数据结构

（C语言版）（第2版）

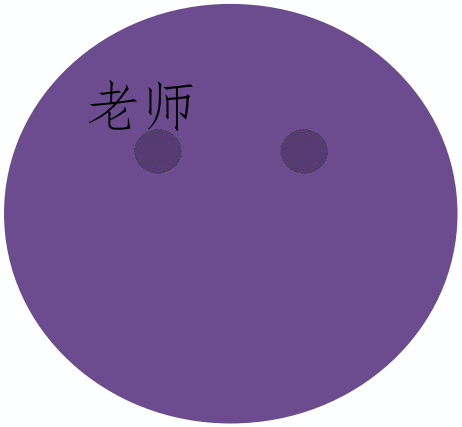
栈和队列

队列的表示和操作

主讲教师：汪红松

教 学 内 容 Contents

一、循环队列
二、链队列



队列的抽象数据类型

ADT Queue {

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

数据关系: $R_1=\{<a_{i-1},a_i> \mid a_{i-1},a_i \in D, i=1,2,\cdots,n\}$

约定a_1端为队列头,a_n端为队列尾

基本操作:

(1) InitQueue (&Q) //构造空队列

(2) DestroyQueue (&Q) //销毁队列

(3) ClearQueue (&S) //清空队列

(4) QueueEmpty(S) //判空. 空--TRUE,

队列的抽象数据类型

(5) QueueLength(Q)　　//取队列长度

(6) GetHead (Q,&e)　　//取队头元素,

(7) EnQueue (&Q,e)　　//入队列

(8) DeQueue (&Q,&e)　　//出队列

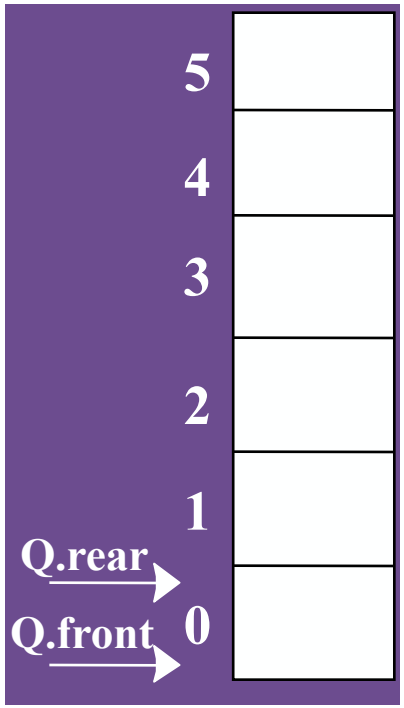
(9) QueueTraverse(Q,visit())　　//遍历

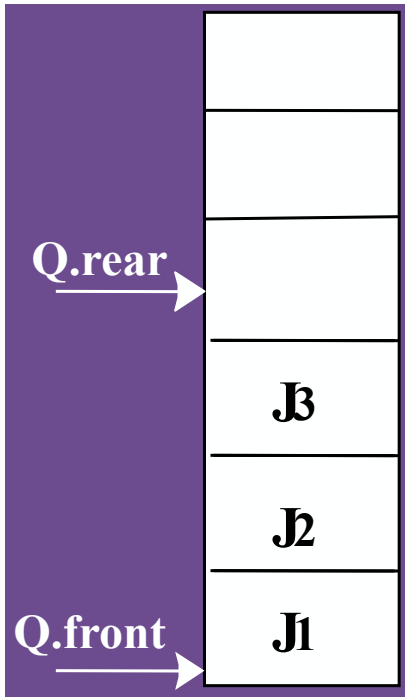
}ADT Queue

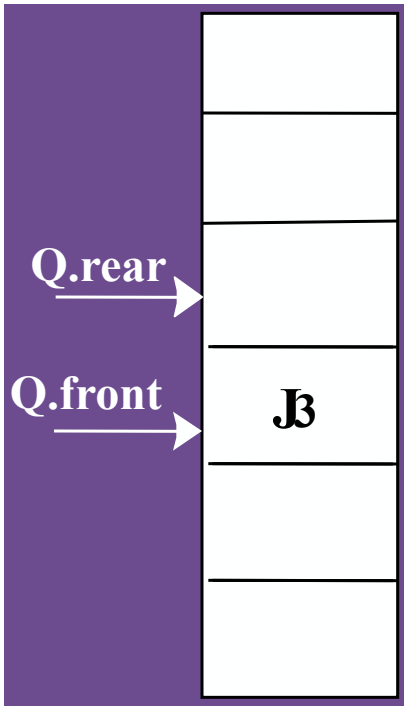
队列的顺序表示 - - 用一维数组base[M]

```
#define M  100   //最大队列长度
Typedef struct {
   QElemType *base;  //初始化的动态分配存储空间
   int  front;            //头指针
   int  rear;             //尾指针
}SqQueue;
```

