CHAPTER 3

栈与队列 第3章

3.1 学习要点

线性结构是最常用的一种数据结构，线性结构中还有一些限定操作的线性表被大量运用在算法的实现中，栈与队列就是两种重要的限定操作的线性表。栈和队列广泛应用在各种类型的软件系统中，如编译软件、操作系统等，学习它们的结构特征和各种操作的实现，有着重要的意义。

3.1.1 栈

栈是一种限定操作的线性表，限定只能在线性表的一端进行插入和删除操作。如图 3-1 所示，栈像一个一端封闭的井，不管插入元素或删除元素都只能从有开口的那边来进行，称为栈顶，封闭的一边称为栈底。栈中最迟插入的元素称为栈顶元素，插入元素只能插入在栈顶元素之后，成为新的栈顶元素。删除元素则是指删除栈顶元素，次顶元素成为新的栈顶元素。栈的插入删除被形象地称为进栈和出栈。

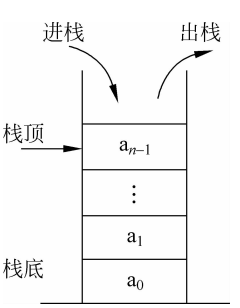

图 3-1 栈示意图

通过定义要注意以下几点。

(1) 栈还是线性表，不过在操作中不能随心所欲，只能做限定的某些操作。

(2) 栈的操作原则是后进先出(LIFO)。

(3) 进栈和出栈只能从栈的一端进行，这端称为栈顶。另外不能操作的一端叫栈底。栈的中间也不能进、出元素。

(4) 同一个栈中的元素类型是相同的。

1. 栈的顺序存储结构

栈的顺序存储结构规定将变量 top 指向栈顶元素所在位置，这样方便操

作,对 top 的上下移动就体现了进栈和出栈的动作。注意栈的顺序存储结构在进栈出栈的时候要考虑栈满和栈空的情况。

栈顺序存储结构的 C 语言描述如下:

```
00   #define MAXSIZE 100          /* MAXSIZE 指的是栈的最大长度 */
01   typedef struct
02    {
03     DataType elem[MAXSIZE];    /* 定义数组依次存放栈里的数据元素 */
04     int top;                   /* 指向栈顶元素的下标 */
05    }Stack;
```

假设定义了顺序栈 Stack S,S.elem 为存储数据元素的数组,可以通过 S.elem[top]来读取栈中的栈顶元素。S.top+1 是栈的元素个数。顺序栈的基本操作实现如下。

(1) 初始化:
```
Stack S;
S.top = -1;
```

(2) x 进栈:
```
S.top = S.top + 1;
S.elem[S.top] = x;
```

(3) 出栈:
```
S.top = S.top - 1;
```

(4) 取栈顶:
```
x = S.elem[S.top];
```

(5) 栈空条件:
```
S.top == -1;
```

(6) 栈满条件:
```
S.top == max - 1;
```

顺序栈是有容量的,如栈元素个数较多且无法正确估计栈的容量时,容易存在因栈满而无法插入的上溢现象,这在栈的应用中往往是一个致命错误。可以采用多栈共享空间的方法来解决这个问题,这就是多栈共享技术。最常用的是两个栈共享一个足够大的数组的方法,称为双端栈,如图 3-2 所示。双端栈的原理是:利用栈只能在栈顶端进行操作的特性,将两个栈的栈底分别设在数组的头和尾,两个栈的栈顶在数组中动态变化,栈 1 元素比较多时就占用比较多的存储单元,元素少时就让出存储单元供栈 2 可用,提高了数组的利用率。

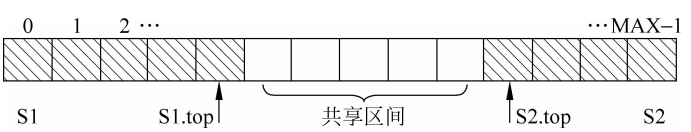

图 3-2 双端栈

在 S1 中做入栈操作时,S1.top 要加 1,在 S2 中做入栈操作时,S2.top 要减 1。随着 S1.top 和 S2.top 的动态变化,数组中间一段形成了共享区间。不过要注意的是,当 S1 中元素较多,S2 中元素也较多,两者之和超出数组容量时,还是有可能出现上溢的。多栈共享技术只是减少了上溢的发生,是解决上溢的一种折中手段,但不能完全解决问题。

2. 栈的链式存储结构

栈的链式存储结构,即用链表存储栈中的数据元素,称为链栈。一般采用带头结点的单链表存储栈。栈的操作仅限在栈顶进行,可将头结点指针域指向栈顶结点,如用 top 指针指向头结点,称为 top 结点。如图 3-3 所示栈的逻辑图相对应的链式存储如图 3-4 所示。

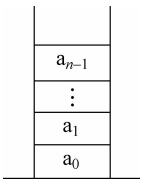

图 3-3　栈逻辑图

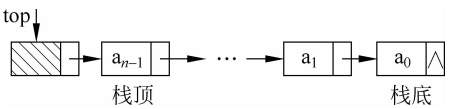

图 3-4　链栈

链栈中结点结构用 C 语言可定义如下：

```
00   struct node
01   {
02     DataType data;              /* 链栈结点数据域类型及名称 */
03     struct node *next;          /* 指针域类型及名称,指向次顶结点 */
04   };
05   typedef struct node StackNode;
06   StackNode *top;
```

栈的 top 结点，类似单链表中的 head 结点，指向栈顶结点，进栈出栈都在表头进行。这里结点的指针域指向的是次顶结点。注意栈的链式存储结构在进栈时不需要考虑栈满的情况，只要还有可用空间，就可以根据需要申请空间来存放元素，故不需要事先估计栈的最大容量，避免了冗余。链栈的基本操作实现如下。

（1）初始化：`top = (StackNode *) malloc(sizeof(StackNode));`
`top->next = NULL;`

（2）p 结点进栈：`p->next = top->next;`
`top->next = p;`

（3）出栈：`top->next = top->next->next;`

（4）取栈顶：`x = top->next->data;`

（5）栈空条件：`top->next == NULL;`

栈也是线性表，并且操作比线性表更有限，所以实际上是更容易掌握了。

3. 栈的应用场合

在分析实际问题时，如操作的对象符合“后出现的反而先处理”的情况，就可考虑用栈来进行辅助设计。

3.1.2　队列

队列限定只能在线性表的一端进行插入，另一端进行删除操作。允许插入的一端叫队尾，允许删除的一端叫队头，这和日常生活中的排队非常类似，可以作为类比，排队时如果加入到队伍中总是加入到队列的尾部。如图 3-5 所示，队列的插入删除都只能在指定位置进行，并称在队头删除元素为出队操作，在队尾插入元素为进队操作。

