数 据 结 构

第三章 栈和队列 [队列]

任课老师：郭艳

数据结构课程组

计算机学院

中国地质大学（武汉）2020年秋

上堂课要点回顾

- 栈
  - 概念、特征*
  - 抽象数据类型定义ADT Stack
    - 抽象基类**class Stack<T>** (“Stack .h”)
- ADT Stack的实现方案1——顺序栈
  - 模型以及变化
  - 顺序栈类的定义和实现（“SeqStack.h“）
    - **class SeqStack<T>** **数据成员**/SeqStack();bool IsEmpty();bool IsFull();int getSize();void makeEmpty();void Push(T&);bool Pop(T&);bool getTop(T&)及性能
- ADT Stack的实现方案2——单链栈
  - 模型以及变化
  - 顺序栈类的定义和实现（“LinkedStack.h“）
    - **class LinkedStack<T>** **数据成员**/LinkedStack();bool IsEmpty();int getSize();void Push(T&);bool Pop(T&);bool getTop(T&);void makeEmpty();及性能
- 栈的应用*
  - **计算算术表达式的值**　　问题分解为两个子问题：
    - **中缀表达式转为后缀表达式**
    - **计算后缀表达式的值**

第 五 次 课

阅读：

殷人昆，第114-126页

练习：

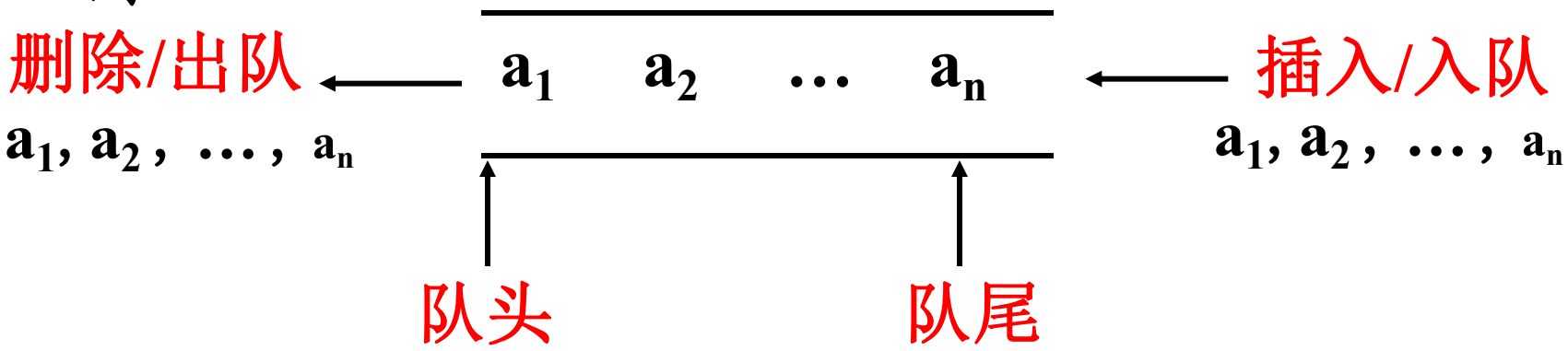
作业5

3.3 队列

3.3.1 队列（Queue）的概念

- 定义：限定所有的插入操作在表的一端进行，而删除操作在表的另一端进行的线性表。

例：



◆ 队列的特征

先进先出 （First In First Out， FIFO）

或

后进后出（Last In Last Out，LILO）

返回

队列的抽象数据类型 P114 程序3. 28

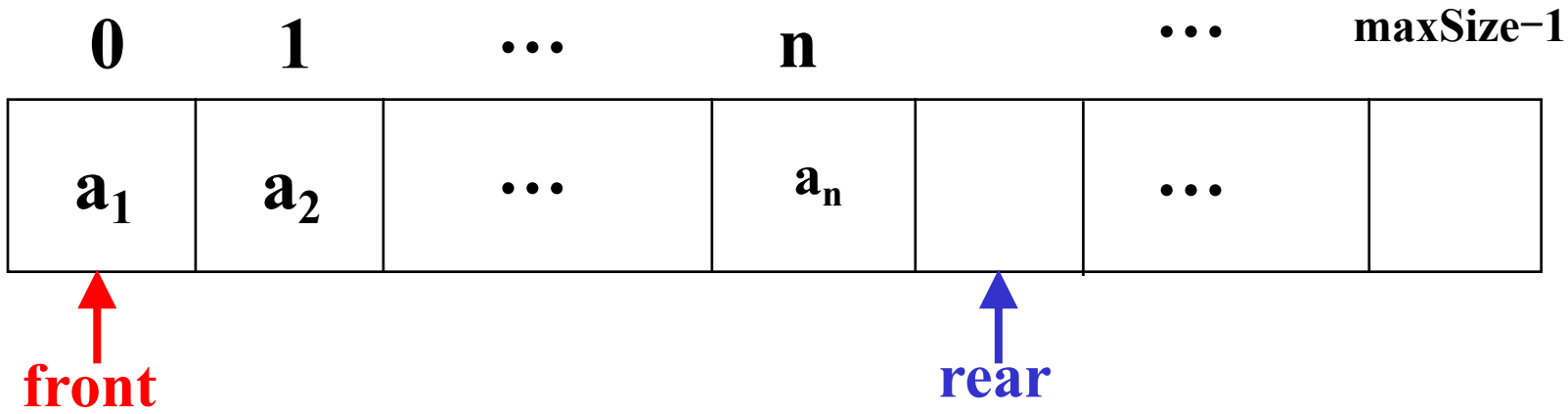
```
template <class T>
class Queue {
public:
    Queue() { };              //构造函数
    ~Queue() { };             //析构函数
    virtual bool EnQueue(const T& x) = 0;  //进队列
    virtual bool DeQueue(T& x) = 0;        //出队列, 输出x
    virtual bool getFront(T& x) = 0;       //取队头, 输出x
    virtual bool IsEmpty() const = 0;      //判队列空
    virtual bool IsFull() const = 0;       //判队列满
    virtual int getSize() const=0;   //求队列元素个数
};
```

返回

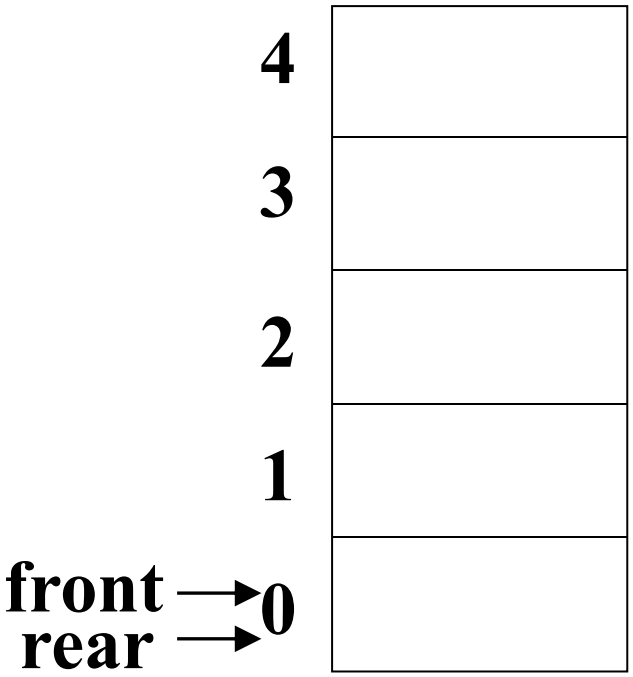
3.3.2 顺序队列和循环队列

- 存储结构：动态一维数组
- 表头表尾位置动态变化，需要设置两个指示器指示表头和表尾位置。约定：
  - 队头指示器**front**指向实际队头元素；
  - 队尾指示器**rear**指向实际队尾元素的下一个位置。