队列的顺序表示 - - 用一维数组base[M]







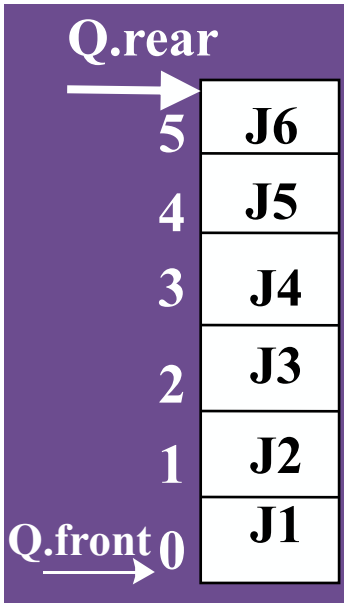
空队标志：front= =rear

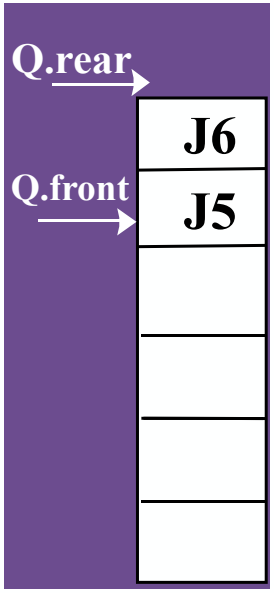
入队：base[rear++]=x;

出队：x=base[front++];

存在的问题

设大小为M





front = 0

rear = M 时

再入队—真溢出

front > 0

rear = M 时

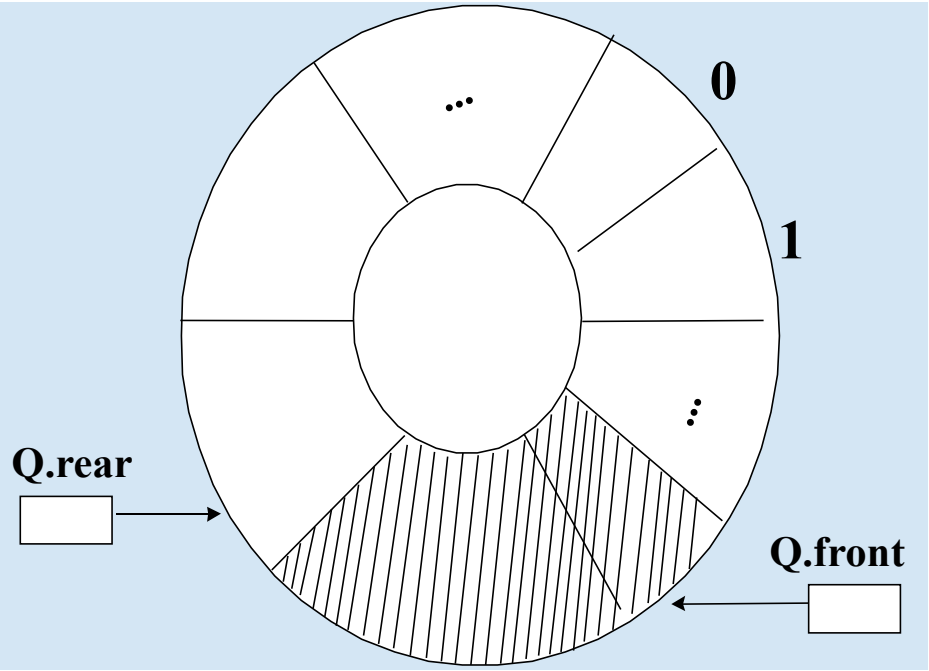
再入队—假溢出

解决的方法——循环队列

base[0]接在base[M-1]之后

若rear+1==M

则令rear=0;



