字符数组 a[0..$n-1$]的所有元素逆置。

(2) 设计一个算法,将一个十进制正整数 d 转换为相应的八进制数。

(3) 设计一个算法,利用栈的基本运算返回给定栈中的栈底元素,要求仍保持栈中元素次序不变。这里只能使用栈 st 的基本运算来完成,不能直接用 st.data[0]来得到栈底元素。

(4) 设计一个算法,利用循环队列的基本运算和《教程》中例 3.13 求循环队列元素个数的算法,求指定循环队列中的队尾元素,要求队列中元素次序不改变,算法的空间复杂度为 $O(1)$。

(5) 对于循环队列,假设队中所有元素为数字字符,利用循环队列的基本运算和《教程》中的例 3.13 求循环队列元素个数的算法,删除其中所有奇数字符元素。

3.1.2 练习题参考答案

1. 单项选择题

(1) A	(2) D	(3) C	(4) D	(5) A
(6) D	(7) C	(8) A	(9) D	(10) A
(11) C	(12) ① A ② D	(13) B	(14) A	(15) C
(16) D	(17) D	(18) B	(19) C	(20) D

2. 填空题

(1) ①栈顶 ②栈底

(2) n

(3) 1 进栈,1 出栈,2 进栈,2 出栈,3 进栈,3 出栈,4 进栈,4 出栈,5 进栈,5 出栈

(4) *cdbae*、*cdeba*、*cdbea*

(5) data[top]=x; top++;

(6) top--; x=data[top];

(7) 队列

(8) rear=(rear+1)%(m+1); A[rear]=x;

(9) front=(front+1)%(m+1); x=A[front];

(10) (rear−front+M)%M=(11−20+70)%70=61

3. 简答题

(1) **答**: 相同点: 都属于线性结构,都可以用顺序表存储或链表存储; 栈和队列是两种特殊的线性表,即受限的线性表,只是对插入、删除运算加以限制。

不同点: ①运算规则不同,线性表为随机存取,而栈是只允许在一端进行插入、删除运算,因而是后进先出表 LIFO; 队列是只允许在一端进行插入、另一端进行删除运算,因而是先进先出表 FIFO。②用途不同,栈用于子程序调用和保护现场等,队列用于多道作业处理、指令寄存及其他运算等。

(2) **答**: 栈的主要操作是进栈和出栈,栈底总是不变的,元素进栈是从栈底向栈顶一个方向伸长的,如果栈底设置在中间,伸长方向的另一端空间难以使用,所以栈底总是设置在数组的一端而不是中间。

(3) **答**: 栈中元素之间的逻辑关系属线性关系,可以采用单链表、循环单链表和双链表之一来存储,而栈的主要运算是进栈和出栈,当采用①时,前端作为栈顶,进栈和出栈运算的时间复杂度为 $O(1)$。当采用②时,前端作为栈顶,当进栈和出栈时,首结点都发生变化,还需要找到尾结点,通过修改其 next 域使其变为循环单链表,算法的时间复杂度为 $O(n)$。当采用③时,前端作为栈顶,进栈和出栈运算的时间复杂度为 $O(1)$。

但单链表和双链表相比,其存储密度更高,所以本题中最适合用作链栈的是带头结点的单链表即①。

(4) **答**: 在循环队列中插入和删除元素时,不需要移动队中任何元素,只需要修改队尾

或队头指针，并向队尾插入元素或从队头取出元素。

(5) **答**：采用一维数组实现队列时，当队尾指针已经到了数组的上界，不能再有进队操作，但其实数组中还有空位置，这就产生了“假溢出”，所以假溢出是队满条件设置不当造成的。

采用循环队列是解决假溢出的途径，解决循环队列是空还是满的办法如下。

① 设置一个布尔变量以区别队满还是队空；

② 浪费一个元素的空间，用于区别队满还是队空；

③ 使用一个计数器记录队列中元素个数(即队列长度)。

通常采用方法②，让队头指针 front 指向队首元素的前一位置，队尾指针 rear 指向队尾元素的位置，这样判断循环队列队空的标志是 front＝rear，队满的标志是(rear＋1)％MaxSize＝front。

4. 算法设计题

(1) **解**：定义一个栈 st，正向扫描 a 的所有字符，将每个字符进栈。再出栈所有字符，将其写入 $a[0..n-1]$中。对应的算法如下。

```
void Reverse(char a[],int n)                    //逆置 a[0..n－1]
{   int i; char e;
    SqStack st;                                 //定义一个顺序栈 st
    InitStack(st);
    for (i=0;i＜n;i++)                          //依次将 a[0..n－1]进栈
        Push(st,a[i]);
    for (i=0;i＜n;i++)                          //将 a[n－1]～a[0]依次存放到 a[0..n－1]
    {   Pop(st,e);
        a[i]＝e;
    }
    DestroyStack(st);                           //销毁栈 st
}
```