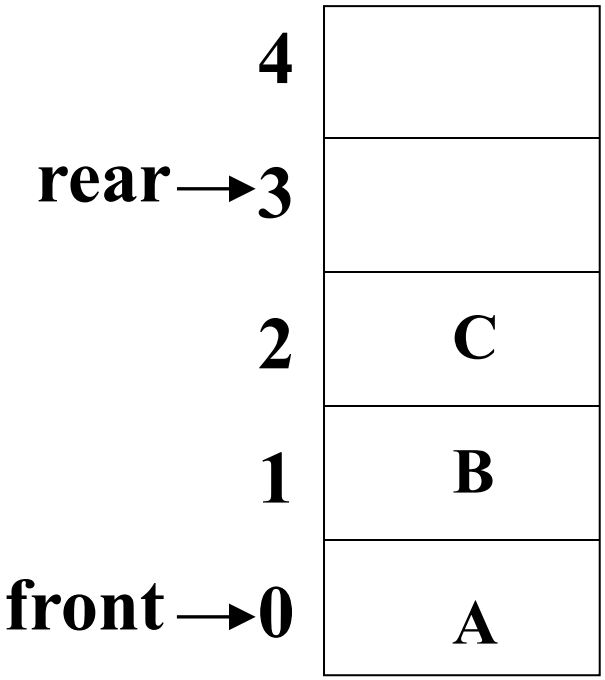
例如：



顺序队列入队、出队时指针的变化方向与赋值：



队空

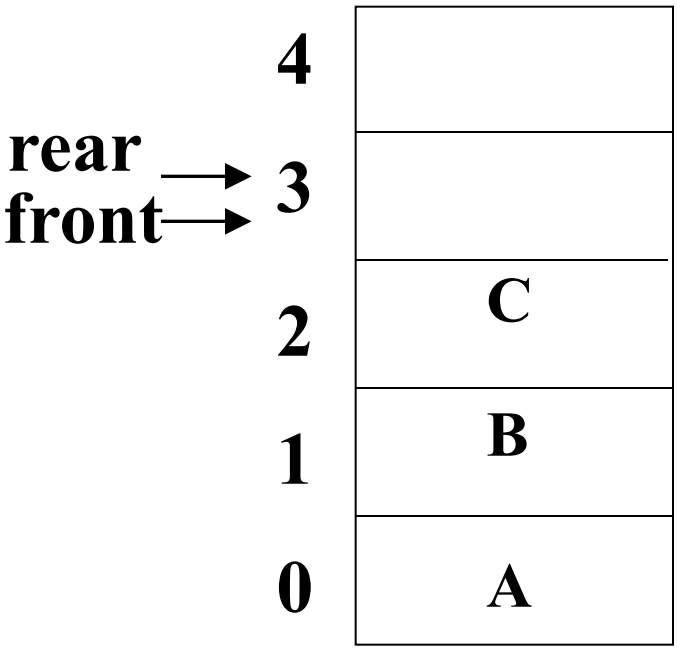


元素A、B、C入队后

初始：front=0;
rear=0;

