第3章　栈和队列

3.1　栈

3.1.1　知识点复习

1. 栈的定义及基本运算

(1) 栈(Stack)可定义为只允许在表的末端进行插入和删除的线性表。允许插入和删除的一端叫做栈顶(top),而不允许插入和删除的另一端叫做栈底(bottom)。当栈中没有任何元素时则成为空栈。

(2) 栈只能通过它的栈顶进行访问。故栈叫做后进先出(LIFO,Last In First Out)或先进后出(FILO,First In Last Out)的线性表。

(3) 栈的基本运算包括以下5种:

① 栈初始化 void initStack(Stack& S):创建一个空栈 S 并使之初始化。

② 判栈空否 int StackEmpty(Stack& S):当栈 S 为空时函数返回1,否则返回0。

③ 进栈 int Push(Stack& S,type x):当栈 S 未满时,函数将元素 x 加入并使之成为新的栈顶,若操作成功,则函数返回1,否则函数返回0。

④ 出栈 int Pop(Stack& S,type& x):当栈 S 非空时,函数将栈顶元素从栈中退出并通过引用参数 x 返回退出元素的值。若操作成功,则函数返回1,否则函数返回0。

⑤ 读栈顶元素 int getTop(Stack& S,type& x):当栈 S 非空时,函数将通过引用参数 x 返回栈顶元素的值(但不退出)。若操作成功,则函数返回1,否则函数返回0。

利用上述5种基本运算,可以实现基于栈结构的应用问题的求解。

注意,栈操作 Pop(S,x)与 getTop(S,x)是有区别的。前者退出栈顶的元素,因此每执行一次 Pop 操作,栈中的元素个数就少一个;后者仅复制出一份栈顶元素,栈中元素个数不发生变化。

2. 栈的混洗(shuffle)

(1) 当一串元素通过栈时,出栈元素的次序如何排列,与进栈和出栈操作的排列组合有关。如果每个元素进栈后立即出栈,所有出栈元素的排列与进栈顺序完全相同;如果所有元素都进栈后才开始出栈,所有出栈元素的排列与进栈顺序完全颠倒。

(2) 栈的进出规则:利用铁路调车轨道的栈式结构分析,当列车车厢按照 $1,2,\cdots,n$ 的编号进栈时,可能的不同出栈序列数目的计算可利用 Catalan 函数算出:

$$\frac{1}{n+1}C_{2n}^{n}=\frac{1}{n+1}\times\frac{(2n)!}{n!\times n!}$$

3. 栈的顺序存储结构

栈的顺序存储,简称顺序栈,是指用一组地址连续的存储单元依次存储栈中元素,同时附设指针 top 指示栈顶元素。顺序栈的存储分配有两种:

(1) 顺序栈的静态存储分配,它的存储数组采用静态方式定义。

```
#define maxSize 100
typedef char SElemType;
typedef struct{
    SElemType elem[maxSize];
    int top;
}SeqStack;
```

顺序栈的静态存储结构需要预先定义或申请栈的存储空间，也就是说，栈空间的容量有限且一旦装满不能扩充。因此在顺序栈中，当一个元素进栈之前，需要判断是否栈满（栈空间中没有空闲单元），若栈满，则元素进栈会发生上溢现象。

（2）顺序栈的动态存储分配，它的存储数组在使用时动态存储分配，其好处是一旦栈满可以扩充。顺序栈的动态存储结构定义为：

```
#define initSize 100
typedef char SElemType;
typedef struct{
    SElemType * elem;
    int maxSize,top;
}SeqStack;
```

在初创基于动态存储分配的顺序栈时必须立即为它动态分配存储空间：

```
elem=new SElemType[initSize]
```

并以 initSize 作为最初的 maxSize。在扩充时需要申请一个新的具有 2×maxSize 的连续的存储空间取代原来的存储空间，把原来存储空间内存放的所有栈元素转移到新的存储空间后释放原来的存储空间，再让 elem 指针指示新的存储空间，并让 maxSize＝2 * maxSize。这样扩充需要足够的内存空间。

