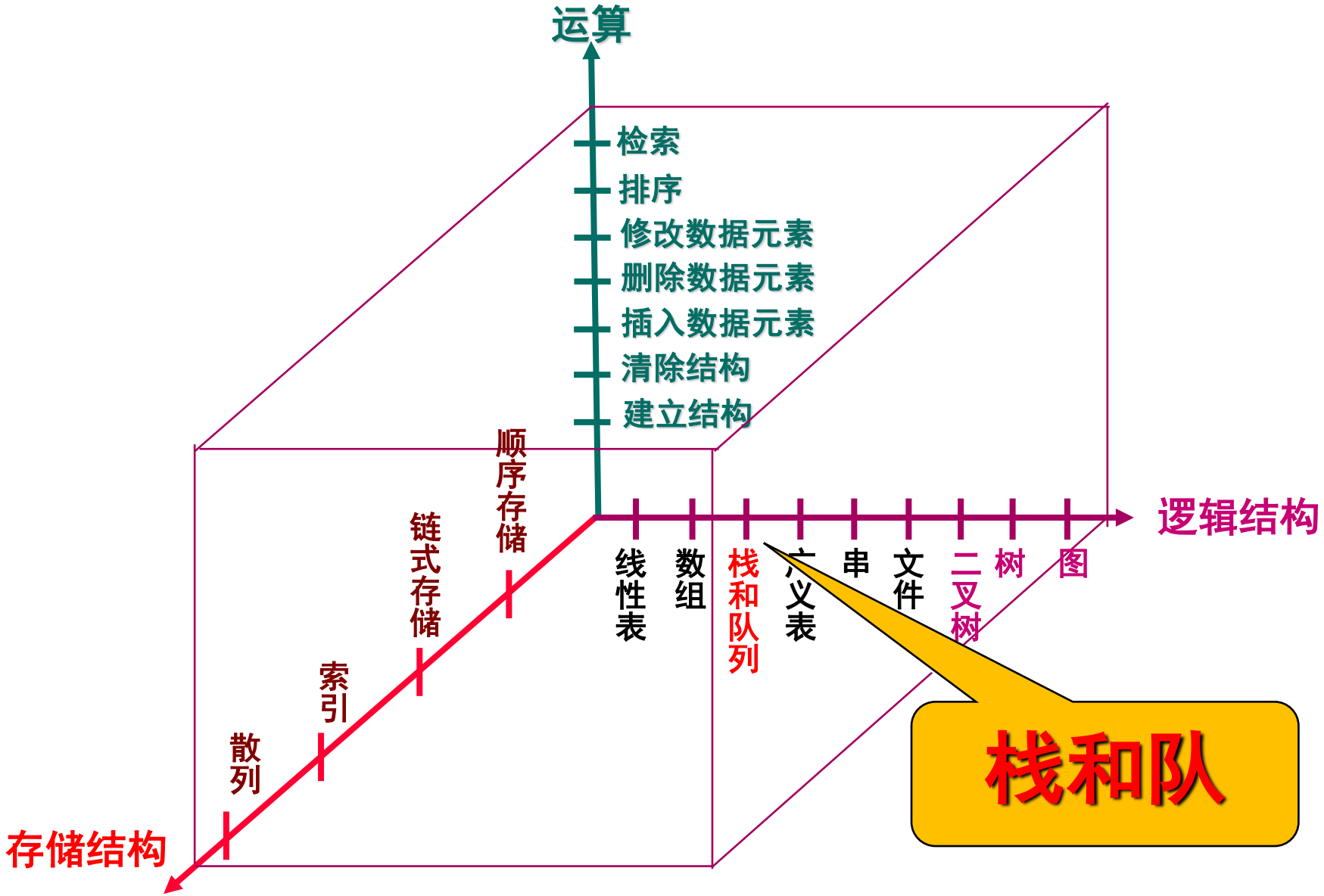
数据结构与程序设计
(Data Structure and Programming)

栈与队
(Stack and Queue)

