

第3章

栈和队列



3.1

本章知识体系



1. 知识结构图

本章的知识结构如图 3.1 所示。

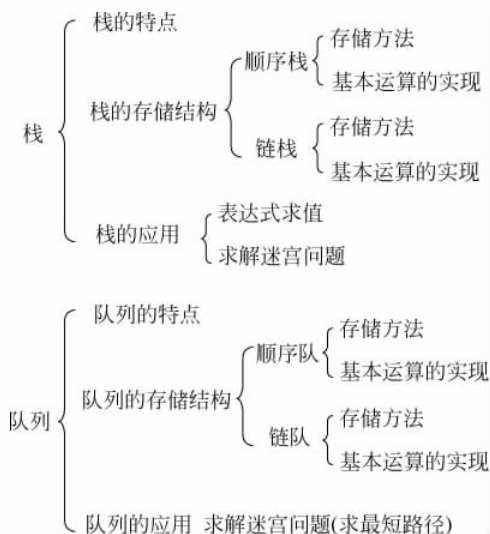


图 3.1 第 3 章知识结构图

2. 基本知识点

- (1) 栈、队列和线性表的异同。
- (2) 顺序栈的基本运算算法设计。
- (3) 链栈的基本运算算法设计。
- (4) 顺序队的基本运算算法设计。
- (5) 环形队列和非环形队列的特点。
- (6) 链队的基本运算算法设计。
- (7) 利用栈/队列求解复杂的应用问题。

3. 要点归纳

- (1) 栈和队列的共同点是它们的数据元素都呈线性关系,且只允许在端点处插入和删除元素。
- (2) 栈是一种“后进先出”的数据结构,只能在同一端进行元素的插入和删除。
- (3) 栈可以采用顺序栈和链栈两类存储结构。
- (4) n 个不同元素的进栈顺序和出栈顺序不一定相同。
- (5) 在顺序栈中通常用栈顶指针指向当前栈顶的元素。
- (6) 在顺序栈中用数组 $\text{data}[0..\text{MaxSize}-1]$ 存放栈中元素,只能将一端作为栈底,另一端作为栈顶,通常的做法是将 $\text{data}[0]$ 端作为栈底, $\text{data}[\text{MaxSize}-1]$ 端作为栈顶。用户

也可以将 $\text{data}[\text{MaxSize}-1]$ 端作为栈底, $\text{data}[0]$ 端作为栈顶, 但不能将中间位置作为栈底或者栈顶。

(7) 初始时栈顶指针 top 设置为 -1 , 栈空的条件为 $\text{top} = -1$, 栈满的条件为 $\text{top} = \text{MaxSize}-1$, 元素 x 的进栈操作是 $\text{top}++$; $\text{data}[\text{top}] = x$, 出栈操作是 $x = \text{data}[\text{top}]$; $\text{top}--$ 。这是经典做法, 但不是唯一的方法, 如果初始时 top 设置为 0 , 可以设置栈空的条件为 $\text{top} = 0$, 栈满的条件为 $\text{top} = \text{MaxSize}$, 元素 x 的进栈操作是 $\text{data}[\text{top}] = x$; $\text{top}++$, 出栈操作是 $\text{top}--$; $x = \text{data}[\text{top}]$ 。

(8) 在顺序栈或链栈中, 进栈和出栈操作不涉及栈中元素的移动。

(9) 在链栈中, 由于每个结点是单独分配的, 通常不考虑上溢问题。

(10) 无论是顺序栈还是链栈, 进栈和出栈运算的时间复杂度均为 $O(1)$ 。

(11) 队列是一种“先进先出”的数据结构, 只能从一端插入元素, 从另一端删除元素。

(12) 队列可以采用顺序队和链队两类存储结构。

(13) n 个元素进队的顺序和出队顺序总是一致的。

(14) 在顺序队中的元素个数可以由队头指针和队尾指针计算出来。

(15) 环形队列也是一种顺序队, 是通过逻辑方法使其首尾相连的, 解决非环形队列的假溢出现象。

(16) 在环形队列中, 队头指针 f 指向队头元素的前一个位置, 队尾指针 r 指向队尾元素, 这是一种经典做法, 但不是唯一的方法, 也可以让队头指针 f 指向队头元素。

(17) 无论是顺序队还是链队, 进队和出队运算的时间复杂度均为 $O(1)$ 。

(18) 在实际应用中, 一般栈和队列都是用来存放临时数据的, 如果先保存的元素先处理, 应该采用队列; 如果后保存的元素先处理, 应该采用栈。

3.2

教材中的练习题及参考答案 *

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 $st1$ 栈中, 将所有偶数进到 $st2$ 栈中。
- (2) 先将 $st1$ 的所有元素(奇数元素)退栈, 放到数组 a 中并覆盖原有位置的元素; 再将 $st2$ 的所有元素(偶数元素)退栈, 放到数组 a 中并覆盖原有位置的元素。
- (3) 销毁两个栈 $st1$ 和 $st2$ 。

所以本算法的功能是利用两个栈将数组 a 中的所有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, e);
    }
    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 中的所有元素, 将不为 x 的元素进栈到 $tmps$ 中。
- (3) 退栈 $tmps$ 中的所有元素, 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 个出队的元素以外, 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: 在用数组表示队列时把数组看成是一个环形的, 即令数组中的第一个元素紧跟在最末一个单元之后就形成了一个环形队列。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? 在什么情况下使用非环形队列?

答: 队列主要用于保存中间数据, 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现象, 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, 这样克服了假溢出现象。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, 在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, 栈的初态和终态均为空, 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) 在下面所示的序列中哪些是合法的?

- A. IOIOIOIO B. IOOIOIO C. IIOIOIO D. IIOOIOIO

(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);
        }
        i++;
    }
    return flag;
}
```

```

        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圆括号、方括号和大括号 3 种括号,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,当遇到'(',')'或'{'时将其进栈;当遇到')'时,若栈顶是'(',则继续处理,否则以不配对返回假;当遇到']'时,若栈顶是'[',则继续处理,否则以不配对返回假;当遇到'}'时,若栈顶是'{',则继续处理,否则以不配对返回假。在 exp 扫描完毕后,若栈不空,则以不配对返回假;否则以括号配对返回真。本题的算法如下:

```

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'{'时,将其进栈
        if (exp[i] == ')')               //遇到')',若栈顶是'(',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'[',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'{',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 $a_1, a_2, \dots, a_n$ 。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, 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 循环接收用户的输入, 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。本题的算法如下:

```

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) 为倒置前的队列 ($n=10$), 图 3.2(b) 为倒置后的队列。

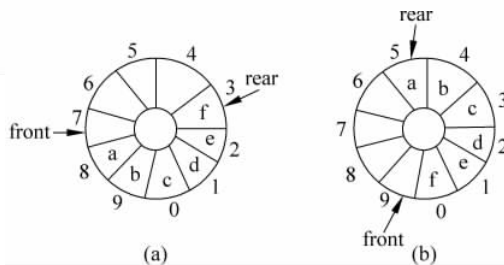


图 3.2 一个环形队列倒置前后的状态

解：使用一个临时栈 st, 先将 qu 队列中的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 ($0 \leq i \leq 9$) 就把它插入到第 i 号队列中, 最后把 10 个队中的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 ($0 \leq i \leq 9$) 队列的队头和队尾, 先将它们的所有元素置为 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] 所指的结点之后
        qut[x] = s;                                //让 qut[x] 仍指向尾结点
    }
}
```

