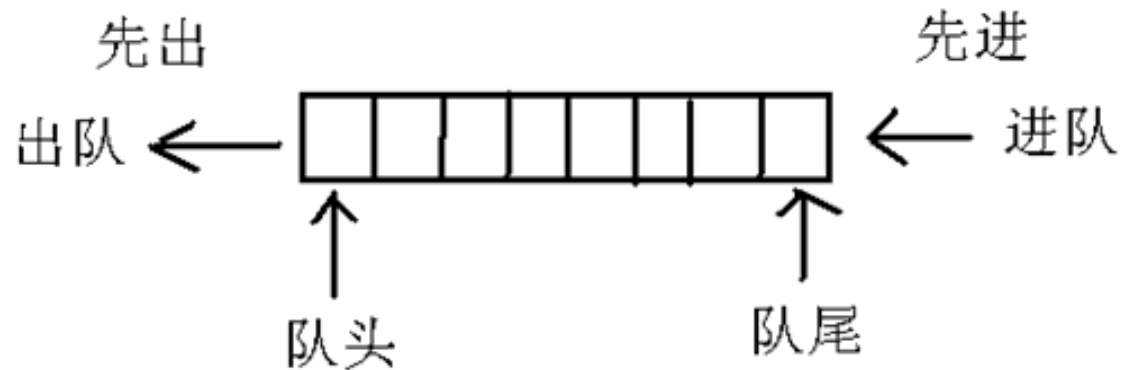


第四章 队列





本章节目录

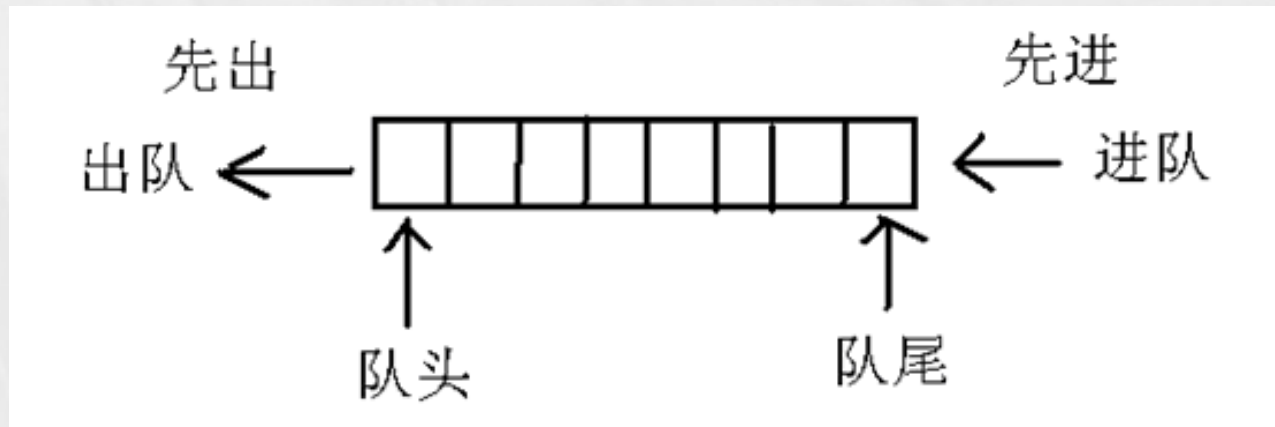
[4.1 队列的抽象数据类型](#)

[4.2 队列的顺序存储结构](#)

[4.3 队列的链式存储结构](#)

4.1 队列的抽象数据类型

- ❖ **定义：**队列(Queue)是限制在表的一端进行插入，而在表的另一端进行删除的**线性表**，通常称允许进行插入的一端为**队尾(Rear)**，允许进行删除的一端为**队头(Front)**。
- ❖ 队列的修改原则：先进先出（FIFO），举例
- ❖ 队列的抽象数据类型定义（参见教材）





ADT Queue{

数据对象: $D=\{a_i \mid a_i \in \text{ElemSet}, i=1,2,\dots,n, n \geq 0\}$

数据关系: $R=\{ \langle a_i, a_{i+1} \rangle \mid a_i, a_{i+1} \in D, a_1 \text{为队头}, a_n \text{为队尾} \}$

基本操作:

InitQueue(&Q); 创建一个空的队列


Destroy(&Q); 销毁一个存在的队列

ClearQueue(&Q); 清空一个已经存在的队列

QueueEmpty(Q); 判断队列是否为空

//读长度, 读队头, 在队尾插入

}



❖ 队列的两种实现方式，即两种存储结构

❖ 顺序

❖ 链式

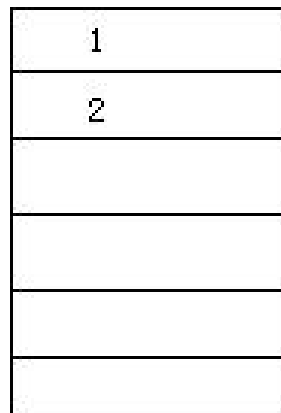
4.2 队列的顺序存储结构

- ❖ 用一组地址连续的存储单元依次存放自队头到队尾的数据元素
- ❖ 设两个变量`front`和`rear`分别指示队头元素和队尾元素的位置，这两个变量分别称为“队头指针”和“队尾指针”。
- ❖ 通常约定：初始化队列时，令`front=rear=0`，每当有新元素入队列时，队尾指针增1；每当删除队头元素时，队头指针增1。
- ❖ 在**非空队列**中，**队头指针**始终指向**队头元素**的当前位置，而**队尾指针**始终指向队尾**下一个空**的存储单元。