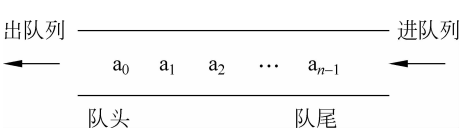

图 3-5　队列示意图

通过定义要注意以下几点。

(1) 队列还是线性表,不过在操作中不能随心所欲,只能做限定的某些动作。

(2) 队列的操作原则是先进先出(FIFO)。

(3) 入队操作只能从称为队尾的一端进行;出队操作只能从队列的另外一端,即称为队头的一端进行;队列的中间不能进、出元素。

(4) 同一个队列中的元素类型是相同的。

1. 队列的顺序存储结构

队列的顺序存储结构,即用数组依次存储队列中的数据元素,称为顺序队列。按照习惯,进队列时从低地址到高地址的次序存放,所以,队头元素是数组中的第一个元素,队尾元素是数组中的最后一个元素。由于队列的基本操作是进队列和出队列,常常要扫描数组中队头元素和队尾元素的位置,故需要设置两个变量,一个变量 front 来指示队头元素的位置,另外一个变量 rear 来指示队尾元素的下一个位置。注意队列的顺序存储结构在入队出队的时候要考虑队满和队空的情况。

front 和 rear 随着队列中元素的进出也动态地变化着。出队列时,front 要后移一位,指向新的队头;进队列时,元素放入 rear 所示位置后,rear 也要后移一位,保证指向新队尾的下一个空位。

顺序队列经过了一系列的进队出队操作后,front 之前的位置都是空的,但 rear 已经指示到了 n(数组容量),这时就会出现"假溢出"的问题。为了解决假溢出的现象,可以采用循环队列,也就是将顺序队列看成是一个首尾相连的数组,下标 $n-1$ 的位置的下一个位置是 0,队满标志不再是 rear=n 了。如图 3-6(a)和图 3-6(b)显示了一个长度为 8 的循环队列。循环队列中 front 和 rear 的表示意义不变。

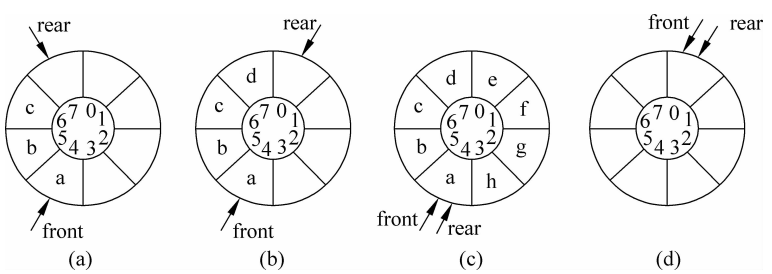

图 3-6　长度为 8 的循环队列

如何实现队列中的 rear 值在 0～$n-1$ 之间循环呢? 即 0 的下一个 rear 值是 1,1 的下一个 rear 值是 2,…,$n-1$ 的下一个 rear 值是 0。一个巧妙的方法是利用数学中的求模运算,下一个 rear=(rear+1)mod n,这里的 n 是数组的长度。这样,借助了求模运算,就能实现 front、rear 的循环变化。所以,出队时 front 的变化是:front=(front+1)mod n;进队时 rear 的变化是:rear=(rear+1)mod n。

在如图 3-6(b)所示循环队列的情况下,再入 4 个元素 e、f、g、h 后,如图 3-6(c)所示,这时似乎可以得出循环队列队满的标志是 front=rear。继续往下看,在如图 3-6(b)所示循环队列的情况下,4 个元素 a、b、c、d 依次出队列后,如图 3-6(d)所示,这时已经队空,此时 front=rear,可见,在循环队列中,会产生队空和队满标志冲突的问题。为了解决这个问题,

可以以牺牲一个存储单元为代价，当判断到当前 rear 值的下一个位置是 front 时，就认为队满，停止进队。这样，rear 永远在 front 之后，不会因不断进队而从后面追上 front。所以，循环队列队满的标志是(rear+1)mod n=front；队空的标志是 rear=front，冲突解决！这是以牺牲一个存储单元为代价的解决方案，长度为 n 的循环队列最多只能存放 $n-1$ 个元素。

基于 C 语言的顺序队列的类型定义如下：

```
00   #define MAX 100                /* MAX 指的是队列的最大长度 */
01   typedef struct
02   {
03     DataType elem[MAX];          /* 定义数组依次存放队列里的数据元素 */
04     int front;                   /* 指向队头元素的下标 */
05     int rear;                    /* 指向队尾元素的下一个空位 */
06   }Queue;
```

假设定义了一个循环队列 Queue Q，可以通过 Q.elem[front]来读取队列中的队头元素。循环队列的基本操作实现如下。

(1) 初始化：`Queue Q;`
`Q.front = Q.rear = 0;`

(2) x 进队列：`Q.elem[Q.rear] = x;`
`Q.rear = (Q.rear + 1) % max;`

(3) 出队：`Q.front = (Q.front + 1) % max;`

(4) 取队头：`x = Q.elem[Q.front];`

(5) 取队尾：`x = (Q.rear + max - 1) % max;`

(6) 队满条件：`Q.front == (Q.rear + 1) % max;`

(7) 队空条件：`Q.front == Q.rear;`

循环队列只是将存放队列的数组想象成首尾相连，以解决假溢出的问题，实际在内存空间中并无区别。

2. 队列的链式存储结构

队列的链式存储结构，即用链表存储队列中的数据元素，称为链队列。一般采用带头结点的单链表存储队列。队列的基本操作包括进队和出队，常要对队头和队尾进行操作，需要设两个指针变量，一个指向头结点，头结点指针域指向队头结点，如用 front 指针指向头结点，称为 front 结点；另一个指向队尾结点，可设为 rear，称为 rear 指针。如图 3-7 所示为队列的链式存储结构示意图。注意队列的链式存储结构在入队时不需要考虑队满的情况。