北航计算机学院 林学练

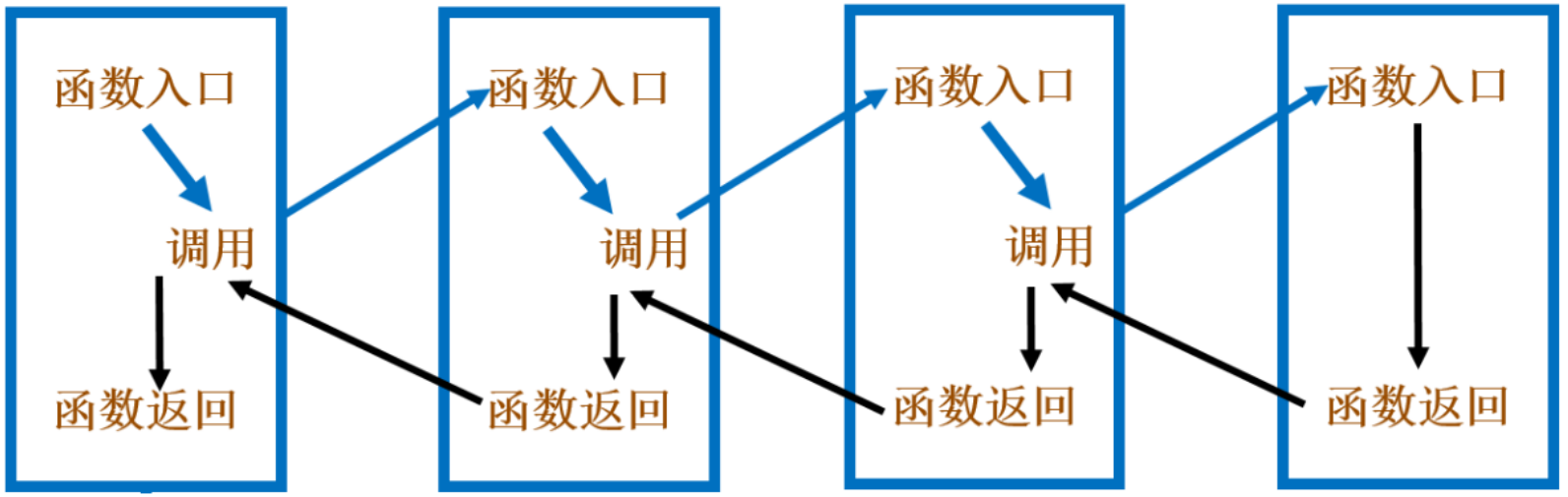
数据结构的基本问题空间



例1：


HP LaserJet P2055dn [374B33] - 脱机
打印机(P) 文档(D) 查看(V)
文档名 状态 所有者 页数 大小 提交时间
http://news.sina.com.cn/china/ YHH 2 1.25 MB 18:00:14 2015/9/22
大话数据结构.pdf YHH 1 155 KB 18:01:20 2015/9/22
Microsoft PowerPoint - DS第... YHH 64 6.37 MB 17:59:27 2015/9/22
Microsoft PowerPoint - DS第... YHH 64 6.37 MB 17:59:10 2015/9/22
队列中有 4 个文档

例2：


函数入口
调用
函数返回
函数入口
调用
函数返回
函数入口
调用
函数返回
函数入口
函数返回

在计算机领域

程序设计

深度优先搜索/回溯法

函数调用、递归程序的执行过程

编译程序

变量的存储空间的分配

表达式的翻译与求值计算

栈

操作系统

作业调度、进程调度

后续章节

二叉树的层次遍历……

队

本章内容

3.1 栈的基本概念

3.2 栈的顺序存储结构

3.3 栈的链式存储结构

3.4 队的基本概念

3.5 队的顺序存储结构

3.6 队的链式存储结构

3.4 队(Queue)的基本概念

(一) 队的定义

先进先出

队列 简称**队**，是一种只允许在表的一端进行插入操作，而在表的另一端进行删除操作的线性表。

允许插入的一端称为**队尾**，其位置由rear指出；

允许删除的一端称为**队头**，其位置由front指出。

(二) 队的基本操作

1. 队的插入 (进队、入队) √
2. 队的删除 (出队、退队) √
3. 测试队是否为空 √
4. 检索当前队头元素
5. 创建一个空队

特殊性

1. 其操作仅是一般线性表的操作的一个子集。
2. 插入和删除操作的位置受到限制。

队列的基本操作

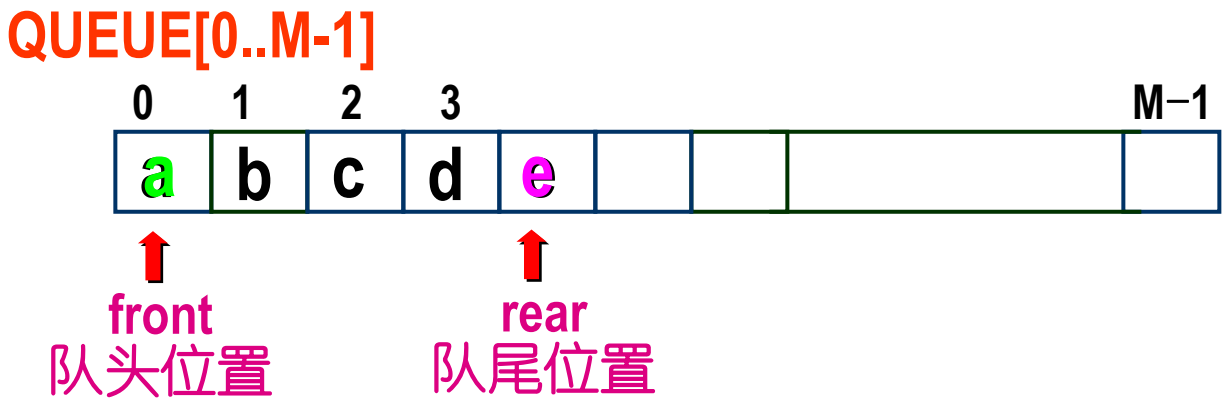
- void enQueue(Queue q, ElemType); //元素进队，即在队尾插入一个元素
- ElemType deQueue(Queue q);　　//元素出队，即队头删除一个元素