Front,rear

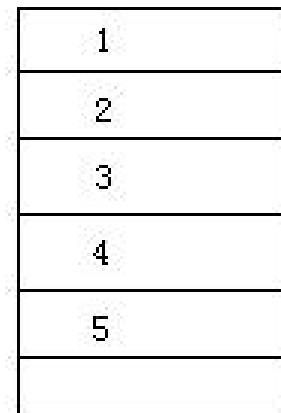
队列为空的情况



front

rear

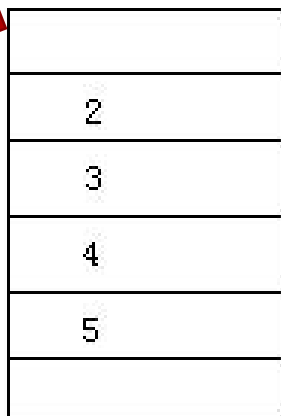
两个元素加对后的情况



front

rear

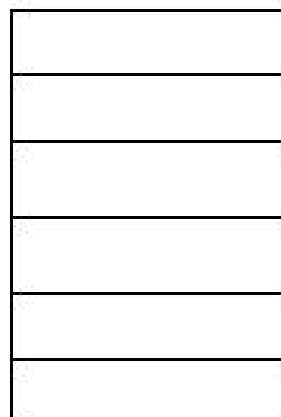
队列满的情况



front

rear

第一个元素出队



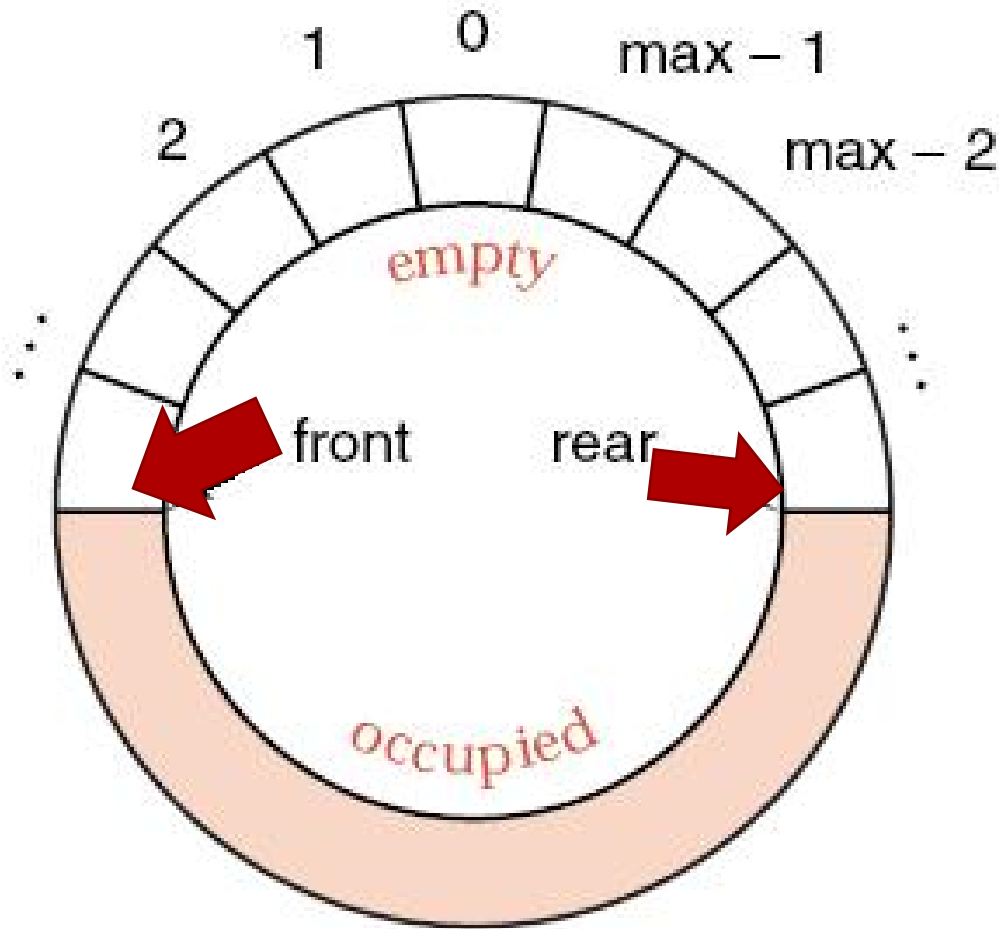
Front,rear

全部元素都出队

队尾指针始终指向队尾下一个空的存储单元

❖ “假溢出”现象：队列的实际可用空间未占满，但不能再进行入队列操作了。

解决假溢出的方法之一：将队列的数据区看成首尾相接的循环结构，形成一个环形的空间，称之为循环队列。（ max 为长度）



循环缓冲区

4.2.1 循环队列的类型定义

```
#define QueueSize 100 // 循环队列的初始分配空间
```

```
typedef struct
```

```
{
```

```
    ElemType data[QueueSize]; // 保存队列中元素
```

```
    int front;                // 队头指针
```

```
    int rear;                 // 队尾指针
```

```
} SqQueue;
```