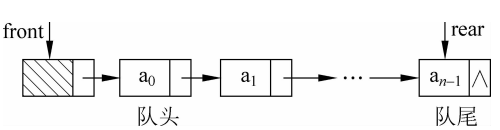

图 3-7　链队列

链队列中结点结构和单链表一样，指针域指向队列中的下一个元素。用 C 语言可定义如下：

```
00   struct node
01   {
02     DataType data;               /* 链队列结点数据域类型及名称 */
```

```
03     struct node * next;          /* 指针域类型及名称,指向下一结点 */
04   };
05   typedef struct node QueueNode;
06   struct node2
07   {
08     QueueNode * front;
09     QueueNode * rear;
10   } ;
11   typedef struct node2 Queue;
```

假设定义了链队类型 Queue,QueueNode 代表一个队列元素结点,可通过 Queue Q 来定义一个队列变量。链队列没有队满的情况,只要还有可用空间,就可以根据需要申请空间来存放元素,故不需要事先估计队列的最大容量,避免了冗余。链队列的基本操作实现如下。

(1) 初始化:
```
Queue Q;
Q.front = (QueueNode * ) malloc(sizeof(QueueNode));
Q.front -> next = NULL;
Q.rear = Q.front;
```

(2) p 进队列:
```
Q.rear -> next = p;
Q.rear = p;
```

(3) 出队:
```
q = Q.front -> next;
Q.front -> next = q -> next;
if(q == Q.rear)          /* 注意特殊情况的处理: 如出队的元素正好是队尾元素 */
    Q.rear = Q.front;
```

(4) 取队头:
```
x = front -> link -> data;
```

(5) 取队尾:
```
x = rear -> data;
```

(6) 队空条件:
```
front -> link == rear;
```

3. 队列的应用场合

在分析实际问题时,如操作的对象符合“先出现的先处理”的情况,就可考虑用队列来进行辅助设计。

3.2 重点难点分析

3.2.1 学习要求与线索

栈和队列属于线性结构,符合线性结构要求的几个特征。将栈和队列独立成章来学习,是因为栈和队列在计算机科学与技术的许多领域呈现出的重要性。栈是限定只能按后进先出(LIFO)原则进行操作的线性表,在编译系统、操作系统中断处理、递归调用等程序设计中,大量使用栈存储程序信息。队列是限定只能按先进先出(FIFO)原则进行操作的线性表,在叫号系统、操作系统资源分配等软件开发中,大量使用队列存储程序信息。在本章的学习中,有以下几点学习要求。

(1) 掌握栈和队列的结构特点及操作原则。

(2) 掌握栈的两种存储结构的各种基本操作及算法。

(3) 熟悉栈的应用。

(4) 掌握队列的两种存储结构的各种基本操作及算法。

(5) 熟悉队列的应用。

学习者在学习过程中,可以沿着如下的学习线索进行学习。

(1) 学习栈的定义,修改原则,了解生活和软件系统中关于栈使用的例子。

(2) 学习栈的顺序存储结构以及关于栈顶的规定。

(3) 学习栈的链式存储结构,注意遵循后进先出的操作原则。

(4) 学习栈的一些应用,如在递归中的应用,在表达式求值中的应用等。

(5) 学习队列的定义,修改原则,了解生活和软件系统中关于使用队列的例子。

(6) 学习队列的顺序存储结构(循环队列)以及关于队头、队尾的规定,注意学习循环队列中相关公式的由来和推导。

(7) 学习队列的链式存储结构,注意进队、出队操作对队头、队尾的影响。

(8) 注重对栈和队列的比较学习,总结相同点和区别。

3.2.2　重点难点解析

1. 链栈与单链表的操作有何区别和联系

链栈是采用单链表作为存储结构的栈,在结点类型的定义上是一样的,同样每个结点包含两个域,一个是数据域,一个是指针域。单链表结点的指针域指向逻辑上的下一个结点地址,链栈结点的指针域指向逻辑上的次顶结点地址。

在操作上,单链表的操作灵活,根据提出的要求进行操作。链栈只能限定在栈顶一端进行进栈和出栈,对应单链表的操作就是在表头进一个元素和在表头出一个元素,其他操作则都不允许,所以,单链表的操作丰富得多。

2. 栈在递归调用中的作用

递归调用的基本思想是将一个难以直接解决的“大”问题逐层分解为越来越小、越来越容易解决的“小”问题,每次分解的问题的解决,都要依赖于下次分解问题的解决。也就是说,“小”一级的问题不解决,比之“大”一级的问题也无法解决。不过这种分解不能无限分解下去,一定要有一个明确的“不能”再分解的位置,而且该位置上的问题是有明确解答的,然后通过逐步回溯,通过“较小”问题的解决来解决比之“较大”一级的问题,从而层层回溯,达到原本“大”问题的解决。

上述“分解”与“不能再分解”表现为递归过程需要解决的递归计算公式和递归结束条件两个问题。

(1) 递归计算公式:是大问题在变成次大问题中所表现出来的规律,要表达清楚这个大问题与下一级的次大问题有什么联系。

(2) 递归结束条件:解决递归问题中的分解不能无终止地分解下去,需有一个结束的条件。这样才可以由结束递归再返回层层解套,最终解决整个问题。递归的结束条件也称为递归出口。

对于上述递归问题的解决大致做下述的算法描述:

```
if (递归结束条件)
```

```
        return  (递归结束条件下的返回值);
else
        return  (递归计算公式);
```

那么在这个递归调用的流程中,栈何时介入呢? 在每次进入下一级程序调用时,都需要将当前的程序信息保存在一个栈里,越迟出现的递归调用程序信息,越迟被保存,但当程序一层层退出递归时,越先被调出,参与问题的解决。

3. 栈在表达式求值中的作用及工作原理

在计算表达式求值的过程中需要根据算术四则运算的规则来进行计算,包括从左算到右,先乘除、后加减,先括号内、后括号外等原则,所以必须将四则运算法则融入计算机的计算过程中,这就需要建立运算符优先表(详见课本表 3-1)作为计算表达式计算顺序的依据。