- isFull(Queue q);　//测试队列是否已满
- isEmpty(Queue q); //测试队列是否为空

3.5 队的顺序存储结构

(一) 构造原理

在实际程序设计过程中，通常借助一个一维数组QUEUE[0..M–1]来描述队的顺序存储结构，同时，设置两个变量 front与rear分别指出当前队头元素与队尾元素的位置。

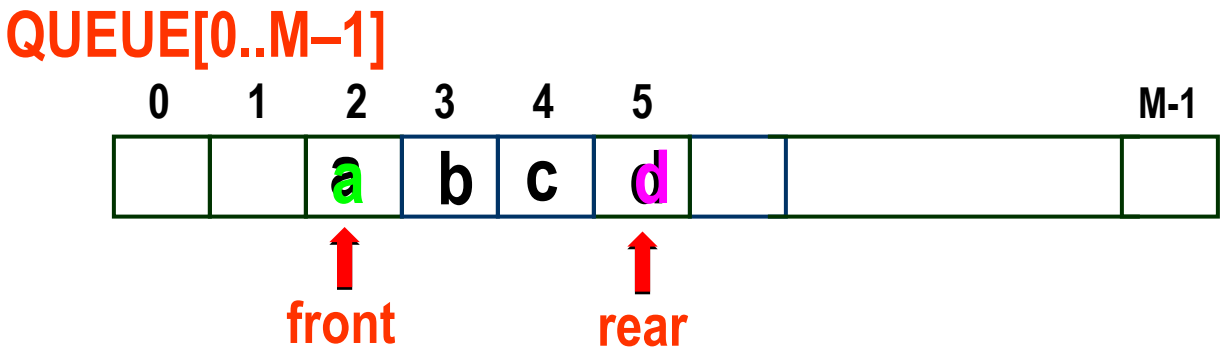


约定 **当队列中有元素时：**

rear 指出实际队尾元素所在的位置，

front 指出实际队头元素所在位置，

(count 指出实际队中元素个数)



测试队为空的条件是

rear< front （或 *count==0*）

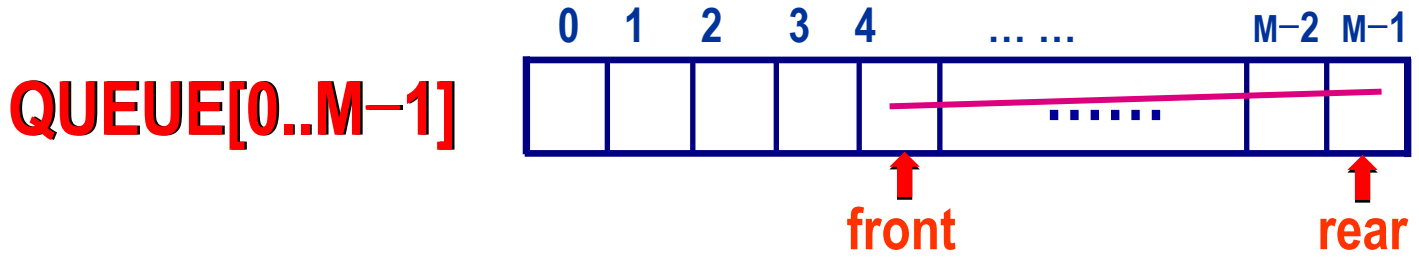
初始时，队为空，令

front= 0 rear= －1 （*count=0*）

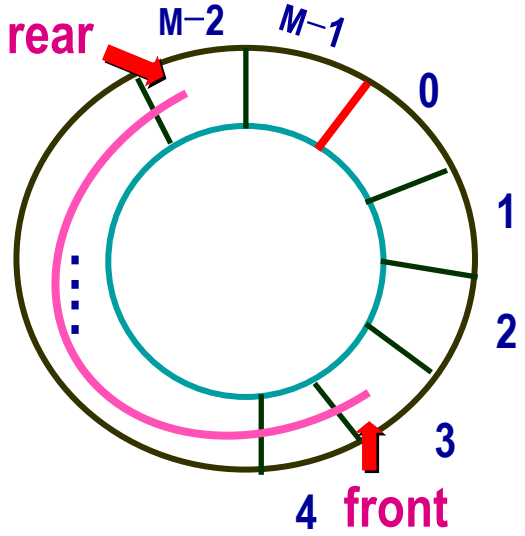
注意队为空时front的位置

(二)循环队列

实际应用中，由于队元素需要频繁的进出，上述结构很容易造成溢出，即rear到达数组尾，而实际队中元素并没有超出数组大小。



把队列(数组)设想成头尾相连的循环表，使得数组前部由于删除操作而导致的无用空间尽可能得到重复利用，这样的队列称为**循环队列** 。



类型定义

```
#define MAXSIZE    1000
QElemType  QUEUE[MAXSIZE];
int  front, rear, count;
```