```

    }
}

void Create(QNode * quh[], QNode * qut[]) //根据用户的输入创建队列
{
    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;
}

```

3.3

补充练习题及参考答案



3.3.1 单项选择题

1. 以下数据结构中元素之间为线性关系的是_____。

- A. 栈 B. 队列 C. 线性表 D. 以上都是

答: D。

2. 栈和队列的共同点是_____。

- A. 都是先进后出 B. 都是先进先出
C. 只允许在端点处插入和删除元素 D. 没有其特点

答: 栈和队列都是受限线性表, 所谓“受限”指的是在端点处插入和删除元素, 所以本题的答案为 C。

3. 经过以下栈运算后 x 的值是_____。

InitStack(s); Push(s, a); Push(s, b); Pop(s, x); GetTop(s, x);

- A. a B. b C. 1 D. 0

答: A。

4. 经过以下栈运算后 StackEmpty(s) 的值是_____。

InitStack(s); Push(s, a); Push(s, b); Pop(s, x); Pop(s, y)

- A. a B. b C. 1 D. 0

答: C。

5. 设一个栈的输入序列为 a, b, c, d , 则借助一个栈所得到的输出序列不可能的是_____。

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

答: 以 d 开头的出栈序列只有 d, c, b, a 一种。本题的答案为 D。

6. 已知一个栈的进栈序列是 $1, 2, 3, \dots, n$, 其输出序列是 $p_1, p_2, \dots, p_n$, 若 $p_1 = n$, 则 p_i 的值是_____。

- A. i B. $n-i$ C. $n-i+1$ D. 不确定

答: 当 $p_1=n$ 时输出序列只有一种, 即 $n, n-1, \dots, 3, 2, 1$, 则 $p_2=n-1, p_3=n-2, \dots, p_n=1$, 推断出 $p_i=n-i+1$, 本题的答案为 C。

7. 设 n 个元素进栈序列是 $1, 2, 3, \dots, n$, 其输出序列是 $p_1, p_2, \dots, p_n$, 若 $p_1=3$, 则 p_2 的值_____。

- A. 一定是 2 B. 一定是 1 C. 不可能是 1 D. 以上都不对

答: 当 $p_1=3$ 时, 说明 1、2、3 先依次进栈, 出栈 3, 然后可能出栈 2, 也可能是 4 或后面的元素进栈后再出栈。因此, p_2 可能是 2, 也可能是 4, $\dots, n$, 但一定不是 1。本题的答案为 C。

8. 设 n 个元素进栈序列是 $p_1, p_2, p_3, \dots, p_n$, 其输出序列是 $1, 2, 3, \dots, n$, 若 $p_n=1$, 则 $p_i (1 \leq i \leq n-1)$ 的值是_____。

- A. $n-i+1$ B. $n-i$ C. i D. 有多种可能

答: 当 $p_n=1$ 时, 表示它是第一个出栈元素, 因此这样的输出序列是唯一的, 即有 $p_{n-1}=2, p_{n-2}=3, \dots, p_1=n$, 也就是说 $p_i=n-i+1$ 。本题的答案为 A。

9. 一个栈的入栈序列为 $1, 2, 3, \dots, n$, 其出栈序列是 $p_1, p_2, p_3, \dots, p_n$ 。若 $p_2=3$, 则 p_3 可能取值的个数是_____。

- A. $n-3$ B. $n-2$ C. $n-1$ D. 无法确定

答: 若 1 进栈, 1 出栈(p_1), 2 进栈, 3 进栈, 3 出栈(p_2), 之后可以 2 出栈(p_3), 也可以 4~ n 的任何元素进栈再出栈(p_3), 所以 p_3 可以是 2 或者 4~ n 。另外, 1、2 依次进栈, 2 出栈(p_1), 3 进栈, 3 出栈(p_2), 1 出栈(p_3)。也就是说, p_3 可以是 3 以外的任何元素。本题的答案为 C。

10. 设栈 S 和队列 Q 的初始状态为空, 元素 $e_1 \sim e_6$ 依次通过栈 S , 一个元素出后即进队列 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

答: 操作过程为 e_1 进栈, e_2 进栈, e_2 出栈后进队, e_3 进栈, e_4 进栈, e_4 出栈后进队, e_3 出栈后进队, e_5 进栈, e_6 进栈, e_6 出栈后进队, e_5 出栈后进队, e_1 出栈后进队, 栈中元素最多时为 3 个。本题的答案为 C。

11. 判定一个顺序栈 st (元素的个数最多为 $MaxSize$) 为空的条件可以设置为_____。

- A. $st \rightarrow top == MaxSize/2$ B. $st \rightarrow top != MaxSize/2$
C. $st \rightarrow top != MaxSize-1$ D. $st \rightarrow top == MaxSize-1$

答: 顺序栈总是以一端(0 或者 $MaxSize-1$ 端)作为栈底, 栈空是指栈不存在元素, 合适的栈空条件为 $st \rightarrow top == MaxSize-1$ 。本题的答案为 D。

12. 若一个栈用数组 $data[1..n]$ 存储, 初始栈顶指针 top 为 $n+1$, 则以下元素 x 进栈的操作正确的是_____。

- A. $top++$; $data[top]=x$; B. $data[top]=x$; $top++$;
C. $top--$; $data[top]=x$; D. $data[top]=x$; $top--$;

答: 初始栈顶指针 top 为 $n+1$, 说明是将 $data[n]$ 端作为栈底、 $data[1]$ 端作为栈顶, 在进栈时 top 应递减, 由于不存在 $data[n+1]$ 的元素, 所以在进栈时应先将 top 递减, 再将 x 放在 top 处。本题的答案为 C。

13. 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 进栈的操

作正确的是_____。

- A. $\text{top}++$; $\text{data}[\text{top}]=x$; B. $\text{data}[\text{top}]=x$; $\text{top}++$;
C. $\text{top}--$; $\text{data}[\text{top}]=x$; D. $\text{data}[\text{top}]=x$; $\text{top}--$;

答: 初始栈顶指针 top 为 n , 说明是将 $\text{data}[n]$ 端作为栈底、 $\text{data}[1]$ 端作为栈顶, 在进栈时 top 应递减, 由于存在 $\text{data}[n]$ 的元素, 所以在进栈时应先将 x 放在 top 处, 再将 top 递减。本题的答案为 D。

14. 若一个栈用数组 $\text{data}[1..n]$ 存储, 初始栈顶指针 top 为 0, 则以下元素 x 进栈的操作正确的是_____。

- A. $\text{top}++$; $\text{data}[\text{top}]=x$; B. $\text{data}[\text{top}]=x$; $\text{top}++$;
C. $\text{top}--$; $\text{data}[\text{top}]=x$; D. $\text{data}[\text{top}]=x$; $\text{top}--$;

答: 初始栈顶指针 top 为 0, 说明是将 $\text{data}[1]$ 端作为栈底、 $\text{data}[n]$ 端作为栈顶, 在进栈时 top 应递增, 由于不存在 $\text{data}[0]$ 的元素, 所以在进栈时应先将 top 递增, 再将 x 放在 top 处。本题的答案为 A。

15. 若一个栈用数组 $\text{data}[1..n]$ 存储, 初始栈顶指针 top 为 1, 则以下元素 x 进栈的操作正确的是_____。

- A. $\text{top}++$; $\text{data}[\text{top}]=x$; B. $\text{data}[\text{top}]=x$; $\text{top}++$;
C. $\text{top}--$; $\text{data}[\text{top}]=x$; D. $\text{data}[\text{top}]=x$; $\text{top}--$;

答: 初始栈顶指针 top 为 1, 说明是将 $\text{data}[1]$ 端作为栈底、 $\text{data}[n]$ 端作为栈底, 在进栈时 top 应递增, 由于存在 $\text{data}[1]$ 的元素, 所以在进栈时应先将 x 放在 top 处, 再将 top 递增。本题的答案为 B。

说明: 从 12~15 小题可以看出, 顺序栈的设计并不是唯一的, 只要能满足栈的操作特点又能充分利用存储空间就是一种合适的设计。

16. 以下各链表均不带有头结点, 其中最不适合用作链栈的链表是_____。

- A. 只有表头指针没有表尾指针的循环双链表
B. 只有表尾指针没有表头指针的循环双链表
C. 只有表尾指针没有表头指针的循环单链表
D. 只有表头指针没有表尾指针的循环单链表

答: 只有表头指针没有表尾指针的循环单链表(不带头结点)在进栈和出栈操作后需要保持循环单链表形式不变, 实现进栈和出栈运算的时间复杂度均为 $O(n)$ 。本题的答案为 D。

17. 由两个栈共享一个数组空间的好处是_____。

- A. 减少存取时间, 降低上溢出发生的几率