那么在表达式求值过程中,为什么要用栈呢? 因为运算符是有优先级的,迟出现的运算符不一定迟处理,只有当前运算符优先级高于后面紧跟的运算符优先级时,才参与计算。并且在表达式中也可以嵌套表达式,如括号里的也是一个表达式,则必须括号内的处理完毕了,才能处理括号外面的,所以需要将运算符存储在栈里面,而当前处理的运算符就是栈顶运算符。

算术表达式求值的工作原理可以描述如下。

(1) 设置两个工作栈:一个是运算符栈,用于存放暂时未参与运算的运算符;一个是操作数栈,存放暂时未参与运算的操作数和中间结果。

(2) 对栈初始化:为方便判别表达式结束,在表达式的最前最后增加(虚拟)起始符"#"。栈初始化时设操作数栈为空,运算符栈的栈顶元素为表达式起始符"#",表示计算即将开始。

(3) 从左到右依次读入表达式中的字符,字符只有两种:操作数或运算符。判断读取到的字符是属于操作数还是运算符,分别做如下处理。

① 操作数:压入操作数栈。

② 运算符:要将运算符栈栈顶元素与该读取到的运算符比较优先级,结果有以下三种。

- 栈顶运算符优先级>待处理运算符优先级:运算符栈出一运算符,操作数栈出两操作数,计算,结果压入操作数栈。
- 栈顶运算符优先级=待处理运算符优先级:运算符栈出一运算符与读取到的运算符抵消。
- 栈顶运算符优先级<待处理运算符优先级:压入运算符栈。

(4) 重复执行(3),直到读取到"#"号,与运算符栈顶的"#"相抵消,最后运算符栈为空,操作数栈只有一个数值,即为结果。

4. 长度为 n 的循环队列为何最多只能存储 $n-1$ 个元素

队列的操作原则是先进先出,基本操作是进队列和出队列,常常需要扫描数组中队头元素和队尾元素的位置,故设置变量 front 来指示队头元素的位置和变量 rear 来指示队尾元素的下一个位置。front 和 rear 随着队列中元素的进出而动态地变化。出队列时,front 要后移一位,指向新的队头;进队列时,元素放入 rear 所示位置后,rear 也要后移一位,保证指

向新队尾的下一个空位。但如果 front 和 rear 只是单纯地后移，会造成顺序队列假溢出的问题，所以采用了循环队列，计算新的 front 和 rear 时要加上对数组容量的取模操作。

但循环队列又引发了新的问题，即队空和队满标志冲突的问题。为了解决这个问题，通常是以牺牲一个存储单元为代价，当判断到当前 rear 值的下一个位置是 front 时，就认为队满，停止进队。这样，rear 永远在 front 之后，不会因不断进队而从后面追上 front。这是以牺牲一个存储单元为代价的解决方案，所以，长度为 n 的循环队列最多只能存放 $n-1$ 个元素。

5. 循环队列元素个数计算公式的推导

循环队列中，rear 的值有可能大于 front，也有可能小于 front（等于时说明队列为空），但不管如何，rear 以及 rear 之后的位置是空位。当 rear＞front 时，元素个数为（rear－front）＜m，当 rear＜front 时，元素个数为[(rear－front)＋m]＜m，要为两个式子推导一个统一的公式，要将简单式子向复杂式子靠拢且不影响结果。将（rear－front）向[(rear－front)＋m]靠拢，当 rear＞front 时，[(rear－front)＋m]大于 m，如再对 m 取模，则会恢复原本(rear－front)的值。同时，当 rear＜front 时，[(rear－front)＋m]＜m 对 m 取模也不会影响结果。所以，不管 rear 和 front 的大小关系如何，循环队列元素个数的计算公式都可以为：[(rear－front)＋m] ％ m。

6. 链队出队时对队头、队尾指针有何影响

采用单链表存储队列时，常设置两个指针变量，一个指向头结点，头结点指针域指向队头结点，如用 front 指针指向头结点，称为 front 结点；另一个指向队尾结点，可设为 rear，称为 rear 指针。空队列时，front 和 rear 均指向 front 结点。

队列出队列从队头出，当队列元素个数大于 1 时，出队列是不会影响到 rear 的，但当队列元素个数等于 1 时，即出队列的元素恰好也是尾结点时，会影响 rear。由于 rear 被出队列了，没有队尾了，rear 要回到空队列的状态，即指向 front 结点。

7. 列举一两个队列在计算机系统中的应用实例

（1）当计算机主机与外部设备速度不匹配时，需要采用队列。如主机输出数据给打印机，主机的输出速度要远高于打印机的打印速度，解决方案是设置一个打印数据缓冲区，主机将要打印的数据写入缓冲区，写满后就暂停输出，将时间片分配给别的进程。打印机则按先来先服务的原则从缓冲区中依次取出数据并打印，打印完再向主机发出请求。实际上，这个打印数据缓冲区，就是一个打印队列，解决了主机与打印机速度不匹配的问题。

（2）在计算机操作系统中，有一个队列的典型应用就是解决多用户引起的资源竞争问题。如多个用户进程请求使用 CPU 资源，操作系统则按照请求的时间顺序建立一个进程队列，按先来先服务的原则，每次把 CPU 分配给队首用户使用并令其出队。当然，请求 CPU 资源的用户还有优先级的问题，更复杂的解决方案在操作系统课程中会有详细叙述。

3.3　例题

例 1　以下与数据的存储结构无关的术语是（　　）。

A. 循环队列　　B. 链表　　C. 散列表　　D. 栈

答案：D

解析：计算机有4种基本的存储结构，即顺序存储结构、链式存储结构、索引存储结构、散列存储结构。循环队列是采用顺序存储结构的队列，不过将队列想象成首尾相连的模式。栈是一个限定操作的线性表，是与数据结构有关的术语，可以根据情况决定采用顺序存储或者链式存储。

例2 有6个元素按6,5,4,3,2,1的顺序进栈，下列(　　)不是合法的出栈序列。

A. 5 4 3 6 2 1　　B. 4 5 3 1 2 6　　C. 3 4 6 5 2 1　　D. 2 3 4 1 5 6

答案：C

解析：本题容易忽视的是元素进栈顺序恰好与自然序相反。C选项中的3若要第一个出栈，说明6,5,4,3已经依次进栈，所以不可能出现选项中的4 6 5的出栈顺序。

例3 某循环队列空间大小为25，当队头为18，队尾为9时，其中元素个数为_____个。

答案：16

解析：题目中队尾下标小于队头下标，队列中真正存放了元素的位置是[18～24]和[0～8]，一共是16个元素。也可以直接套用到公式里，循环队列元素个数为[(rear－front)＋m] % m(m为队列容量)。