由于变量front和rear需要在多个操作（函数）间共享，为了方便操作，在此将其设为全局变量。
count为队列中元素个数。

初始时，三个变量为：

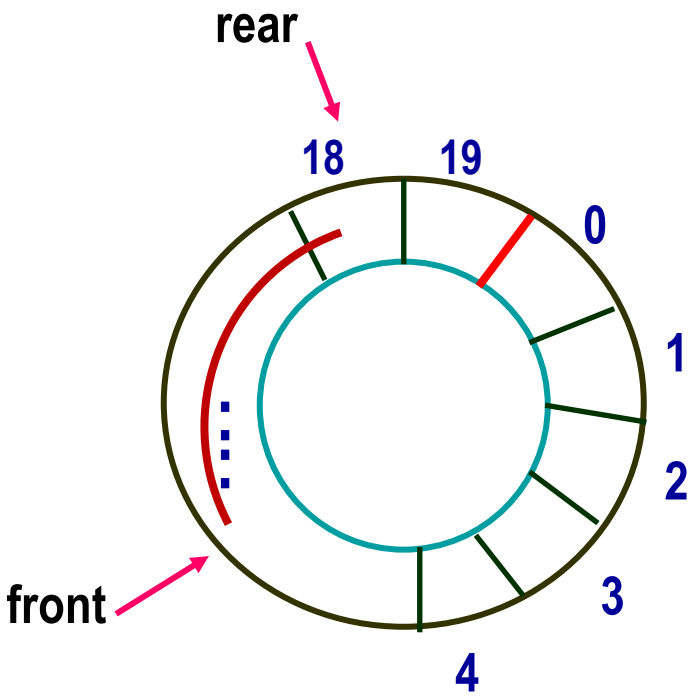
```
front = 0;
rear = MAXSIZE - 1;
count = 0;
```

练习

若用一个大小为20的数组（下标从0开始）来实现循环队列，且当前rear和front的值分别为18和10，当从队列中出队2个元素，再入队5个元素后，rear和front的值分别为 _____ 。

A. 3和12　　B. 12和3

C. 0和15　　D. 15和0



（三）（循环队列）基本算法

1. 初始化队列

```
void  initQueue(  ) {
        front = 0;
        rear = MAXSIZE-1;
        count = 0;
}
```

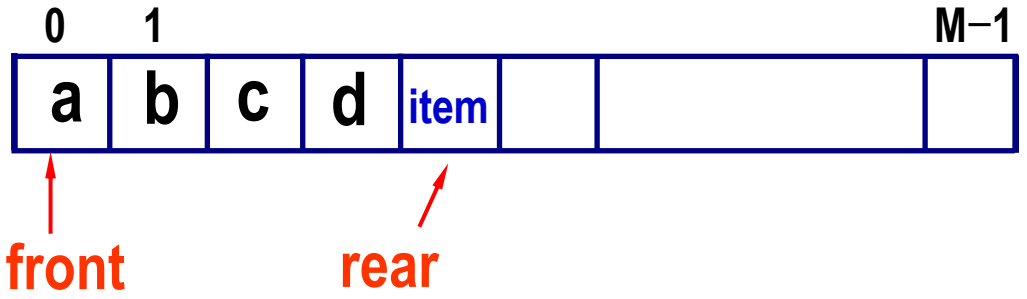
2、测试队列是否为空或满

```
int  isEmpty( ){
        return count == 0;
}
```

```
int  isFull( ){
        return count == MAXSIZE;
}
```

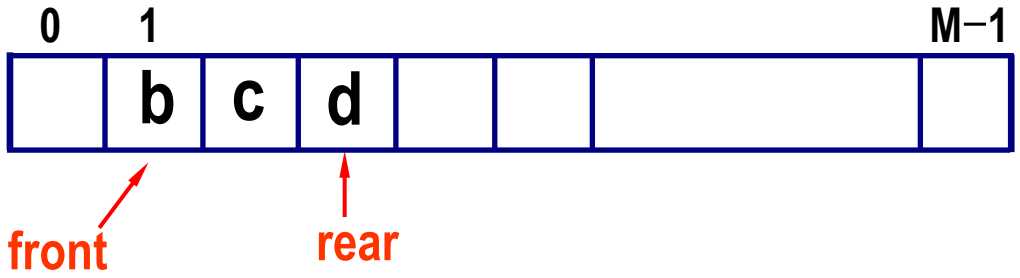
队空或满，返回1，
否则，返回0。

3. 插入(进队)算法




```
void enQueue(ElemType queue[ ], ElemType item){
    if(isFull())                          /* 队满，插入失败 */
        Error("Full queue!");
    else{                                 /* 队未满，插入数据 */
        rear = (rear+1) % MAXSIZE;
        queue[rear]=item;
        count++;
    }
}
```

4. 删除(出队)算法




```
ElemType  deQueue(ElemType queue[ ]){
   ElemType e;
     if(isEmpty())
          Error("Empty queue!");    /* 队空，删除失败 */
     else{
          e=queue[Front];
          front = (front+1)%MAXSIZE;
          count--;                  /* 队非空，删除成功 */
          return e;
     }
}
```