入队时：rear=rear+1；
front不变化

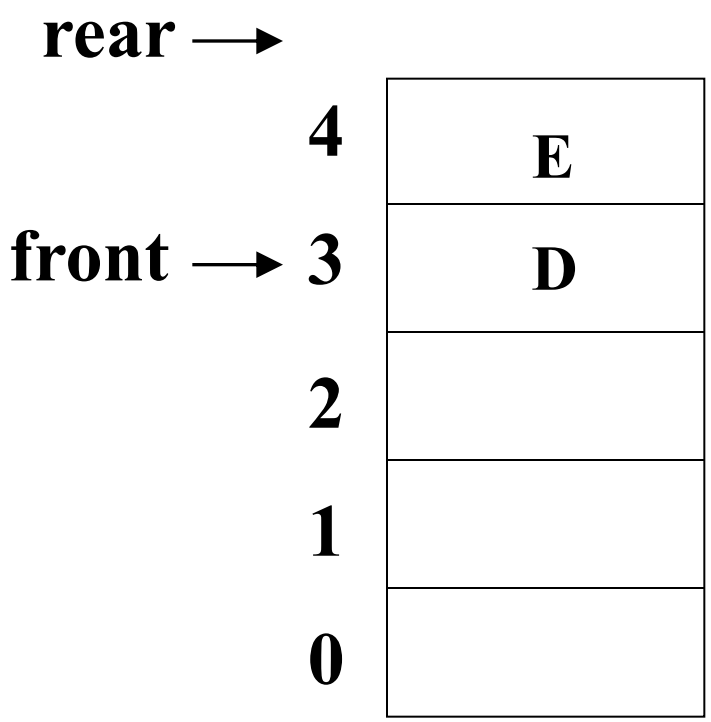
顺序队列入队、出队时指针的变化情况（续）：



A、B、C出队，队空

出队时：front=front+1;
rear不变化

队空时：front==rear



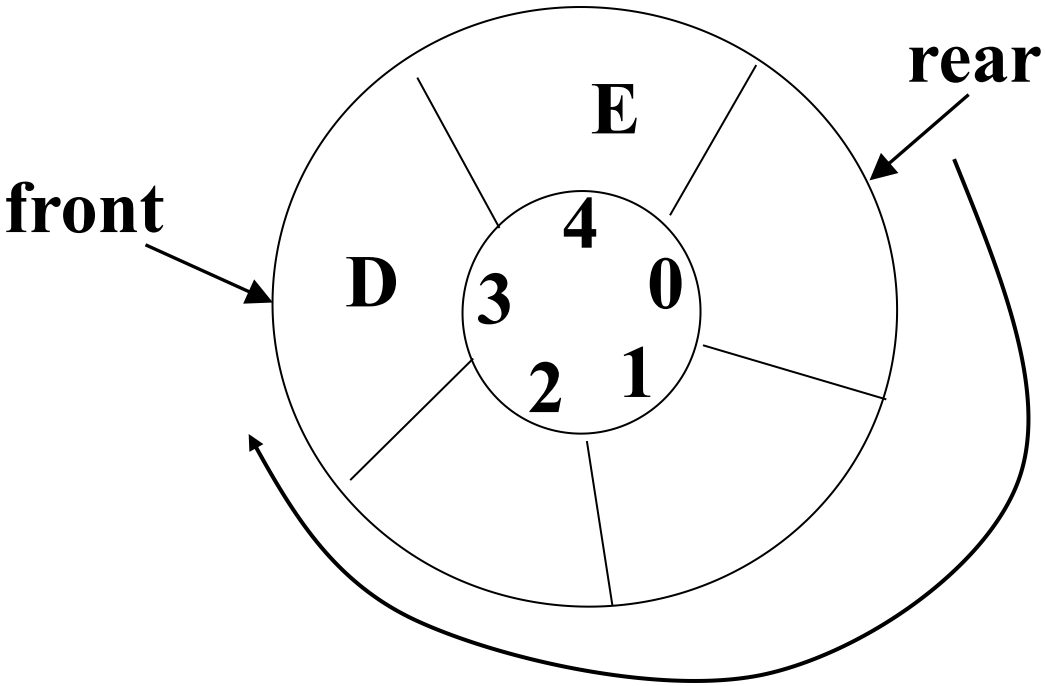
元素D、E入队，队满，F入队

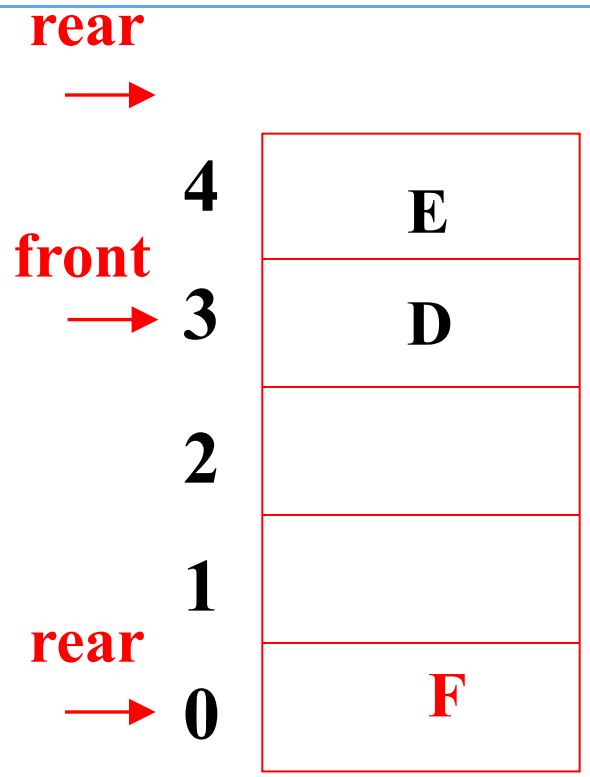
队满时：rear==maxSize

浪费空间，“假溢出”

顺序循环队列

为解决“假溢出”现象，
将队列逻辑上形成环形。





元素F 入循环队列第 0号单元

入队时：rear=rear+1;
如果rear==maxSize则rear=0;
出队时：front=front+1;
