第 3 章 栈和队列

教材中练习题及参考答案

1. 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（即*C*第一个且*D*第二个出栈）的次序有哪几个？

答：要使*C*第一个且*D*第二个出栈，应是*A*进栈，*B*进栈，*C*进栈，*C*出栈，*D*进栈，*D*出栈，之后可以有以下几种情况：

（1）*B*出栈，*A*出栈，*E*进栈，*E*出栈，输出序列为*CDBAE*；

（2）*B*出栈，*E*进栈，*E*出栈，*A*出栈，输出序列为*CDBEA*；

（3）*E*进栈，*E*出栈，*B*出栈，*A*出栈，输出序列为*CDEBA*。

所以可能的次序有：*CDBAE*、*CDBEA*、*CDEBA*。

2. 在一个算法中需要建立多个栈(假设 3 个栈或以上)时可以选用以下 3 种方案之一，试问这些方案之间相比各有什么优缺点？

（1）分别用多个顺序存储空间建立多个独立的顺序栈。

（2）多个栈共享一个顺序存储空间。

（3）分别建立多个独立的链栈。

答：（1）优点是每个栈仅用一个顺序存储空间时，操作简单。缺点是分配空间小了，容易产生溢出，分配空间大了，容易造成浪费，各栈不能共享空间。

（2）优点是多个栈仅用一个顺序存储空间，充分利用了存储空间，只有在整个存储空间都用完时才会产生溢出。缺点是当一个栈满时要向左、右查询有无空闲单元。如果有，则要移动元素和修改相关的栈底和栈顶指针。当接近栈满时，要查询空闲单元、移动元素和修改栈底、栈顶指针，这一过程计算复杂且十分耗时。

（3）优点是多个链栈一般不考虑栈的溢出。缺点是栈中元素要以指针相链接，比顺序存储多占用了存储空间。

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

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。

4. 简述以下算法的功能（假设 ElemType 为 int 类型）：

```
void fun(ElemType a[],int n)
{   int i; ElemType e;
    SqStack *st1,*st2;
    InitStack(st1);
    InitStack(st2);
    for (i=0;i<n;i++)
        if (a[i]%2==1)
            Push(st1,a[i]);
        else
            Push(st2,a[i]);
    i=0;
    while (!StackEmpty(st1))
    {   Pop(st1,e);
        a[i++]=e;
    }
    while (!StackEmpty(st2))
    {   Pop(st2,e);
        a[i++]=e;
    }
    DestroyStack(st1);
    DestroyStack(st2);
}
```

答：算法的执行步骤如下：

（1）扫描数组 *a*，将所有奇数进到 *st*1 栈中，将所有偶数进到 *st*2 栈中。

（2）先将 *st*1 的所有元素（奇数元素）退栈，并放到数组 *a* 中并覆盖原有位置的元素；再将 *st*2 的所有元素（偶数元素）退栈，并放到数组 *a* 中并覆盖原有位置的元素。

（3）销毁两个栈 *st*1 和 *st*2。

所以本算法的功能是，利用两个栈将数组 *a* 中所有的奇数元素放到所有偶数元素的前面。例如，ElemType *a*[]={1,2,3,4,5,6}，执行算法后数组 *a* 改变为{5,3,1,6,4,2}。

5. 简述以下算法的功能（顺序栈的元素类型为 ElemType）。

```
void fun(SqStack *&st,ElemType x)
{   SqStack *tmps;
    ElemType e;
    InitStack(tmps);
    while(!StackEmpty(st))
    {   Pop(st,e);
        if(e!=x) Push(tmps,d);
    }
    while (!StackEmpty(tmps))
    {   Pop(tmps,e);
```

```
            Push(st,e);
        }
        DestroyStack(tmps);
}
```

答：算法的执行步骤如下：

（1）建立一个临时栈 *tmps* 并初始化。

（2）退栈 *st* 中所有元素，将不为 *x* 的元素进栈到 *tmps* 中。

（3）退栈 *tmps* 中所有元素，并进栈到 *st* 中。

（4）销毁栈 *tmps*。

所以本算法的功能是，如果栈 *st* 中存在元素 *x*，将其从栈中清除。例如，*st* 栈中从栈底到栈顶为 *a*、*b*、*c*、*d*、*e*，执行算法 fun(*st*，'*c*')后，*st* 栈中从栈底到栈顶为 *a*、*b*、*d*、*e*。