3.6 队列的链式存储结构

链接队列
链队

(一) 构造原理

队列的链式存储结构是用一个线性链表表示一个队列，指针front与rear分别指向实际队头元素与实际队尾元素所在的链结点。

约定

~~rear指出实际队尾元素所在的位置；~~

~~front指出实际队头元素所在位置。~~

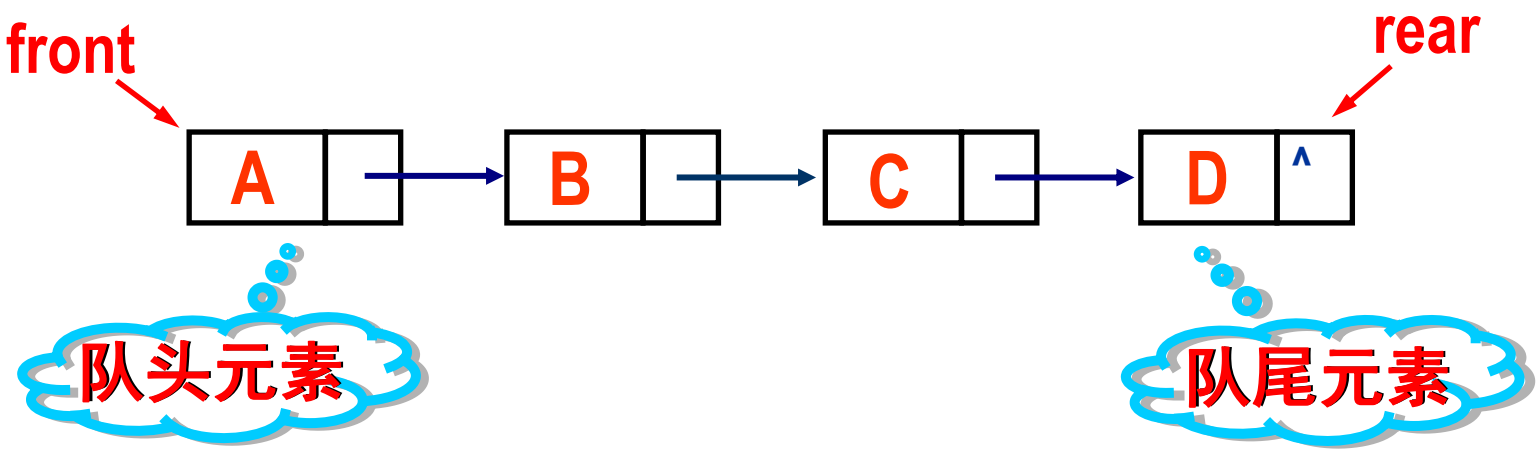
front与rear指针分别指向实际队头和队尾元素

例

在一个初始为空的链接队列中依次插入数据元素

A, B, C, D

以后，队列的状态为



空队对应的链表为空链表，空队的标志是

front == NULL

类型定义

```
struct  node {
        ElmeType   data;
        struct  node   *link;
}
typedef struct node   QNode;

typedef struct node *QNodeptr;
```

队头及队尾指针front和rear定义如下：

```
QNodeptr front, rear;
```

为了操作方便，经常将它们定义为全局变量

(二) 基本算法

1. 初始化队列

```
void  initQueue(){
        front=NULL;
        rear=NULL;
}
```

2. 测试队列是否为空

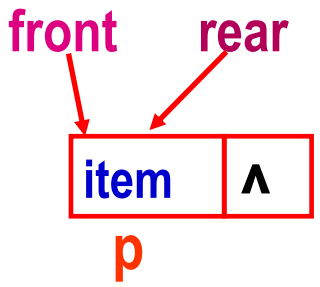
```
int  isEmpty(){
        return front==NULL;
}
```

队空,返回1
否则,返回0

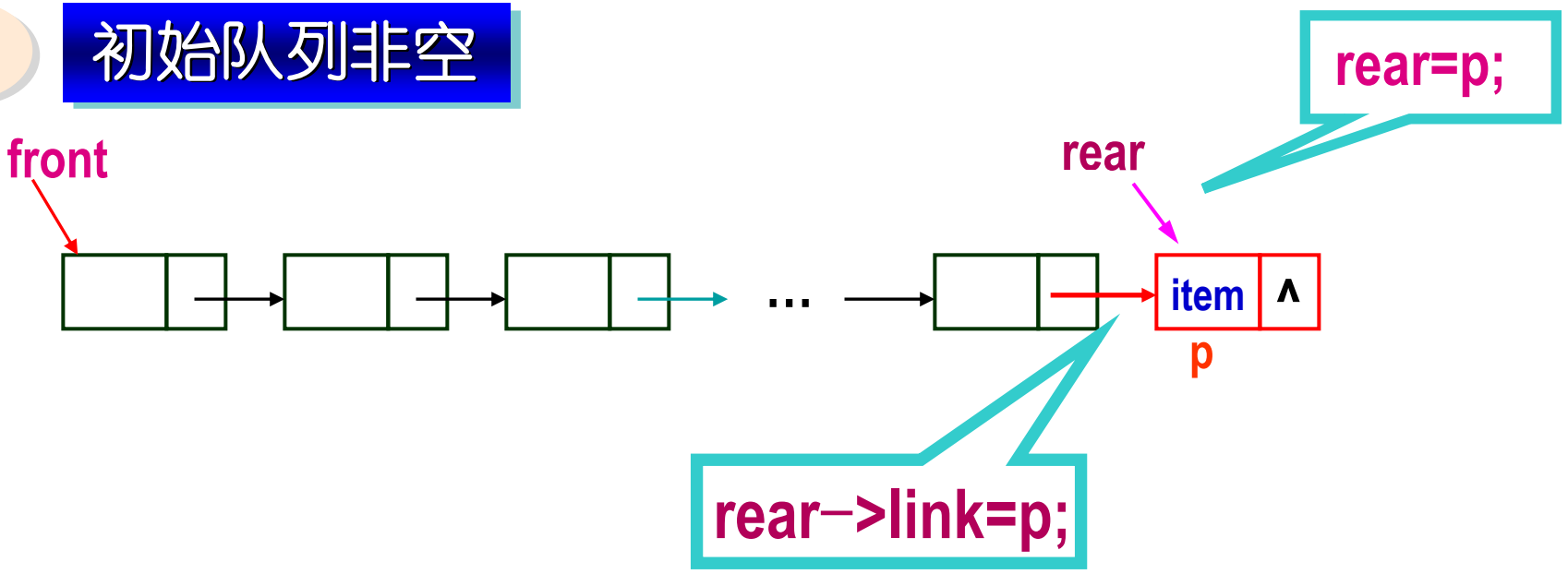
3. 插入(进队)