（3）进栈和出栈有两种处理方式。

① 先让栈顶指针进一，再按栈顶指针所指位置把进栈元素加入。这样，栈顶是最后加入元素所处的位置。它的示意图如图 3.1 所示。

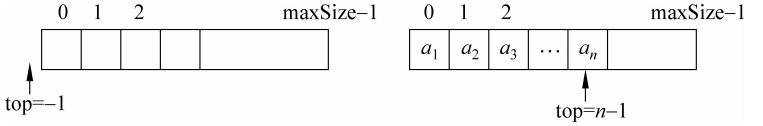

图 3.1　顺序栈的示意图之一

按照这种处理方式，在使用栈之前，需要做初始化工作，即置空栈。因为一维数组下标从 0 开始，栈空时应该 top＜0，因此空栈时栈顶指针 top＝＝－1。

② 先按栈顶指针所指位置把进栈元素加入，再把栈顶指针加一。这样，栈顶是最后加入元素的下一位置，是下一次加入元素的位置。它的示意图如图 3.2 所示。

按照这种处理方式，空栈时应该 top＝0。本书采用前一种处理方式。

（4）两个顺序栈共享一个存储空间。

利用栈底位置相对不变的特性，可让两个顺序栈共享一个一维数据空间，以互补余缺，

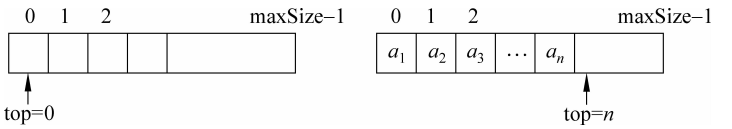

图 3.2　顺序栈的示意图之二

实现方法是：将两个栈的栈底位置分别设在存储空间的两端，让它们的栈顶各自向中间延伸，如图 3.3 所示。这样，两个栈的空间就可以相互调节，只有在整个存储空间被占满时才发生上溢，这样一来产生上溢的概率要小得多。

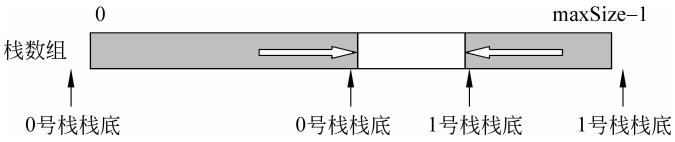

图 3.3　两个栈共享一个空间的结构示意图

若设 0 号栈的栈底指针为 b[0]，栈顶指针为 t[0]；1 号栈的栈底指针为 b[1]，栈顶指针为 t[1]，则各栈初始化的条件为 b[0]＝t[0]＝－1，b[1]＝t[1]＝maxSize；各栈判栈空的条件为 t[0]＝＝b[0] 和 t[1]＝＝b[1]。判栈满的条件为 t[0]＋1＝＝t[1]，注意，绝对不是 t[0]＋t[1]＝＝maxSize。

4. 栈的链式存储结构

利用单链表作为存储结构的栈称为链式栈。链式栈的表示如图 3.4 所示。

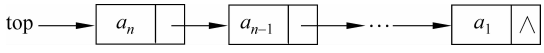

图 3.4　链式栈

从图 3.3 可知，链式栈的栈顶在链表的表头。因此，新结点的插入和栈顶结点的删除都在链表的表头，即栈顶进行。通常情况下，作为链式栈使用的链表不要头结点，链表的表头指针就是栈顶指针。链式栈的类型定义如下：

```
typedef int SElemType;
typedef struct node{
    SElemType data;
    struct node * link;
}LinkNode, * LinkStack;
```