(2) **解**：算法原理与《教程》中例 3.9 相同，仅将二进制改为八进制。对应的算法如下。

```
void trans(int d,char b[])                      //b 用于存放 d 转换成的八进制数的字符串
{   SqStack st;                                 //定义一个顺序栈 st
    InitStack(st);                              //栈初始化
    char ch;
    int i=0;
    while (d!=0)
    {   ch='0'＋d％8;                           //求余数并转换为字符
        Push(st,ch);                            //字符 ch 进栈
        d/＝8;                                  //继续求更高位
    }
    while (!StackEmpty(st))
    {   Pop(st,ch);                             //出栈并存放在数组 b 中
        b[i]＝ch;
        i++;
    }
    b[i]＝'\0';                                 //添加字符串结束标志
    DestroyStack(st);                           //销毁栈 st
}
```

(3) **解**：对于给定的栈 st，设置一个临时栈 tmpst，先退栈 st 中所有元素并进入栈 tmpst 中，最后的一个元素即为所求，然后将临时栈 tmpst 中的元素逐一出栈并进入 st 中，这样恢复 st 栈中原来的元素。对应算法如下。

```
int GetBottom(SqStack st,ElemType &x)          //x 在算法返回 1 时保存栈底元素
{   ElemType e;
    SqStack tmpst;                             //定义临时栈
    InitStack(tmpst);                          //初始化临时栈
    if (StackEmpty(st))                        //空栈返回 0
    {   DestroyStack(tmpst);
        return 0;
    }
    while (!StackEmpty(st))                    //临时栈 tmpst 中包含 st 栈中逆转元素
    {   Pop(st,x);
        Push(tmpst,x);
    }
    while (!StackEmpty(tmpst))                 //恢复 st 栈中原来的内容
    {   Pop(tmpst,e);
        Push(st,e);
    }
    DestroyStack(tmpst);
    return 1;                                  //返回 1 表示成功
}
```

(4) **解**：由于算法要求空间复杂度为 $O(1)$，所以不能使用一个临时队列。先利用《教程》中例 3.13 的算法求出队列 qu 中的元素个数 m。循环 m 次，将一个元素 x 出队，再将元素 x 进队。最后的 x 即为队尾元素。对应的算法如下。

```
int Count(SqQueue sq)                          //求循环队列 qu 中元素个数
{
    return (sq.rear+MaxSize-sq.front) % MaxSize;
}
int Last(SqQueue qu,ElemType &x)               //求解算法
{   int i,m=Count(qu);
    if (m==0) return 0;                        //队中元素个数为 0 返回 0
    for (i=1;i<=m;i++)
    {   DeQueue(qu,x);                         //出队元素 x
        EnQueue(qu,x);                         //将元素 x 进队
    }
    return 1;                                  //成功找到队尾元素返回 1
}
```

(5) **解**：先求出队列 qu 中的元素个数 m。i 从 1～m 循环，出队第 i 个元素 e，若 e 不为奇数，将其进队。对应的算法如下。

```
int Count(SqQueue sq)                          //求循环队列 qu 中元素个数
{
    return (sq.rear+MaxSize-sq.front) % MaxSize;
}
void DelOdd(SqQueue &qu)                       //求解算法
{   char e;
    int i,m=Count(qu);
    for (i=1;i<=m;i++)                         //出队 m 次
    {   DeQueue(qu,e);
        if ((e-'0')%2!=1)                      //将不是奇数的数字字符进队
            EnQueue(qu,e);
    }
}
```

3.2 上机实验题及参考答案

3.2.1 上机实验题

1. 基础实验题

(1) 设计字符顺序栈的基本运算程序,并用相关数据进行测试。

(2) 设计字符链栈的基本运算程序,并用相关数据进行测试。

(3) 设计字符循环队列的基本运算程序,并用相关数据进行测试。

(4) 设计字符链队的基本运算程序,并用相关数据进行测试。

2. 应用实验题

(1) 设计一个算法,利用顺序栈的基本运算删除栈 st 中所有值为 e 的元素(这样的元素可能有多个),并且保持其他元素次序不变。并用相关数据进行测试。

(2) 设计一个算法,利用顺序栈的基本运算求栈中从栈顶到栈底的第 k 个元素,要求仍保持栈中元素次序不变,并用相关数据进行测试。

(3) 设进栈序列是 1,2,…,n(n 为一个大于 2 的正整数),编写一个程序判断通过一个栈能否得到由 $a[0..n-1]$指定出栈序列,如果能够得到指定的出栈序列,请给出栈操作的步骤,并用相关数据进行测试。