6. 简述以下算法的功能（栈 *st* 和队列 *qu* 的元素类型均为 ElemType）。

```
bool fun(SqQueue *&qu,int i)
{   ElemType e;
    int j=1;
    int n=(qu->rear-qu->front+MaxSize)%MaxSize;
    if (j<1 || j>n) return false;
    for (j=1;j<=n;j++)
    {   deQueue(qu,e);
        if (j!=i)
            enQueue(qu,e);
    }
    return true;
}
```

答：算法的执行步骤如下：

（1）求出队列 *qu* 中的元素个数 *n*。参数 *i* 错误时返回假。

（2）*qu* 出队共计 *n* 次，除了第 *i* 个出队的元素外，其他出队的元素立即进队。

（3）返回真。

所以本算法的功能是，删除 *qu* 中从队头开始的第 *i* 个元素。例如，*qu* 中从队头到队尾的元素是 *a*、*b*、*c*、*d*、*e*，执行算法 fun(*qu*，2)后，qu 中从队头到队尾的元素改变为 *a*、*c*、*d*、*e*。

7. 什么是环形队列？采用什么方法实现环形队列？

答：当用数组表示队列时，把数组看成是一个环形的，即令数组中第一个元素紧跟在最末一个单元之后，就形成一个环形队列。环形队列解决了非环形队列中出现的“假溢出”现象。

通常采用逻辑上求余数的方法来实现环形队列，假设数组的大小为 *n*，当元素下标 *i* 增 1 时，采用 *i*=(*i*+1)%*n* 来实现。

8. 环形队列一定优于非环形队列吗？什么情况下使用非环形队列？

答：队列主要是用于保存中间数据，而且保存的数据满足先产生先处理的特点。非环形队列可能存在数据假溢出，即队列中还有空间，可是队满的条件却成立了，为此改为环形队列，这样克服了假溢出。但并不能说环形队列一定优于非环形队列，因为环形队列中

出队元素的空间可能被后来进队的元素覆盖，如果算法要求在队列操作结束后利用进队的所有元素实现某种功能，这样环形队列就不适合了，这种情况下需要使用非环形队列，例如利用非环形队列求解迷宫路径就是这种情况。

9. 假设以 I 和 O 分别表示进栈和出栈操作，栈的初态和终栈均为空，进栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列。

（1）下面所示的序列中哪些是合法的？

A.IOIIOIOO　B.IOOIOIIO　C.IIIOIOIO　D.IIIOOIOO

（2）通过对（1）的分析，设计一个算法判定所给的操作序列是否合法。若合法返回真；否则返回假。（假设被判定的操作序列已存入一维数组中）。

解：（1）选项A、D均合法，而选项B、C不合法。因为在选项B中，先进栈一次，立即出栈3次，这会造成栈下溢。在选项C中共进栈5次，出栈3次，栈的终态不为空。

（2）本题使用一个链栈来判断操作序列是否合法，其中 *str* 为存放操作序列的字符数组，*n* 为该数组的字符个数（这里的 ElemType 类型设定为 char）。对应的算法如下：

```
bool judge(char str[],int n)
{   int i=0; ElemType x;
    LinkStNode *ls;
    bool flag=true;
    InitStack(ls);
    while (i<n && flag)
    {   if (str[i]=='I')            //进栈
            Push(ls,str[i]);
        else if (str[i]=='O')       //出栈
        {   if (StackEmpty(ls))
                flag=false;         //栈空时
            else
                Pop(ls,x);
        }
        else
            flag=false;             //其他值无效
        i++;
    }
    if (!StackEmpty(ls)) flag=false;
    DestroyStack(ls);
    return flag;
}
```