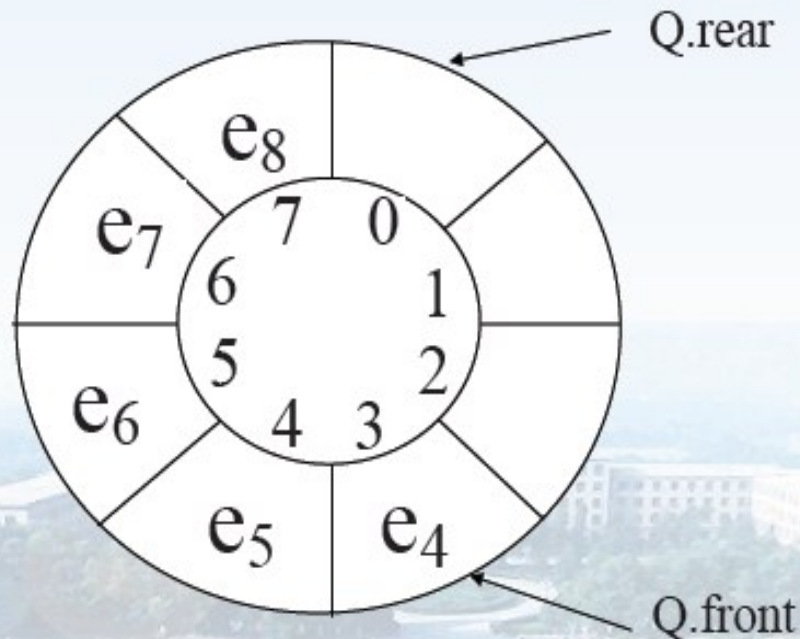
- ❖ 其中，QueueSize是指循环队列的初始分配空间，是循环队列的最大容量。数组data用于存储队列中的元素，front和rear分别为“队头指针”和“队尾指针”。

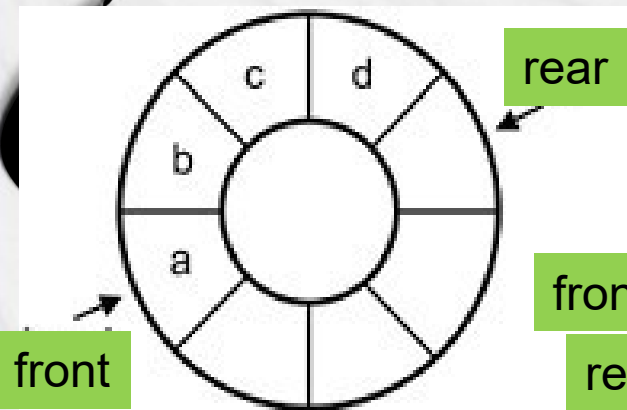
❖ 在循环队列Q中:

队头指针增1: $Q.front = (Q.front + 1) \% QueueSize$

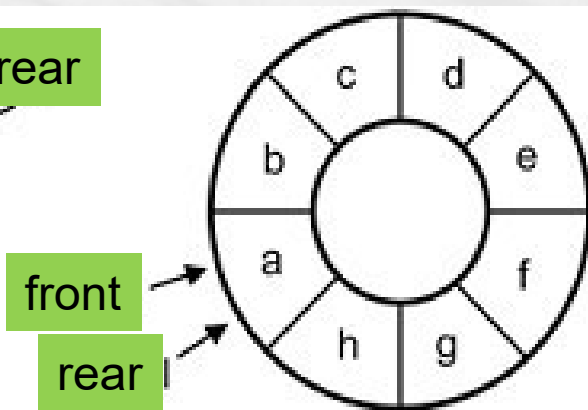
队尾指针增1: $Q.rear = (Q.rear + 1) \% QueueSize$

将队列空间看成是环形结构,
当 $Q.rear == MAXSIZE$ 时, 将其置为0,
即 $Q.rear = 0$; 或 $Q.rear = Q.rear \% MAXSIZE$;

