# 第3章　栈和队列

## 一、基本内容

栈和队列的结构特性；在两种存储结构上如何实现栈和队列的基本操作，以及栈和队列在程序设计中的应用。

## 二、学习要点

（1）掌握栈和队列这两种抽象数据类型的特点，并能在相应的应用问题中正确选用它们。

（2）熟练掌握栈类型的两种实现方法，即两种存储结构表示时的基本操作实现算法，特别应注意栈满和栈空的条件以及它们的描述方法。

（3）熟练掌握循环队列和链队列的基本操作实现算法，特别注意队满和队空的描述方法。

**（4）理解递归算法执行过程中栈的状态变化过程。

**（5）理解递归算法到非递归算法的机械转换过程。

其中，后两点属于较高难度的学习内容，在标号左上角加上双重星号(**)，以示区别。

本章的习题明显地可看出有三类：第一类涉及栈的类型特点及其应用，如解答题1～10题，算法设计题1～9题；第二类涉及递归算法执行过程中栈的状态和递归的消除，如算法设计题10～13题；第三类涉及队列的类型特点和应用，以及在不同存储结构上的实现方法，如解答题12～14题和算法设计题14～20题。

## 三、可视交互学习内容与解析

### 1. 顺序栈的初建、入栈和出栈的结构形态变化

图3.1是AnyviewC的结构窗中对一个顺序栈的操作导致的结构形态变化的过程示例：图3.1(a)初建；图3.1(b)A入栈；图3.1(c)B、C和D依次入栈；图3.1(d)E入栈，栈满；图3.1(e)F入栈导致扩容5个单元；图3.1(f)F和E出栈。

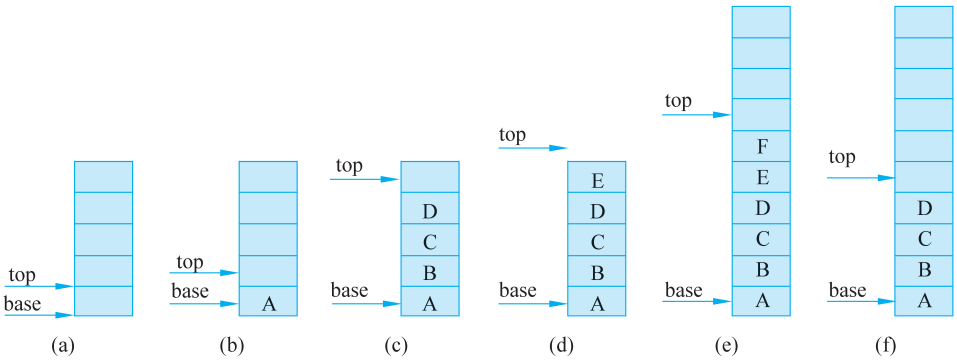



图3.1　顺序栈操作导致的结构形态变化的过程示例

### 2. 算法 3.5——汉诺塔问题递归求解过程中对运行栈的可视交互观察

作为首个关于递归函数运行栈的例子，教科书用较大篇幅讲述了汉诺塔问题递归求解过程中，运行栈的作用和变化，并在图 3.7 给出了示意图。在 AnyviewC 通过可视交互观察与算法语句执行同步的栈区变化，更容易加深理解。图 3.2 是与教科书图 3.7 前半部分对应的栈区状态变化截图，并包括调用移动一个盘的函数 move 的“栈迹”，栈中 RA 单元格内的值是函数（运行代码）返回地址。