采用链式栈来表示一个栈，便于结点的插入与删除。在程序中同时使用多个栈的情况下，用链接表示不仅能够提高效率，还可以达到共享存储空间的目的。

5. 理解栈的存储结构的要点

(1) 顺序栈的操作一般在栈顶指针处进行。在顺序栈的实现中，初始化函数用于为栈赋初值，令 top＝－1，置栈为空。

(2) 在实现进栈操作时，应先判断栈是否已满。如果栈顶指针 top＝＝maxSize－1 (maxSize 是栈的容量)，则说明栈已满。如再加入新元素将发生栈溢出，程序转入出错处理。如果 top＜maxSize－1，则先让栈顶指针进 1，指到当前可加入新元素的位置，再按栈顶

指针所指位置将新元素插入。这个新插入的元素将成为新的栈顶元素。

(3) 如果在退栈时发现是空栈,即 top==-1,则退栈操作将执行栈空处理。栈空处理一般不是出错处理,而是使用这个栈的算法结束时需要执行的处理。

(4) 链式栈的栈顶在链表的表头。因此,新结点的插入和栈顶结点的删除都在链表的表头,即栈顶进行。

(5) 因为所有的操作都限定在链表的表头进行,不需要设定链表的表尾指针。

(6) 在不能预知栈的大小的情况下,使用链式栈比顺序栈更合适。

3.1.2 疑难点辨析

1. 栈是一种先进后出的顺序存取结构,它是顺序存储结构吗?

【辨析】 顺序存取和顺序存储是两个不同的概念。顺序存取是指只能逐个存或逐个取结构中的元素,例如栈只能在栈顶位置存或取;顺序存储是指利用一个连续的空间相继存放结构中的元素,例如栈可基于一维数组存放栈的元素。

2. 一个较早进栈的元素能否先于在它之后进栈的元素从栈中取出?

【辨析】 如果它进栈后一直未退出,在它之后进栈的元素一旦压在它的上面,它就不能先于后进栈的元素取出。如果在它之后要进栈的元素还未进栈,它是可以退出的。

3. 一般来讲,只允许栈顶元素从栈中退出,在什么情况下元素可以从栈底泄出?

【辨析】 在操作系统中设计调度算法时有此特例。

4. 以 1,2,…,n 的顺序进栈,如何判断可能的出栈序列?

【辨析】 要注意各进栈元素出栈的时机。假如 5 先于 3 和 4 出栈,那么 3 就不可能在 4 之前出栈,因为 4 一定压在 3 的上面。分析可能的出栈序列有多少,类似于分析不同二叉树有多少,最终推导出 Catalan 函数。

5. 当一个顺序栈已满,如何才能扩充栈长度,使得程序能够继续使用这个栈?

【辨析】 只限于顺序栈的动态存储分配方式。可另分配一个长度大一倍的连续存储空间,用它来取代原来的存储空间。

6. 当两个栈共享同一个存储空间 V[m]时,可设栈顶指针数组 t[2]和栈底指针数组 b[2]。如果进栈采用两个栈相向前行的方式,则任一栈的栈满条件是什么?

【辨析】 当两个栈的栈顶指针碰面了,即 t[0]+1==t[1],则表明栈已满。此时假定 0 号栈在 1 号栈的左边。

7. 顺序栈的优点是什么? 缺点是什么?

【辨析】 顺序栈的优点是存取速度快。缺点是当栈满时要发生溢出,为了避免这种情况,需要为栈设立一个足够大的空间。但如果空间设置得过大,而栈中实际只有几个元素,也是一种空间浪费。此外,当程序中需要使用多个栈时,因为各个栈所需的空间在运行中是动态变化着的。如果给几个栈分配同样大小的空间,可能在实际运行时,有的栈膨胀得快,很快就产生了溢出,而其他的栈可能此时还有许多空闲的空间。这时就必须调整栈的空间,防止栈的溢出。

8. 链式栈可否增加头结点? 如果增设了头结点,链式栈的栈顶在链表的什么位置? 栈底在链表的什么位置? 栈指针如何设置? 栈空条件是什么? 栈满条件是什么?