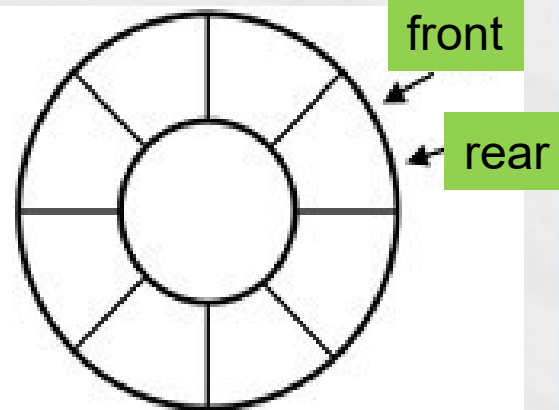




(a)一般情况



(b)队满



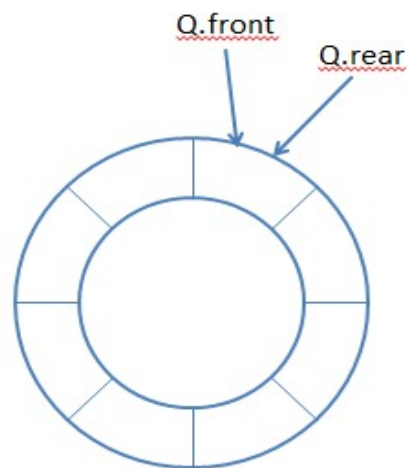
(c)队空

注意：如何判断队列空还是满？两个指针间的关系？

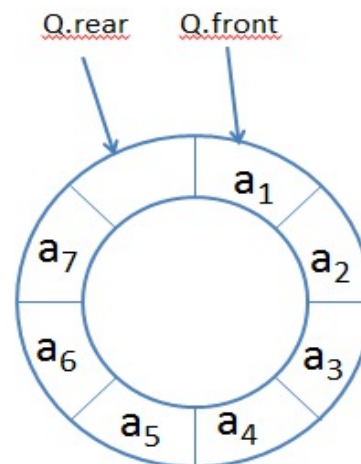
规定：留出一个空存储单元

Empty: $\text{front} == \text{rear}$

Full: $\text{front} == \text{rear} + 1$



(a) 空的循环队列



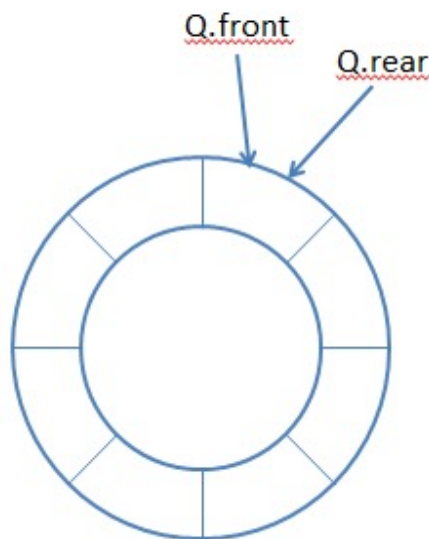
(b) 满的循环队列

为了使用 $\text{Q.rear} = \text{Q.front}$ 来区别是队空还是队满，我们常常认为出现左图时的情况即为队满的情况，此时： $\text{rear} + 1 = \text{front}$

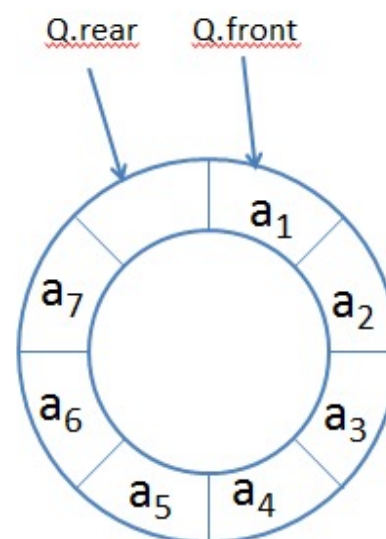
4.2.2 队列基本运算在循环队列上的实现

1. 初始化队列运算

```
void InitQueue(SqQueue *qu)
{
    qu->rear=qu->front=0;
}
```



(a) 空的循环队列



(b) 满的循环队列

为了使用 $Q.rear=Q.front$ 来区别是队空还是队满，我们常常认为出现左图时的情况即为队满的情况，此时： $rear+1=front$

2. 入队列运算

```
int EnQueue(SqQueue *qu, ElemType e)
```

```
{
```

```
if ((qu->rear+1)%QueueSize==qu->front)
```

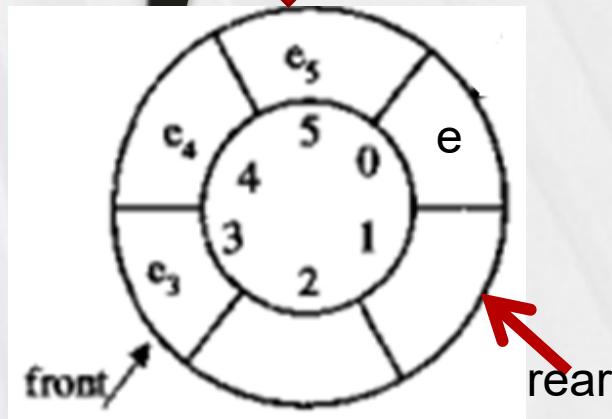
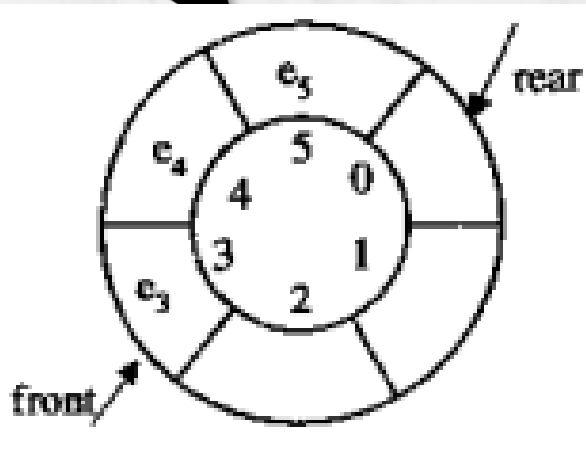
```
    return 0; //full
```

```
qu->data[qu->rear]=e;
```

```
qu->rear=(qu->rear+1)%QueueSize;
```

```
return 1;
```

```
}
```