(4) 设计一个算法,利用循环队列的基本运算和《教程》中例 3.13 求循环队列元素个数的算法,删除指定队列中的队尾元素,要求算法的空间复杂度为 $O(1)$。

(5) 设计一个循环队列,用 front 和 rear 分别作为队头和队尾指针,另外用一个标志 tag 标识队列可能空(tag=0)或可能满(tag=1),这样加上 front==rear 可以作为队空或队满的条件。要求设计队列的相关基本运算算法。

(6) 用循环队列求解约瑟夫问题:设有 n 个人站成一圈,其编号从 1～n。从编号为 1 的人开始按顺时针方向 1、2……循环报数,数到 m 的人出列,然后从出列者的下一个人重新开始报数,数到 m 的人又出列,如此重复进行,直到 n 个人都出列为止。要求输出这 n 个人的出列顺序,并用相关数据进行测试。

3.2.2 上机实验题参考答案

1. 基础实验题

(1) **解**:字符顺序栈的基本运算算法设计原理参见《教程》的 3.1.2 节。包含顺序栈基本运算函数的文件 SqStack.cpp 如下。

```
#include <stdio.h>
#define MaxSize 100                    //顺序栈的初始分配空间大小
typedef char ElemType;                 //假设顺序栈中所有元素为 char 类型
typedef struct
{    ElemType data[MaxSize];           //保存栈中元素
     int top;                          //栈顶指针
} SqStack;                             //顺序栈类型
```

```
void InitStack(SqStack &st)                    //初始化顺序栈 st
{
    st.top=-1;
}
void DestroyStack(SqStack st)                  //销毁顺序栈 st
{ }
int Push(SqStack &st,ElemType x)               //进栈元素 x
{   if (st.top==MaxSize-1)                     //栈满
        return 0;
    else
    {   st.top++;
        st.data[st.top]=x;
        return 1;
    }
}
int Pop(SqStack &st,ElemType &x)               //出栈元素 x
{   if (st.top==-1)                            //栈空
        return 0;
    else
    {   x=st.data[st.top];
        st.top--;
        return 1;
    }
}
int GetTop(SqStack st,ElemType &x)             //取栈顶元素 x
{   if (st.top==-1)                            //栈空
        return 0;
    else
    {   x=st.data[st.top];
        return 1;
    }
}
int StackEmpty(SqStack st)                     //判断栈是否为空
{   if (st.top==-1) return 1;
    else return 0;
}
```

设计如下应用主函数。

```
#include "SqStack.cpp"                         //包含顺序栈基本运算函数
int main()
{   SqStack st;
    ElemType e;
    printf("初始化栈 st\n");
    InitStack(st);
    printf("栈%s\n",(StackEmpty(st)==1?"空":"不空"));
    printf("a 进栈\n");Push(st,'a');
    printf("b 进栈\n");Push(st,'b');
    printf("c 进栈\n");Push(st,'c');
    printf("d 进栈\n");Push(st,'d');
    printf("栈%s\n",(StackEmpty(st)==1?"空":"不空"));
    GetTop(st,e);
    printf("栈顶元素:%c\n",e);
    printf("出栈次序:");
    while (!StackEmpty(st))                    //栈不空循环
    {   Pop(st,e);                             //出栈元素 e 并输出
        printf("%c ",e);
```

```
    }
    printf("\n 销毁栈 st\n");
    DestroyStack(st);
}
```