B. 节省存储空间, 降低上溢出发生的几率
C. 减少存取时间, 降低下溢出发生的几率
D. 节省存储空间, 降低下溢出发生的几率

答: B。

18. 表达式 $a * (b + c) - d$ 的后缀表达式是_____。

- A. $abcd * + -$ B. $abc + * d -$
C. $abc * + d -$ D. $- + * abcd$

答：选项 A 对应的中缀表达式为 $a - (b + c * d)$ ，选项 B 对应的中缀表达式为 $a * (b + c) - d$ ，选项 C 对应的中缀表达式为 $(a + b * c) - d$ ，选项 D 不是合法的后缀表达式。本题的答案为 B。

19. 在将算术表达式“ $1 + 6 / (8 - 5) * 3$ ”转换成后缀表达式的过程中，当扫描到 5 时运算符栈（从栈顶到栈底次序）为_____。

- A. $- / +$ B. $- (/ +$ C. $/ +$ D. $/ - +$

答：算术表达式“ $1 + 6 / (8 - 5) * 3$ ”的后缀表达式是“ $1\ 6\ 8\ 5\ -\ /\ 3\ *\ +$ ”，当扫描到 5 时，前面的运算符 $+、/、($ 和 $-$ 均在栈中，运算符栈中从栈顶到栈底次序为 $- (/ +$ 。本题的答案为 B。

20. 在利用栈求表达式的值时，设立运算数栈 OPND，设 OPND 只有两个存储单元，在求下列表达式中不发生上溢出的是_____。

- A. $a - b * (c + d)$ B. $(a - b) * c + d$
C. $(a - b * c) + d$ D. $(a - b) * (c + d)$

答：选项 A 对应的后缀表达式为 $a\ b\ c\ d\ +\ *\ -$ ，在求值时 OPND 的最少存储单元为 4。选项 B 对应的后缀表达式为 $a\ b\ -\ c\ *\ d\ +$ ，在求值时 OPND 的最少存储单元为 2。选项 C 对应的后缀表达式为 $a\ b\ c\ *\ -\ d\ +$ ，在求值时 OPND 的最少存储单元为 3。选项 D 对应的后缀表达式为 $a\ b\ -\ c\ d\ +\ *$ ，在求值时 OPND 的最少存储单元为 3。本题的答案为 B。

21. 经过以下队列运算后 QueueEmpty(qu) 的值是_____。

InitQueue(qu); enqueue(qu, a); enqueue(qu, b); dequeue(qu, x); dequeue(qu, y);

- A. a B. b C. true D. false

答：C。

22. 环形队列_____。

- A. 不会产生下溢出 B. 不会产生上溢出
C. 不会产生假溢出 D. 以上都不对

答：C。

23. 在环形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

答：A。

24. 某环形队列的元素类型为 char，队头指针 front 指向队头元素的前一个位置，队尾指针 rear 指向队尾元素，如图 3.3 所示，则队中元素为_____。

- A. abcd123456 B. abcd123456c C. dfgbca D. cdfgbcab

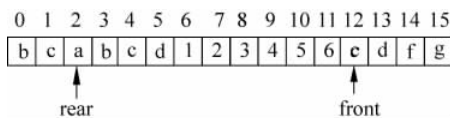


图 3.3 一个环形队列

答: $\text{front}=12$, 队头元素应为下标 13 的元素, $\text{rear}=2$, 队尾元素应为下标 2 的元素, 队中元素从队头到队尾是 $\text{data}[13..2]$, 本题的答案为 C。

25. 已知环形队列存储在一维数组 $A[0..n-1]$ 中, 且队列非空时 front 和 rear 分别指向队头元素和队尾元素。若初始时队列空, 且要求第一个进入队列的元素存储在 $A[0]$ 处, 则初始时 front 和 rear 的值分别是_____。

- A. 0, 0 B. 0, $n-1$ C. $n-1, 0$ D. $n-1, n-1$

答: 在环形队列中, 进队操作是队尾指针 rear 循环加 1, 再在该处放置进队的元素, 这里要求第一个进队元素存储在 $A[0]$ 处, 则 rear 应为 $n-1$, 因为这样 $(\text{rear}+1)\%n=0$ 。而队头指向队头元素, 此时队头位置为 0, 所以 front 的初值为 0。本题的答案为 B。

26. 若某环形队列有队头指针 front 和队尾指针 rear , 在队不满时进队操作仅会改变_____。

- A. front B. rear C. front 和 rear D. 以上都不对

答: B。

27. 设环形队列中数组的下标是 $0 \sim N-1$, 其队头指针为 f (指向队头元素的前一个位置)、队尾指针为 r (指向队尾元素), 则其元素个数为_____。

- A. $r-f$ B. $r-f-1$
C. $(r-f)\%N+1$ D. $(r-f+N)\%N$

答: 对于非环形队列, 每次是先移动指针, 再存取元素, 其中的元素个数 $= r-f$, 但由于是环形队列, r 可能小于 f , 为此求环形队列中元素个数的公式改为 $(r-f+N)\%N$ 。本题的答案为 D。

28. 设环形队列的存储空间为 $a[0..20]$, 且当前队头指针 (f 指向队首元素的前一个位置) 和队尾指针 (r 指向队尾元素) 的值分别为 8 和 3, 则该队列中的元素个数为_____。

- A. 5 B. 6 C. 16 D. 17

答: 这里 $\text{MaxSize}=21$, 其中的元素个数 $= (r-f+\text{MaxSize})\%\text{MaxSize}=16$ 。本题的答案为 C。

29. 设环形队列中数组的下标是 $0 \sim N-1$, 已知其队头指针 f (f 指向队首元素的前一个位置) 和队中元素个数 n , 则队尾指针 r (r 指向队尾元素的位置) 为_____。

- A. $f-n$ B. $(f-n)\%N$
C. $(f+n)\%N$ D. $(f+n+1)\%N$

答: C。

30. 设环形队列中数组的下标是 $0 \sim N-1$, 已知其队尾指针 r (r 指向队尾元素的位置) 和队中元素个数 n , 则队尾指针 f (f 指向队头元素的前一个位置) 为_____。

- A. $r-n$ B. $(r-n)\%N$
C. $(r-n+N)\%N$ D. $(r+n)\%N$

答: C。

31. 若用一个大小为 6 的数组来实现环形队列, rear 作为队尾指针指向队列中的尾部元素, front 作为队头指针指向队头元素的前一个位置。现在 rear 和 front 的值分别是 0 和 3, 当从队列中删除一个元素再加入两个元素后 rear 和 front 的值分别是_____。

- A. 1 和 5 B. 2 和 4 C. 4 和 2 D. 5 和 1

答：删除一个元素时 front 循环增 1，进队两个元素时 rear 循环增 2。本题的答案为 B。

32. 有一个环形队列 qu(存放元素位置 $0 \sim \text{MaxSize}-1$)，rear 作为队尾指针指向队列中的尾部元素，front 作为队头指针指向队头元素的前一个位置，则队满的条件是_____。

- A. $\text{qu} \rightarrow \text{front} == \text{qu} \rightarrow \text{rear}$
- B. $\text{qu} \rightarrow \text{front} + 1 == \text{qu} \rightarrow \text{rear}$
- C. $\text{qu} \rightarrow \text{front} == (\text{qu} \rightarrow \text{rear} + 1) \% \text{MaxSize}$
- D. $\text{qu} \rightarrow \text{rear} == (\text{qu} \rightarrow \text{front} + 1) \% \text{MaxSize}$

答：根据环形队列的结构很快可以排除选项 A 和 B(因为它们与 MaxSize 无关)。环形队列中约定，当进队一个元素后到达了队头就表示队满，进队操作是 rear 循环增 1。本题的答案为 C。

33. 假设用 $Q[0..M]$ 实现环形队列，f 作为队头指针指向队头元素的前一个位置，r 作为队尾指针指向队尾元素。若用“ $(r+1) \% (M+1) == f$ ”作为队满的标志，则_____。

- A. 可用“ $f == r$ ”作为队空的标志
- B. 可用“ $f > r$ ”作为队空的标志
- C. 可用“ $(f+1) \% (M+1) == r$ ”作为队空的标志
- D. 队列中最多可以有 $M+1$ 个元素

答：这里 MaxSize 等于 $M+1$ ，若用“ $(r+1) \% (M+1) == f$ ”作为队满的标志，队列中最多可以有 M 个元素。本题的答案为 A。

34. 环形队列存放在一维数组 $A[0..M-1]$ 中，end1 指向队头元素，end2 指向队尾元素的后一个位置。假设队列两端均可以进行入队和出队操作，队列中最多能容纳 $M-1$ 个元素，初始时为空。下列判断队空和队满的条件中正确的是_____。

- A. 队空： $\text{end1} == \text{end2}$ ；队满： $\text{end1} == (\text{end2} + 1) \bmod M$
- B. 队空： $\text{end1} == \text{end2}$ ；队满： $\text{end2} == (\text{end1} + 1) \bmod (M-1)$
- C. 队空： $\text{end2} == (\text{end1} + 1) \bmod M$ ；队满： $\text{end1} == (\text{end2} + 1) \bmod M$