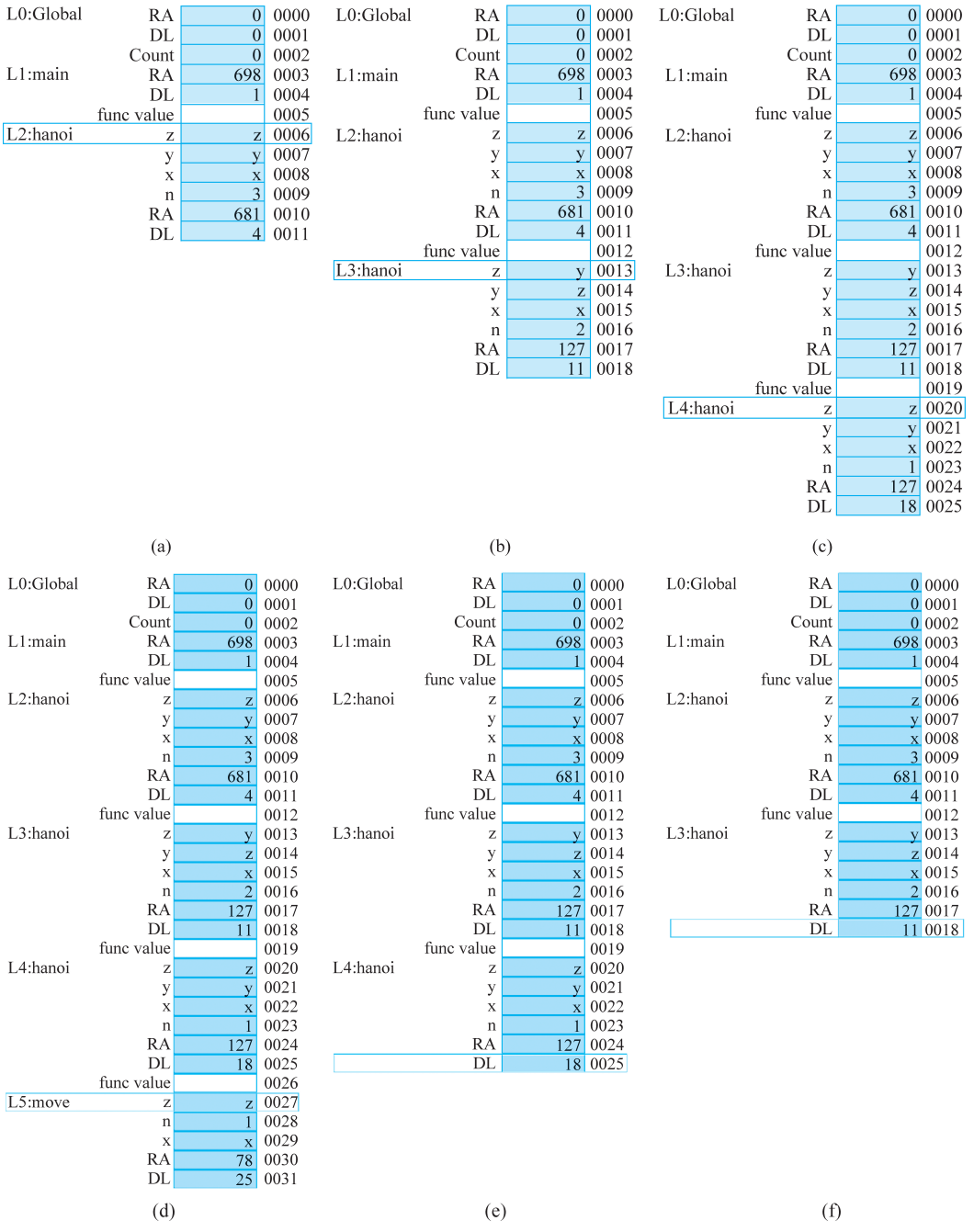
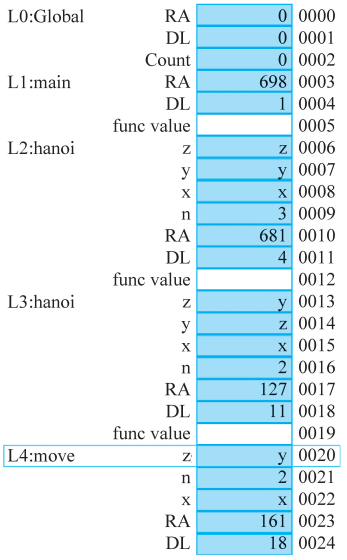
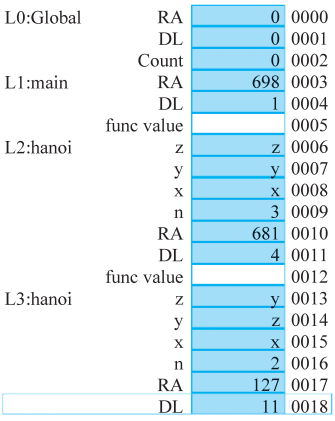


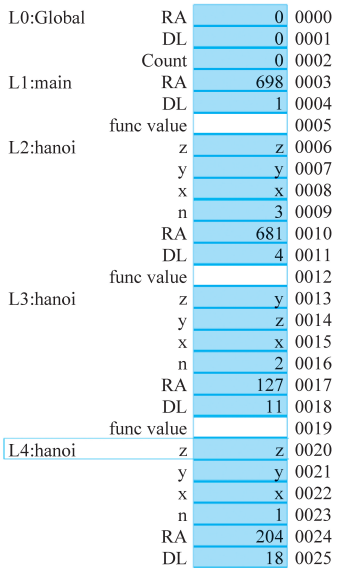

图 3.2　汉诺塔问题递归求解过程中运行栈区状态变化


L0:Global RA 0 0000
DL 0 0001
Count 0 0002
L1:main RA 698 0003
DL 1 0004
func value 0005
L2:hanoi z z 0006
y y 0007
x x 0008
n 3 0009
RA 681 0010
DL 4 0011
func value 0012
L3:hanoi z y 0013
y z 0014
x x 0015
n 2 0016
RA 127 0017
DL 11 0018
func value 0019
L4:move z y 0020
n 2 0021
x x 0022
RA 161 0023
DL 18 0024