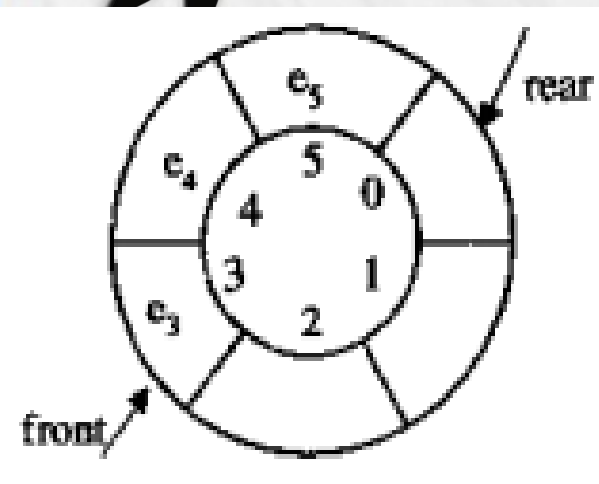
3. 出队列运算

```
int DeQueue(SqQueue *qu, ElemType *e)
{
    if (qu->rear==qu->front)
        return 0;

    *e=qu->data[qu->front];

    qu->front=(qu->front+1)%QueueSize;

    return 1;
}
```

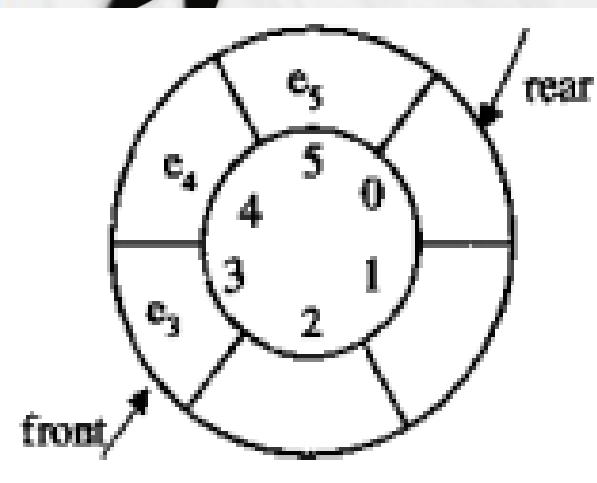


4. 取队头元素运算

```
int GetHead(SqQueue qu, ElemType *e)
{
    if (qu.rear==qu.front) return 0;

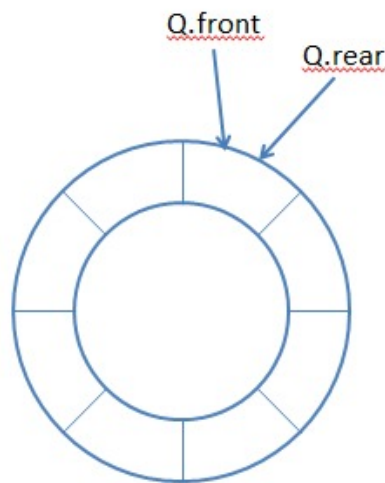
    *e=qu.data[qu.front];

    return 1;
}
```

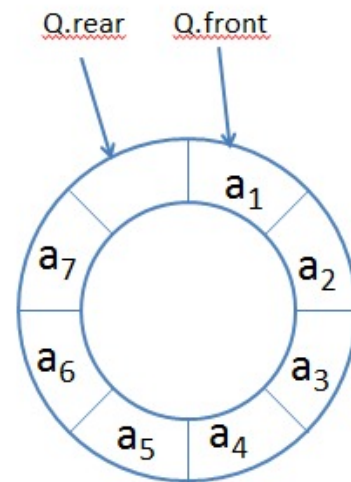


5. 判断队列空运算

```
int QueueEmpty(SqQueue qu)
{
    if (qu.rear==qu.front)
        return 1;
    else return 0;
}
```



(a) 空的循环队列



(b) 满的循环队列

为了使用 $Q.rear=Q.front$ 来区别是队空还是队满，我们常常认为出现左图时的情况即为队满的情况，此时： $rear+1=front$

4.2.3 循环队列的应用举例

【例1】 设从键盘输入一整数序列 a_1, a_2, \dots, a_n ，试编程实现：

当 $a_i > 0$ 时， a_i 进队；

当 $a_i < 0$ 时，将队首元素出队；

当 $a_i = 0$ 时，表示输入结束。

要求将队列处理成循环队列，并在异常情况时（如队满）打印出错信息。


- ❖ 解题思路：先建立一个循环队列，用while循环接收用户输入。若输入值大于0，将该数入队列；若小于0，则该数出队列并输出；若等于0，则退出循环。
- ❖ 具体算法

数据类型的定义：


```
#define QueueSize 100 // 循环队列的初始分配空间

typedef struct
{
    ElemType data[QueueSize]; // 保存队列中元素
    int front;                // 队头指针
    int rear;                 // 队尾指针
} SqQueue;
```

SqQueue qu;



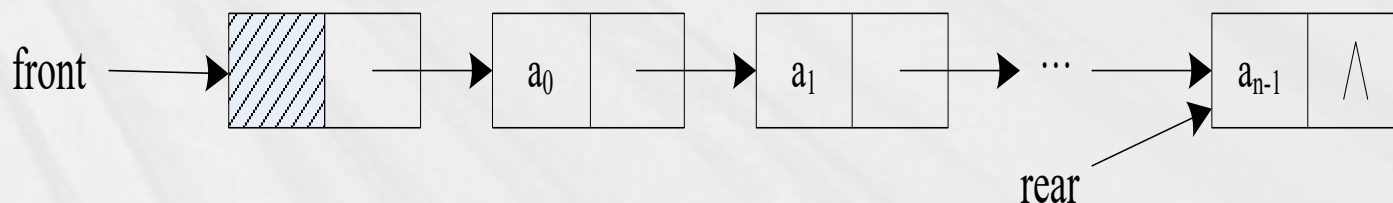
```
Void main( )  
{ int a, x;  
  SqQueue  qu;  
  qu.front=qu.rear=0;  
  scanf("%d", &a);  // input 0 means end
```