- D. 队空： $\text{end1} == (\text{end2} + 1) \bmod M$ ；队满： $\text{end2} == (\text{end1} + 1) \bmod (M-1)$

答：这里环形队列是让队头指针指向队头元素、队尾指针指向队尾元素的后一个位置，和经典做法让队头指针指向队头元素的前一个位置、队尾指针指向队尾元素，在判断队空和队满的条件上是相同的，都是通过少放一个元素来区分队空和队满。本题的答案为 A。

35. 若用 $\text{data}[0..n-1]$ 数组来实现环形队列，初始时队头指针 front(指向队头元素的前一个位置)和队尾指针 rear(指向队列中的尾部元素)均为 0，现有 1~6 的 6 个元素进队，然后出队 8 次，发现原来存放元素 4 的位置变为队头，则 n 为_____。

- A. 5
- B. 4
- C. 8
- D. 10

答：初始时， $\text{front} = \text{rear} = 0$ ，进队 1~6 元素，此时元素 4 的位置为 3，出队 8 次后，队头指针 $\text{front} = (\text{front} + 8) \% n = 8 \% n$ ，即 $8 \% n = 3$ ，则 $n = 5$ 。本题的答案为 A。

36. 假设用一个不带头结点的单链表表示队列，队尾应该在链表的_____位置。

- A. 链头
- B. 链尾
- C. 链中
- D. 以上都可以

答：B。

37. 最适合用作链队的链表是_____。

- A. 带队首指针和队尾指针的循环单链表

- B. 带队首指针和队尾指针的非循环单链表
- C. 只带队首指针的非循环单链表
- D. 只带队首指针的循环单链表

答：对于链队，进队操作是在队尾插入结点，出队操作是删除队首结点。对于带队首指针和队尾指针的非循环单链表，这两种操作的时间复杂度均为 $O(1)$ ，所以本题的答案为 B。

38. 最不适合用作链队的链表是_____。

- A. 只带队首指针的非循环双链表
- B. 只带队首指针的循环双链表
- C. 只带队尾指针的循环双链表
- D. 只带队尾指针的循环单链表

答：选项 A 和 B 均可用作链式队列，但不必采用循环单链表，这样反而降低了队列基本运算的效率。本题的答案为 A。

3.3.2 填空题

1. 栈是一种具有_____特性的线性表。

答：后进先出或先进后出。

2. 设栈 S 和队列 Q 的初始状态均为空，元素 a, b, c, d, e, f, g 依次进栈 S 。若每个元素出栈后立即进入队列 Q ，且 7 个元素出列的顺序是 b, d, c, f, e, a, g ，则栈 S 的容量至少是_____。

答：3。由于队列不改变进出序列，这里变为求通过一个栈将 a, b, c, d, e, f, g 序列变为 b, d, c, f, e, a, g 序列时栈空间至少多大？从其进出栈过程可以看到，栈中最多有 3 个元素，即栈大小至少为 3。

3. 一个初始输入序列 $1, 2, \dots, n$ ，出栈序列是 $p_1, p_2, \dots, p_n$ ，若 $p_1=1$ ，则 p_2 的可能取值个数为_____。

答： $n-1$ 。 p_2 不可能取值 1，其他 $2 \sim n$ 的值都可能。

4. 一个初始输入序列 $1, 2, \dots, n$ ，出栈序列是 $p_1, p_2, \dots, p_n$ ，若 $p_1=4$ ，则 p_2 的可能取值个数为_____。

答： $n-3$ 。 p_2 不可能取值 4、2、1，其他值都可能。

5. 栈的常用运算是进栈和出栈，设计栈的一种好的存储结构应尽可能保证进栈和出栈运算的时间复杂度为_____。

答： $O(1)$ 。

6. 当利用大小为 n 的数组 $\text{data}[0..n-1]$ 存储一个顺序栈时，假定用 $\text{top}==n$ 表示栈空，则向这个栈插入一个元素时首先应执行_____语句修改 top 指针。

答： $\text{top}--$ 。

7. 当利用大小为 n 的数组 $\text{data}[0..n-1]$ 存储一个顺序栈时，假定用 $\text{top}== -1$ 表示栈空，则向这个栈插入一个元素时首先应执行_____语句修改 top 指针。

答： $\text{top}++$ 。

8. 若用 $\text{data}[1..m]$ 作为顺序栈的存储空间，栈空的标志是栈顶指针 top 的值等于 $m+1$ ，则每进行一次 ① 操作，需将 top 的值加 1；每进行一次 ② 操作，需将 top 的值减 1。

答：①出栈 ②进栈。这里以 $\text{data}[m]$ 端作为栈底、 $\text{data}[1]$ 端作为栈顶。

9. 当两个栈共享一个存储区时,栈利用一维数组 $\text{data}[1..n]$ 表示,栈 1 在低下标处,栈 2 在高下标处。两栈顶指针为 top1 和 top2 ,初始值分别为 0 和 $n+1$,则当栈 1 空时 top1 为 ①,栈 2 空时 ②,栈满时为 ③。

答: ① $\text{top1}=0$ ② $\text{top2}=n+1$ ③ $\text{top1}+1=\text{top2}$ 。

10. 表达式“ $a+((b*c-d)/e+f*g/h)+i/j$ ”的后缀表达式是_____。

答: $a\ b\ c\ *\ d\ -\ e\ /\ f\ g\ *\ h\ /\ +\ +\ i\ j\ /\ +$ 。

11. 如果栈的最大长度难以估计,则其存储结构最好使用_____。

答: 链栈。

12. 若用带头结点的单链表 st 来表示链栈,则栈空的标志是_____。

答: $st \rightarrow \text{next} == \text{NULL}$ 。

13. 若用不带头结点的单链表 st 来表示链式栈,则创建一个空栈所要执行的操作是_____。

答: $st = \text{NULL}$ 。

14. 在用栈求解迷宫路径时,当找到出口时,栈中所有方块_____。

答: 构成一条迷宫路径。

15. 若用 $Q[1..m]$ 作为非环形顺序队列的存储空间,则最多只能执行_____次进队操作。

答: m 。

16. 若用 $Q[1..100]$ 作为环形队列的存储空间, f 、 r 分别表示队头和队尾指针, f 指向队头元素的前一个位置, r 指向队尾元素,则当 $f=70$, $r=20$ 时,队列中共有_____个元素。

答: 50。这里 $\text{MaxSize}=100$,元素个数 $= (r-f+\text{MaxSize})\% \text{MaxSize} = 50$ 。

17. 环形队列用数组 $A[m..n]$ ($m < n$) 存储元素,其中队头指针 f 指向队头元素的前一个位置、队尾指针 r 指向队尾元素,则该队列中的元素个数是_____。

答: $(r-f+n-m+1)\%(n-m+1)$ 。

18. 用一个大小为 8 的数组来实现环形队列,队头指针 front 指向队头元素的前一个位置,队尾指针 rear 指向队尾元素位置。当前 front 和 rear 的值分别为 0 和 5,现在进队 3 个元素,又出队 3 个元素, front 和 rear 的值分别是_____。

答: 3,0。这样操作后, $\text{front}=(0+3)\%8=3$, $\text{rear}=(5+3)\%8=0$ 。

19. 在实现顺序队的时候,通常将数组看成是一个首尾相连的环,这样做的目的是为了产生_____现象。

答: 假溢出。

20. 已知环形队列的存储空间大小为 m ,队头指针 front 指向队头元素、队尾指针 rear 指向队尾元素,则在队列不满的情况下队中元素个数是_____。

答: $(\text{rear}-\text{front}+1+m)\%m$ 。这样的队列当 $\text{front}=\text{rear}$ 时,队中有一个元素。

21. 假设用一个不带头结点的单链表表示队列,进队结点 p 的操作是_____。

答: 将结点 p 插入到单链表末尾。

22. 假设用一个不带头结点的单链表表示队列,非空队列的出队操作是_____。

答: 删除单链表的首结点。

3.3.3 判断题

1. 判断以下叙述的正确性。

- (1) 栈底元素是不能删除的元素。
- (2) 顺序栈中元素值的大小是有序的。
- (3) 在 n 个元素连续进栈以后, 它们的出栈顺序和进栈顺序一定正好相反。
- (4) 栈顶元素和栈底元素有可能是同一个元素。
- (5) 若用 $\text{data}[1..m]$ 表示顺序栈的存储空间, 则对栈的进栈、出栈操作最多只能进行 m 次。
- (6) 栈是一种对进栈、出栈操作总次数做了限制的线性表。
- (7) 对顺序栈进行进栈、出栈操作不涉及元素的前、后移动问题。
- (8) n 个元素通过一个栈产生 n 个元素的出栈序列, 其中进栈和出栈操作的次数总是相等的。
- (9) 空的顺序栈没有栈顶指针。
- (10) n 个元素进队的顺序和出队的顺序总是一致的。
- (11) 环形队列中有多少个元素可以根据队首指针和队尾指针的值来计算。
- (12) 若采用“队首指针和队尾指针的值相等”作为环形队列为空的标志, 则在设置一个空队时只需将队首指针和队尾指针赋同一个值, 不管什么值都可以。
- (13) 无论是顺序队还是链队, 插入、删除运算的时间复杂度都是 $O(1)$ 。
- (14) 若用不带头结点的非循环单链表来表示链队, 则可以用“队首指针和队尾指针的值相等”作为队空的标志。

答: (1) 错误。栈底元素可以出栈即删除。

- (2) 错误。顺序栈是指用顺序存储结构实现的栈, 栈中的元素不一定是有序的。
- (3) 正确。后进栈的元素先出栈, 先进栈的元素后出栈, 注意这里是连续进栈。
- (4) 正确。当栈中只有一个元素时就是这种情况。
- (5) 错误。可以进行任意多次交替的进栈、出栈操作, 但栈中最多只有 m 个元素。
- (6) 错误。栈是只能在一端进行进栈、出栈操作的线性表。
- (7) 正确。
- (8) 正确。