如果front==maxSize则front=0;

•顺序循环队列入队、出队时指针的变化

① 入队：

```
rear++;
if (rear==maxSize)
        rear=0;
```

还可利用数学上的“模(%)运算”来实现上述过程：

```
rear＝(rear+1) % maxSize ;
```

② 出队：

```
front++;
if (front==maxSize)
        front=0;
```

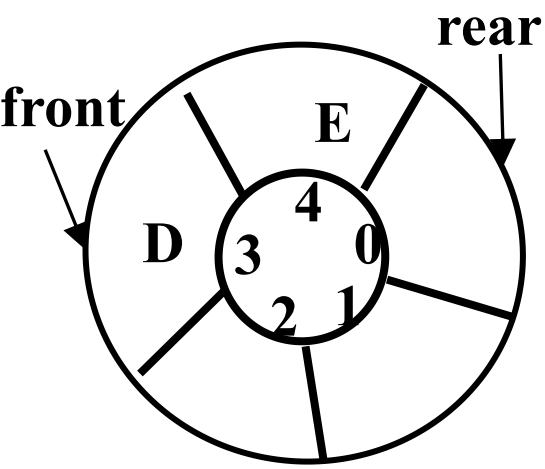
还可利用数学上的“模(%)运算”来实现上述过程：

```
front＝(front+1) % maxSize;
```

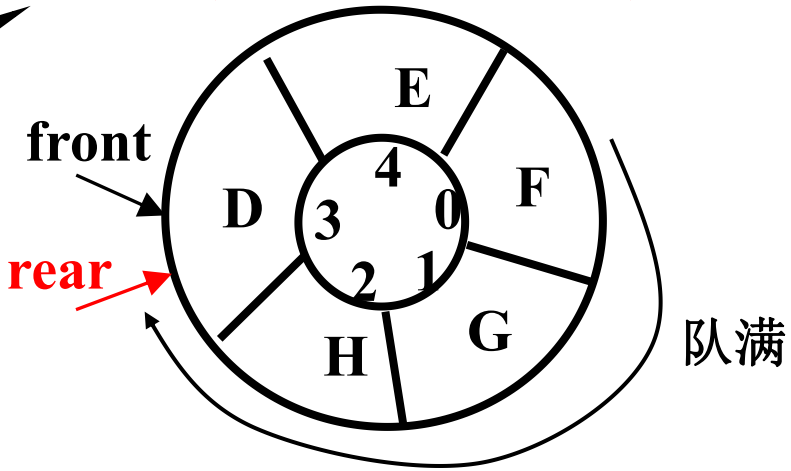
顺序循环队列入队、出队时指针的变化

入队：rear=rear+1;