例4 以下说法正确的是(　　)。

A. 顺序队和循环队的队满和队空判断条件是一样的

B. 栈可以作为实现过程调用的一种数据结构

C. 插入和删除是数据操作中最基本的两种操作，所以这两种操作在数组中也经常使用

D. 在循环队列中，front指向队列中队头元素所在位置，rear指向队尾元素下一个空位，队列为满的条件是front＝rear

答案：B

解析：循环队列是将顺序队列视为首尾相连，自然会有些不一样。顺序队判满的条件是rear＝maxsize，循环队列判满的条件是(rear＋1) % maxsize＝front。在过程调用中栈的作用类似于中断处理中栈的应用，保留程序现场，以便返回，越迟发生的调用越早返回。插入和删除是数据操作中最基本的两种操作，但是在数组中实现这两种操作效率很低，所以不常使用，如果需要经常执行插入和删除操作，则应采用链式存储结构。

例5 即使不设置尾指针，链队也能进行入队、出队操作。(判断对/错)

答案：对

解析：链队设置尾指针，是因为常常需要从队尾进队列，为了节省时间开销，才设置尾指针，设置了尾指针后，进队列的时间复杂度是$O(1)$。如不设置尾指针，则需要找到队尾才能进队列，找队尾的时间开销是$O(n)$。即使不设置尾指针，链队也能进行入队操作，只是时间开销大了。

3.4 习题

一、填空题

1. 栈修改的原则是_____。

2. 对于顺序栈，如果在空栈情况下作退栈运算，则产生“_____”。

3. 一般地，栈和线性表类似，有两种实现方法，即_____实现和_____实现。

4. 实现在顺序栈上判栈空的条件为_____。

5. 实现在顺序栈上取栈顶元素，主要语句为_____。

6. 设一个链栈的栈顶结点为 top，判断栈空的条件是_____；如果栈不为空，则取栈顶操作为_____。

7. 实现在循环队列 queue 上的出队列，主要语句为_____。

8. 实现在循环队上判队空的条件为_____。

9. 链队判断空队列的条件为_____。

10. 在具有 n 个单元的循环队列中，队满时共有_____个元素。

11. 某循环队列空间大小为 20，当队头为 10，队尾为 5 时，其中元素个数为____个。

12. 循环队列下一个队尾的计算公式为_____。

二、判断题

(　　)1. 栈是一种对所有插入、删除操作限于在表的一端进行的线性表，是一种后进先出型结构。

(　　)2. 栈和链表属于两种不同的数据结构。

(　　)3. 栈和队列的存储方式既可以是顺序存储方式，也可以是链式存储方式。

(　　)4. 两个栈共享一片连续内存空间时，为提高内存利用率，减少溢出机会，应把两个栈的栈底分别设在这片内存空间的两端。

(　　)5. 队是一种插入与删除操作分别在表的两端进行的线性表，是一种先进后出型结构。

(　　)6. 一个栈的输入序列是 12345，则栈的输出序列不可能是 12345。

(　　)7. 因链栈本身没有容量限制，故在用户内存空间的范围内不会出现栈满情况。

(　　)8. 对于顺序栈而言在栈满状态下如果此时再作进栈运算，则会发生“下溢”。

三、单项选择题

1. 设有 4 个数据元素 a1、a2、a3 和 a4，对它们分别进行栈操作。在进栈时，按 a1、a2、a3、a4 次序每次进入一个元素。假设栈的初始状态是空。现要进行的栈操作是进栈两次，出栈一次，再进栈两次，出栈一次。这时，栈顶元素为(　　)。