【辨析】 可以增设头结点。这时链式栈的栈顶在头结点的 link 指针所指向的结点,即

首结点；栈底在链表的表尾。栈顶指针在头结点，每次进栈/出栈都在头结点之后进行，不必修改栈顶指针。因为所有的栈操作都限定在栈顶，即链表的表头进行，所以不需要设置链表的表尾指针。栈空条件为 top->link==NULL，无栈满条件。

9. 链式栈的优点是什么？缺点是什么？

【辨析】 采用链式栈的优点是便于结点的插入与删除，不存在栈满的问题。在程序中同时使用多个栈的情况下，用链接表示不仅能够提高效率，还可以达到共享存储空间的目的。缺点是需要额外的存储空间存放每个结点的链接指针。

10. 链式栈只能顺序存取，而顺序栈不但能顺序存取，还能直接存取，这种说法对吗？

【辨析】 不对。栈本身是一种顺序存取结构，不论是链式栈还是顺序栈，都不允许直接存取。

11. 理论上链式栈没有栈满问题，但在进栈时还有一个后置条件，是何条件？

【辨析】 该后置条件放在用 new()或 malloc()动态分配栈结点存储空间的语句之后，判断动态分配是否成功。如果动态分配成功，则新元素可正确插入，进栈成功；否则报告结点动态存储分配失败，进栈失败。

3.1.3 选择题解析

题 3.1.1 栈的插入和删除操作在(　　)进行。

A. 栈顶　　B. 栈底　　C. 任意位置　　D. 指定位置

【解析】 选 A。栈的定义要求在栈顶插入与删除。

题 3.1.2 对一个初始为空的栈 s 执行操作 Push(s,5)，Push(s,2)，Push(s,4)，Pop(s,x)，getTop(s,x)后，x 的值应是(　　)。

A. 5　　B. 2　　C. 4　　D. 0

【解析】 选 B。连续 3 次进栈后，栈内数据为 5,2,4，经过 1 次退栈，栈内还有 5,2，栈顶读到 x 值应为 2。

题 3.1.3 用 S 表示进栈操作，用 X 表示出栈操作，若元素的进栈顺序是 1234，为了得到 1342 的出栈顺序，相应的 S 和 X 的操作序列为(　　)。

A. SXSXSSXX　　B. SSSXXSXX　　C. SXSSXXSX　　D. SXSSXSXX

【解析】 选 D。用排除法选择答案。选项 A、B、C 所得的出栈序列分别为 1243、3241 和 1324，都不是正确答案。选项 D 得到的出栈序列为 1342。

题 3.1.4 假设一个栈的输入序列是 1,2,3,4，则不可能得到的输出序列是(　　)。

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

【解析】 选 B。借用上一小题的符号，用 S 表示进栈操作，用 X 表示出栈操作，则选项 A、C 和 D 的输出序列可以分别用操作序列 SXSXSXSX、SSSSXXXXX 和 SXSSXSXX 得到，这都是合法的操作序列。只有选项 B 的输出序列找不到合法的操作序列。

题 3.1.5 已知一个栈的进栈序列为 1,2,3,…,n，其输出序列的第一个元素是 i，则第 j 个出栈元素是(　　)。

A. $j-i$　　B. $n-i$　　C. $j-i+1$　　D. 不确定

【解析】 选 D。当 i 第一个出栈时，1,2,…,$i-1$ 都压在栈内，之后还可能有 i 之后的元素进栈，究竟第 j 个出栈元素是哪个，不一定。

题 3.1.6 已知一个栈的进栈序列为 1,2,3,…,n,其输出序列是 $p_1,p_2,p_3,\cdots,p_n$。若 $p_1=n$,则 p_i 的值是(　　)。

A. i　　B. $n-i$　　C. $n-i+1$　　D. 不确定

【解析】 选 C。当第一个出栈元素是 n 时,1,2,3,…,$n-1$ 都压在栈内,后续出栈的元素依次为 $n-1,n-2,\cdots$,第 i 个出栈元素应是 $n-i+1$。

题 3.1.7 已知一个栈的进栈序列为 1,2,3,…,n,其输出序列是 $p_1,p_2,p_3,\cdots,p_n$。若 $p_1=3$,则 p_2 的值(　　)。

