第3章 队 列

CHAPTER

队列是限定允许在表的一端进行插入操作，而在另一端进行删除操作的线性表。允许插入的一端称为**队尾**，允许删除的一端称为**队首**。向队列中插入新元素称为**进队**或**入队**，新元素入队后就成为新的队尾元素。从队列中删除元素称为**出队**或**离队**，出队后，出队元素的后一个元素为新的队首元素。

队列具有“先进先出”或“后进后出”的特性，与日常生活中的队列相似。

3.1 循环队列

3.1.1 循环队列存储定义和特性

采用顺序存储方式实现的队列称为顺序队列。顺序队列定义如下：

```
template <class DT>
struct  SqQueue
{
    DT * base;              //存储空间基地址
    int  front;             //队头指针,指向队首元素
    int  rear;              //队尾指针,指向队尾元素的后面
    int  queuesize;         //队列容量
}
```

为了解决顺序队列中的“假溢出”问题，假设顺序队列首尾相连，如图 1.3.1 所示，此时，顺序队列称为“**循环队列**”。本定义中，队尾指针指向队尾元素后，队头指针指向队首元素。

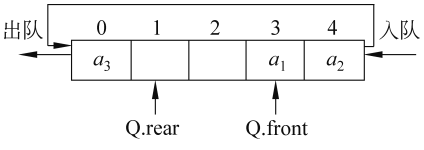


图 1.3.1 循环队列存储示意图

循环队列中的入队和出队，指针的运算为对队容量求模，入队时队尾指针变化为

```
Q.rear=(Q.rear+1)%Q.queuesize
```

出队时队头指针变化为

```
Q.front=(Q.front+1)%Q.queuesize
```

为了区分队空和队满，将队空条件设为 Q.front==Q.rear；入队中当 Q.rear 与 Q.front 之间差一个单元时，认为队满，即队满条件为(Q.rear+1) % Q.queuesize==Q.front。

3.1.2 循环队列操作实现原理

1. 初始化队列　bool InitQueue (SqQueue &Q,int m)

初始化队列指在内存中创建一个空队。操作步骤如下：

Step 1　申请一组连续的内存空间作为队列的存储空间，首地址为队列基址 Q.base。

Step 2　申请失败，退出。

Step 3　申请成功，给队列属性赋值。

3.1　队头 Q.front=0，队尾 Q.rear=0。

3.2　队列容量 Q.queuesize=申请的容量。

3.3　返回 true，表示创建成功。

2. 销毁队列　void DestroyQueue (SqQueue &Q)

销毁队列指释放队列所占的内存空间。操作步骤如下：

Step 1　对应申请内存空间的 new 命令，用 delete 释放循环队列 Q 所占内存空间。

Step 2　设置队列属性值：

2.1　队头指针 Q.front=0，队尾指针 Q.rear=0。

2.2　队列容量 Q.queuesize =0，表示队列不可用。

3. 入队　bool EnQueue (SqQueue &Q, DT e)

入队指在队尾插入元素 *e*。操作步骤如下。

Step 1　如果队满，返回 false，表示入队失败。

Step 2　否则在队尾插入元素，操作如下：

2.1　将新元素插入队尾，即 Q.base[Q.rear]←e。

2.2　队尾指针后移 1，循环队列中，Q.rear=(Q.rear+1) %Q.queuesize。

2.3　返回 true，表示入队成功。

4. 出队　bool DeQueue(SqQueue &Q, DT &e)

出队指在队头删除元素 *e*。操作步骤如下。

Step 1　如果队空，返回 false，表示出队失败。

Step 2　否则，删除队首元素，操作如下：

2.1　取队头元素赋给 *e*，即 e←Q.base[Q.front]。

2.2　队头指针后移 1，循环队列中，Q.front=(Q.rornt+1) %Q.queuesize。

2.3　返回 true，表示出队成功。

5. 取队头元素　bool GetHead (SqQueue Q, DT &e)