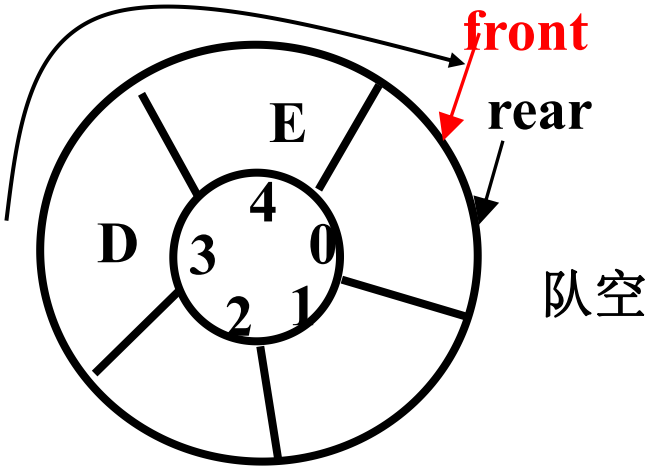
if (rear==Maxsize)rear=0;



F、G、H入队



D、E出队

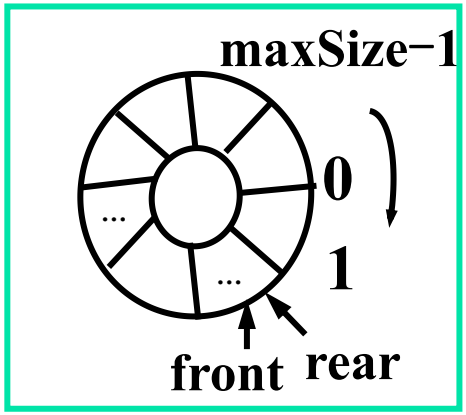


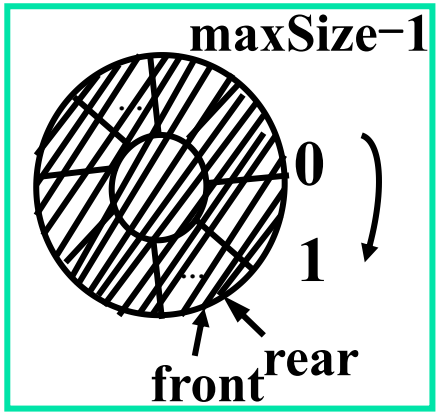
出队：front=front+1;

if (front==Maxsize)front=0;

如何判队满、队空？

方法一：





队空 (front==rear)　　队满 (front==rear)

出问题了：

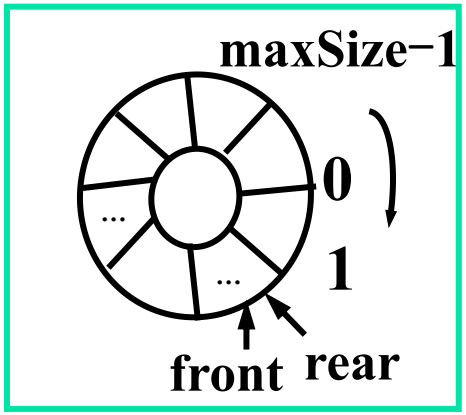
当front==rear时

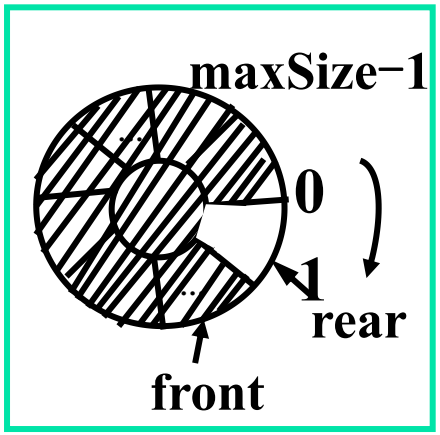
是队满？

是队空？

方法一失败，修正方法一

方法二：空一个存储单元不使用





队空 (front==rear)　　队满((rear+1)%maxSize == front)