分两种情况

1 初始队列为空　front=rear=NULL

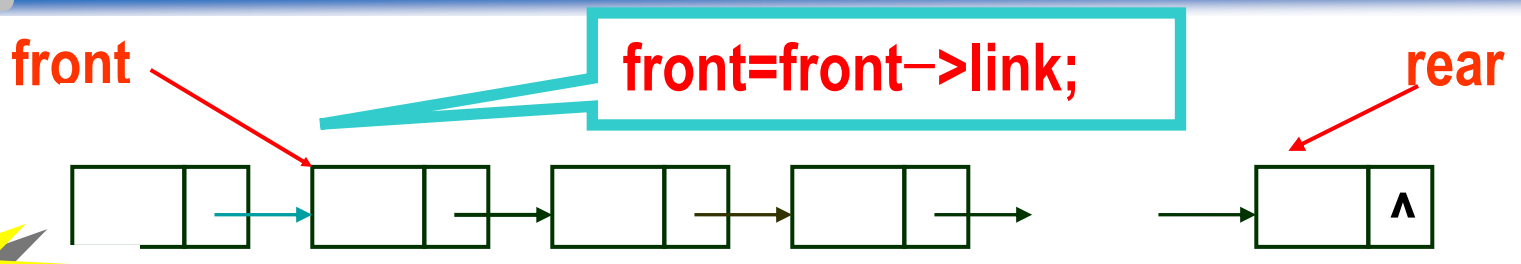


2 初始队列非空



```
void  enLQueue(ElemType  item ){
     QNodeptr  p;
      if((p=(QNodeptr)malloc(sizeof(QNode))) ==NULL)
           Error("No memory! ");
      p->data=item;
      p->link=NULL;
      if(front==NULL)
           front=p;                    /* 插入空队的情况 */
      else
           rear->link=p;
      rear=p;                          /* 插入非空队的情况 */
}
```

4. 删除(出队)

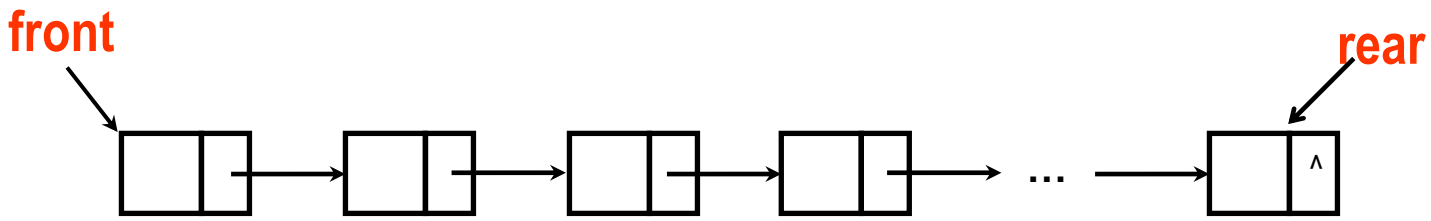


算法

```
ElemType  deLQueue( ){
      QNodeptr  p;
      ElemType item;
      if(isEmpty() )
          Error("Empty queue!");   /* 队为空，删除失败 */
      else{
          p=front;
          front=front->link;
          item=p->data;
          free(p);
          return  item;            /* 队非空，删除成功 */
      }
}
```