A. 一定是 2　　B. 一定是 1　　C. 可能是 1　　D. 可能是 2

【解析】 选 D。当第一个出栈元素为 3 时,1,2 一定压在栈内,下一个出栈的元素可能是 2,不可能是 1。当然也可能 2 暂不出栈,4,5,…进栈,所以第二个出栈的元素也可能不是 2。

题 3.1.8 已知一个栈的进栈序列为 $p_1,p_2,p_3,\cdots,p_n$,其输出序列是 1,2,3,…,n。若 $p_3=1$,则 p_1 的值(　　)。

A. 一定是 2　　B. 可能是 2　　C. 不可能是 2　　D. 一定是 3

【解析】 选 C。当 $p_3=1$ 时,输入序列为 $p_1,p_2,1,p_4,\cdots$,因为输出的第一个元素是 1,则 p_1,p_2 一定在栈内。如果下一个出栈的是 p_2,可以断定 $p_2=2$,则 p_1 不可能是 2。当然可能是 3,但不一定是 3:也许 p_1 暂不出栈,$p_4,p_5,\cdots,p_i=3$ 进栈,p_i 再出栈。

题 3.1.9 已知一个栈的进栈序列为 $p_1,p_2,p_3,\cdots,p_n$,其输出序列是 1,2,3,…,n。若 $p_3=3$,则 p_1 的值(　　)。

A. 一定是 2　　B. 可能是 2　　C. 不可能是 1　　D. 一定是 1

【解析】 选 B。当 $p_3=3$ 时,输入序列为 $p_1,p_2,3,p_4,\cdots$,因为输出的第 3 个元素是 3,则有多种可能:当 $p_1=1,p_2=2$ 时,允许的进/出栈顺序是 p_1 进栈 p_1 出栈 p_2 进栈 p_2 出栈。当 $p_1=2,p_2=1$ 时,允许的进/出栈顺序是 p_1 进栈 p_2 进栈 p_2 出栈 p_1 出栈。如果 p_1、p_2、3 进栈不出,$p_4=2,p_5=1$,允许的进出栈顺序可以是 p_4 进栈 p_5 进栈 p_5 出栈 p_4 出栈。因此可以断定 $p_1=1$ 或 $p_1=2$ 是可能的选择,但不是一定的选择。

题 3.1.10 已知一个栈的进栈序列为 $p_1,p_2,p_3,\cdots,p_n$,其输出序列是 1,2,3,…,n。若 $p_n=1$,则 p_1 的值(　　)。

A. $n-i+1$　　B. $n-i$　　C. i　　D. 不确定

【解析】 选 A。当 $p_n=1$ 时,输入序列为 $p_1,p_2,p_3,p_4,\cdots,p_{n-1},1$,因输出序列为 1,2,3,…,$n$,所以第一次输出 1 时,$p_1,p_2,p_3,p_4,\cdots,p_{n-1}$ 都应在栈内,这样可得 $p_{n-1}=2,p_{n-2}=3,\cdots,p_{n-i}=i+1,\cdots,p_i=n-i+1$。

题 3.1.11 当利用大小为 n 的数组顺序存储一个栈时,假定用 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。