取队头元素的操作步骤如下。

Step 1　如果队空，返回 false，表示无队头元素。

Step 2　队非空，取队头元素，操作如下：

2.1　把队头元素值赋给 e，即 e←Q.base[Q.front]。

2.2　返回 true，表示操作成功。

注意：相比于出队，此操作不删除队头元素，也无须移动队头指针。

3.2 链　　队

3.2.1 链队的存储定义和特性

链队是采用链式存储实现的队列，可用有头结点的单链表表示。

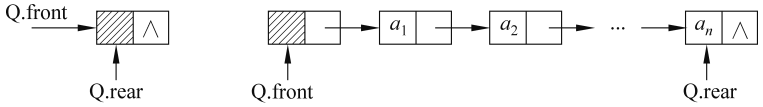
链队的结点结构与单链表的结点结构相同，定义如下：

```
template <class DT>
struct QNode                    //结点类型名
{
    DT data;                    //数据域,存储数据元素
    QNode * next;               //指针域,指向后继结点
};
```

队列的链式存储结构定义如下：

```
template <class DT>
struct LinkQueue
{
    QNode<DT> * front;          //队头
    QNode<DT> * rear;           //队尾
}
```

链队的存储示意图如图 1.3.2 所示。



(a) 空队　　(b) 非空队

图 1.3.2　链队存储示意图

因为有头结点，队头指针指向头结点，队头元素是 Q.front->next 所指结点，所以元素出队时，不需要调整队头指针。

入队时，是在 Q.rear 后增加结点；出队时是删除 Q.front 后的结点。如果队列中只有一个元素，出队后为空队，需把 Q.rear 指向头结点。

3.2.2 链队的操作实现原理

1. 初始化队列　bool InitQueue（LinkQueue &Q）

初始化队列指在内存中创建一个空队。操作步骤如下。

Step 1 创建一个结点作为头结点。

Step 2 队头指针和队尾指针均指向该结点。

2. 销毁队列 void DestroyQueue（LinkQueue &Q）

销毁队列指释放队列所占的内存空间。从头结点开始，依次销毁，与单链表的销毁类似。

3. 入队 bool EnQueue（LinkQueue &Q，DT e）

入队指在队尾插入一个元素。操作步骤如下。

Step 1 创建一个结点 p。

Step 2 创建失败，返回 false，表示入队失败。

Step 3 创建成功，给结合点 p 赋值 e 后在队尾插入新元素结点，操作如下：

3.1 p 结点链在队尾 Q.rear 之后，为链表的尾结点。

3.2 队尾指针指向 p。

3.3 返回 true，表示入队成功。

4. 出队 bool DeQueue（LinkQueue &Q，DT &e）

出队指删除队头元素 e。操作步骤如下。

Step 1 如果队空，返回 false，表示不能出队。

Step 2 否则，删除队头元素，操作如下：

2.1 取队头元素赋给 *e*，即 e←Q.front→next→data。

2.2 删除队头结点。

2.3 如果出队后队列为空，调整队尾指针指向头结点。

2.4 返回 true，表示出队成功。

5. 取队头元素 bool GetHead（LinkQueue Q，DT &e）

如果队空，返回 false，表示无队头元素；否则把队头元素值赋给 *e*。相比于出队，此操作不删除队头元素。

3.3 队列的应用

以舞伴匹配作为队列的应用示例。

设周末舞会上需男女搭配跳舞。依据先来先配对原则，分别设立男队和女队，入场时男队和女队的队头各出一人配成舞伴。若两队初始人数不等，则较多的那队中未配对者等待下一轮。操作步骤如下。

Step 1 创建两个空队，一个为男队 GenQueue，一个为女队 LadyQueue。

Step 2 反复循环，依次将等待的跳舞者根据其性别插入男队或女队。

Step 3 舞曲开始，只要男队和女队都不空，重复执行下列操作：

男队和女队各出队一人，配对入场。

Step 4 配对结束，从非空队列中输出下一轮第一个出场的未配对者的姓名。