10. 假设表达式中允许包含 3 种括号：圆括号、方括号和大括号。编写一个算法判断表达式中的括号是否正确配对。

解：设置一个栈 *st*，扫描表达式 *exp*，遇到‘(’、‘[’或‘{’，则将其进栈；遇到‘)’，若栈顶是‘(’，则继续处理，否则以不配对返回假；遇到‘]’，若栈顶是‘[’，则继续处理，否则以不配对返回假；遇到‘}’，若栈顶是‘{’，则继续处理，否则以不配对返回假。在 *exp* 扫描完毕，若栈不空，则以不配对返回假；否则以括号配对返回真。本题算法如下：

```
bool Match(char exp[],int n)
{   LinkStNode *ls;
```

```
    InitStack(ls);
    int i=0;
    ElemType e;
    bool flag=true;
    while (i<n && flag)
    {   if (exp[i]=='(' || exp[i]='[' || exp[i]='{')
            Push(ls,exp[i]);   //遇到'('、'['或'{',则将其进栈
        if (exp[i]==')')       //遇到')',若栈顶是'(',则继续处理,否则以不配对返回
        {   if (GetTop(ls,e))
            {   if (e=='(') Pop(ls,e);
                else flag=false;
            }
            else flag=false;
        }
        if (exp[i]==']')       //遇到']',若栈顶是'[',则继续处理,否则以不配对返回
        {   if (GetTop(ls,e))
            {   if (e=='[') Pop(ls,e);
                else flag=false;
            }
            else flag=false;
        }
        if (exp[i]=='}')       //遇到'}',若栈顶是'{',则继续处理,否则以不配对返回
        {   if (GetTop(ls,e))
            {   if (e=='{') Pop(ls,e);
                else flag=false;
            }
            else flag=false;
        }
        i++;
    }
    if (!StackEmpty(ls)) flag=false; //若栈不空,则不配对
    DestroyStack(ls);
    return flag;
}
```

11. 设从键盘输入一序列的字符 a_1、a_2、…、a_n。设计一个算法实现这样的功能：若 a_i 为数字字符，a_i 进队，若 a_i 为小写字母时，将队首元素出队，若 a_i 为其他字符，表示输入结束。要求使用环形队列。

解：先建立一个环形队列 *qu*，用 while 循环接收用户输入，若输入数字字符，将其进队；若输入小写字母，出队一个元素，并输出它；若为其他字符，则退出循环。本题算法如下：

```
void fun()
{   ElemType a,e;
    SqQueue *qu;                        //定义队列指针
    InitQueue(qu);
    while (true)
    {   printf("输入 a:");
        scanf("%s",&a);
        if (a>='0' && a<='9')           //为数字字符
```

```
        {   if (!enQueue(qu,a))
                printf("  队列满,不能进队\n");
            }
            else if (a>='a' && a<='z')       //为小写字母
            {   if (!deQueue(qu,e))
                printf("  队列空,不能出队\n");
                else
                printf("  出队元素:%c\n",e);
            }
            else break;                              //为其他字符
        }
        DestroyQueue(qu);
    }
```

12. 设计一个算法，将一个环形队列（容量为 *n*，元素下标从 0 到 *n*−1）的元素倒置。例如，图 3.2（a）中为倒置前的队列（*n*=10），图 3.2（b）中为倒置后的队列。

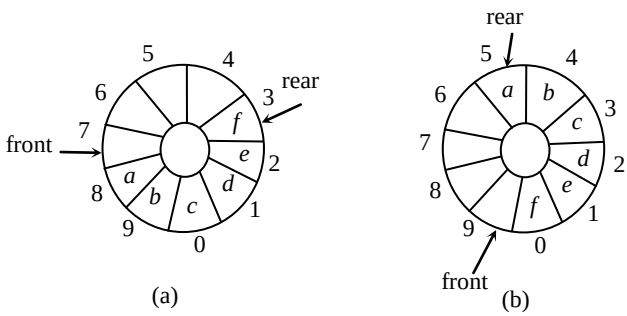

图 3.2　一个环形队列倒置前后的状态

解：使用一个临时栈 *st*，先将 *qu* 队列中所有元素出队并将其进栈 *st*，直到队列空为止。然后初始化队列 *qu*（队列清空），再出栈 *st* 的所有元素并将其进队 *qu*，最后销毁栈 *st*。对应的算法如下：

```
void Reverse(SqQueue *&qu)
{   ElemType e;
    SqStack *st;
    InitStack(st);
    while (!QueueEmpty(qu))      //队不空时,出队并进栈
    {   deQueue(qu,e);
        Push(st,e);
    }
    InitQueue(qu);               //队列初始化
    while (!StackEmpty(st))      //栈不空时,出栈并将元素入队
    {   Pop(st,e);
        enQueue(qu,e);
    }
    DestroyStack(st);
}
```