方法二成功判断队空和队满

？其它的方法

循环队列的类定义 P116 程序3.29

```
template <class T>                    //SeqQueue.h文件实现顺序循环队列类
class SeqQueue:Public Queue<T> {
protected:
    T *elements;                      //存放队列元素的动态数组
    int front;        //队头指示器，指向实际队头元素位置*/
    int rear ;        //队尾指示器，指向实际队尾元素的下一个位置*/
    int maxSize;                      //最大元素个数
public:
    SeqQueue ( int sz= 10 );
    ~SeqQueue ( ) { delete [ ] elements; }
    bool EnQueue ( const T & x); //进队
    bool DeQueue (T& x );             //出队
    bool getFront (T& x);       //取队头元素
    void MakeEmpty ( ) { front = rear = 0; }
    bool IsEmpty ( ) const  { return front == rear; }
    bool IsFull ( ) const  { return (rear+1) % maxSize == front; }
    int getSize( ) const   { return (rear-front+maxSize) % maxSize;}
};
```

【顺序循环队列的构造算法】P116 程3.29(2)自学

```
template <class T>
SeqQueue<T>::SeqQueue ( int sz) :
                                front(0), rear(0), maxSize(sz)
{
    elements=new T[maxSize];
    assert(elements!=NULL);
}
```

【顺序循环队列的入队算法】P116 程3.29(3)

```
template <class T>
bool SeqQueue<T>::EnQueue ( const T & x) {
    if  (IsFull ( )==true)  return false;
    elements[rear] = x;
    rear = (rear+1) % MaxSize;
    return true;
}//O(?)
```

【顺序循环队列的出队算法】P116 程3.29(4)

```
template <class T>
bool SeqQueue<T> :: DeQueue(T& x) {
    if ( IsEmpty ( )==true) return false;
    x = elements[front];
    front = ( front+1) % MaxSize;
    return true;
}//O(?)
```

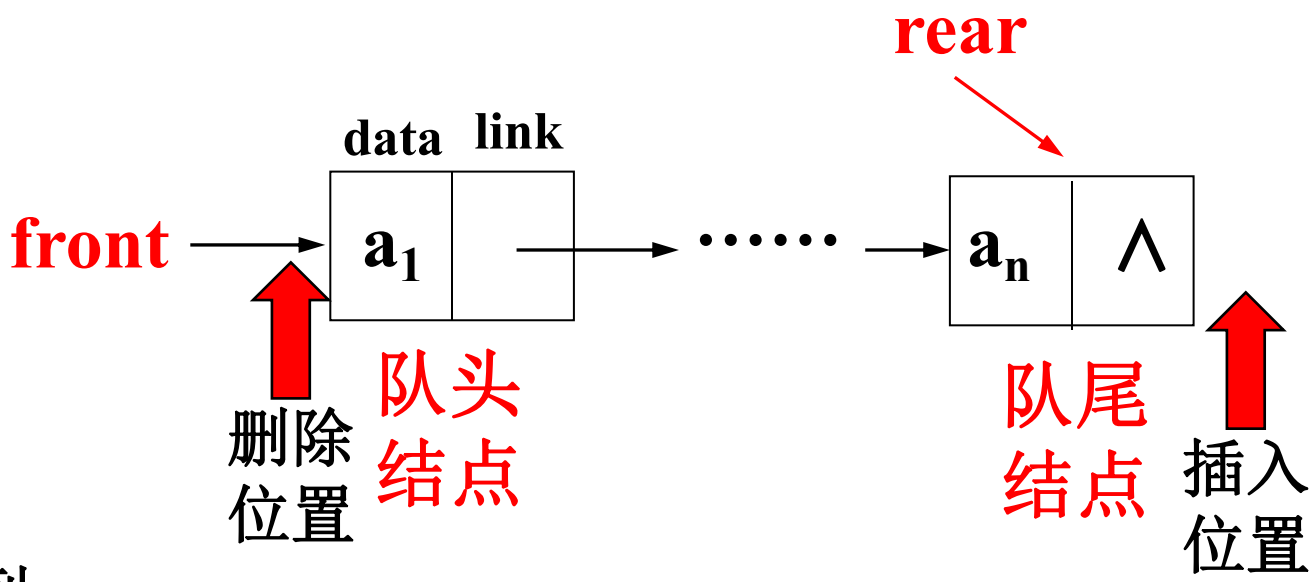
【顺序循环队列的取队头算法】P116 程序3.29(5)自学

```
template <class T>
bool SeqQueue<T> :: getFront(T& x) const{
    if ( IsEmpty ( )==true) return false;
    x = elements[front];
    return true;
}//O(?)
```