A. a1　　B. a2　　C. a3　　D. a4

2. 数组栈中，若用 top 指向栈顶元素，x 入栈，操作为(　　)。

A. top=top+1;stack[top]=x;　　B. stack[top]=x;top=top+1;

C. top=top-1;stack[top]=x;　　D. stack[top]=x;top=top-1;

3. 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是(　　)。

A. A,B,C,D　　B. D,C,B,A

C. A,C,D,B　　D. D,A,B,C

4. 循环队列的入队操作应为(　　)。

A. sq.rear=sq.rear+1;　sq.data[sq.rear]=x;

B. sq.data[sq.rear]=x;　sq.rear=sq.rear+1;

C. sq.rear=(sq.rear+1)% maxsize;　sq.data[sq.rear]=x;

D. sq.data[sq.rear]=x; sq.rear=(sq.rear+1)% maxsize;

5. 假设用一维数组变量 sq[0..n-1]实现循环队列,front 和 rear 分别是队头和队尾指针,则出队的操作是(　　)。

A. sq.front=(sq.ftont+1)%n

B. sq.front=sq.front+1

C. sq.rear=(sq.rear+1)%n

D. sq.rear=sq.rear+1

6. 循环队列的队满条件为(　　)。

A. (sq.rear+1) % maxsize==(sq.front+1) % maxsize

B. (sq.rear+1 % maxsize==sq.front+1

C. sq.(rear+1) % maxsize==sq.front

D. sq.rear==sq.front

7. 如果以链表作为栈的存储结构,则执行退栈操作时(　　)。

A. 必须判别栈是否满　　B. 必须判别栈是否空

C. 判别栈元素的类型　　D. 对栈不做任何操作

8. 设有一顺序栈已含三个元素,a1,a2,a3,a1 在栈底,元素 a4 正等待进栈。那么下列 4 个序列中不可能出现的出栈序列是(　　)。

A. a3,a1,a4,a2　　B. a3,a2,a4,a1　　C. a3,a4,a2,a1　　D. a4,a3,a2,a1

9. 向一个栈顶结点为 Top 的链中插入一个 s 所指结点时,其操作步骤为(　　)。

A. Top->next=s;

B. s->next=Top->next;Top->next=s;

C. s->next=Top;Top=s;

D. s->next=Top;Top=Top->next;

10. 从链栈 Top 中删除一个结点,并将被删结点的值保存到 x 中,其操作步骤为(　　)。

A. x=Top->data;Top=Top->next;

B. Top=Top->next;x=Top->data;

C. x=Top->next->data;Top->next=Top->next->next;

D. x=Top->data; Top->next=Top->next->next;

11. 在一个链队中,若 f,r 分别为队首、队尾指针,则插入 s 所指结点的操作为(　　)。

A. f->next=c;f=s　　B. r->next=s;r=s

C. s->next=r;r=s　　D. s->next=f;f=s

12. 链栈与顺序栈相比,有一个比较明显的优点即(　　)。

A. 插入操作更方便　　B. 通常不会出现栈满的情况

C. 不会出现栈空的情况　　D. 删除操作更方便

13. 一个队列的入队序列是 1,2,3,4,则出队序列可能是(　　)。

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

14. 设计一个判别表达式中左、右括号是否配对的算法,采用(　　)数据结构最佳。

A. 线性表的顺序存储结构　　B. 栈

C. 队列　　D. 线性表的链式存储结构

15. 用数组 $Q[n]$表示一个循环队列，f 为当前队头元素的位置，r 为队尾元素的下一个空位，计算队列中元素的公式为(　　)。

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

四、应用题

1. 设循环队列的容量为 40(序号从 0 到 39)，现经过一系列的入队和出队操作后，问在以下两种情况下，循环队列中各有多少个元素？

(1) front=11，rear=19

(2) front=19，rear=11

2. 按照四则运算优先关系的法则，写出计算表达式 # 6 * 8 /(7−5) # 时栈的变化情况。

五、算法设计题

称正读和反读都相同的字符序列为“回文”，如 abcrrcba 为回文，ababab 则不是回文。试编写一个算法，判断一个字符序列是不是回文。

3.5　实验指导

3.5.1　栈的应用之单括号匹配

1. 实验目的

(1) 会定义顺序栈和链栈的结点类型；

(2) 熟悉栈的进栈和出栈代表的实际意义；

(3) 掌握两种存储结构下栈的基本操作：进栈、出栈、取栈顶等；

(4) 了解在什么情况下会应用到栈。

2. 实验内容

输入一包含(和)的字符串，检测括号是否匹配(其中括号中能嵌套括号)，并输出括号是否匹配的信息(匹配、缺少左括号、缺少右括号)。

3. 程序分析

(1) 先考虑一个匹配的字符串的特点，如“fha(bh(gf(hg)gt)q)gfr”，其中第三个左括号是与第一个右括号相匹配的，即越迟出现的左括号越先与右括号匹配。

(2) 还没有匹配成功的左括号要存储起来，匹配成功的左括号可以忽略，即删除。出现了越后存储起来的左括号越先删除的情况，符合栈的操作原则。

(3) 可以得到大概的程序设计要点：遇到“(”，入栈操作；遇到“)”，如栈空则缺少左括号，否则出栈操作。当所有字符处理完毕时，栈是空的说明所有左右括号相匹配，否则缺少右括号。

(4) 可能用到的函数如下。

gets(string)：其功能为键盘输入一个字符串，内容赋值给 string 字符数组，程序会自动在字符串末尾添加'\\0'作为结束标志。

strlen(string)：求字符串 string 的长度。

4. 调试测试

(1) 输入：按提示输入包含(和)的字符串。

(2) 输出：是否匹配的信息。

情况(1)：匹配，如图 3-8 所示。

图 3-8　匹配的情况

情况(2)：缺少左括号，如图 3-9 所示。

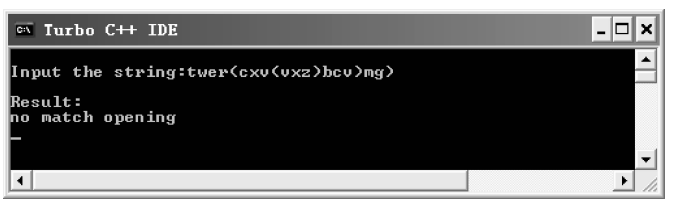

图 3-9　缺少左括号的情况

情况(3)：缺少右括号，如图 3-10 所示。

图 3-10　缺少右括号的情况

5. 思考题

在发生“缺少左括号”的错误时，如何确定错误发生的具体位置？

3.5.2　栈的应用之多级括号匹配

1. 实验目的

(1) 熟练掌握两种存储结构下栈的基本操作：进栈、出栈、取栈顶等；

(2) 进一步了解在什么情况下会应用到栈；

(3) 能考虑更复杂一些的关于栈的应用的实例。

2. 实验内容

输入一个包含(、[、{和)、]、}的字符串，检测括号是否匹配(其中大括号中能嵌套中括号，中括号中能嵌套小括号，反之不行)，并输出括号是否匹配的信息(匹配、缺少左括号类型、缺少右括号类型)。

3. 程序分析

(1) 提示：这个程序不能用三个栈解决问题，因为三种括号并不是互相独立匹配的，而是有一定制约条件的。

(2) 同样是越迟出现的左括号越先与右括号匹配。

(3) 根据括号嵌套原则，分析如下。

① 对于左边括号而言：如出现的是 [，则之前最迟出现的不允许是(；如出现的是{，则之前最迟出现的不允许是 [或者 (。

② 对于右边括号而言：如出现的是)，则最迟出现的是左边括号必须是 (才能与之匹配；如出现的是] ，则最迟出现的是左边括号必须是 [才能与之匹配；如出现的是 } ，则最迟出现的是左边括号必须是 { 才能与之匹配。

③ 只要是允许的情况，就遇左边括号进栈，遇右边括号出栈，否则提示出错。

4. 调试测试

(1) 输入：按提示输入包含(、[、{和)、]、}的字符串。

(2) 输出：是否匹配的信息。

情况(1)：匹配，如图 3-11 所示。

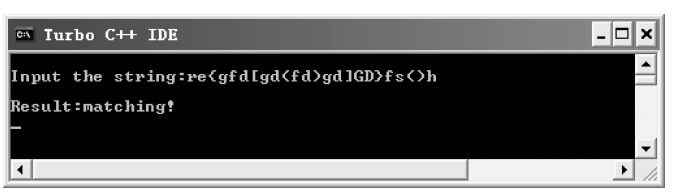

图 3-11　匹配的情况

情况(2)：错误出现“{”，如图 3-12 所示。

图 3-12　错误出现“{”的情况

情况(3)：缺少右边括号，如图 3-13 所示。

图 3-13　缺少右括号的情况

5. 思考题

当所有字符处理完毕时,栈是空的说明所有左右括号相匹配,否则缺少右边括号,如何判断缺少右边哪个括号?