实现：利用“模”运算

入队：

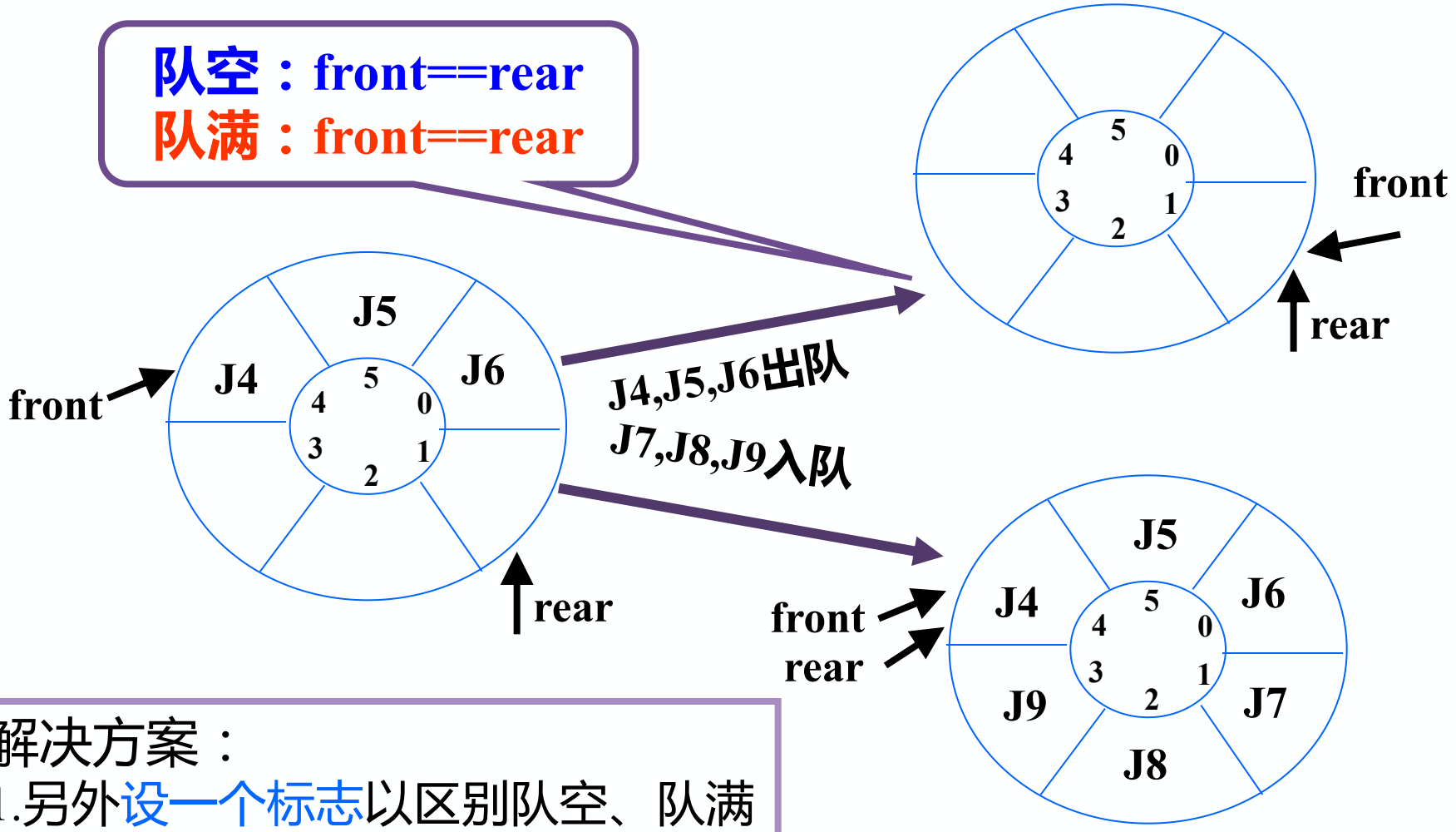
```
base[rear]=x;
rear=(rear+1)%M;
```

出队：

```
x=base[front];
front=(front+1)%M;
```

解决的方法——循环队列

队空：front==rear
队满：front==rear



解决方案：

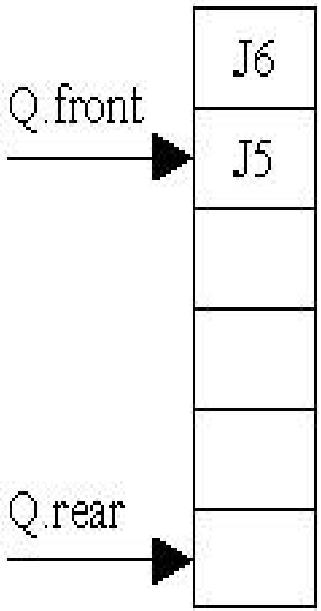
1.另外设一个标志以区别队空、队满

2.少用一个元素空间：

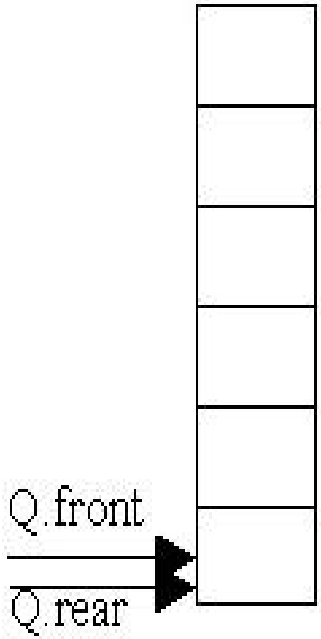
队空：front==rear

队满：(rear+1)%M==front

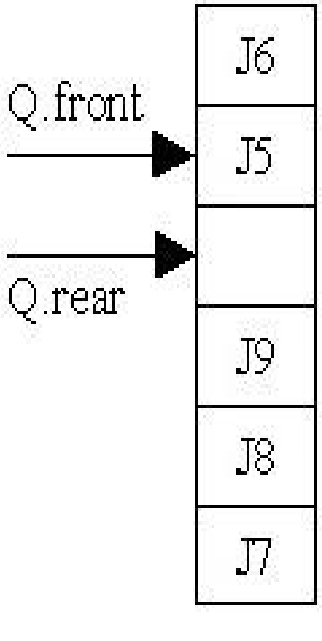
一、循环队列



一般情况



队空：front==rear



队满：(rear+1)%M==front

一、循环队列

1.循环队列初始化

```
Status InitQueue (SqQueue &Q){
    Q.base =new QElemType[MAXQSIZE]
   if(!Q.base) exit(OVERFLOW);
    Q.front=Q.rear=0;
    return OK;
}
```

一、循环队列

2.求循环队列的长度

```
int  QueueLength (SqQueue Q){
   return (Q.rear-Q.front+MAXQSIZE)%MAXQSIZE;
 }
```