- (9) 错误。空栈指栈中没有元素, 但顺序栈一定要有栈顶指针。
- (10) 正确。后进队的元素后出队, 先进队的元素先出队。
- (11) 正确。
- (12) 正确。因为无论出队和入队都要进行求余运算, 将队首指针和队尾指针转化为有效的顺序队下标值, 所以是正确的。
- (13) 正确。
- (14) 错误。应该用“队首指针和队尾指针的值均为 NULL”作为队空的标志, 因为当链队中只有一个结点时队首指针和队尾指针的值也相等。

2. 判断以下叙述的正确性。

- (1) 栈和线性表是两种不同的数据结构, 它们的数据元素的逻辑关系也不同。

(2) 有 n 个不同的元素通过一个栈,产生的所有出栈序列恰好构成这 n 个元素的全排列。

(3) 对于 $1, 2, \dots, n$ 的 n 个元素通过一个栈,则以 n 为第一个元素的出栈序列只有一种。

(4) 在顺序栈中,将栈底放在数组的任意位置不会影响运算的时间性能。

(5) 若用 $s[1..m]$ 表示顺序栈的存储空间,以 $s[1]$ 为栈底,变量 top 指向栈顶元素的前一个位置,当栈未满时,将元素 e 进栈的操作是 $top--$; $s[top]=e$ 。

(6) 在采用单链表作为链栈时必须带有头结点。

(7) 环形队列不存在空间上溢出的问题。

(8) 在队空间大小为 n 的环形队列中最多只能进行 n 次进队操作。

(9) 顺序队采用数组存放队中元素,而数组具有随机存取特性,所以在顺序队中可以随机存取元素。

(10) 对于链队,可以根据队头、队尾指针的值计算队中元素的个数。

答: (1) 错误。

(2) 错误。

(3) 正确。

(4) 错误。在顺序栈中,如果将栈底放在数组的两端,其进栈、出栈运算的时间性能都是最好的。如果将栈底放在数组的中间,要么将数组改为循环的(需要保存该栈底位置),要么移动元素,其时间性能都不如将栈底放在数组两端。

(5) 错误。以 $s[1]$ 为栈底,进栈操作时 top 应远离栈底方向移动,所以进栈操作为“ $s[top]=e$; $top++$ ”。

(6) 错误。可以用不带头结点的单链表表示链栈。

(7) 错误。环形队列只是不存在假溢出,它仍然存在上溢出的问题。

(8) 错误。

(9) 错误。顺序队采用数组存放队中元素,尽管数组具有随机存取特性,但队列的操作特性是顺序存取,只能存取两端的元素。

(10) 错误。

3.3.4 简答题

1. 试各举一个实例,简要说明栈和队列在程序设计中所起的作用。

答: 栈的特点是先进后出,所以在解决的实际问题涉及后进先出的情况时可以考虑使用栈。例如求解表达式括号匹配问题时通常使用一个栈,将读到的左括号进栈,每读入一个右括号,判断栈顶是否为左括号,若是,则出栈;否则,表示不匹配。

队列的特点是先进先出。例如求解操作系统中的作业排队问题时通常使用队列,因为在允许多道程序运行的计算机系统中同时有几个作业运行,如果运行的结果都需要通过通道输出,那就要按请求输出的先后次序排队。每当通道传输完毕并可以接受新的输出任务时,队头的作业先从队列中退出做输出操作(出队)。凡是申请输出的作业都从队尾进入队列(进队)。

2. 假设有 4 个元素 a, b, c, d 依次进栈,进栈和出栈操作可以交替进行,试写出所有可

能的出栈序列。

答：当进栈的元素为 n 个时，经过栈运算后可得到的输出序列个数为 $\frac{1}{n+1}C_{2n}^n$ ，其中 $C_{2n}^n = \frac{(2n)!}{n!n!}$ 。这里 $n=4$ 时，出栈序列个数为 $\frac{1}{5} \times \frac{8!}{4! \times 4!} = 14$ 种，如表 3.1 所示。

表 3.1 出栈序列

以 a 开头	abcd abdc acbd acdb adcb
以 b 开头	bacd badc bcad bcda bdca
以 c 开头	cbad cbda cdba
以 d 开头	dcba

3. 假设以 S 和 X 分别表示进栈和出栈操作，则初态和终态均为栈空的进栈和出栈的操作序列，可以表示为仅由 S 和 X 组成的序列，称可以实现的栈操作序列为合法序列（例如 $SSXX$ 为合法序列，而 $SXXS$ 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，并证明对同一输入序列的两个不同的合法序列不可能得到相同的输出序列。

答：合法的栈操作序列必须满足以下两个条件。

(1) 在操作序列的任何前缀（从开始到任何一个操作时刻）中， S 的个数不得少于 X 的个数。

(2) 整个操作序列中 S 和 X 的个数相等。

要求证明：对同一输入序列 $a_1a_2 \cdots a_n$ 的两个不同的合法操作序列 $p = p_1p_2 \cdots p_{j-1}p_j \cdots p_{2n}$ ， $q = q_1q_2 \cdots q_{j-1}q_j \cdots q_{2n}$ ，不可能得到相同的输出序列。

证明：因为 $p \neq q$ ，所以一定存在一个 $j (1 \leq j \leq 2n)$ ，使得 $p_1p_2 \cdots p_{j-1} = q_1q_2 \cdots q_{j-1}$ ，而 $p_j \neq q_j$ ，假设操作子序列 $p_1p_2 \cdots p_{j-1}$ 已将 $a_1a_2 \cdots a_{i-1}$ 进栈且将其中的某些元素出栈，而 $a_ia_{i+1} \cdots a_n$ 尚未进栈。

因为 p 和 q 都是合法的栈操作序列，且 $p_j \neq q_j$ ，所以 p_j 和 q_j 中必有一个为 S 操作，另一个为 X 操作（不失一般性，不妨设 p_j 为 S 操作， q_j 为 X 操作），而且栈不必为空（否则就不能进行 X 操作）。设栈顶元素为 $a_f (1 \leq f \leq i)$ 。因此对于操作序列 p 来说，在其对应的输出序列中 a_i 必领先于 a_f （因为 p_j 为 S 操作，它使 a_i 进栈而 a_f 尚在栈中），对于操作序列 q 来说，在其对应的输出序列中 a_f 必领先于 a_i （因为 q_j 为 X 操作，它使 a_f 出栈而 a_i 尚未进栈），所以 p 和 q 必定对应不同的输出序列。

4. 什么是队列的上溢现象和假溢出现象？解决假溢出有哪些方法？

答：在队列的顺序存储结构中，设队头指针为 front、队尾指针为 rear、队的容量（存储空间的大小）为 MaxSize。当有元素进队时，若 $rear = \text{MaxSize}$ （初始时 $rear = 0$ ），则发生队列的上溢现象，不能做进队操作。所谓队列假溢出现象是指队列中还有剩余空间但元素却不能进入队列，造成这种现象的原因是队列的设计不合理。

解决队列假溢出的方法有以下几种：

(1) 建立一个足够大的存储空间，但这样做会造成空间的使用效率降低。

(2) 当出现假溢出时可采用以下几种方法。

① 采用平移元素的方法：每当进队一个元素时，队列中已有的元素向队头移动一个位

置(当然要有空闲的空间可供移动)。这种方法对应进队运算的时间复杂度为 $O(n)$ 。

② 每当出队一个元素时,依次移动队中的元素,始终使 front 指针指向队列中的第一个元素位置。这种方法对应出队运算的时间复杂度为 $O(n)$ 。

③ 采用环形队列方式:把队列看成一个首尾相接的环形队列,在环形队列上进行进队或出队运算时仍然遵循“先进先出”的原则。这种方法对应进队和出队运算的时间复杂度均为 $O(1)$ 。

5. 在利用两个栈 S_1 、 S_2 模拟一个队列时如何用栈的基本运算实现队列的进队、出队以及队列的判空等基本运算,请简述算法思想。

答:利用两个栈 S_1 和 S_2 模拟一个队列的基本思想是用栈 S_1 作为输入栈、栈 S_2 作为输出栈。进队时,总是将元素进栈到 S_1 ,出队时,若输出栈 S_2 已空,则将 S_1 中的元素全部出栈到 S_2 中,然后由 S_2 出栈元素。若输出栈 S_2 不空,则直接由 S_2 出栈元素。显然,只有当输入栈、输出栈均为空时队列才为空。

6. 设输入元素为 1、2、3、P 和 A,输入次序为 123PA,元素经过一个栈后产生输出序列,在所有输出序列中有哪些序列可作为高级语言的变量名(以字母开头的字母数字串)。

答:AP321,PA321,P3A21,P32A1,P321A。

7. 用栈实现将中缀表达式 $8-(3+5)*(5-6/2)$ 转换成后缀表达式,画出栈的变化过程图。

答:栈的变化过程如表 3.2 所示。最后生成的后缀表达式为 $8\ 3\ 5\ +\ 5\ 6\ 2\ /\ -\ *\ -$,其求值结果为 -8。

表 3.2 将中缀表达式 $8-(3+5)*(5-6/2)$ 转换成后缀表达式时栈的变化过程

op 栈	postexp	说 明
	8 #	将 8 # 存入 postexp 中
-	8 #	'-' 进栈
-(	8 #	'(' 进栈
-(	8 # 3 #	将 3 # 存入 postexp 中
-(+	8 # 3 #	'+' 进栈
-(+	8 # 3 # 5 #	将 5 # 存入 postexp 中
-	8 # 3 # 5 # +	遇到 ')', 将 '+' 和 '(' 出栈
-*	8 # 3 # 5 # +	'*' 进栈
-*(	8 # 3 # 5 # +	'(' 进栈
-*(	8 # 3 # 5 # + 5 #	将 5 # 存入 postexp 中
-*(-	8 # 3 # 5 # + 5 #	'-' 进栈
-*(-	8 # 3 # 5 # + 5 # 6 #	将 6 # 存入 postexp 中
-*(-/	8 # 3 # 5 # + 5 # 6 #	'/' 进栈
-*(-/	8 # 3 # 5 # + 5 # 6 # 2 #	将 2 # 存入 postexp 中
-*	8 # 3 # 5 # + 5 # 6 # 2 # / -	遇到 ')', 将 '/'、'-' 和 '(' 出栈
	8 # 3 # 5 # + 5 # 6 # 2 # / - * -	exp 扫描完毕,将栈中的所有运算符依次出栈并存入数组 postexp 中,得到后缀表达式

8. 简述以下算法的功能:

```
void fun(int a[], int n)
{
    int i = 0, e;
    SqStack * st;
    InitStack(st);
    for (i = 0; i < n; i++)
        Push(st, a[i]);
    i = 0;
    while (!StackEmpty(st))
    {
        Pop(st, e);
        a[i++] = e;
    }
    DestroyStack(st);
}
```

答: 算法的执行步骤如下。

- (1) 扫描整数数组 a , 将所有元素进栈到 st 栈中。
- (2) 将 st 的所有元素退栈, 放到数组 a 中并覆盖原有位置的元素, 从而将数组 a 的所有元素逆置。

(3) 销毁栈 st 。

所以本算法的功能是利用栈 st 将数组 a 中的所有元素逆置。

9. 阅读以下程序, 给出其输出结果:

```
char * fun(int d)
{
    char e; int i = 0, x;
    static char b[MaxSize];
    SqStack * st;
    InitStack(st);
    while (d != 0)
    {
        x = d % 16;
        if (x < 10) e = '0' + x;
        else e = 'A' + x - 10;
        Push(st, e);
        d /= 16;
    }
    while (!StackEmpty(st))
    {
        Pop(st, e);
        b[i++] = e;
    }
    b[i] = '\0';
    DestroyStack(st);
    return b;
}

int main()
{
    int d = 1000, i;
    char * b;
    b = fun(d);
}
```

```
for (i = 0; b[i]; i++)
    printf("%c", b[i]);
printf("\n");
return 1;
}
```

答: fun(*d*)算法的功能是采用辗转相除法将十进制数 *d* 转换成十六进制数,并用数组 *b* 存放十六进制数串。本程序的功能是将十进制数 1000 转换成十六进制数并输出,其结果为 3E8。

10. 算法 fun 的功能是借助栈结构实现整数从十进制到八进制的转换,阅读算法并回答问题:

- (1) 画出 *n* 为十进制数 1348 时算法执行过程中栈的动态变化情况。
- (2) 说明算法中 while 循环完成的操作。

```
void fun(int n)                //n 为非负的十进制整数
{
    int e;
    SqStack * S;
    InitStack(S);
    do
    {
        Push(S, n % 8);
        n = n / 8;
    } while (n);
    while (!StackEmpty(S))
    {
        Pop(S, e);
        printf("%ld", e);
    }
}
```

答: (1) *n* 为十进制数 1348 时算法执行过程中栈的动态变化情况如图 3.4 所示,产生对应的八进制数为 2504。

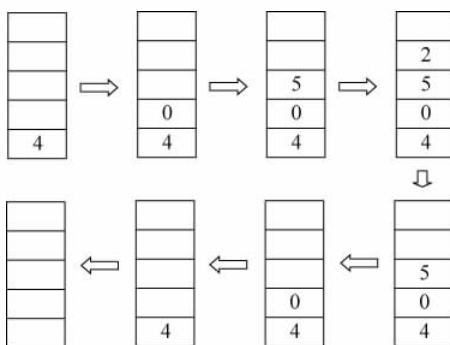


图 3.4 栈的动态变化情况

(2) 算法中 while 循环的操作是退栈所有元素并输出,即从高位到低位输出八进制数。

11. 简述以下算法的功能(栈的元素类型为 int)。

```
void fun(SqStack * &st)
{
    int i, j = 0, A[MaxSize];
```

```

while (!StackEmpty(st))
{
    Pop(S, A[j]);
    j++;
}
for(i = 0; i < j; i++)
    Push(S, A[i]);
}

```

答：算法的执行步骤如下。

- (1) 将栈 st 中的所有元素退栈并存放于数组 A 中。
 - (2) 将 A 中的元素一一进栈, 达到逆置栈 st 的目的。
- 所以本算法的功能是逆置栈 st 的所有元素。

3.3.5 算法设计题

1. 【顺序栈算法】设计一个算法将一个十进制正整数 d 转换为相应的二进制数。

解：将十进制正整数转换成二进制数通常采用除 2 取余数法。在转换过程中, 二进制数是按照从低位到高位次序得到的, 这和通常的从高位到低位输出二进制的次序相反。为此设计一个栈 st , 用于暂时存放每次得到的余数, 当转换过程结束时, 退栈所有元素便得到从高位到低位的二进制数。图 3.5 所示为十进制数 12 转换为二进制数 1100 的过程。

栈底 $\Rightarrow$ 栈顶			
$12 \% 2 = 0$, 0 进栈, $12 / 2 = 6$	0		
$6 \% 2 = 0$, 0 进栈, $6 / 2 = 3$	0, 0		
$3 \% 2 = 1$, 1 进栈, $3 / 2 = 1$	0, 0, 1		
$1 \% 2 = 1$, 1 进栈, $1 / 2 = 0$	0, 0, 1, 1	退栈并输出	转换结果
			1100

图 3.5 整数 12 转换为二进制数的过程

对应的算法如下：

```

#include "SqStack.cpp"           //包含顺序栈的定义及运算函数
void trans(int d, char b[])      //b 用于存放 d 转换成的二进制数串
{
    char e;
    SqStack * st;
    InitStack(st);
    int i = 0;
    while (d != 0)
    {
        e = '0' + d % 2;         //求余数并转换为字符
        Push(st, e);
        d /= 2;                  //继续求更高位
    }
    while (!StackEmpty(st))
    {
        Pop(st, e);              //出栈元素 e
        b[i] = e;                //将 e 存放在数组 b 中
        i++;
    }
}

```

```

        b[i] = '\0';                //加入字符串结束标志
        DestroyStack(st);           //销毁栈
    }

```

2. 【顺序栈算法】设计一个算法,利用顺序栈的基本运算从栈顶到栈底输出栈中的所有元素,要求仍保持栈中元素不变。

解: 先建立并初始化一个临时栈 tmpst。退栈 st 中的所有元素,输出这些元素并进栈到 tmpst 中,然后将临时栈 tmpst 中的元素逐一出栈并进栈到 st 中,这样恢复 st 栈中原来的元素。注意本题要求只能使用栈的基本运算来完成,不能直接用 `st->data[i]` 输出栈中的元素。对应的算法如下:

```

#include "SqStack.cpp"                //包含顺序栈的定义及运算函数
void DispStack(SqStack * st)
{
    ElemType x;
    SqStack * tmpst;                 //定义临时栈
    InitStack(tmpst);                //初始化临时栈
    while (!StackEmpty(st))           //临时栈 tmpst 中包含 st 栈中的逆转元素
    {
        Pop(st, x);
        printf("%d ", x);
        Push(tmpst, x);
    }
    printf("\n");
    while (!StackEmpty(tmpst))         //恢复 st 栈中原来的内容
    {
        Pop(tmpst, x);
        Push(st, x);
    }
    DestroyStack(tmpst);
}

```

3. 【顺序栈算法】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不变。

解: 先建立并初始化一个临时栈 tmpst。退栈 st 中的所有元素 x ,并用 i 累计元素个数,当 $i=k$ 时置 $e=x$,并将所有元素进栈到 tmpst 中,然后将临时栈 tmpst 中的元素逐一出栈并进栈到 st 中,这样恢复 st 栈中原来的元素。如果栈中没有第 k 个元素,返回假;否则返回真,并通过引用型参数 e 保存第 k 个元素。注意本题要求只能使用栈的基本运算来完成,不能直接用 `st->data[i]` 求第 k 个栈中元素。对应的算法如下:

```

#include "SqStack.cpp"                //包含顺序栈的定义及运算函数
bool Findk(SqStack * st, int k, ElemType &e)
{
    int i = 0;
    bool flag = false;
    ElemType x;
    SqStack * tmpst;                 //定义临时栈
    InitStack(tmpst);                //初始化临时栈
    while (!StackEmpty(st))           //临时栈 tmpst 中包含 st 栈中的逆转元素
    {
        i++;
        Pop(st, x);
    }

```