3.3.3 单链队列

◆ 单链队列（不带头结点）模型

非空链队列：满足front!=NULL && rear!=NULL



空链队列：

空链队列满足front==rear==NULL

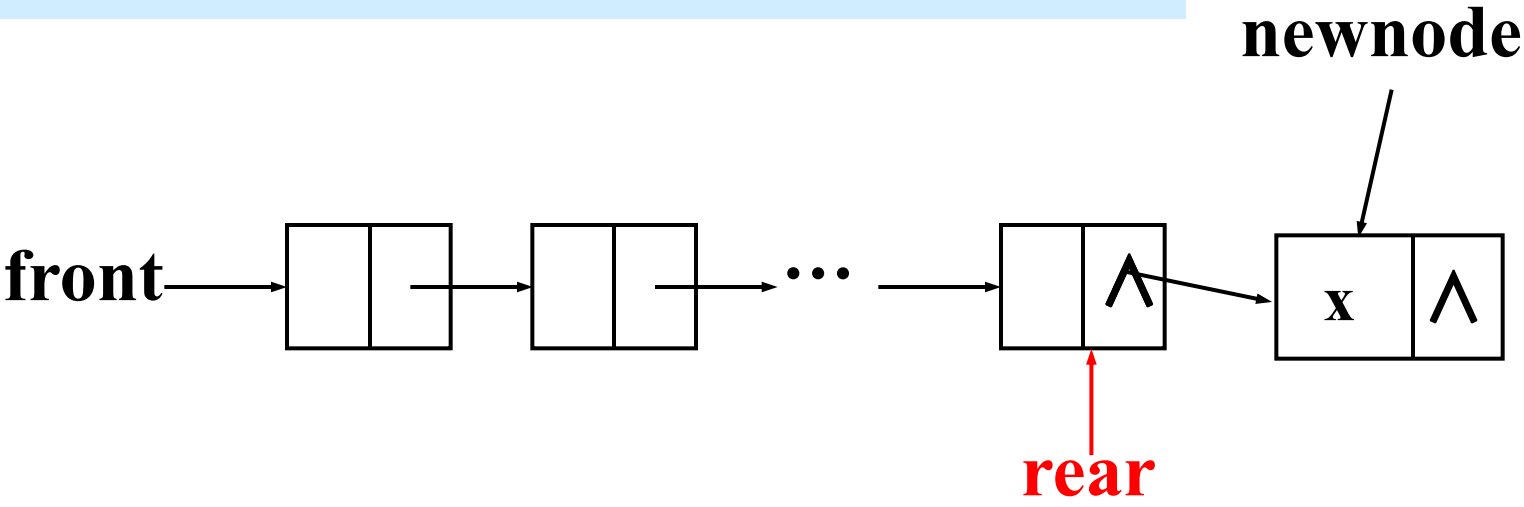
链式队列类的定义 P118 程序3.30

```
#include "LinkedList.h"      //LinkNode类定义
#include "Queue.h"           //Queue类定义
template <class T>
class LinkedQueue:public Queue<T>{
protected:
   LinkNode<T> * front,* rear;
public:
    LinkedQueue():rear(NULL),front(NULL){}//不带头结点
    ~LinkedQueue(makeEmpty());
    bool EnQueue(const T& x);
    bool DeQueue(T& x);
    void makeEmpty();
    bool getFront(T& x) const {if (IsEmpty()) return false;
                               else {x=front->data;return true};}
    bool IsEmpty() const { return front==NULL;}
    int getSize() const;          }
```

【链队列的置空算法】P118 程序3.30(2)自学

```
template <class T>
void LinkedQueue<T>::makeEmpty() {
//
{  LinkNode<T> *p;
    while(front!= NULL)
    { p=front;
       front=front->link;
      delete p;
    }
   rear=NULL;   //增加
}//O(?)
```

◆ 链队列的插入运算（不带头结点）



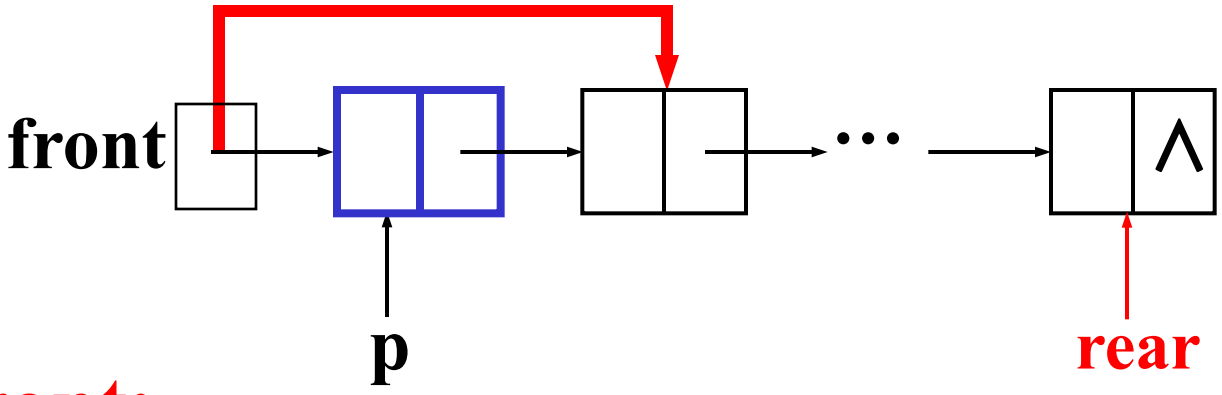
rear->link=newnode;

rear=newnode;