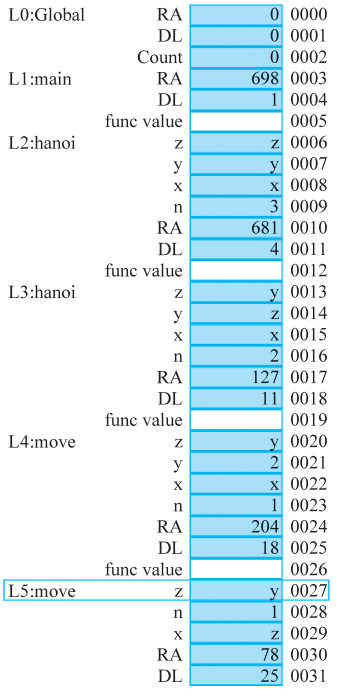
(g)


L0:Global RA 0 0000
DL 0 0001
Count 0 0002
L1:main RA 698 0003
DL 1 0004
func value 0005
L2:hanoi z z 0006
y y 0007
x x 0008
n 3 0009
RA 681 0010
DL 4 0011
func value 0012
L3:hanoi z y 0013
y z 0014
x x 0015
n 2 0016
RA 127 0017
DL 11 0018

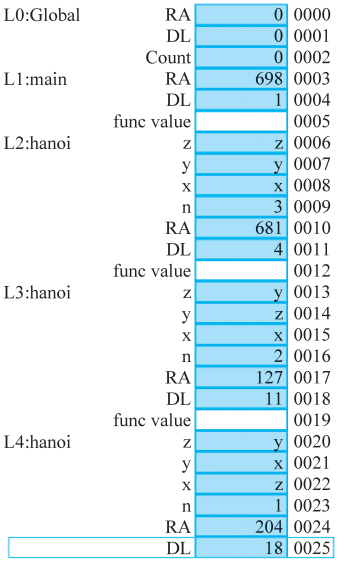
(h)


L0:Global RA 0 0000
DL 0 0001
Count 0 0002
L1:main RA 698 0003
DL 1 0004
func value 0005
L2:hanoi z z 0006
y y 0007
x x 0008
n 3 0009
RA 681 0010
DL 4 0011
func value 0012
L3:hanoi z y 0013
y z 0014
x x 0015
n 2 0016
RA 127 0017
DL 11 0018
func value 0019
L4:hanoi z z 0020
y y 0021
x x 0022
n 1 0023
RA 204 0024
DL 18 0025

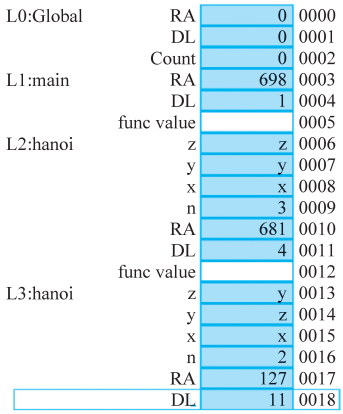
(i)


L0:Global RA 0 0000
DL 0 0001
Count 0 0002
L1:main RA 698 0003
DL 1 0004
func value 0005
L2:hanoi z z 0006
y y 0007
x x 0008
n 3 0009
RA 681 0010
DL 4 0011
func value 0012
L3:hanoi z y 0013
y z 0014
x x 0015
n 2 0016
RA 127 0017
DL 11 0018
func value 0019
L4:move z y 0020
y 2 0021
x x 0022
n 1 0023
RA 204 0024
DL 18 0025
func value 0026
L5:move z y 0027
n 1 0028
x z 0029
RA 78 0030
DL 25 0031

(j)


L0:Global RA 0 0000
DL 0 0001
Count 0 0002
L1:main RA 698 0003
DL 1 0004
func value 0005
L2:hanoi z z 0006
y y 0007
x x 0008
n 3 0009
RA 681 0010
DL 4 0011
func value 0012
L3:hanoi z y 0013
y z 0014
x x 0015
n 2 0016
RA 127 0017
DL 11 0018
func value 0019
L4:hanoi z y 0020
y x 0021
x z 0022
n 1 0023
RA 204 0024
DL 18 0025

(k)