```

        if (i == k)
        {
            e = x;
            flag = true;
        }
        Push(tmpst, x);
    }
    while (!StackEmpty(tmpst))    //恢复 st 栈中原来的内容
    {
        Pop(tmpst, x);
        Push(st, x);
    }
    DestroyStack(tmpst);
    return flag;
}

```

4. 【顺序栈算法】有 abcde 共 $n(n=5)$ 个字符,通过一个栈可以产生多种出栈序列,设计一个算法判断序列 str 是否为一个合适的出栈序列,并给出操作过程,要求用相关数据进行测试。

解: 先建立一个字符顺序栈 st,将进栈序列 abcde 存放到字符数组 A 中。用 i 、 j 分别扫描数组 A 和 str,它们的初始值均为 0。当数组 A 和 str 都没有扫描完时循环:比较栈顶元素 e 和 $str[j]$,若两者不相同,则将 $A[i]$ 进栈, i 加 1;若两者相同,则出栈栈顶元素 e, j 加 1。上述循环结束后退栈所有元素。如果序列 str 是一个合适的出栈序列,必有 $j == n$,否则 str 不是一个合适的出栈序列。对应的算法如下:

```

#include "SqStack.cpp"    //包含顺序栈的定义及运算函数
bool isSerial(char str[], int n)
{
    int i, j;
    char A[MaxSize], e;
    SqStack *st;          //建立一个顺序栈
    InitStack(st);
    for (i = 0; i < n; i++)
        A[i] = 'a' + i;    //将 abcde 放入数组 A 中
    i = 0; j = 0;
    while (i < n && j < n)
    {
        if (StackEmpty(st) || (GetTop(st, e) && e != str[j]))
        {
            Push(st, A[i]);
            printf("  元素 %c 进栈\n", A[i]);
            i++;
        }
        else
        {
            Pop(st, e);
            printf("  元素 %c 出栈\n", e);
            j++;
        }
    }
    while (!StackEmpty(st) && GetTop(st, e) && e == str[j])
    {
        Pop(st, e);
        printf("  元素 %c 出栈\n", e);
    }
}

```

```
        j++;
    }
    DestroyStack(st);
    if (j == n) return true;        //是出栈序列时返回 true
    else return false;            //不是出栈序列时返回 false
}
void Disp(char str[], int n)      //输出 str
{
    int i;
    for (i = 0; i < n; i++)
        printf("%c ", str[i]);
}
int main()
{
    int n = 5;
    char str[] = "acbed";
    Disp(str, n); printf("的操作序列:\n");
    if (isSerial(str, n))
    {
        Disp(str, n);
        printf("是合适的出栈序列\n");
    }
    else
    {
        Disp(str, n);
        printf("不是合适的出栈序列\n");
    }
    return 1;
}
```

本程序的执行结果如下：

```
acbed 的操作序列：
元素 a 进栈
元素 a 出栈
元素 b 进栈
元素 c 进栈
元素 c 出栈
元素 b 出栈
元素 d 进栈
元素 e 进栈
元素 e 出栈
元素 d 出栈
acbed 是合适的出栈序列
```

5. 【共享栈算法】用一个一维数组 S (设大小为 $MaxSize$) 作为两个栈的共享空间, 说明共享方法, 以及栈满、栈空的判断条件, 并用 C/C++ 语言设计公用的初始化栈运算 $InitStack1(st)$ 、判栈空运算 $StackEmpty1(st, i)$ 、进栈运算 $Push1(st, i, x)$ 和出栈运算 $Pop1(st, i, x)$, 其中 i 为 1 或 2, 用于表示栈号, x 为进栈或出栈元素。

解：设用一维数组 $S[MaxSize]$ 作为两个栈 $S1$ 和 $S2$ 的共享空间, 整型变量 $top1$ 、 $top2$ 分别作为两个栈的栈顶指针, 并约定栈顶指针指向当前元素的下一个位置。 $S1$ 的栈底位置设在 $S[0]$, $S2$ 的栈底位置设在 $S[MaxSize-1]$, 如图 3.6 所示。

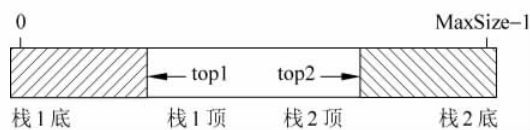


图 3.6 共享栈示意图

栈 S1 空的条件是 $top1 == -1$ ，栈 S1 满的条件是 $top1 == top2 - 1$ ；栈 S2 空的条件是 $top2 == MaxSize$ ，栈 S2 满的条件是 $top2 == top1 + 1$ 。归纳起来，栈 S1 和 S2 满的条件都是 $top1 == top2 - 1$ 。

元素 x 进栈 S1 的算法是 $Push1(\&st, 1, x)$ ，当不满时，执行 $st.top1++$ ， $st.S[st.top1] = x$ ；元素 x 进栈 S2 的算法是 $Push1(\&st, 2, x)$ ，当不满时，执行 $st.top2--$ ， $st.S[st.top2] = x$ 。

元素 x 退栈 S1 的算法是 $Pop1(\&st, 1, \&x)$ ，当不空时，执行 $x = st.S[st.top1]$ ， $st.top1--$ ；元素 x 退栈 S2 的算法是 $Pop1(\&st, 2, \&x)$ ，当不空时，执行 $x = st.S[st.top2]$ ， $st.top2++$ 。

共享栈的类型定义和相关运算算法如下：

```
#include <stdio.h>
#define MaxSize 100
typedef char ElemType;
typedef struct
{
    ElemType S[MaxSize];           //存放共享栈中的元素
    int top1, top2;                 //两个栈顶指针
} StackType;                       //声明共享栈类型
// ----- 栈初始化算法 -----
void InitStack1(StackType &st)
{
    st.top1 = -1;
    st.top2 = MaxSize;
}
// ----- 判栈空算法: i=1:栈 1, i=2:栈 2 -----
bool StackEmpty1(StackType st, int i)
{
    if (i == 1)
        return(st.top1 == -1);
    else //i = 2
        return(st.top2 == MaxSize);
}
// ----- 进栈算法: //i=1:栈 1, i=2:栈 2 -----
bool Push1(StackType &st, int i, ElemType x)
{
    if (st.top1 == st.top2 - 1) //栈满
        return false;
    if (i == 1) //x 进栈 S1
    {
        st.top1++;
        st.S[st.top1] = x;
    }
    else if (i == 2) //x 进栈 S2
    {
        st.top2--;
```

```

        st.S[st.top2] = x;
    }
    else //参数 i 错误返回 false
        return false;
    return true; //操作成功返回 true
}
// ----- 出栈算法: i = 1: 栈 1, i = 2: 栈 2 -----
bool Pop1(StackType &st, int i, ElemType &x)
{
    if (i == 1) //S1 出栈
    {
        if (st.top1 == -1) //S1 栈空
            return false;
        else //出栈 S1 的元素
        {
            x = st.S[st.top1];
            st.top1 -- ;
        }
    }
    else if (i == 2) //S2 出栈
    {
        if (st.top2 == MaxSize) //S2 栈空
            return false;
        else //出栈 S2 的元素
        {
            x = st.S[st.top2];
            st.top2++;
        }
    }
    else //参数 i 错误返回 false
        return false;
    return true; //操作成功返回 true
}

```

6. 【环形队列算法】设计一个算法,利用环形队列的基本运算返回指定队列中的队尾元素,要求算法的空间复杂度为 $O(1)$ 。

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 。循环 m 次,出队一个元素 x ,再将元素 x 进队,最后的 x 即为队尾元素。对应的算法如下:

```

#include "SqQueue.cpp" //包含顺序队的类型定义和运算函数
ElemType Last(SqQueue * qu)
{
    ElemType x;
    int i, m = (qu->rear - qu->front + MaxSize) % MaxSize;
    for (i = 1; i <= m; i++)
    {
        deQueue(qu, x); //出队元素 x
        enQueue(qu, x); //将元素 x 进队
    }
    return x;
}

```

7. 【环形队列算法】对于环形队列,利用队列的基本运算设计删除队列中从队头开始的第 k 个元素的算法。

解：先求出队列 qu 中的元素个数 count，若 k 小于 0 或大于 count，返回假。出队所有元素，并记录元素的序号 i ，当 $i=k$ 时对应的元素只出不进，否则将出队的元素又进队。对应的算法如下：

```
#include "SqQueue.cpp"           //包含顺序队的类型定义和运算函数
bool Delk(SqQueue * &qu, int k)
{   ElemType e;
    int i, count = (qu->rear - qu->front + MaxSize) % MaxSize;
    if (k <= 0 || k > count)
        return false;
    for (i = 1; i <= count; i++)
    {   deQueue(qu, e);           //出队元素 e
        if (i != k)              //第 k 个元素只出不进
            enQueue(qu, e);      //其他元素出队后又进队
    }
    return true;
}
```

说明：在设计本题算法时不能通过移动元素的方式直接对数组 data 删除第 k 个元素，这样是把顺序队看成一个顺序表，没有作为一个队列看待。

8. 【环形队列算法】对于环形队列来说，如果知道队尾元素的位置和队列中元素的个数，则队头元素所在的位置显然是可以计算的。也就是说，可以用队列中元素的个数代替队头指针。编写出这种环形顺序队列的初始化、进队、出队和判空算法。

解：当已知队头元素的位置 rear 和队列中元素的个数 count 后，队空的条件为 $\text{count} == 0$ ；队满的条件为 $\text{count} == \text{MaxSize}$ ；计算队头位置为 $\text{front} = (\text{rear} - \text{count} + \text{MaxSize}) \% \text{MaxSize}$ 。对应的算法如下：