上述程序的执行结果如图 3.1 所示。

图 3.1 实验程序的执行结果

(2) **解**：字符链栈的基本运算算法设计原理参见《教程》的 3.1.3 节。包含链栈基本运算函数的文件 LinkStack.cpp 如下。

```
#include <stdio.h>
#include <malloc.h>
typedef char ElemType;                          //假设链栈中所有元素为 char 类型
typedef struct node
{   ElemType data;                              //存储结点数据
    struct node *next;                          //指针域
} LinkStack;                                    //链栈结点类型
void InitStack(LinkStack *&ls)                  //初始化链栈 ls
{
    ls=NULL;
}
void DestroyStack(LinkStack *&ls)               //销毁链栈 ls
{   LinkStack *pre=ls, *p;
    if (pre==NULL) return;
    p=pre->next;
    while (p!=NULL)
    {   free(pre);                              //释放 pre 结点
        pre=p;p=p->next;                        //pre、p 同步后移
    }
    free(pre);                                  //释放尾结点
}
int Push(LinkStack *&ls,ElemType x)             //进栈元素 x
{   LinkStack *p;
    p=(LinkStack *)malloc(sizeof(LinkStack));
    p->data=x;                                  //创建结点 p 用于存放 x
    p->next=ls;                                 //插入 p 结点作为栈顶结点
    ls=p;
    return 1;
}
int Pop(LinkStack *&ls,ElemType &x)             //出栈元素 x
{   LinkStack *p;
    if (ls==NULL)                               //栈空,下溢出返回 0
        return 0;
    else                                        //栈不空时出栈元素 x 并返回 1
    {   p=ls;                                   //p 指向栈顶结点
        x=p->data;                              //取栈顶元素 x
        ls=p->next;                             //删除结点 p
        free(p);                                //释放 p 结点
        return 1;
    }
}
int GetTop(LinkStack *ls,ElemType &x)           //取栈顶元素 x
{   if (ls==NULL)                               //栈空,下溢出时返回 0
        return 0;
    else                                        //栈不空,取栈顶元素 x 并返回 1
    {   x=ls->data;
```

```
        return 1;
    }
}
int StackEmpty(LinkStack *ls)                    //判断栈是否为空栈
{   if (ls==NULL) return 1;
    else return 0;
}
```

设计如下应用主函数。

```
#include "LinkStack.cpp"                         //包含前面的链栈基本运算函数
int main()
{   LinkStack *st;
    ElemType e;
    printf("初始化栈 st\n");
    InitStack(st);
    printf("栈%s\n",(StackEmpty(st)==1?"空":"不空"));
    printf("a 进栈\n");Push(st,'a');
    printf("b 进栈\n");Push(st,'b');
    printf("c 进栈\n");Push(st,'c');
    printf("d 进栈\n");Push(st,'d');
    printf("栈%s\n",(StackEmpty(st)==1?"空":"不空"));
    GetTop(st,e);
    printf("栈顶元素:%c\n",e);
    printf("出栈次序:");
    while (!StackEmpty(st))                      //栈不空循环
    {   Pop(st,e);                               //出栈元素 e 并输出
        printf("%c ",e);
    }
    printf("\n 销毁栈 st\n");
    DestroyStack(st);
}
```

上述程序的执行结果如图 3.1 所示。

(3) **解**：字符循环队列的基本运算算法设计原理参见《教程》的 3.2.2 节。包含循环队列基本运算函数的文件 SqQueue.cpp 如下。

```
#include <stdio.h>
#define MaxSize 100                              //循环队列的初始分配空间大小
typedef char ElemType;                           //假设循环队列中所有元素为 char 类型
typedef struct
{   ElemType data[MaxSize];                      //保存队中元素
    int front,rear;                              //队头和队尾指针
} SqQueue;
void InitQueue(SqQueue &sq)                      //初始化循环队列 sq
{
    sq.rear=sq.front=0;                          //指针初始化
}
void DestroyQueue(SqQueue sq)                    //销毁循环队列 sq
{ }
int EnQueue(SqQueue &sq,ElemType x)              //进队列元素 x
{   if ((sq.rear+1) % MaxSize==sq.front)         //队满上溢出
        return 0;
    sq.rear=(sq.rear+1) % MaxSize;               //队尾循环进 1
```

```
    sq.data[sq.rear]=x;
    return 1;
}
int DeQueue(SqQueue &sq,ElemType &x)          //出队元素 x
{   if (sq.rear==sq.front)                    //队空下溢出
        return 0;
    sq.front=(sq.front+1) % MaxSize;          //队头循环进 1
    x=sq.data[sq.front];
    return 1;
}
int GetHead(SqQueue sq,ElemType &x)           //取队头元素 x
{   if (sq.rear==sq.front)                    //队空下溢出
        return 0;
    x=sq.data[(sq.front+1) % MaxSize];
    return 1;
}
int QueueEmpty(SqQueue sq)                    //判断循环队列 sq 是否为空
{   if (sq.rear==sq.front) return 1;
    else return 0;
}
```

设计如下应用主函数。

```
#include "SqQueue.cpp"                        //包含销毁队列的基本运算函数
int main()
{   SqQueue sq;
    ElemType e;
    printf("初始化队列");
    InitQueue(sq);
    printf("队%s\n",(QueueEmpty(sq)==1?"空":"不空"));
    printf("a 进队\n");EnQueue(sq,'a');
    printf("b 进队\n");EnQueue(sq,'b');
    printf("c 进队\n");EnQueue(sq,'c');
    printf("d 进队\n");EnQueue(sq,'d');
    printf("队%s\n",(QueueEmpty(sq)==1?"空":"不空"));
    GetHead(sq,e);
    printf("队头元素:%c\n",e);
    printf("出队次序:");
    while (!QueueEmpty(sq))                   //队不空循环
    {   DeQueue(sq,e);                        //出队元素 e
        printf("%c ",e);                      //输出元素 e
    }
    printf("\n 销毁队列\n");
    DestroyQueue(sq);
}
```