一、循环队列

3.循环队列入队

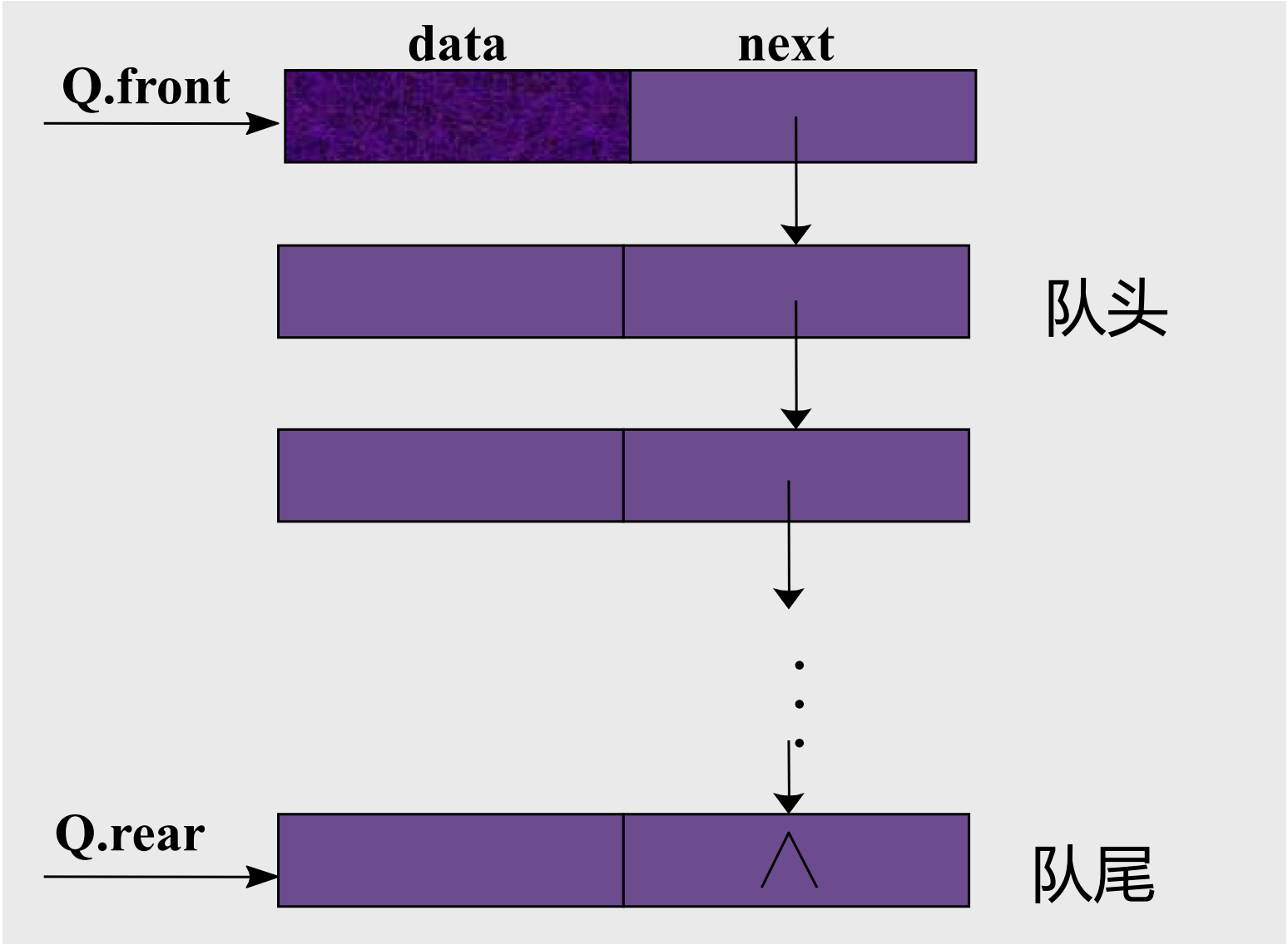
```
Status EnQueue(SqQueue &Q,QElemType e){
    if((Q.rear+1)%MAXQSIZE==Q.front)  return ERROR;
    Q.base[Q.rear]=e;
    Q.rear=(Q.rear+1)%MAXQSIZE;
     return OK;
}
```

一、循环队列

4.循环队列出队

```
Status DeQueue (LinkQueue &Q,QElemType &e){
   if(Q.front==Q.rear) return ERROR;
   e=Q.base[Q.front];
   Q.front=(Q.front+1)%MAXQSIZE;
   return OK;
}
```

二、链队列



二、链队列

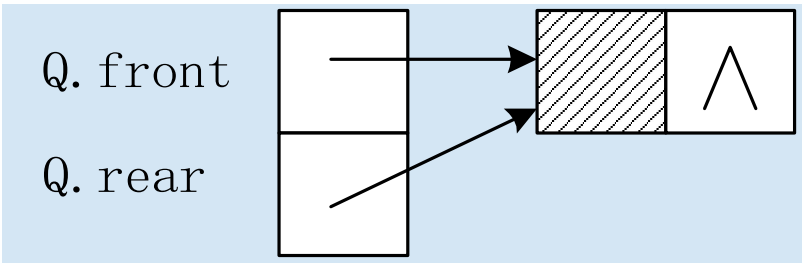
```
typedef struct QNode{
   QElemType   data;
   struct Qnode  *next;
}Qnode, *QueuePtr;

typedef struct {
   QueuePtr  front;          //队头指针
   QueuePtr  rear;           //队尾指针
}LinkQueue;
```

二、链队列

1.链队列初始化

```
Status InitQueue (LinkQueue &Q){
   Q.front=Q.rear=(QueuePtr) malloc(sizeof(QNode));
    if(!Q.front) exit(OVERFLOW);
    Q.front->next=NULL;
     return OK;
}
```



二、链队列

2.销毁链队列

```
Status DestroyQueue (LinkQueue &Q){
  while(Q.front){
    Q.rear=Q.front->next;
    free(Q.front);
    Q.front=Q.rear;   }
  return OK;
}
```