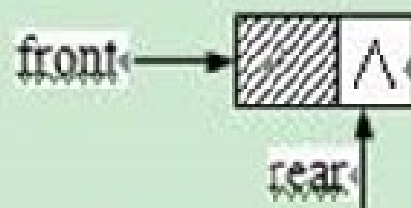
```
while(a!=0)
{ if(a>0)
    { if((qu.rear+1)%QueueSize ==qu.front)
        {printf("queue full"); return; }
      qu.data[qu.rear]=a;
      qu.rear=(qu.rear+1)%QueueSize;
    }
  else
    { if((qu.rear ==qu.front)
        {printf("queue empty"); return; }
      x=qu.data[qu.front];
      qu.front=(qu.front+1)%QueueSize;
      printf("%d 出队列", x);
    }
    scanf("%d",&a);
  }
printf("program end!\n");
}
```

4.3 队列的链式存储结构

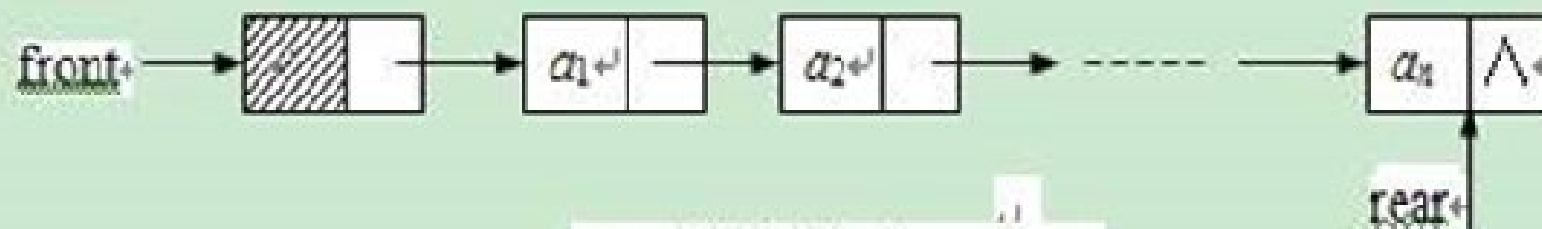
- ❖ 队列的链式存储结构称为**链队列**，它实际上是一个同时带有首指针和尾指针的**单链表**，首指针指向队头结点，尾指针指向队尾结点，如下图所示。



- ◆ 链队列的插入操作只能在队尾进行，删除操作只能在队头进行。



(a) 空链队列



(b) 非空链队列

图 3-10 链队列示意图

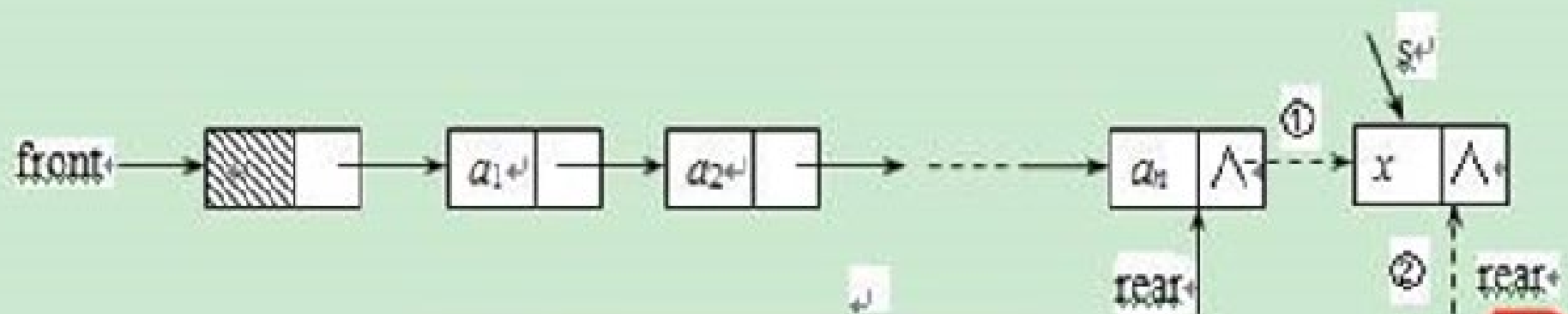


图 3-11 链队列入队操作示意图

4.3.1链队列的类型定义

typedef struct Qnode //定义链队列的**结点结构**

```
{ ElemType data;  
    struct Qnode *next;  
} QNode, *QueuePtr;
```