3.5.3 十进制整数转化为 R 进制整数

1. 实验目的

(1) 熟练掌握栈的基本操作;

(2) 会用栈解决一些实际应用;

(3) 掌握十进制整数转化为 R 进制整数的工作原理。

2. 实验内容

任意输入一个十进制数,利用栈的思想,完成实现将其转化为 R 进制整数的算法并输出其 R 进制整数。

3. 程序分析

(1) 十进制整数转化成 R 进制整数的方法为:除 R 取余数,得到的余数即为 R 进制数个位的数码;得到的商继续除 R 取余数,得到的余数为 R 进制数十位的数码;……,如此反复,直到商为 0,最后得到的余数是 R 进制数最高位的数码。

(2) 得到的余数从右到左排列(反序排列)就是所求的 R 进制数,故可用栈作为 R 进制数的存储方式。

(3) 实现十进制整数转化成 R 进制整数的过程如下。

十进制整数 X 和 R 作为已知条件,初始化栈,只要 X 不为 0 重复做下列动作:

$$\text{将 } X\%R \text{ 入栈} \rightarrow X = X/R$$

(4) 输出得到的 R 进制数的方法如下。

只要栈不为空,重复做下列动作:

$$\text{输出栈顶元素} \rightarrow \text{栈顶出栈}$$

4. 调试测试

(1) 输入:按提示要求,输入需要转换的十进制数和 R 进制数的基数 R,如图 3-14 所示。

图 3-14 输入

(2) 输出结果,如图 3-15 所示。

5. 思考题

如果是十进制的小数转换为 R 进制数,该如何转换?工作原理如何?需要用到何种存储方式?

图 3-15　输出

3.5.4　队列基本操作

1. 实验目的

(1) 会定义顺序队和链队的结点类型；

(2) 熟悉队列的入队和出队代表的实际意义；

(3) 掌握两种存储结构下队列的基本操作：入队、出队、输出队列等；

(4) 熟悉 C 语言中子程序的编写与调用。

2. 实验内容

建立队列并输出；然后插入元素再输出队列；最后删除元素再输出队列。要求：在输入数据的过程中程序做相应的是否继续输入的提示。

3. 程序分析

(1) 入队与出队操作都不是只入一个或只出一个，还要求询问是否继续操作，所以出队和入队都要编写成子程序形式，方便需要时调用。

(2) 每入队一次或出队一次，要输出队列，以便查看结果是否正确，所以输出队列也要编写成子程序形式。

(3) 读懂并掌握该语句写法：

```
printf("\n insert? 1/0");
scanf(" %d",&k);
while (k == 1)
 {  insert(); display();
    printf("\n insert again? 1/0");
    scanf(" %d",&k);
 }
```