上述程序的执行结果如图 3.2 所示。

图 3.2 实验程序的执行结果

(4) **解**：字符链队的基本运算算法设计原理参见《教程》的 3.2.3 节。包含链队基本运算函数的文件 LinkQueue.cpp 如下。

```
#include <stdio.h>
#include <malloc.h>
```

```
typedef char ElemType;                          //假设链队中所有元素为 char 类型
typedef struct QNode
{   ElemType data;
    struct QNode * next;
} QType;                                        //链队中数据结点的类型
typedef struct qptr
{   QType * front;                              //队头指针
    QType * rear;                               //队尾指针
} LinkQueue;                                    //链队结点类型
void InitQueue(LinkQueue * &lq)                 //初始化链队 lq
{   lq=(LinkQueue * )malloc(sizeof(LinkQueue));
    lq->rear=lq->front=NULL;                    //初始时队头和队尾指针均为空
}
void DestroyQueue(LinkQueue * &lq)              //销毁链队 lq
{   QType * pre=lq->front, * p;
    if (pre!=NULL)                              //非空队的情况
    {   if (pre==lq->rear)                      //只有一个数据结点的情况
            free(pre);                          //释放 pre 结点
        else                                    //有两个或多个数据结点的情况
        {   p=pre->next;
            while (p!=NULL)
            {   free(pre);                      //释放 pre 结点
                pre=p; p=p->next;               //pre、p 同步后移
            }
            free(pre);                          //释放尾结点
        }
        free(lq);                               //释放链队结点
    }
}
int EnQueue(LinkQueue * &lq,ElemType x)         //进队元素 x
{   QType * s;
    s=(QType * )malloc(sizeof(QType));          //创建新结点,插入链队的末尾
    s->data=x;s->next=NULL;
    if (lq->front==NULL)                        //原队为空队的情况
        lq->rear=lq->front=s;                   //front 和 rear 均指向 s 结点
    else                                        //原队不为空队的情况
    {   lq->rear->next=s;                       //将 s 链到队尾
        lq->rear=s;                             //rear 指向它
    }
    return 1;
}
int DeQueue(LinkQueue * &lq,ElemType &x)        //出队元素 x
{   QType * p;
    if (lq->front==NULL)                        //原队为空队的情况
        return 0;
    p=lq->front;                                //p 指向队头结点
    x=p->data;                                  //取队头元素值
    if (lq->rear==lq->front)                    //若原队中只有一个结点,删除后队列变空
        lq->rear=lq->front=NULL;
    else                                        //原队有两个或以上结点的情况
        lq->front=lq->front->next;
    free(p);
    return 1;
}
int GetHead(LinkQueue * lq,ElemType &x)         //取队头元素 x
{   if (lq->front==NULL)                        //原队为队空的情况
        return 0;
```

```
    x=lq->front->data;                    //原队不空的情况
    return 1;
}
int QueueEmpty(LinkQueue *lq)             //判断队列 ls 是否是空
{   if (lq->front==NULL) return 1;        //队空返回 1
    else return 0;                        //队不空返回 0
}
```

设计如下应用主函数。

```
#include "LinkQueue.cpp"                  //包含链队的基本运算函数
int main()
{   LinkQueue *lq;
    ElemType e;
    printf("初始化队列\n");
    InitQueue(lq);
    printf("队%s\n",(QueueEmpty(lq)==1?"空":"不空"));
    printf("a 进队\n");EnQueue(lq,'a');
    printf("b 进队\n");EnQueue(lq,'b');
    printf("c 进队\n");EnQueue(lq,'c');
    printf("d 进队\n");EnQueue(lq,'d');
    printf("队%s\n",(QueueEmpty(lq)==1?"空":"不空"));
    GetHead(lq,e);
    printf("队头元素:%c\n",e);
    printf("出队次序:");
    while (!QueueEmpty(lq))               //队不空循环
    {   DeQueue(lq,e);                    //出队元素 e
        printf("%c ",e);                  //输出元素 e
    }
    printf("\n 销毁队列\n");
    DestroyQueue(lq);
}
```