L0:Global RA 0 0000
DL 0 0001
Count 0 0002
L1:main RA 698 0003
DL 1 0004
func value 0005
L2:hanoi z z 0006
y y 0007
x x 0008
n 3 0009
RA 681 0010
DL 4 0011
func value 0012
L3:hanoi z y 0013
y z 0014
x x 0015
n 2 0016
RA 127 0017
DL 11 0018

(l)

图 3.2 （续）

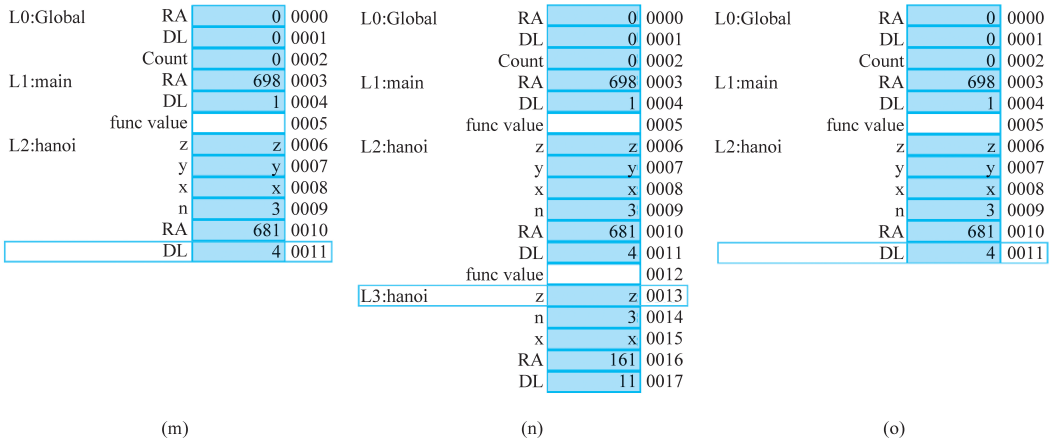



图 3.2 （续）

学习后续章节的众多递归算法或做算法设计题，都可以通过可视交互观察栈区，加深理解或辅助调试。

### 3. 循环队列的初建、入队和出队的结构形态变化

图 3.3 是 AnyviewC 的结构窗中对一个循环队列的操作导致的结构形态变化示例：图 3.3(a)初建；图 3.3(b)11、22 和 33 入队；图 3.3(c)44、55 入队，队列满；图 3.3(d)11 出队(front 增 1，不“抹去”11，可以体会内存不那么“干净”)；图 3.3(e)22 出队；图 3.3(f)77 入队(注意 rear 的指向)；图 3.3(g)88 入队，队列又满了(22 不在对内，所占的是空单元)。

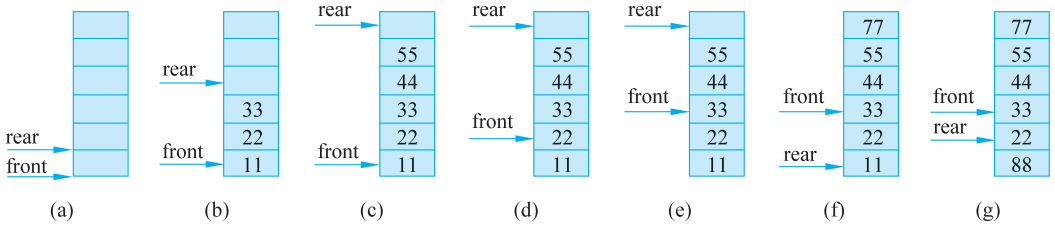



图 3.3 循环队列操作导致的结构形态变化示例

### 4. 算法 3.8——银行业务模拟过程的可视交互观察和理解

在教科书第 3.5 节，图 3.15 已经给出的算法 3.8 运行过程的主要数据结构的形态变化。在 AnyviewC 进行可视交互，可跟踪该算法对各数据结构操作的细节和观察结构形态变化，从而帮助理解这个多数据结构且较复杂的综合应用。图 3.4(a)是银行开门营业前各数据结构的初始状态；图 3.4(b)是模拟过程中的结构窗的截图。