A. top++;　　B. top--;　　C. top=0;　　D. top=n;

【解析】 选 B。根据题意,栈底应在第 n 个位置,因此进栈时栈顶指针应该先减 1,向栈空间的低端延伸。

题 3.1.12 以下有关顺序栈的操作中正确的是(　　)。

A. n 个元素进入一个栈后,它们的出栈顺序一定与进栈顺序相反

B. 若一个栈的存储空间为 S[n],则对栈的进栈和出栈操作最多只能执行 n 次

C. 栈是一种对进栈、出栈操作的次序做了限制的线性表

D. 空栈没有栈顶指针

【解析】 选 A。当 n 个元素进栈后，出栈顺序当然与当初进栈的顺序相反。

题 3.1.13 在实现顺序栈的操作时，在进栈之前应先判断栈是否（ 1 ），在出栈之前应先判断栈是否（ 2 ）。

A. 空　　B. 满　　C. 上溢　　D. 下溢

【解析】 (1) 选 B，(2) 选 A。在进栈之前应先判断是否满，如果栈已满，再进栈就会发生上溢。在出栈之前应先判断是否空，如果栈已空，再出栈就会发生下溢。

题 3.1.14 为了增加内存空间的利用率和减少溢出的可能，在两个栈共享一片连续的存储空间时，应将两个栈的栈顶分设在这片存储空间的两端，当（　　）时才产生上溢。

A. 两个栈的栈顶同时到达栈空间的中心点

B. 其中一个栈的栈顶到达栈空间的中心点

C. 两个栈的栈顶在栈空间的某一位置相遇

D. 两个栈的栈顶相加超过了栈空间的最大容量

【解析】 选 C。两个栈的栈顶一开始设在栈空间的两端，在进栈时一个栈的栈顶按“加 1”方式向栈空间中心前进，另一个栈的栈顶则按“减 1”方式向栈空间中心前进。两个栈的进栈速度可能不同，当两个栈的栈顶在栈空间的某一个位置相遇时才算栈满。

题 3.1.15 设链式栈中结点的结构为(data，link)，且 top 是指向栈顶的指针。若想在链式栈的栈顶插入一个由指针 s 所指的结点，则应执行的操作是（　　）。

A. top->link=s;

B. s->link=top->link; top->link=s;

C. s->link=top; top=s;

D. s->link=top; top=top->link;

【解析】 选 C。在栈顶插入就是在链表首元结点前插入，必须首先让 s 后面链接上原栈顶 top，再让栈顶指针指向新栈顶 s，所以有 s->link=top;top=s;。

题 3.1.16 设链式栈中结点的结构为(data，link)，且 top 是指向栈顶的指针。若想摘除链式栈的栈顶结点，并将被摘除结点的值保存到 x 中，则应执行的操作是（　　）。

A. x=top->data; top=top->link;

C. top=top->link; x=top->data;

C. x=top; top=top->link;

D. x=top->data;

【解析】 选 A。为了从链式栈的栈顶 top 摘下栈顶结点并把该结点存放的值保存在 x 中，首先做 x=top->data，然后让栈顶指针退到次栈顶位置 top=top->link 即可。

3.1.4 应用题选讲

题 3.1.17 请综述栈的基本性质。

【解答】 栈的基本性质如下：

(1) 集合性。栈是由若干个元素集合而成，没有元素的空集合称为空栈。

(2) 线性。除栈底和栈顶元素外，栈中任一元素均有唯一的前趋元素和后继元素。

(3) 运算受限。只允许在栈顶实施进栈或出栈操作,且栈顶位置由栈顶指针指示。

(4) 数学性质。当多个编号元素依某种顺序进栈,且可任意时刻出栈时,所获得的编号元素排列的数目恰好满足 Catalan 函数的计算,即

$$\frac{1}{n+1}C_{2n}^{n}=\frac{1}{n+1}\times\frac{(2n)!}{(n!)^2}$$

其中,n 为编号元素的个数。

题 3.1.18 若一个栈的输入序列为 1,2,…,n,输出序列的第一个元素是 n,则第 i 个输出元素是什么($1\leqslant i\leqslant n$)?

【解答】 $n-i+1$。按照题意,输出序列应该是 n,$(n-1)$,$(n-2)$,…,1,需要把 1,2,…,$(n-1)$一个一个压在底下,再后进先出地一个一个退出栈。这样,第 1 个进的第 n 个出,第 2 个进的第 $n-1$ 个出,第 3 个进的第 $n-2$ 个出,依次类推,可知第 i 个进的,第 $n-i+1$ 个出。

题 3.1.19 铁路进行列车调度时,常把站台设计成栈式结构的站台,如图 3.5 所示。试问:

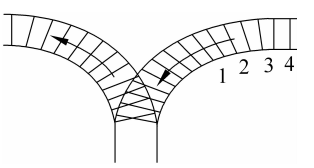

图 3.5 列车调度示例

(1) 设有编号为 1,2,3,4,5,6 的 6 辆列车顺序开入栈式结构的站台,则可能的出栈序列有多少种?