上述程序的执行结果如图 3.2 所示。

2. 应用实验题

(1) **解**：给定栈 st，建立一个临时栈 tmpst 并初始化。退栈 st 的所有元素 x，当 x 不为 e 时将其进栈到 tmpst 中。将 tmpst 的所有元素退栈并进栈到 st 中。最后销毁临时栈 tmpst。对应的实验程序如下。

```
#include "SqStack.cpp"                    //包含顺序栈的基本运算函数
void Dele(SqStack &st,ElemType e)         //求解算法
{   ElemType x;
    SqStack tmpst;
    InitStack(tmpst);
    while (!StackEmpty(st))               //栈 st 不空循环
    {   Pop(st,x);                        //出栈元素 x
        if (x!=e)Push(tmpst,x);
    }
    while (!StackEmpty(tmpst))            //栈 tmpst 不空循环
    {   Pop(tmpst,x);                     //出栈元素 x
        Push(st,x);
    }
    DestroyStack(tmpst);                  //销毁栈 tmpst
}
int main()
```

```
{   SqStack st;
    printf("初始化栈 st\n");
    InitStack(st);
    printf("元素 1,2,2,1,2,3 依次进栈\n");
    Push(st,'1'); Push(st,'2');
    Push(st,'2'); Push(st,'1');
    Push(st,'2'); Push(st,'3');
    char e='2';
    printf("删除所有元素%c\n",e);
    Dele(st,e);
    printf("出栈序列: ");
    while (!StackEmpty(st))
    {   Pop(st,e);
        printf("%c ",e);
    }
    printf("\n 销毁栈\n");
    DestroyStack(st);
}
```

上述程序的执行结果如图 3.3 所示。

图 3.3 实验程序的执行结果

(2) **解**：设计 Pushk(SqStack &st,int k,ElemType &e)算法，如果栈中没有第 k 个元素，返回 0；否则返回 1，并通过引用型参数 e 保存第 k 个元素。

先建立并初始化一个临时栈 tmpst，置表示栈中能否找到第 k 个元素的变量 tag 为 0。出栈 st 中所有元素，并用 i 记录出栈元素 x 的序号，当 $i==k$ 时置 $e=x$ 和 tag=1；否则将元素 x 进栈到 tmpst 中。出临时栈 tmpst 的所有元素并进栈到 st 中，最后销毁 tmpst 并返回 tag。对应的实验程序如下。

```
#include "SqStack.cpp"                          //包含顺序栈的基本运算函数
int Pushk(SqStack &st,int k,ElemType &e)        //求解算法
{   int i=0,tag=0;
    ElemType x;
    SqStack tmpst;                              //定义临时栈
    InitStack(tmpst);                           //初始化临时栈
    while (!StackEmpty(st))          //将 st 中所有元素出栈并进栈到 tmpst 中(除第 k 个元素外)
    {   i++;
        Pop(st,x);
        if (i==k)
        {   e=x;
            tag=1;                              //存在第 k 个元素
        }
        else Push(tmpst,x);
    }
    while (!StackEmpty(tmpst))                  //出栈 tmpst 元素并进栈 st
    {   Pop(tmpst,x);
        Push(st,x);
    }
    DestroyStack(tmpst);                        //销毁临时栈
    return tag;
}
int main()
{   SqStack st;
```

```
    printf("初始化栈 st\n");
    InitStack(st);
    printf("元素 6 到 1 依次进栈\n");
    Push(st,'6'); Push(st,'5');
    Push(st,'4'); Push(st,'3');
    Push(st,'2'); Push(st,'1');
    int k=5;
    char e;
    printf("删除第%d 个元素,",k);
    if (Pushk(st,k,e))
        printf("对应的元素是:%c\n",e);
    else
        printf("该元素不存在\n");
    printf("出栈序列: ");
    while (!StackEmpty(st))
    {   Pop(st,e);
        printf("%c ",e);
    }
    printf("\n 销毁栈\n");
    DestroyStack(st);
}
```

上述程序的执行结果如图 3.4 所示。

图 3.4 实验程序的执行结果

(3) **解**：采用一个临时栈 st(用于暂时存放还没有与 a 中元素匹配的整数)，从 $i=1\sim n$ 循环，$j=0$ 扫描数组 a 的元素。先将 i 进栈，栈不空时取栈顶元素 e，若 $a[j]==e$，则元素 e 退栈，$j++$ 继续栈元素的判断；若 $a[j]\neq e$，退出栈不空的循环，让 $i++$ 继续下一个 i 的判断。整个循环结束，栈空表示可以产生 a 的出栈序列；否则不能产生 a 的出栈序列。

说明：由于 SqStack.cpp 文件中的顺序栈对应的栈元素类型为 char，这里要求栈元素类型为 int，将 SqStack.cpp 复制到 SqStack1.cpp 文件，将其中的 ElemType 声明改为 int 即可。