读者可修改柜台数目、时间随机参数等，观察营业节奏的变化，加深对随机事件模拟方法的理解。

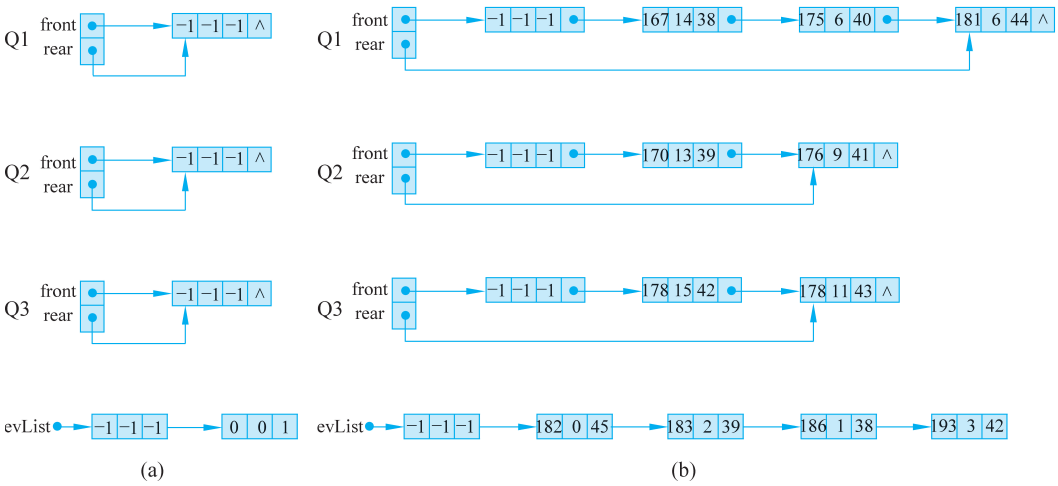



图 3.4　银行业务模拟过程中各数据结构的典型形态示例

## 四、基础知识题

### （一）单项选择题

**1.** 若进栈序列为 x，y，z，则通过入、出栈操作可能得到的 x，y，z 的不同排列的个数为(　　)。

A. 4　　B. 5　　C. 6　　D. 7

**2.** 一个栈的输入序列为 1，2，3，4，5，其合法的输出序列是(　　)。

A. 1，4，2，5，3　　B. 2，3，4，1，5　　C. 3，1，2，4，5　　D. 5，4，1，3，2

**3.** 若进栈序列为 1，2，3，4 且进栈和出栈可以穿插进行，则不可能出现的出栈序列是(　　)。

A. 2，4，3，1　　B. 3，2，4，1　　C. 2，4，1，3　　D. 2，3，1，4

**4.** 栈的两种常用的存储结构分别为(　　)。

A. 顺序存储结构和链式存储结构　　B. 顺序存储结构和散列存储结构

C. 链式存储结构和索引存储结构　　D. 链式存储结构和散列存储结构

**5.** 链栈与顺序栈相比，比较明显的优点是(　　)。

A. 插入操作更加方便　　B. 删除操作更加方便

C. 不会出现下溢的情况　　D. 不会出现上溢的情况

**6.** 导致栈上溢的操作是(　　)。

A. 栈满时执行的出栈　　B. 栈满时执行的入栈

C. 栈空时执行的出栈　　D. 栈空时执行的入栈

**7.** 上溢现象通常出现在(　　)。

A. 顺序栈的入栈操作过程中　　B. 顺序栈的出栈操作过程中

C. 链栈的入栈操作过程中　　D. 链栈的出栈操作过程中

**8.** 由两个栈共享一个向量空间的好处是(　　)。

A. 减少存取时间，降低下溢发生的概率

B. 节省存储空间，降低上溢发生的概率

C. 减少存取时间，降低上溢发生的概率

D. 节省存储空间，降低下溢发生的概率

**9.** 若数组 s[0..n－1]为两个栈 s1 和 s2 的共用存储空间，且仅当 s[0..n－1]全满时，各栈才不能进行进栈操作，则为这两个栈分配空间的最佳方案是：s1 和 s2 的栈顶指针的初值分别为(　　)。

A. 1 和 n＋1　　B. 1 和 n/2　　C. －1 和 n　　D. －1 和 n＋1

**10.** 栈和队列都是(　　)。

A. 限制存取位置的线性结构　　B. 顺序存储的线性结构

C. 链式存储的线性结构　　D. 限制存取位置的非线性结构

**11.** 队列和栈的主要区别是(　　)。

A. 逻辑结构不同　　B. 所包含的运算个数不同

C. 存储结构不同　　D. 限定插入和删除的位置不同

**12.** 判定“带头结点的链队列为空”的条件是(　　)。

A. Q->front＝＝NULL　　B. Q->rear＝＝NULL

C. Q->front＝＝Q->rear　　D. Q->front!＝Q->rear

**13.** 引起循环队列队头位置发生变化的操作是(　　)。

A. 出队　　B. 入队　　C. 取队头元素　　D. 取队尾元素

**14.** 在链队列的出队操作中，需要修改尾指针的条件是队列(　　)。

A. 原来已满　　B. 原来已空　　C. 变为满　　D. 变为空

**15.** 设数组 A[m]作为循环队列 Q 的存储空间，front 为队头指针，rear 为队尾指针，则判定 Q 为空队列的条件是(　　)。

A. (rear－front)％m＝＝1　　B. front＝＝rear

C. (rear－front)％m＝＝m－1　　D. front＝＝(rear＋1)％m

**16.** 设数组 data[m]作为循环队列 SQ 的存储空间，front 为队头指针，rear 为队尾指针，则执行出队操作后其头指针 front 值为(　　)。

A. front＝front＋1　　B. front＝(front＋1)％(m－1)

C. front＝(front－1)％m　　D. front＝(front＋1)％m

**17.** 假设以数组 A[m]存放循环队列的元素，其头、尾指针分别为 front 和 rear，则当前队列中的元素个数为(　　)。

A. (rear－front＋m)％m　　B. rear－front＋1

C. (front－rear＋m)％m　　D. (rear－front)％m