```
typedef struct
{   ElemType data[MaxSize];
    int rear;           //队尾指针
    int count;          //队列中元素的个数
} QuType;              //队列类型
void InitQu(QuType * &q) //队列的初始化运算
{   q = (QuType *) malloc(sizeof(QuType));
    q->rear = 0;
    q->count = 0;
}
bool EnQu(QuType * &q, ElemType x) //进队运算
{   if (q->count == MaxSize)      //队满上溢出
        return false;
    else
    {   q->rear = (q->rear + 1) % MaxSize;
        q->data[q->rear] = x;
        q->count++;
        return true;
    }
}
bool DeQu(QuType * &q, ElemType &x) //出队运算
```

```

{   int front;                                //局部变量
    if (q->count == 0)                        //队空下溢出
        return false;
    else
    {   front = (q->rear - q->count + MaxSize) % MaxSize;
        front = (front + 1) % MaxSize;        //队头位置进 1
        x = q->data[front];
        q->count--;
        return true;
    }
}

bool QuEmpty(QuType *q)                      //判空运算
{
    return(q->count == 0);
}

```

9. 【环形队列算法】设计一个环形队列,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(0)或可能满(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。

解:设计的队列类型如下:

```

typedef struct
{   ElemType data[MaxSize];
    int front,rear;           //队头和队尾指针
    int tag;                  //为 0 表示队可能空,为 1 时表示队可能满
} QueueType;

```

初始时 tag=0,front=rear=0,成功的进队操作后 tag=1(任何进队操作后队列可能满,但不一定满,任何进队操作后队列不可能空),成功的出队操作后 tag=0(任何出队操作后队列可能空,但不一定空,任何出队操作后队列不可能满),因此这样的队列的 4 要素如下。

- ① 队空条件: qu.front==qu.rear && qu.tag==0;
- ② 队满条件: qu.front==qu.rear && qu.tag==1;
- ③ 元素 x 进队: qu.rear=(qu.rear+1)%MaxSize; qu.data[qu.rear]= x ; qu.tag=1;
- ④ 元素 x 出队: qu.front=(qu.front+1)%MaxSize; x =qu.data[qu.front]; qu.tag=0。

对应的算法如下:

```

void InitQueue1(QueueType &qu)                //初始化队列算法
{   qu.front = qu.rear = 0;
    qu.tag = 0;                                //为 0 表示队空可能为空
}

bool QueueEmpty1(QueueType qu)                //判队空算法
{
    return(qu.front == qu.rear && qu.tag == 0);
}

bool QueueFull1(QueueType qu)                //判队满算法

```

```

{
    return(qu.tag == 1 && qu.front == qu.rear);
}

bool EnQueue1(QueueType &qu, ElemType x)           //进队算法
{
    if (QueueFull1(qu) == 1)                       //队满
        return false;
    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 true;
}

bool DeQueue1(QueueType &qu, ElemType &x)          //出队算法
{
    if (QueueEmpty1(qu) == 1)                      //队空
        return false;
    qu.front = (qu.front + 1) % MaxSize;
    x = qu.data[qu.front];
    qu.tag = 0;                                     //出队一个元素,可能空
    return true;
}

```

10. 【双端队列应用】假设有一个整型数组存放 n 个学生的分数,将分数分为 3 个等级,分数高于或等于 90 的为 A 等,分数低于 60 的为 C 等,其他为 B 等。要求采用双端队列,先输出 A 等分数,再输出 C 等分数,最后输出 B 等分数。

解: 设计双端队列的从队头出队算法 deQueue1、从队头进队算法 enQueue1 和从队尾进队算法 enQueue2。对于含有 n 个分数的数组 a ,扫描所有元素 $a[i]$,若 $a[i]$ 为 A 等,直接输出;若为 B 等,将其从队尾进队;若为 C 等,将其从队头进队。最后从队头出队并输出所有的元素。对应的算法如下:

```

#include "SqQueue.cpp"                               //包含顺序队的类型定义和运算函数

bool deQueue1(SqQueue *q, ElemType &e)              //从队头出队算法
{
    if (q->front == q->rear)                          //队空下溢出
        return false;
    q->front = (q->front + 1) % MaxSize;               //修改队头指针
    e = q->data[q->front];
    return true;
}

bool enQueue1(SqQueue *q, ElemType e)               //从队头进队算法
{
    if ((q->rear + 1) % MaxSize == q->front)           //队满
        return false;
    q->data[q->front] = e;                             //e 元素进队
    q->front = (q->front - 1 + MaxSize) % MaxSize;    //修改队头指针
    return true;
}

bool enQueue2(SqQueue *q, ElemType e)              //从队尾进队算法
{
    if ((q->rear + 1) % MaxSize == q->front)           //队满上溢出
        return false;
    q->rear = (q->rear + 1) % MaxSize;                 //修改队尾指针
    q->data[q->rear] = e;                             //e 元素进队
}

```

```

    return true;
}

void fun(int a[],int n)
{
    int i;
    ElemType e;
    SqQueue *qu;
    InitQueue(qu);
    for (i = 0; i < n; i++)
    {
        if (a[i] >= 90)
            printf(" %d ", a[i]);
        else if (a[i] >= 60)
            enqueue2(qu, a[i]);           //从队尾进队
        else
            enqueue1(qu, a[i]);           //从队头进队
    }
    while (!QueueEmpty(qu))
    {
        dequeue1(qu, e);                  //从队头出队
        printf(" %d ", e);
    }
    printf("\n");
    DestroyQueue(qu);
}

```

11. 【顺序栈和顺序队算法】用于列车编组的铁路转轨网络是一种栈结构,如图 3.7 所示,其中右边轨道是输入端、左边轨道是输出端。当右边轨道上的车皮编号顺序为 1、2、3、4 时,如果执行操作进栈、进栈、出栈、进栈、进栈、出栈、出栈、出栈,则在左边轨道上的车皮编号顺序为 2、4、3、1。

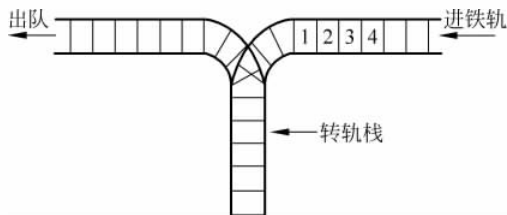


图 3.7 铁路转轨网络

设计一个算法,输入 n 个整数,表示右边轨道上 n 节车皮的编号,用上述转轨栈对这些车皮重新编排,使得编号为奇数的车皮都排在编号为偶数的车皮的前面。

解: 将转轨栈看成一个栈,将左边轨道看成是一个队列。从键盘逐个输入表示右边轨道上车皮编号的整数,根据其奇偶性做以下处理:若是奇数,则将其插入到表示左边轨道的顺序队列的队尾;若是偶数,则将其插入到表示转轨栈的顺序栈的栈顶。当 n 个整数都检测完之后,这些整数已全部进入队列或栈中。此时,首先按先进先出的顺序输出队列中的元素,然后再按后进先出的顺序输出栈中的元素。

算法中直接使用两个数组 st 和 qu 分别存放栈和队列中的元素。对应的算法如下:

```

#include <stdio.h>
#define MaxSize 100
void fun1()
{
    int i,n,x;
    int st[MaxSize],top = -1;           //顺序栈和栈顶指针
    int qu[MaxSize],front = 0, rear = 0; //队列和队指针
    printf("n:");
    scanf("%d",&n);
    for (i = 0; i < n; i++)
    {
        printf("第 %d 个车皮编号:", i + 1);
        scanf("%d",&x);
        if (x % 2 == 1)                 //编号为奇数,则进队列
        {
            qu[rear] = x;
            rear++;
            printf("  %d 进队\n",x);
        }
        else                             //编号为偶数,则进栈
        {
            top++;
            st[top] = x;
            printf("  %d 进栈\n",x);
        }
    }
    printf("出栈操作:\n  ");
    while (front != rear)                //队列中的所有元素出队
    {
        printf(" %d 出队 ",qu[front]);
        front++;
    }
    while (top >= 0)                     //栈中的所有元素出栈
    {
        printf(" %d 出栈 ",st[top]);
        top--;
    }
    printf("\n");
}
int main()
{
    fun1();
    return 1;
}

```

本程序的一次求解结果如下：

```

n:4 ✓
第 1 个车皮编号:4 ✓ 4 进栈
第 2 个车皮编号:1 ✓ 1 进队
第 3 个车皮编号:3 ✓ 3 进队
第 4 个车皮编号:2 ✓ 2 进栈
出栈操作:
    1 出队  3 出队  2 出栈  4 出栈

```