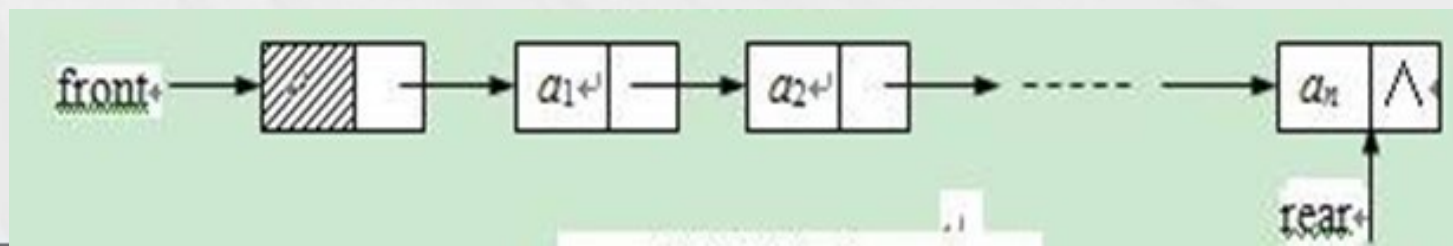


typedef struct { //定义队列

QueuePtr front; //队头指针

QueuePtr rear; //队尾指针

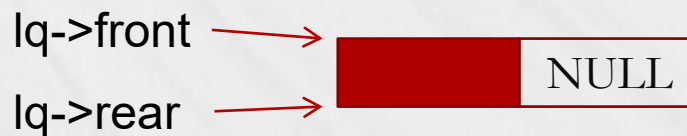
} LinkQueue; //链队列类型



4.3.2 队列基本运算在链队列上的实现

1. 初始化队列运算---建立一个空队列

```
void InitQueue(LinkQueue *lq )  
{  
    lq->front = lq->rear = (QueuePtr)malloc(sizeof(QNode));  
    lq->front->next=NULL;  
}
```



2. 入队列运算

会溢出吗？

```
void EnQueue(LinkQueue *lq, ElemType e)
```

```
{ QueuePtr p;
```

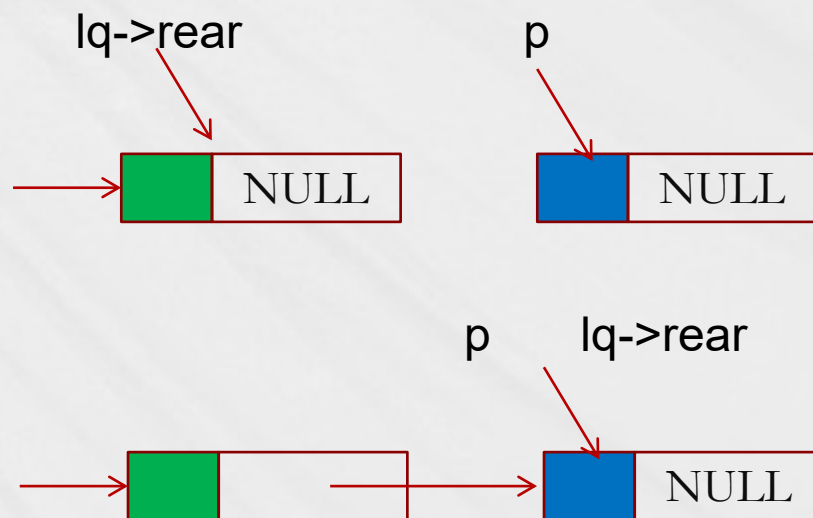
```
  p= (QueuePtr)malloc(sizeof(QNode));
```

```
  p->data=e; p->next=NULL;
```

```
  lq->rear->next=p;
```

```
  lq->rear=p;
```

```
}
```



3. 出队列运算，删除结点

```
int DeQueue(LinkQueue *lq, ElemType *e)
```

```
{ QueuePtr p;
```

```
  if (lq->rear==lq->front)    return 0;
```

```
  p=lq->front->next;
```

```
  *e=p->data;
```

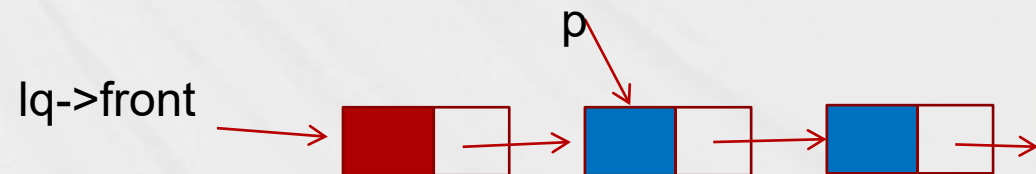
```
  lq->front->next=p->next;
```

```
  if(lq->rear==p) lq->rear=lq->front;  //仅有一个节点
```

```
  free(p);
```

```
  return 1;
```

```
}
```



4. 取队头结点中的数据-----不删除

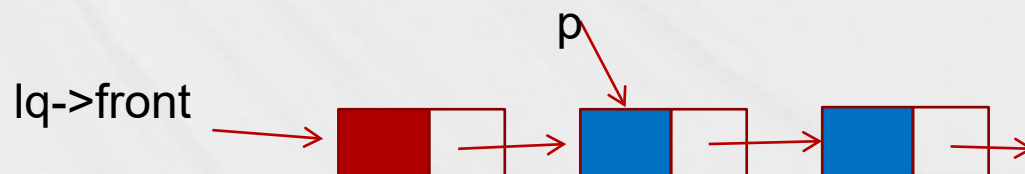
```
int GetHead(LinkQueue lq, ElemType *e)
{
    QueuePtr p;

    if (lq.rear==lq.front) return 0;

    p=lq.front->next;

    *e=p->data;

    return 1;
}
```



5. 判断队列空运算

```
int QueueEmpty(LinkQueue lq)
{
    if (lq.rear==lq.front) return 1;
    else return 0;
}
```