**18.** 已知循环队列的存储空间为数组 data[21]，且当前队列的头指针和尾指针的值分别为 8 和 3，则该队列的当前长度为(　　)。

A. 5　　B. 6　　C. 16　　D. 17

**19.** 如果循环队列 Q 只设尾指针 rear 和长度域 length，而且循环向量的长度为 m，那么队头的位置是(　　)。

A. Q->rear－Q->length　　B. (Q->rear－Q->length)％m

C. Q->rear＋Q->length　　D. (Q->rear－Q->length＋m)％m

**20.** 若允许表达式内多种括号混合嵌套，则通常为检查表达式中括号是否正确配对的算法选用的辅助结构是(　　)。

A. 栈　　B. 线性表　　C. 队列　　D. 二叉排序树

**21.** 设栈 S 和队列 Q 的初态均为空，元素 a,b,c,d,e,f 依次进入栈 S，若每个元素出栈后立即进入队列 Q，且 6 个元素的顺序是 b,d,c,f,e,a，则栈 S 的容量至少是(　　)。

A. 1　　B. 2　　C. 3　　D. 4

**22.** 若元素 a,b,c,d,e 依次进入输出受限双端队列，则不可能得到的出队序列是(　　)。

A. b,a,c,d,e　　B. d,b,a,c,e　　C. d,b,c,a,e　　D. e,c,b,a,d

**23.** 若以 a,b,c,d 作为输入序列，则能由输出受限的双端队列得到，但不能由输入受限的双端队列得到的输出序列是(　　)。

A. a,b,c,d　　B. d,c,b,a　　C. b,a,d,c　　D. d,b,a,c

### (二) 解答题

◆**1.** ①　若按教科书 3.1.1 节中图 3.1(b)所示铁道进行车厢调度(注意：两侧铁道均为单向行驶道)，则请回答：

(1) 如果进站的车厢序列为 1,2,3，可能得到的出站车厢序列是什么？

(2) 如果进站的车厢序列为 1,2,3,4,5,6，能否得到 4,3,5,6,1,2 和 1,3,5,4,2,6 的出站序列，并请说明为什么不能得到或者如何得到(即写出以'S'表示进栈和以'X'表示出栈的栈操作序列)。

**2.** ①　简述栈和线性表的差别。

**3.** ②　写出下列程序段的输出结果(栈的元素类型 SElemType 为 **char**)。

```
void main(){
    Stack S=InitStack();
    char x='c', y='k';
    Push(S, x);  Push(S, 'a');  Push(S, y);
    x=Pop(S);  Push(S, 't');  Push(S, x);
    x=Pop(S);  Push(S, 's');
    while (!StackEmpty(S)) { y=Pop(S);  printf(" %c", y); };
    printf(" %c\n", x);
}
```

**4.** ②　简述以下算法的功能(栈的元素类型 SElemType 为 **int**)。

(1)

```
Status algo1(Stack S) {
    int  n=0, A[255];
    while (!StackEmpty(S)) { n++;   A[n]=Pop(S); }
    for (int i=1; i<=n; i++) Push(S, A[i]);
}
```

(2)

```
Status algo2(Stack S,  int e) {
    int d;
```

```
    StackT=InitStack();
    while (!StackEmpty(S)) {
        d=Pop(S);
        if (d!=e) Push(T, d);
    }
    while (!StackEmpty(T)) {
        d=Pop(T);
        Push(S, d);
    }
}
```

◆5. ④　假设以 S 和 X 分别表示入栈和出栈的操作，则初态和终态均为栈空的入栈和出栈的操作序列可以表示为仅由 S 和 X 组成的序列。称可以操作的序列为合法序列（例如，SXSX 为合法序列，SXXS 为非法序列）。试给出区分给定序列为合法序列或非法序列的一般准则，并证明：两个不同的合法（栈操作）序列（对同一输入序列）不可能得到相同的输出元素（注意：在此指的是元素实体，而不是值）序列。