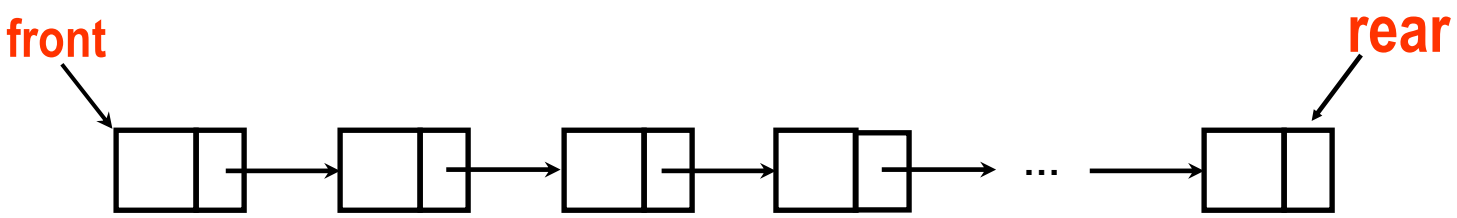
5. 销毁一个队

所谓销毁一个队是指将队列所对应的链表中所有结点都删除，并且释放其存储空间，使队成为一个空队(空链表)。



归结为一个线性链表的删除

```
void destroyLQueue(){
    while(front != NULL){    /* 队非空时 */
        rear=front->link;
        free(Front);          /* 释放一个结点空间 */
        front=rear;
    }
}
```



问题3.2： 银行排队模拟(Simulation)

背景知识：一个系统模仿另一个系统行为的技术称为**模拟**，如飞行模拟器。模拟可以用来进行方案认证、人员培训和改进服务。计算机技术常用于模拟系统中。

生产者-消费者（Server-Custom）是常见的应用模式，见于银行、食堂、打印机、医院、超市…提供服务和使用服务的应用中。

这类应用的主要问题是消费者如果等待（排队）时间过长，会引发用户抱怨，影响服务质量；如果提供服务者（服务窗口）过多，将提高运管商成本。（排队论-queuing theory）

问题3.2a： 银行排队模拟(简化版)

问题描述：某银行网点有五个服务窗口，设计一个程序用来模拟银行服务。

输入：首先输入一个整数表示时间周期数(注：一个时间周期指的是银行处理一笔业务的平均处理时间，可以是一分钟、三分钟或其它)；然后再依次输入每个时间周期的到达客户数。例如：

6

2 5 13 11 15 9

说明：表明在6个时间周期内，第1个周期来了2个（ID分别为1,2），第2个来了5人（ID分别为3,4,5,6,7），以此类推。

输出：每个客户等待服务的时间周期数。

问题3.2： 银行排队模拟(简化版问题分析)

在**生产者-消费者**应用中消费者显然是先来先得到服务

(注：该问题中，消费者指银行职员。她（他）们被简化了，固化为一个时间周期处理一笔业务，因而得以省略)。

（1）可用一个**队列**来存放等待服务的**客户**。每个客户有2个基本属性：排队序号和等待时间（周期数）：

```
struct  cust {
        int id;    //客户排队序号
        int wtime;        //客户等待服务的时间（时间周期数）
};
struct cust Cqueue[MAXSIZE];    //等待服务的客户队列，一个循环队列
```

（2）可用一个变量来表示银行当前提供服务的**窗口数**：

```
int  snum;  //在本问题中，该变量的取值范围为 5
```

问题3.2： 银行排队模拟(简化版模拟器核心程序)


```
for(clock=1; ; clock++){//依次处理每个时间周期，在每个时间周期内
    1. If 客户等待队列非空
          将每个客户的等待时间wtime增加一个时间单元;

    2. If(clock <= simulationtime) //总周期数之内
          2.1 如果有新客户到来（从输入中读入本周期内新来客户数），将
          其入队;

    3. If 客户等待队列非空
          3.1 从客户队列中取（出队）相应数目（按实际服务窗口数）客户
          获得服务;
       Else 结束模拟
}
```

问题3.2： 银行排队模拟（简化版代码实现）


```
//main.c
#include <stdio.h>
#include  <stdlib.h>
#define  MAXSIZE 200  //队列容量
#define MAXSVR  5   //最大服务窗口数
typedef struct {
    int id;
    int wtime;
} CustType;
int Winnum=MINSVR; //提供服务的窗口数
void  updateCustqueue(); //更新等待队列中客户等待时间
void arriveCust(); //获取新客户，并加至等待队列中
int  service(); //银行从队列中获取客户进行服务
void  enCustqueue(CustType c); //客户入等待队列
CustType deCustqueue(); //客户出队
int  getCustnum();   //获取队中等待客户人数
int  isFull();
int  isEmpty();
```

```
//main.c
int main(){
    int clock, simulationtime;
    scanf("%d",&simulationtime);
    for(clock=1; ; clock++) {

        //更新等待队列中客户等待时间
        updateCustqueue();

        //获取新客户，并加至等待队列中
        if(clock <= simulationtime )
            arriveCust();

        //银行从队列中获取客户进行服务
        if(service()==0 && clock > simulationtime)
            break;  //等待队列为空且不会有新客户
    }
    return 0;
}
```

问题3.2： 银行排队模拟(加强版)

