实验五　队列实验

【实验目的】

（1）掌握队列的两种不同的存储结构。

（2）掌握循环队列的基本运算方法。

【实验背景】

队列也是一种重要且常用的数据结构，其逻辑结构与顺序表、链表、堆栈相同，但其运算较顺序表、链表有更多的限制，与堆栈一样，也是运算受限的线性表。

用地址连续的向量空间依次存储队列中的元素，同时记录当前队头元素及队尾元素在向量中的位置，这样顺序存储的队列简称**顺序队列**。

由于队列中元素的插入与删除分别在队列的两端进行，为了记录变化中的队头和队尾元素，我们分别用两个整型变量记录当前队头及队尾元素在向量中的位置。这样，顺序队列的数据类型可以定义如下：

```
＃define maxlen 100
typedef struct{
    Datatype data[maxlen]；
    int front；
    int rear；
} SeqQueue；
```

我们分别称 front 和 rear 为队头指针和队尾指针。为方便起见，我们规定队头指针 front 总是记录队头元素的前一个位置，队尾指针记录当前队尾元素的位置。

顺序队列在入队运算时，rear 的值加 1；当 rear = maxlen-1 时，认为队满。但此时不一定是真的队满，因为随着队头元素的不断出队，data 数组前面会出现一些空单元；而入队运算只能在队尾进行，使得这些空单元无法使用。我们将这个时候进行入队运算而产生的溢出称为“假溢出”。

为解决假溢出现象而使顺序队列的空间得以充分利用，我们规定：如果队头元素前有空单元而 rear = maxlen-1 时，我们将下一个入队的元素放在 data[0]的位置，且 rear 的值为 0。也就是说，我们将数组 data 想象成一个首尾相接的环形，data[0]就是 data[maxlen-1]的后继单元。我们形象地称这样的顺序队列为**循环队列**。

如何判断循环队列的队空和队满呢？

对于这个问题，可以有两种处理方法：一种方法是少用一个元素空间，也就是说，让 front 指向的元素空间永远是空的，不存放任何元素；这样，队满时的状态描述为：

```
if(Q->front = = (Q->rear + 1)%maxlen)；
```

而队空的条件不变，描述为：

```
if(Q->front = = Q->rear) ；
```

另一种方法是增设一个标志量 flag：每进行一次入队运算，标志量 flag 置 1；每进行一次出队运算，标志量 flag 置 0。这样，当出现 front 与 rear 的值相等时，若 flag = 1，表示最后一次进行的是入队运算，front 与 rear 相等表示队满；若 flag = 0，表示最后一次进行的是出队运算，front 与 rear 相等表示队空。

循环队列的基本运算可以基于这两种方法来分别实现。

与链表的存储结构类似，我们也用链式存储结构存储队列，称为**链队列**。

与单链表不同的是，队列仅限制在队头删除数据元素、队尾插入数据元素。为此，我们设置一个队头指针、一个队尾指针，分别指向链队列的队头和队尾。这样，一个链队列就由

一个头指针和一个尾指针唯一确定。我们将这两个指针封装在一起，将链队列定义为一个结构类型：

```
typedef struct{
    LinkList *front，*rear；
}LinkQueue；
LinkQueue *Q；    //Q 是该类型的指针
```

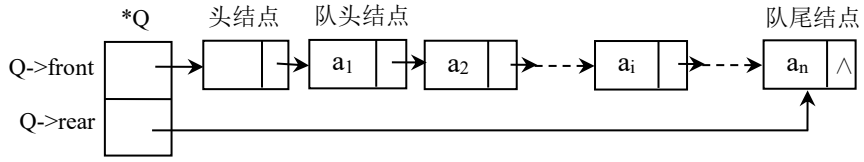

图 7.1 链队列示意图

和单链表一样，为了方便运算，我们也在队头节点前附加一个节点，称为头节点，且令 front 指向该头节点。图 7.1 所示为这样存储的链队列示意图。

我们在这样的链队列存储结构上实现队列的基本运算。

【实验任务】

1、程序验证

（1）分别建立含有若干个元素的循环队列和链队列，并分别实现循环队列和链队列的入队和出队操作。

（2）写出下列程序段的输出结果，并通过运行来验证。

```
void print( ) {
    SeqQueue   *q;
    char   x, y;
    x='e';    y='c';
    InitQueue(q);
    Add(q, 'h');
    Add(q, 'r');
    Add(q, y);
    x=GetHead(q);
    Delete(q);
    Add(q,   x);    x=GetHead(q);
    Delete(q);        Add(&q,'a');
    while(QueueEmpty(q)= =0) {
        y=GetHead(q);
        Delete(q);
        printf("%c",y);
    }
    printf("%c",x);
}
```

2、算法设计

（1）假设以数组 se[m]存放循环队列的元素，同时设变量 rear 和 num 分别作为队尾指针和队中元素个数记录。试讨论判别此循环队列的队满条件，写出相应入队和出队算法，并通过运行验证之。

（2）假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾结点。试编写相应

的置空队、判队空、入队和出队等算法，并通过运行验证之。

（3）设顺序栈 S 中有 2n 个元素，从栈顶到栈底的元素依次为 $a_{2n}, a_{2n-1}, \cdots, a_1$，要求通过一个循环队列重新排列栈中元素，使得从栈顶到栈底的元素依次为 $a_{2n}, a_{2n-2}, \cdots, a_4, a_2$， a_{2n-1}， a_{2n-3}， …, a_3, a_1，请设计算法实现该操作，要求空间复杂度和时间复杂度均为 O(n)。

3、实例演练：停车场管理

[问题描述]

设停车场内只有一个的停放 n 辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满 n 辆汽车，则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入；当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它让路，待该辆车开出大门外，其它车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序。

[测试数据]

设 n=2,输入数据为：（‘A’，1，5），（‘A’，2，10），（‘D’，1，15），（‘A’，3， 20），（‘A’，4，25），（‘A’，5，30），（‘D’，2，35），（‘D’，4，40），（‘E’，0，0）。其中，‘A’表示到达；‘D’表示离去，‘E’表示输入结束。

[基本要求]

①以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到达或离去的时刻，对每一组输入数据进行操作后的输出数据为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置；若是车离去；则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表实现。

② 设计算法完成问题求解；

③ 分析算法的时间复杂度和空间复杂度。

[实现提示]

需另设一个栈，临时停放为给要离去的汽车让路而从停车场退出来的汽车，也用顺序存储结构实现。输入数据按到达或离去的时刻有序。栈中每个元素表示一辆汽车，包含两个数据项：汽车的牌照号码和进入停车场的时刻。

[思维扩展]

（1） 两个栈共享空间，思考应开辟数组的空间是多少？

（2） 汽车可有不同种类，则它们的占地面积不同，收费标准也不同，如 1 辆客车和 1.5 辆小汽车的占地面积相同，1 辆十轮卡车占地面积相当于 3 辆小汽车的占地面积。

（3） 汽车可以直接从便道上开走，此时派在它前面的汽车要先开走让路，然后再依次排到队尾。

（4） 停放在便道上的汽车也收费，收费标准比停放在停车场的车低，请思考如何修改结构以满足这种要求。