◆6. ④　试证明：若借助栈由输入序列 1，2，…，$n$ 得到的输出序列为 $p_1$，$p_2$，…，$p_n$（它是输入序列的一个排列），则在输出序列中不可能出现这样的情形：存在着 $i<j<k$ 使 $p_j<p_k<p_i$。

◆7. ①　按照四则运算加、减、乘、除和幂运算（^）优先关系的惯例，并仿照教科书 3.2 节例 3-1 的格式，画出对下列算术表达式求值时操作数栈和运算符栈的变化过程：

A－B×C/D＋E^F

8. ③　试推导求解 $n$ 阶梵塔问题至少要执行的 move 操作的次数。

9. ③　试将下列递推过程改写为递归过程。

```
void ditui(int n) {
    int i=n;
    while (i>1)
        printf("%d", i--);
}
```

◆10. ③　试将下列递归过程改写为非递归过程。

```
int test(int sum) {
    int x;   scanf("%d", x);
    if (x==0) sum=0;
    else { sum=test(sum);  sum+=x; }
    printf(" %d", sum);
    return sum;
}
```

11. ①　简述队列和栈这两种数据类型的相同点和差异处。

12. 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();
    char  x='e', y='c';
    EnQueue(Q, 'h');   EnQueue(Q, 'r');   EnQueue(Q, y);
```

```
    x=DeQueue(Q);   EnQueue(Q, x);
    x=DeQueue(Q);   EnQueue(Q, 'a');
    while (!QueueEmpty(Q)) {
        y = DeQueue(Q);
        printf(" %c", y);
    }
    printf(" %c", x);
}
```

**13.** ② 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) {
    int d;
    Stack S = InitStack(S);
    while (!QueueEmpty(Q)) {
        d = DeQueue(Q);  Push(S, d);
    }
    while (!StackEmpty(S)) {
        d = Pop(S);  EnQueue(Q, d);
    }
}
```

**14.** ④ 若以 1,2,3,4 作为双端队列的输入序列,试分别求出满足以下条件的输出序列。

(1) 能由输入受限的双端队列得到,但不能由输出受限的双端队列得到的输出序列。

(2) 能由输出受限的双端队列得到,但不能由输入受限的双端队列得到的输出序列。

(3) 既不能由输入受限的双端队列得到,也不能由输出受限的双端队列得到的输出序列。

## 五、算法设计题

◆**1.** ③ 假设以顺序存储结构实现一个双向栈,即在一维数组的存储空间中存在着两个栈,它们的栈底分别设在数组的两个端点。试编写实现这个双向栈 tws 的 3 个操作:初始化 InitStack(tws)、入栈 Push(tws,i,x)和出栈 Pop(tws,i)的算法,其中 i 为 0 或 1,用于分别指示设在数组两端的两个栈,并讨论按过程(正/误状态变量可设为指针参数)或函数设计这些操作算法各有什么优缺点。

**2.** ② 假设如解答题 1 所述火车调度站的入口处有 $n$ 节硬席或软席车厢(分别以 H 和 S 表示)等待调度,试编写算法,输出对这 $n$ 节车厢进行调度的操作(即入栈或出栈操作)序列,以使所有的软席车厢都被调整到硬席车厢之前。

◆**3.** ③ 试写一个算法,识别依次读入的一个以@为结束符的字符序列是否为形如 "序列 1&序列 2" 模式的字符序列。其中,序列 1 和序列 2 中都不含字符'&',且序列 2 是序列 1 的逆序列。例如,"a+b&b+a"是属该模式的字符序列,而 "1+3&3-1"则不是。

**4.** ② 试写一个判别表达式中开、闭括号是否配对出现的算法。

◆**5.** ④ 假设一个算术表达式中可以包含 3 种括号:圆括号'('和')'、方括号'['和']'和花括号'{'和'}',且这 3 种括号可按任意的次序嵌套使用(如…[…{…} …[…]…]…[…]…(…)

…)。编写判别给定表达式中所含括号是否正确配对出现的算法(已知表达式已存入数据元素为字符的顺序表中)。

**6.** ③ 假设以二维数组 g[1..m][1..n]表示一个图像区域,g[i,j]表示该区域中点$(i,j)$所具颜色,其值为 0～$k$ 的整数。编写算法置换点 $(i_0,j_0)$所在区域的颜色。约定和 $(i_0,j_0)$同色的上、下、左、右的邻接点为同色区域的点。

