第3章　栈和队列

3.1　栈

1. 栈的基本概念

栈(stack)是限定只在表头进行插入(入栈)与删除(出栈)操作的线性表,表头端称为栈顶,表尾端称为栈底。

设有栈 $S=(a_1,a_2,\cdots,a_n)$,则一般称 a_1 为栈底元素,a_n 为栈顶元素,按 $a_1,a_2,\cdots,a_n$ 的顺序依次进栈,出栈的第一个元素为栈顶元素,也就是说栈是按后进先出的原则进行进栈和出栈操作,如图 1.3.1 所示,所以栈可称为后进先出(last in first out,LIFO)的线性表(简称为 LIFO 结构)。

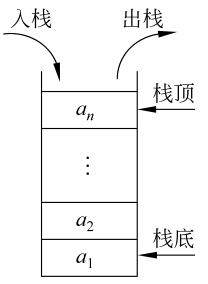

图 1.3.1　栈示意图

在实际应用中,栈包含了如下基本操作:

1) int Length() const

初始条件:栈已存在。

操作结果:返回栈元素个数。

2) bool Empty() const

初始条件:栈已存在。

操作结果:如栈为空,则返回 true,否则返回 false。

3) void Clear()

初始条件:栈已存在。

操作结果:清空栈。

4) void Traverse(void (*visit)(const ElemType &)) const

初始条件:栈已存在。

操作结果:从栈底到栈顶依次对栈的每个元素调用函数(*visit)。

5) bool Push(const ElemType &e)

初始条件:栈已存在。

操作结果:插入元素 e 为新的栈顶元素。

6) bool Top(ElemType &e) const

初始条件:栈已存在且非空。

操作结果:用 e 返回栈顶元素。

7) bool Pop(ElemType &e)

初始条件:栈已存在且非空。

操作结果:删除栈顶元素,并用 e 返回栈顶元素。

8) bool Pop()

初始条件:栈已存在且非空。

操作结果:删除栈顶元素。

2. 顺序栈

在顺序实现中，利用一组地址连续的存储单元依次存放从栈底到栈顶的数据元素，将数据类型为 ElemType 的数据元素存储在数组中，并用 count 存储数组中存储的栈的实际元素个数。当 count = 0 时表示栈为空，每当插入新的栈顶元素时，如栈未满，操作成功，count 的值将加 1。而当删除栈顶元素时，如栈不空，操作成功，并且 count 的值将减 1。

3. 链式栈

链式栈的结构如图 1.3.2 所示，入栈和出栈操作都非常简单，一般都不使用头节点直接实现。

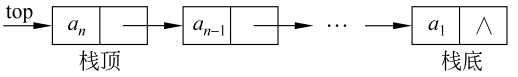

图 1.3.2　链式栈示意图

3.2　队　　列

1. 队列的基本概念

队列(queue)是一种先进先出(first in first out，FIFO)的线性表，只允许在一端进行插入(入队)操作，在另一端进行删除(出队)操作。

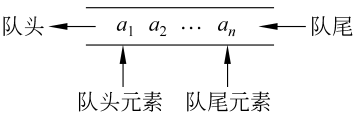

图 1.3.3　队列示意图

在队列中，允许入队操作的一端称为队尾，允许出队操作的一端称为队头，如图 1.3.3 所示。

设有队列 $q=(a_1,a_2,\cdots,a_n)$，则一般 a_1 称为队头元素。a_n 称为队尾元素。队列中元素按 $a_1,a_2,\cdots,a_n$ 的顺序入队，同时也按相同的顺序出队。队列的典型应用是操作系统中的作业排队。在实际应用中，队列包含了如下的基本操作：

1) int Length() const

初始条件：队列已存在。

操作结果：返回队列长度。

2) bool Empty() const

初始条件：队列已存在。

操作结果：如队列为空，则返回 true，否则返回 false。

3) void Clear()

初始条件：队列已存在。

操作结果：清空队列。

4) void Traverse(void (* visit)(const ElemType &)) const

初始条件：队列已存在。

操作结果：依次对队列的每个元素调用函数(* visit)。

5) bool OutQueue(ElemType &e)

初始条件：队列非空。

操作结果：删除队头元素，并用 e 返回其值。

6) bool OutQueue()

初始条件：队列非空。

操作结果：删除队头元素。

7) bool GetHead(ElemType &e) const

初始条件：队列非空。

操作结果：用 e 返回队头元素。

8) bool InQueue(const ElemType &e)

初始条件：队列已存在。

操作结果：插入元素 e 为新的队尾。

2. 链队列

用链表表示的队列称为链队列，一个链队列应有两个分别指示队头与队尾的指针(分别称为头指针与尾指针)，如图 1.3.4 所示。

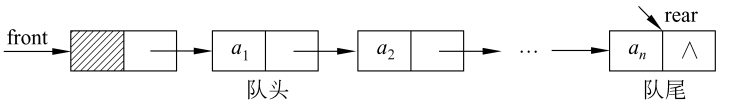

图 1.3.4　链队列示意图

如要从队列中退出一个元素，必须从单链表的首元素节点中取出队头元素，并删除此节点，而入队的新元素是存放在队尾处的，也就是单链表的最后一个元素的后面，并且此节点将成为新的队尾。

3. 循环队列——队列的顺序存储结构

用 C++ 描述队列的顺序存储结构，就是利用一个容量是 maxSize 的一维数组 elems 作为队列元素的存储结构，其中 front 和 rear 分别表示队头和队尾，maxSize 是队列的最大元素个数。当队列的实际可用空间还没有使用完，这种情况下再插入一个新元素产生的溢出称为假溢出，解决假溢出的一个技巧是将顺序队列从逻辑上看成一个环，成为循环队列，循环队列的首尾相接，当队头 front 和队尾 rear 进入 maxSize − 1 时，再进一个位置就自动移动到 0。此操作可用取余运算(%)简单地实现。

队头进 1：front＝(front＋1)％maxSize

队尾进 1：rear＝(rear＋1)％maxSize

只从 front＝rear 无法判断是队空还是队满，有 3 种处理方法：

(1) 另设一个标志符区别队列是空还是满。

(2) 少用一个元素空间，约定队头在队尾指针的下一位置(指环状的下一位置)时作为队满的标志。

(3) 增加一个表示元素个数 count 的成员，队空条件就变成 count＝＝0，队满条件为 count＝＝maxSize。

本书配套软件包采用方法(3)处理循环队列。

*3.3 优先队列

在许多情况下，前面介绍的队列是不够的，先进先出的机制有时需要某些优先规则来完善。比如在医院中，危重病人应具有更高的优先权，也就是当医生有空时，应立刻医治危重病人，而不是排在最前面的病人。

优先队列(priority queue)是一种数据结构，其中元素的固有顺序决定了对基本操作的执行结果，优先队列有两种类型：最小优先队列和最大优先队列。最小优先队列的出队操作 OutQueue()将删除最小的数据元素值。最大优先队列的出队操作 OutQueue()将删除最大的数据元素值。

优先队列有多种实现方法，一种比较简单的实现方法是在做入队操作 InQueue()时，元素不是排在队列的队尾，而是将其插入在队列的适当位置，使队列的元素有序，优先队列类模板可作为队列类的派生来实现。只需要重载入队操作 InQueue()即可。