13. 编写一个程序，输入 *n*（由用户输入）个 10 以内的数，每输入 *i*（0≤*i*≤9），就把

它插入到第 *i* 号队列中。最后把 10 个队中非空队列，按队列号从小到大的顺序串接成一条链，并输出该链的所有元素。

解：建立一个队头指针数组 *quh* 和队尾指针数组 *qut*，*quh*[*i*]和 *qut*[*i*]表示 *i* 号（0≤*i*≤9）队列的队头和队尾，先将它们所有元素置为 NULL。对于输入的 *x*，采用尾插法将其链到 *x* 号队列中。然后按 0～9 编号的顺序把这些队列中的结点构成一个不带头结点的单链表，其首结点指针为 *head*。最后输出单链表 *head* 的所有结点值并释放所有结点。对应的程序如下：

```
#include <stdio.h>
#include <malloc.h>
#define MAXQNode 10                          //队列的个数
typedef struct node
{    int data;
     struct node *next;
} QNode;
void Insert(QNode *quh[],QNode *qut[],int x)  //将 x 插入到相应队列中
{    QNode *s;
     s=(QNode *)malloc(sizeof(QNode));       //创建一个结点 s
     s->data=x; s->next=NULL;
     if (quh[x]==NULL)                       //x 号队列为空队时
     {    quh[x]=s;
          qut[x]=s;
     }
     else                                    //x 号队列不空队时
     {    qut[x]->next=s;                    //将 s 结点链到 qut[x]所指结点之后
          qut[x]=s;                          //让 qut[x]仍指向尾结点
     }
}
void Create(QNode *quh[],QNode *qut[])       //根据用户输入创建队列
{    int n,x,i;
     printf("n:");
     scanf("%d",&n);
     for (i=0;i<n;i++)
     {    do
          {    printf("输入第%d 个数:",i+1);
               scanf("%d",&x);
          } while (x<0 || x>10);
          Insert(quh,qut,x);
     }
}
void DestroyList(QNode *&head)               //释放单链表
{    QNode *pre=head,*p=pre->next;
     while (p!=NULL)
     {
          free(pre);
          pre=p; p=p->next;
     }
     free(pre);
}
void DispList(QNode *head)                   //输出单链表的所有结点值
```

```
{    printf("\n 输出所有元素:");
     while (head!=NULL)
     {    printf("%d ",head->data);
          head=head->next;
     }
     printf("\n");
}
QNode *Link(QNode *quh[],QNode *qut[])    //将非空队列链接起来并输出
{    QNode *head=NULL,*tail;              //总链表的首结点指针和尾结点指针
     int i;
     for (i=0;i<MAXQNode;i++)             //扫描所有队列
          if (quh[i]!=NULL)               //i 号队列不空
          {    if (head==NULL)            //若 i 号队列为第一个非空队列
               {    head=quh[i];
                    tail=qut[i];
               }
               else                       //若 i 号队列不是第一个非空队列
               {    tail->next=quh[i];
                    tail=qut[i];
               }
          }
     tail->next=NULL;
     return head;
}
int main()
{    int i;
     QNode *head;
     QNode *quh[MAXQNode],*qut[MAXQNode]; //各队列的队头 quh 和队尾指针 qut
     for (i=0;i<MAXQNode;i++)
          quh[i]=qut[i]=NULL;             //置初值空
     Create(quh,qut);                     //建立队列
     head=Link(quh,qut);                  //链接各队列产生单链表
     DispList(head);                      //输出单链表
     DestroyList(head);                   //销毁单链表
     return 1;
}
```