**◆7.** ③ 假设表达式由单字母变量和双目四则运算符构成。试写一个算法,将一个通常书写形式且书写正确的表达式转换为逆波兰式。

**8.** ③ 如题 7 的假设条件,试写一个算法,对以逆波兰式表示的表达式求值。

**9.** ⑤ 如题 7 的假设条件,试写一个算法,判断给定的非空后缀表达式是否为正确的逆波兰式(即后缀表达式),如果是,则将它转换为波兰式(即前缀表达式)。

**10.** ③ 试编写如下定义的递归函数的递归算法,并根据算法画出求 $g(5,2)$时栈的变化过程。

$$g(m,n)=\begin{cases}0, & m=0,n\geqslant 0\\ g(m-1,2n)+n, & m>0,n\geqslant 0\end{cases}$$

**11.** ④ 试写出求递归函数 $F(n)$的递归算法,并消除递归:

$$F(n)=\begin{cases}n+1, & n=0\\ n\cdot F(n/2), & n>0\end{cases}$$

**12.** ④ 求解平方根$\sqrt{A}$的迭代函数定义如下:

$$\mathrm{sqrt}(A,p,e)=\begin{cases}p, & |p^2-A|<e\\ \mathrm{sqrt}\left(A,\dfrac{1}{2}\left(p+\dfrac{A}{p}\right),e\right), & |p^2-A|\geqslant e\end{cases}$$

其中,$p$ 是 $A$ 的近似平方根,$e$ 是结果允许误差。试写出相应的递归算法,并消除递归。

**13.** ⑤ 已知 Ackerman 函数的定义如下:

$$\mathrm{Akm}(m,n)=\begin{cases}n+1, & m=0\\ \mathrm{Akm}(m-1,1), & m\neq 0,n=0\\ \mathrm{Akm}(m-1,\mathrm{Akm}(m,n-1)), & m\neq 0,n\neq 0\end{cases}$$

(1) 写出递归算法。

(2) 写出非递归算法。

(3) 根据非递归算法,画出求 Akm(2,1)时栈的变化过程。

**◆14.** ② 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结点(注意不设头指针),试编写相应的队列初始化、入队列和出队列的算法。

**15.** ③ 如果希望循环队列中的元素都能得到利用,则需设置一个标志域 tag,并以 tag 的值为 0 或 1 来区分,尾指针和头指针值相同时的队列状态是“空”还是“满”。试编写与此结构相应的入队列和出队列的算法,并从时间和空间角度讨论设标志和不设标志这两种方法的使用范围(如当循环队列容量较小而队列中每个元素占的空间较多时,哪一种方法较好)。

**◆16.** ② 假设将循环队列定义为:以域变量 rear 和 length 分别指示循环队列中队尾元素的位置和内含元素的个数。试给出此循环队列的队满条件,并写出相应的入队列和出队列的算法(在出队列的算法中要返回队头元素)。

**◆17.** ③ 假设称正读和反读都相同的字符序列为“回文”,例如,"abba" 和 "abcba"是回

文，"abcde" 和 "ababab" 则不是回文。试写一个算法判别读入的一个以'@'为结束符的字符序列是不是“回文”。

◆**18.** ④　试利用循环队列编写求 $k$ 阶斐波那契序列中前 $n+1$ 项（$f_0, f_1, \cdots, f_n$）的算法，要求满足：$f_n \leqslant \max$ 而 $f_{n+1} > \max$，其中 max 为某个约定的常数。（注意：本题所用循环队列的容量仅为 $k$，则在算法执行结束时，留在循环队列中的元素应是所求 $k$ 阶斐波那契序列中的最后 $k$ 项 $f_{n-k+1}, \cdots, f_n$）。

**19.** ③　在顺序存储结构上实现输出受限的双端循环队列的入列和出列（只允许队头出列）算法。设每个元素表示一个待处理的作业，元素值表示作业的预计时间。入队列采取简化的短作业优先原则，若一个新提交的作业的预计执行时间小于队头和队尾作业的平均时间，则插入队头，否则插入队尾。

**20.** ③　假设在如教科书 3.4.1 节中图 3.9 所示的铁道转轨网的输入端有 $n$ 节车厢：硬座、硬卧和软卧（分别以 P、H 和 S 表示）等待调度，要求这 3 种车厢在输出端铁道上的排列次序为：硬座在前，软卧在中，硬卧在后。试利用输出受限的双端队列对这 $n$ 节车厢进行调度，编写算法输出调度的操作序列：分别以字符'E'和'D'表示对双端队列的头端进行入队列和出队列的操作；以字符'A'表示对双端队列的尾端进行入队列的操作。