4.3.3 链队列的应用举例

【例2】编写一个程序，反映病人到医院看病、排队看医生的情况。在病人排队过程中，主要发生两件事：

- (1) 病人排队
- (2) 病人就诊（先来先服务）

要求程序采用菜单方式，其选项及功能说明如下：

- (1) 排队：输入排队病人姓名，加入到病人排队队列中；
- (2) 就诊：病人排队队列中最前面的病人就诊，并将其从队列中删除；
- (3) 查看排队：从队首到队尾列出所有的排队病人；
- (4) 下班：退出运行，排队非空要通知。

❖ 解题思路:

定义链队列的类型；根据题意将ElemType设为char型；在程序中根据功能要求进行插入、删除和输出操作，并使用switch语句实现菜单的选择。

```
typedef char Elemtype  
  
typedef struct Qnode  
{ Elemtype data[15];      //记录患者名字  
  
  struct Qnode  *next; } QNode, *Queueptr;  
  
  
typedef struct { Queueptr front;  
  
                Queueptr rear; } Linkqueue;
```



```
void main()
```

```
{ int choice, flag=1;
```

```
    Linkqueue lq;
```

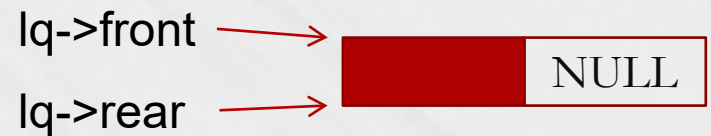
```
    QNode *s, *p;
```

```
    char name[15];
```

```
    lq.front=(Queueptr)malloc(sizeof(QNode)); // create a queue
```

```
    lq.front->next=NULL;
```

```
    lq.rear=lq.front;
```



```
while(flag==1)
```

```
{ printf("1: 排队 2: 看医生 3: 查看排队 0: 下班\n");
```

```
printf("请选择: ") ;      scanf("%d", &choice);
```

```
switch(choice) {
```

```
    case 0: if(lq.front !=lq.rear) printf("work is over");
```

```
        flag=0; break;
```

```
    case 1: printf("input name:"); scanf("%s", name)
```

```
        s=(Queueptr) malloc(sizeof(QNode));
```

```
        strcpy(s->data, name); s->next=NULL;
```

```
        lq.rear->next=s; lq.rear=s; break;
```



case 2: if(lq.front==lq.rear) printf("queue is empty!");

else { s=lq.front->next;

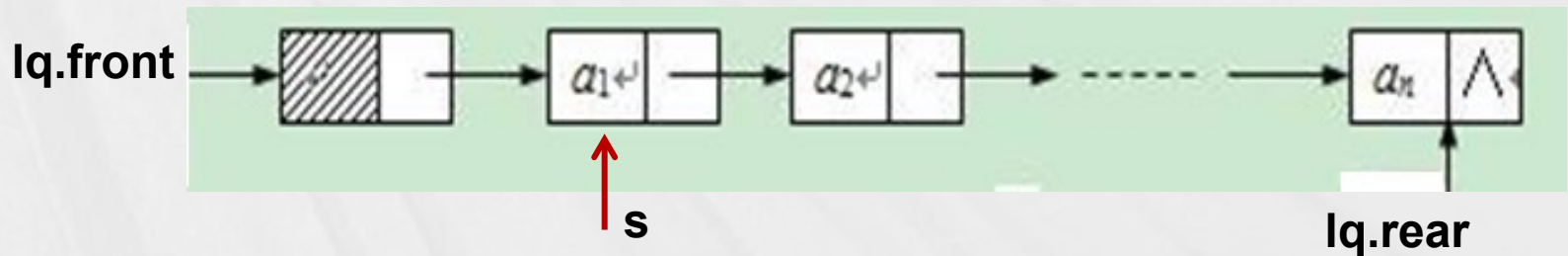
lq.front->next=s->next;

if(lq.rear==s) lq.rear=lq.front; //仅一个元素，输出后队列为空

printf("%s 看医生", s->data);

free(s); }

break;



case 3: //查看队列

```
if(lq.front==lq.rear) ) printf("queue is empty!");
```

```
else {
```

```
p=lq.front->next;
```

```
printf("排队的患者\n");
```

```
while(p!=NULL)
```

```
{ printf( "%s\t" , p->data);
```

```
    p=p->next; }
```

```
}
```

```
break;
```

```
}
```

```
}
```



本章小结

- ❖ **队列**也是一种**操作受限的线性表**。它只允许在表尾进行插入而在表头进行删除操作。
- ❖ 队列的结构特点是**先进先出**，在许多实际问题的解决中和系统软件的设计中，都会用到队列。
- ❖ **队列**也有**顺序存储结构和链式存储结构**。
- ❖ 为了解决**假溢出**的问题，通常将队列的**顺序存储结构**数据区看成首尾相接的循环结构，形成一个环形的空间，称之为**循环队列**。
- ❖ 队列的链式存储结构称为**链队列**，是一个同时带有首指针和尾指针的单链表，其各种操作的实现也类似于单链表。

本章习题

1. 循环队列有什么优点？如何判断它的空和满？
2. 对于一个具有 Q_{size} 个单元的循环队列，写出求队列中元素个数的公式。
3. 假设以一维数组 $sq[m]$ 存储循环队列的元素，同时以 $rear$ 和 $length$ 分别指示循环队列中的队尾位置和队列中所含元素的个数。试给出该循环队列的队空条件和队满条件，并写出相应的入队列和出队列的算法。
4. 假设以带头结点的循环链表表示一个队列，并且只设一个队尾指针指向队尾元素结点（注意不设头指针），试写出相应的初始化、入队列和出队列的算法。