对应的实验程序如下。

```
#include "SqStack1.cpp"                    //包含顺序栈(栈元素为 int 类型)的基本运算函数
int Validseq(int a[],int n)                //求解算法
{   int i,j,e;
    SqStack st;                            //定义一个顺序栈 st
    InitStack(st);                         //初始化栈 st
    j=0;
    for(i=1;i<=n;i++)                      //处理进栈序列 1,2,…,n
    {   Push(st,i);                        //整数 i 进栈
        printf("  %d 进栈\n",i);
        while (!StackEmpty(st))            //栈不空循环
        {   GetTop(st,e);                  //取栈顶元素 e
            if (a[j]==e)                   //匹配的情况
            {   Pop(st,e);                 //退栈元素 e
                printf("  %d 退栈\n",e);
                j++;
            }
            else break;                    //不匹配时退出 while 循环
        }
    }
```

```
    if (StackEmpty(st))                    //栈空返回 true;否则返回 false
    {   DestroyStack(st);                  //销毁栈 st
        return true;
    }
    else
    {   DestroyStack(st);                  //销毁栈 st
        return false;
    }
}
void dispa(int a[],int n)                  //输出 a
{   for (int i=0;i<n;i++)
        printf("%d ",a[i]);
    printf("\n");
}
void display(int a[],int n)                //测试一个序列 a
{   printf("  测试 a:"); dispa(a,n);
    if (Validseq(a,n))
        printf("  a 是合法序列\n");
    else
        printf("  a 不是合法序列\n");
}
int main()
{   printf("测试结果:\n");
    int n=5;                               //假设测试的进栈序列为 1~5
    int a[]={2,1,5,4,3};
    display(a,n);
    int a1[]={5,1,2,3,4};
    display(a1,n);
}
```

上述程序的执行结果如图 3.5 所示。

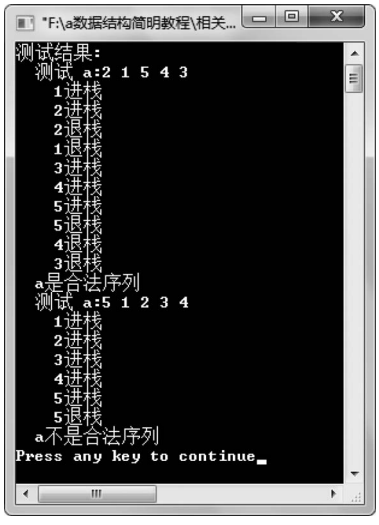

图 3.5 实验程序的执行结果

(4) **解**：由于算法要求空间复杂度为 $O(1)$，所以不能使用临时队列。先求出队列 qu 中的元素个数 m。用 i 循环 m 次，每次出次一个元素 x，若 $i<m$ 则将元素 x 进队；否则元素 x 不进队，从而删除了队尾元素。对应的实验程序如下。

```
#include "SqQueue.cpp"          //包含循环队列的基本运算
                                //函数
int Count(SqQueue sq)           //求循环队列 qu 中元素个数
{
    return(sq.rear+MaxSize-sq.front) % MaxSize;
}
int DelLast(SqQueue &qu,ElemType &x)       //求解算法
{   int i,m=Count(qu);
    if (m==0) return 0;
    for (i=1;i<=m;i++)
    {   DeQueue(qu,x);                     //出队元素 x
        if (i<m)EnQueue(qu,x);             //将元素 x 进队
    }
    return 1;
}
int main()
{   SqQueue qu;
    printf("初始化循环队列 qu\n");
```

```
    InitQueue(qu);
    printf("元素 1 到 5 依次进队\n");
    EnQueue(qu,'1'); EnQueue(qu,'2');
    EnQueue(qu,'3'); EnQueue(qu,'4'); EnQueue(qu,'5');
    char e;
    DelLast(qu,e);
    printf("删除队尾元素%c\n",e);
    printf("出队序列: ");
    while (!QueueEmpty(qu))
    {   DeQueue(qu,e);
        printf("%c ",e);
    }
    printf("\n 销毁队列\n");
    DestroyQueue(qu);
}
```

上述程序的执行结果如图 3.6 所示。

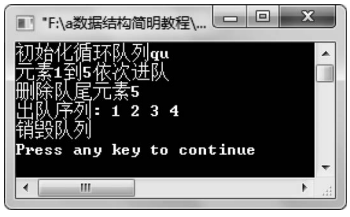

图 3.6 实验程序的执行结果

(5) **解**：设计本实验题中队列的类型如下。

```
typedef struct
{   ElemType data[MaxSize];
    int front,rear;                          //队头和队尾指针
    int tag;                                 //为 0 表示可能队空,为 1 时表示可能队满
} QueueType;                                 //循环队列类型
```