(2)若进站的 6 辆列车顺序如上所述,那么是否能够得到 435612,325641,154623 和 135426 的出站序列,如果不能,说明为什么不能;如果能,说明如何得到(即写出“进栈”或“出栈”的序列)。

【解答】

(1) 编号为 1,2,3,4,5,6 的 6 辆列车顺序开入栈式结构站台,可能的不同出栈序列有

$$\frac{1}{n+1}C_{2n}^{n}=\frac{1}{6+1}C_{12}^{6}=\frac{1}{7}\times\frac{12\times11\times10\times9\times8\times7}{6\times5\times4\times3\times2\times1}=132\text{ 种}$$

(2) 不能得到 435612 和 154623 这样的出栈序列。因为若在 4,3,5,6 之后再将 1,2 出栈,则 1,2 必须一直在栈中,此时 1 先进栈,2 后进栈,2 应压在 1 上面,不可能 1 先于 2 出栈。154623 也是这种情况。出栈序列 325641 和 135426 可以得到。

题 3.1.20★ 试证明:若借助栈可由输入序列 1,2,3,…,n 得到一个输出序列 p_1,p_2,p_3,…,p_n(它是输入序列的某一种排列),则在输出序列中不可能出现以下情况,即存在 $i<j<k$,使得 $p_j<p_k<p_i$(提示:用反证法)。

【解答】 证明过程如下。

(1) 必要性:按照题意,当 $i<j<k$ 时进栈顺序是 i、j、k,这 3 个元素出栈的相对顺序是 p_i、p_j、p_k。例如,当 $i=1$,$j=2$,$k=3$ 时一个合理的出栈序列是 $p_i=2$,$p_j=3$,$p_k=1$。如果 $p_j<p_k<p_i$ 成立,意味着出栈顺序为 3、1、2,这恰恰是不可能的。当较大的数首先出栈时,那些较小的数都是降序压在栈内的,如 2、1,这些数不可能如 1、2 那样正序出栈。

(2) 充分性:如果 $p_j<p_k<p_i$ 成立,表明当 $i<j<k$ 时各元素进栈出栈后的相对顺序为 p_j、p_k、p_i。现做具体分析:

当 $i<j$ 时 $p_j<p_i$,表明 p_j 是在 p_i 进栈后进栈并压在 p_i 上面,且 p_j 在 p_i 出栈前出栈。

当 $j<k$ 时 $p_j<p_k$,表明 p_j 必须在 p_k 入栈之前就出栈,否则 p_j 就被压在 p_k 下面了。

当 $i<k$ 时 $p_k<p_i$,表明 p_i 是先于 p_k 进栈的。

综上所述可知：这与正确的出栈顺序 $p_i<p_j<p_k$ 相矛盾。

证毕。

题 3.1.21 设有一个顺序栈 S，元素 s_1,s_2,s_3,s_4,s_5,s_6 依次进栈，如果 6 个元素的出栈顺序为 s_2,s_3,s_4,s_6,s_5,s_1，则顺序栈的容量至少应为多少？

【解答】 至少为 3。在出栈顺序为 s_2,s_3,s_4,s_6,s_5,s_1 时，顺序栈只需 3 个栈元素即可满足要求，进栈与出栈情况如图 3.6 所示。

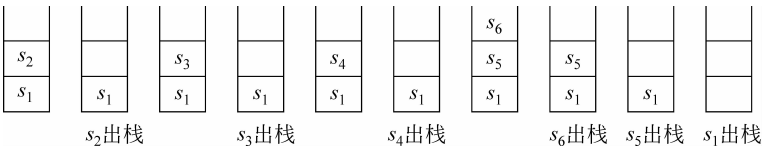

图 3.6 题 3.1.21 的进栈与出栈情况

题 3.1.22 改写顺序栈的进栈成员函数 Push(x)，要求当栈满时执行一个 stackFull() 操作进行溢出处理。其功能是：动态创建一个比原来的栈数组大一倍的新数组，代替原来的栈数组，原来栈数组中的元素占据新数组的前 maxSize 个位置。