问题描述：某银行网点有五个服务窗口，分别为3个对私、1个对公和1个外币窗口。通常对私业务人很多，其它窗口人则较少，可临时改为对私服务。假设当对私窗口客户平均排队人数超过7人时（总排队人数/当前对私窗口数>=7），客户将有抱怨，此时银行可临时将其它窗口中一个或两个改为对私服务；当平均客户数少于7人时，将恢复原有业务。设计一个程序用来模拟银行服务。

输入：首先输入一个整数表示时间周期数(注：一个时间周期指的是银行处理一笔业务的平均处理时间，可以是一分钟、三分钟或其它)，然后再依次输入每个时间周期中因私业务的到达客户数。例如：

6

2 5 13 11 15 9

注意：输入信息中没有对公和外币客户，可认为他们一直为0。

说明：表明在6个时间周期内，第1个周期来了2个（ID分别为1,2），第2个来了5人（ID分别为3,4,5,6,7），以此类推。

输出：每个客户等待服务的时间周期数。

问题3.2： 银行排队模拟(问题分析)

在**生产者-消费者**应用中消费者显然是先来先得到服务

(注：该问题中，消费者指银行职员。她（他）们被简化了，固化为一个时间周期处理一笔业务，因而得以省略)。

(1) 可用一个**队列**来存放等待服务的**客户**。每个客户有2个基本属性：排队序号和等待时间（周期数）：

```
struct  cust {
    int id;         //客户排队序号
    int wtime;   //客户等待服务的时间（时间周期数）
};
struct cust Cqueue[MAXSIZE]; //等待服务的客户队列，一个循环队列
```

(2) 可用一个变量来表示银行当前提供服务的**窗口数**：

```
int  snum;
```

在本问题中，该变量的取值范围为3<= snum <= 5

问题3.2： 银行排队模拟(模拟器核心程序)


```
for(clock=1; ; clock++){ //在每个时间周期内
    1. If 客户等待队列非空
          将每个客户的等待时间wtime增加一个时间单元;
    2. If(clock <= simulationtime)
          2.1 如果有新客户到来（从输入中读入本周期内新来客户数），
          将其入队;
          2.2 根据等待服务客户数重新计算服务窗口数;
    3. If 客户等待队列非空
          3.1 从客户队列中取（出队）相应数目（按实际服务窗口数）
          客户获得服务;
          3.2 根据等待服务客户数重新计算服务窗口数;
       Else 结束模拟
}
```

问题3.2： 银行排队模拟（参考实现）


```
//main.c
#include <stdio.h>
#include  <stdlib.h>
#define  MAXSIZE 200  //队列容量
#define THRESHOLD  7  //窗口增加阈值
#define MAXSVR  5   //最大服务窗口数
#define MINSVR  3   //最小服务窗口数
typedef struct {
    int id;
    int wtime;
} CustType;
int Winnum=MINSVR; //提供服务的窗口数
void  updateCustqueue(); //更新等待队列中客户等待时间
void arriveCust(); //获取新客户，并加至等待队列中
int  service(); //银行从队列中获取客户进行服务
void  enCustqueue(CustType c); //客户入等待队列
CustType deCustqueue(); //客户出队
int  getCustnum();   //获取队中等待客户人数
int  isFull();
int  isEmpty();
```

```
//main.c
int main(){
    int clock, simulationtime;
    scanf("%d",&simulationtime);
    for(clock=1; ; clock++) {
      //更新等待时间
       updateCustqueue();

      //新客户入队，调整窗口数（尝试增加）
       if(clock <= simulationtime )
          arriveCust();

      //提供服务，并调整窗口数（尝试减少）
       if(service()==0 && clock > simulationtime)
          break;  //等待队列为空且不会有新客户
    }
    return 0;
}
```

问题3.2： 扩展与思考

扩展1. 在本问题中，每个业务都占用1个时间周期，这不符合事实。倘若每个业务用时1-k个时间周期，程序需要如何修改？

扩展2. 在本问题中，当前服务窗口平均排队等待服务的客户人员数小于某个阈值时，临时窗口将不再提供服务，一来该策略不是最优，二来也不符合实际情况。现**增加如下规则**：

- 外币和对公窗口应优先处理本业务，即当有对应业务（有客户等待时）时应优先处理，只有当本业务没有排队客户时，才能处理对私业务；
- 当外币和对公窗口没有等待客户同时对私窗口有等待客户排队时，将处理对私业务（资源利用最大化）。

优先队列（Priority queue）*

在实际应用时，前述简单队列结构是不够的，先入先出机制需要使用某些优先规则来完善。如：

- 在服务行业，通常有残疾人、老人优先
- 在公路上某些特殊车辆（如救护车、消防车）优先
- 在操作系统进程调度中，具有高优先级的进程优先执行

优先队列（Priority Queue）：根据元素的优先级及在队列中的当前位置决定出队的顺序。

栈、队

- **栈、队的基本概念**
  - 栈、队的定义
  - 栈、队的基本操作

    栈、队列是特殊线性表(特殊性)
- **栈、队的顺序存储结构**
  - 构造原理、特点
  - 对应的插入、删除操作的算法设计

    (循环队列)
- **栈、队的链式存储结构**
  - 构造原理、特点
  - 对应的插入、删除操作的算法设计
- 栈、队的应用举例

本章结束！