初始时 tag=0,进行任何一次成功的进队操作后 tag=1(因为只有在进队操作后队列才有可能满),进行任何一次成功的出队操作后 tag=0(只有在出队操作后队列才有可能空),因此,这样的队列的基本要素如下。

① 初始时：tag=0,front=rear。

② 队空条件：qu.front==qu.rear && qu.tag==0。

③ 队满条件：qu.front==qu.rear && qu.tag==1。

对应的实验程序如下。

```
#include <stdio.h>
#define MaxSize 100
typedef char ElemType;
typedef struct
{   ElemType data[MaxSize];
    int front,rear;                          //队头和队尾指针
    int tag;                                 //为 0 表示可能队空,为 1 时表示可能队满
} QueueType;                                 //循环队列类型
void InitQueue1(QueueType &qu)               //初始化队列
{   qu.front=qu.rear=0;
    qu.tag=0;                                //为 0 表示可能为空
}
void DestroyQueue1(QueueType qu)             //销毁队列
{ }
int QueueEmpty1(QueueType qu)                //判队空
{
    return(qu.front==qu.rear && qu.tag==0);
}
int QueueFull1(QueueType qu)                 //判队满
{
    return(qu.tag==1 && qu.front==qu.rear);
```

```
}
int EnQueue1(QueueType &qu,ElemType x)    //进队算法
{   if (QueueFull1(qu)==1)                //队满则返回 0
        return 0;
    qu.rear=(qu.rear+1)%MaxSize;
    qu.data[qu.rear]=x;
    qu.tag=1;                             //至少有一个元素,可能满
    return 1;                             //成功进队,返回 1
}
int DeQueue1(QueueType &qu,ElemType &x)   //出队算法
{   if (QueueEmpty1(qu)==1)               //队空则返回 0
        return 0;
    qu.front=(qu.front+1)%MaxSize;
    x=qu.data[qu.front];
    qu.tag=0;                             //出队一个元素,可能空
    return 1;                             //成功出队,返回 1
}
int main()
{   QueueType qu;
    printf("初始化队列 qu\n");
    InitQueue1(qu);
    printf("元素 1 到 5 依次进队\n");
    EnQueue1(qu,'1'); EnQueue1(qu,'2');
    EnQueue1(qu,'3'); EnQueue1(qu,'4'); EnQueue1(qu,'5');
    char e;
    printf("出队序列: ");
    while (!QueueEmpty1(qu))
    {   DeQueue1(qu,e);
        printf("%c ",e);
    }
    printf("\n 销毁队列\n");
    DestroyQueue1(qu);
}
```

上述程序的执行结果如图 3.7 所示。

图 3.7 实验程序的执行结果

(6) **解**：约瑟夫问题已在《教程》例 2.21 中采用循环单链表求解，这里采用循环队列求解。

定义一个整数队列 sq，先将 1～n（对应 n 个人的编号）依次进队，计数器 i 置为 0。在队不空时循环：连续出队元素 p，用 i 累计报数的次数，当 $i\%m\neq 0$ 时表示还没有数到为 m 的人，将出队的 p 进队；当 $i\%m=0$ 时表示恰好数到为 m 的人，将 p 不进队并出列。对应的算法如下。

说明：由于 SqQueue.cpp 文件中的循环队列对应的队列元素类型为 char，这里要求队列元素类型为 int，将 SqQueue.cpp 复制到 SqQueue1.cpp 文件，将其中 ElemType 声明改为 int 即可。

对应的实验程序如下。

```
#include "SqQueue1.cpp"                   //包含循环队列(队列元素为 int 类型)基本运算函数
void Josephus(int n,int m)                //用队列求解约瑟夫问题
{   int i,p;
    SqQueue sq;                           //定义一个顺序队 sq
    InitQueue(sq);                        //初始化队列
    for (i=1;i<=n;i++)                    //n 个人进队
        EnQueue(sq,i);
```

```
    printf(" 出列顺序:");
    i=0;
    while (!QueueEmpty(sq))                    //队不空循环
    {   DeQueue(sq,p);                         //出队元素 p
        i++;                                   //出队个数计数
        if (i % m==0)                          //循环报数到 m
            printf("%d ",p);                   //出列 p
        else
            EnQueue(sq,p);                     //将 p 进队
    }
    printf("\n");
    DestroyQueue(sq);                          //销毁队列
}
void display(int n,int m)                      //测试一个约瑟夫问题
{   printf("   n=%d,m=%d ",n,m);
    Josephus(n,m);
}
int main()
{   printf("测试结果:\n");
    display(6,3);
    display(6,5);
    display(5,8);
    display(4,2);
}
```

上述程序的执行结果如图 3.8 所示。

图 3.8 实验程序的执行结果