【解答】

```
void Push(SeqStack& S,SElemType x){
    if(S.top==maxSize-1) StackFull(S);                    //栈满,做溢出处理
    S.elem[++S.top]=x;                                    //进栈
};
void StackFull(SeqStack& S){
    SElemType * temp=new SElemType[2 * S.maxSize];
    if(temp==NULL){cerr<<"存储分配失败!\n"; exit(1);}
    for(int i=0; i<=S.top; i++) temp[i]=S.elem[i];        //传送原栈内的数据
    delete [] S.elem;                                     //删去原数组
    S.maxSize=2 * S.maxSize;                              //数组最大体积增长一倍
    S.elem=temp;                                          //新数组成为栈的数组空间
};
```

题 3.1.23 借助栈实现单链表上的逆置运算。

【解答】 算法的思路是：首先遍历单链表，逐个结点删除并把被删结点存入栈中，再从栈中取出存放的结点把它们依次链入单链表中。通过栈把结点的次序颠倒过来。

```
void Reverse(LinkList& L){
    LinkStack S; LinkNode * p=L->link, * q;
    if(p!=NULL){                                          //检测原链表
        L->link=p->link; Push(S,p);                       //结点 p 从原链表摘下,进栈
        p=L->link;
    }
    p=L;
    while(!StackEmpty(S)){                                //当栈不空时
        Pop(S,q);                                         //退栈,退出元素由 q 指示
        p->link=q; p=q;                                   //链入结果链尾
```

```
}
p->link=NULL:                                          //链收尾
};
```

题 3.1.24 设一个栈的输入序列为 1,2,…,n,编写一个算法,判断一个序列 p_1,p_2,…,p_n 是否是一个合理的栈输出序列。

【解答】 首先举例说明。设一个输入序列为 1,2,3,输出序列为 2,3,1,可能的进栈/出栈动作如表 3.1 所示。

表 3.1 进栈/出栈动作示例 1

输入序列当前数据	1	2	2	3	3	3	
栈内数据	—	1	1,2	1	1,3	1	—
预期的输出序列		2	2	3	3	1	
栈顶与输出序列当前数据比较		1<2	2=2	1<3	3=3	1=1	
动作	进栈	进栈	出栈	进栈	出栈	出栈	结束

如果输出序列为 3,1,2,这是一个不合理的输出序列,则可能的进栈/出栈动作如表 3.2 所示。从表中可见,当预期的输出序列的数据比当前栈顶的元素小时,一定出现了输出不合理的情形,可以报错并结束检查处理。

表 3.2 进栈/出栈动作示例 2

输入序列当前数据	1	2	3	3	3		
栈内数据	—	1	1,2	1,2,3	1,2	1,2	
预期的输出序列		3	3	3	1		
栈顶与输出序列当前数据比较		1<3	2<3	3=3	2>1		
动作	进栈	进栈	进栈	出栈	报错	结束	

算法中用 $i=1,2,\cdots,n$ 作为栈的输入数据序列,用 $p[0],p[1],\cdots,p[n-1]$作为预期的栈的输出序列。

```
int Decision(int p[],int n){
//算法判断序列 p[0],p[1],…,p[n-1]是否合理的出栈序列,如果合理,则函数返回 0
//如果不合理,则函数返回 1
    SeqStack S; InitStack(S);
    int i=1,k=0,j;
    Push(S,i);
    do{
        getTop(S,j);
        if(j<p[k]){i++; Push(S,i);}                    //输入序列下一数据进栈
        else    if(j==p[k]){Pop(S,j); k++;}            //出栈后检查下一输出数据
            else{cout<<"不合理的"; break;}             //不合理情形
    }while(! StackEmpty(S) && i<=n);
    cout<<"输出序列: ";
```