(4) 此程序题目用循环队列实现便可。

4. 调试测试

(1) 输入：按提示输入队列元素，以－99 结束，如图 3-16 所示。

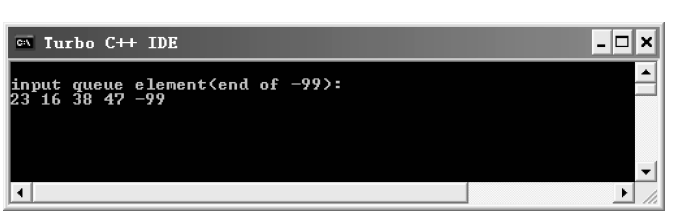

图 3-16　输入

(2) 输出队列,以及询问是否继续入队列,如图 3-17 所示。

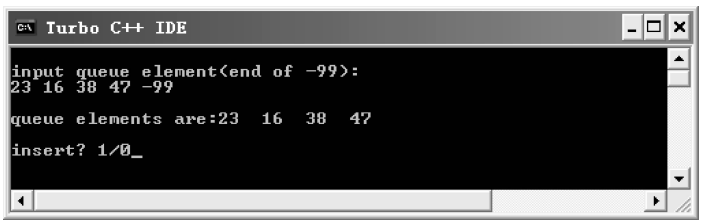

图 3-17 输出

(3) 输入 1,代表继续入队列,按提示输入队列元素。输出插入成功的提示和队列。继续询问是否入队列,如图 3-18 所示。

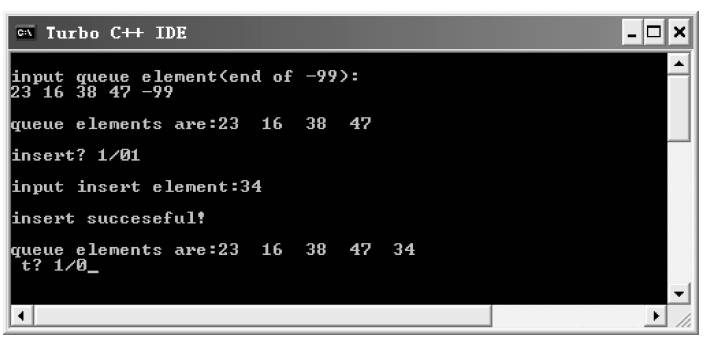

图 3-18 进队列

(4) 输入 0,代表停止入队。提示询问是否出队列,如图 3-19 所示。

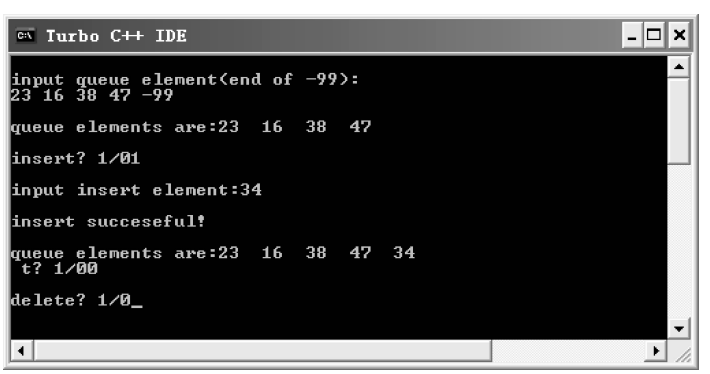

图 3-19 程序提示

(5) 输入 1,代表要出队列。再次提示是否确定要出队列,输入 1,代表确定出队列。输出出队列成功的提示,并输出队列。继续询问是否出队列,如图 3-20 所示。

(6) 输入 0,代表不再出队列,程序结束。

5. 思考题

试采用链队实现上述题目。

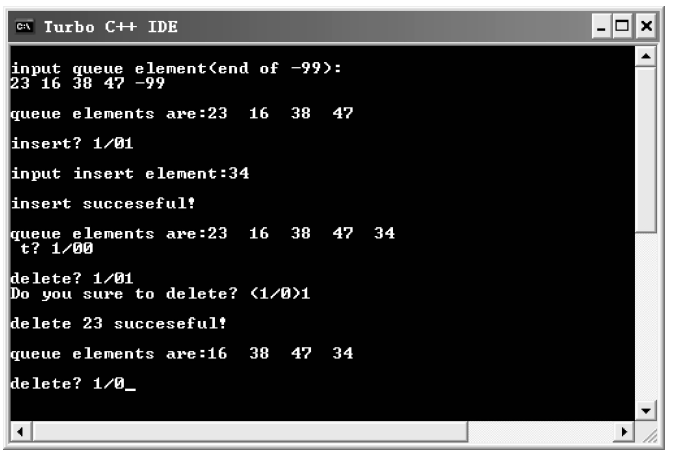

图 3-20　出队列

3.5.5　跳舞配对问题 1

1. 实验目的

(1) 熟练掌握两种存储结构下队列的基本操作：入队、出队、输出队列等；

(2) 掌握队列的原理，会应用队列解决一些实际应用。

2. 实验内容

班上有 m 个女生，有 n 个男生(m 不等于 n)，现要开一个舞会。男女生分别以 1～m 和 1～n 编号，坐在舞池两边的椅子上，每曲开始时，依次从男生和女生中各出一人配对跳舞，本曲没成功配对者坐着等待下一曲找舞伴。配对成功的舞伴跳完舞后排回各自队伍的最后。

请设计一系统模拟动态地显示上述过程，要求输出第 K 支舞曲配对情况。

3. 程序分析

(1) 男生和女生依次排队，排在队头的男生和排在队头的女生先配对，舞曲结束排到各自的队尾。因此该问题具有典型的先进先出特性，可用队列作为数据的存储方式。

(2) 算法需要两个队列，分别是存放男生队列和女生队列，每跳一支舞曲则是男、女队列的分别一次出队列和入队列。

(3) 初始化时男生、女生队列各元素的值分别为他们的编号序列。

(4) 解决实验内容的问题，只需运用循环语句，是男、女队列分别执行 K 次入队、出队即可，第 K 次出队列的数据对就是第 K 支舞曲男、女生配对的情况。

4. 调试测试

(1) 输入：按提示输入女生和男生的人数，以及需要输出配对情况的曲目 K，如图 3-21 所示。

(2) 输出：第 K 曲女生和男生配对跳舞的情况，如图 3-22 所示。

5. 思考题

如果每支舞曲的跳舞人数不是女生、男生各出一人配对，而是各出 5 人配对成 5 对舞伴，程序该如何修改？

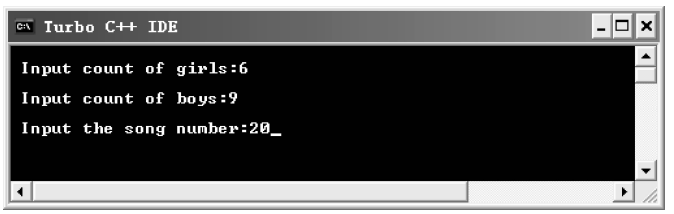

图 3-21　输入

图 3-22　输出

3.5.6　跳舞配对问题 2

1. 实验目的

(1) 熟练掌握两种存储结构下队列的基本操作：入队、出队、输出队列等；

(2) 掌握队列的原理，会应用队列解决一些实际应用。

2. 实验内容

班上有 m 个女生，有 n 个男生(m 不等于 n)，现要开一个舞会。男女生分别以 $1\sim m$ 和 $1\sim n$ 编号，坐在舞池两边的椅子上，每曲开始时，依次从男生和女生中各出一人配对跳舞，本曲没成功配对者坐着等待下一曲找舞伴。配对成功的舞伴跳完舞后排回各自队伍的最后。

请设计一系统模拟动态地显示出上述过程，要求输出计算任何一个女生(编号为 X)和任意男生(编号为 Y)第一次配对跳舞的舞曲编号 K。

3. 程序分析

(1) 男生和女生依次排队，排在队头的男生和排在队头的女生先配对，舞曲结束排到各自的队尾。因此该问题具有典型的先进先出特性，可用队列作为数据的存储方式。

(2) 算法需要两个队列，分别是存放男生队列和女生队列，每跳一支舞曲则是男、女队列的分别一次出队列和入队列。

(3) 初始化时男生、女生队列各元素的值分别为他们的编号序列。

(4) 解决该问题的方法与 3.5.5 类似，将每次配对的女生和男生编号与指定编号比较，相等则目前该舞曲是需要计算的舞曲。注意舞曲的数目必须限制在一定的范围内，避免死循环。例如，在 100 支舞曲内，X 和 Y 都没有配对跳舞，则输出不可能配对成功。

4. 调试测试

(1) 输入：按提示输入女生和男生的人数，以及希望配对跳舞的女生编号和男生编号，

如图 3-23 所示。

图 3-23　输入

(2) 输出：X 女生和 Y 男生配对跳舞成功，并且输出当前 X 和 Y 首次配对跳舞的舞曲序号 K，如图 3-24 所示。

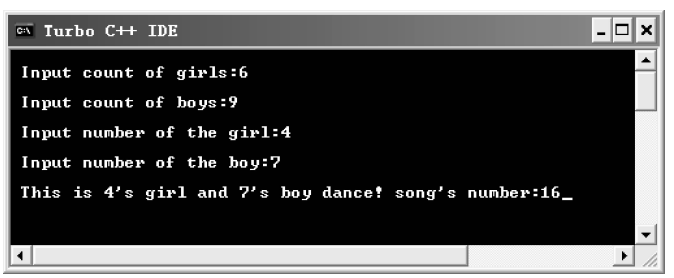

图 3-24　输出

5. 思考题

输入什么数据，会出现 100 支舞曲内 X 和 Y 都不能成功配对跳舞的情况？自己尝试一下。