二、链队列

3.判断链队列是否为空

```
Status QueueEmpty (LinkQueue Q)
{
   return (Q.front==Q.rear);
}
```

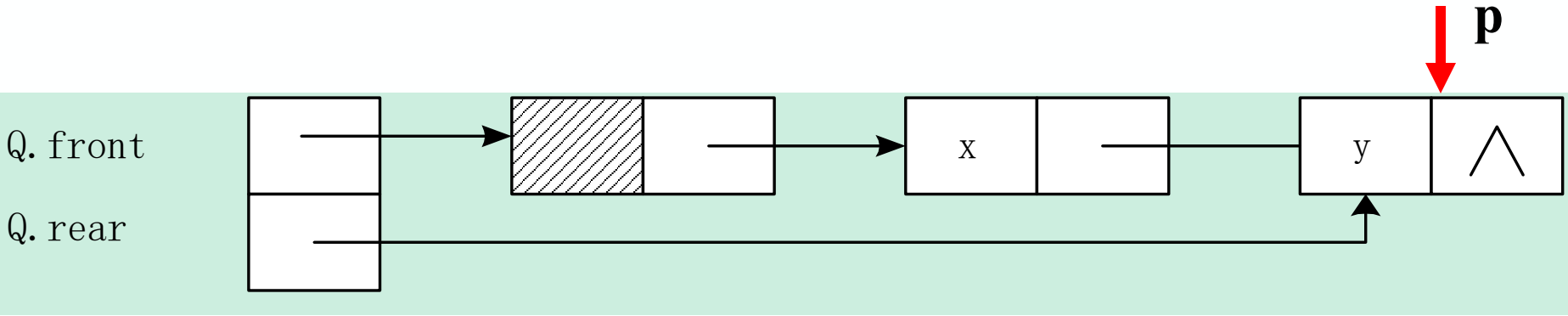
4.求链队列的队头元素

```
Status GetHead (LinkQueue Q, QElemType &e){
   if(Q.front==Q.rear) return ERROR;
   e=Q.front->next->data;
   return OK;
}
```

二、链队列

5.链队列入队

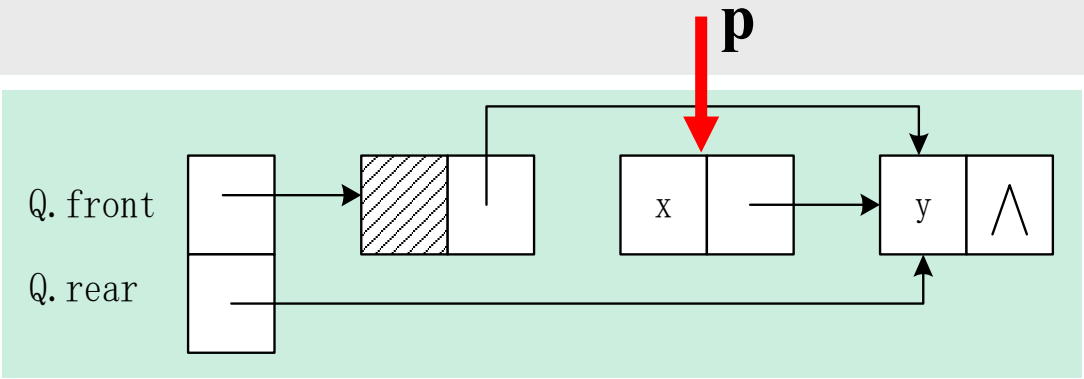
```
Status EnQueue(LinkQueue &Q,QElemType e){
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p) exit(OVERFLOW);
    p->data=e; p->next=NULL;
    Q.rear->next=p;
    Q.rear=p;
    return OK;
}
```



二、链队列

6.链队列出队

```
Status DeQueue (LinkQueue &Q,QElemType &e){
  if(Q.front==Q.rear) return ERROR;
  p=Q.front->next;
  e=p->data;
  Q.front->next=p->next;
  if(Q.rear==p) Q.rear=Q.front;
  delete p;
  return OK;
}
```



小结

1. 队列的表示和操作，包括队列的顺序表示和列链式表示
2. 两种队列的初始化、判断队列是否为空或是否为满、求队列长度、入队和出队等操作