【链队列的进队算法】P118 程序3.30(3)

```
template <class T>
bool LinkedQueue<T>::EnQueue(const T& x) {
//将新元素x插入到队列的队尾
    if (front == NULL)
    {    //创建第一个结点
        front = rear = new LinkNode<T> (x);
        if (front == NULL) return false;//分配失败
    }
    else
    {    //队列不空, 插入
        rear->link = new LinkNode<T> (x);
        if (rear->link == NULL) return false;
        rear = rear->link;
    }
    return true;
}//O(?)
```

◆ 链队列的删除运算（不带头结点）

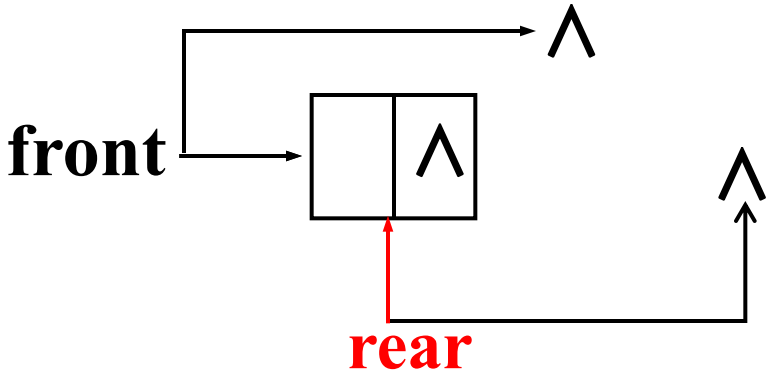


p=front;

front=front->link;

delete(p);

注意：



if (front==NULL) rear=NULL;

【链队列的出队算法】P118 程序3.30(4)

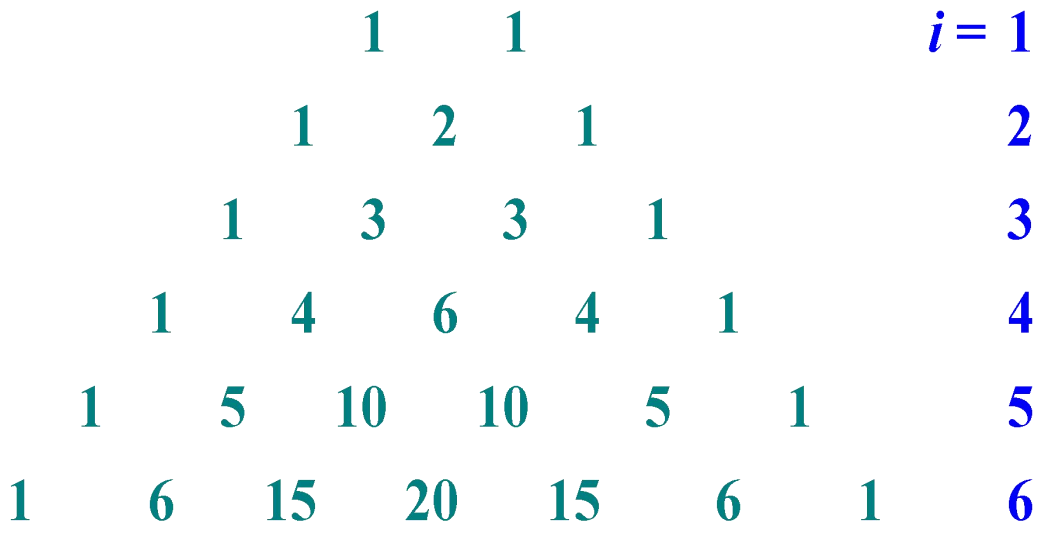
```
template <class T>
bool LinkedQueue<T>::DeQueue(T& x) {
//如果队列不空，将队头结点从链式队列中删去
    if (IsEmpty() == true) return false;//判队空
    LinkNode<T> *p = front;
    x = front->data;
    front = front->link;
    delete p;
    if (front==NULL) rear=NULL;//增加处理rear语句
    return true;
}//O(?)
```

【链队列的取队头算法】P118 程序3.30(5)自学

```
template <class T>
bool LinkedQueue<T>::getFront(T& x) const{
//如果队列不空，将队头结点从链式队列中删去
    if (IsEmpty() == true) return false;//判队空
    x = front->data;
}//O(?)
```

3.3.4 队列的应用——打印杨辉三角形前n行

杨辉三角形

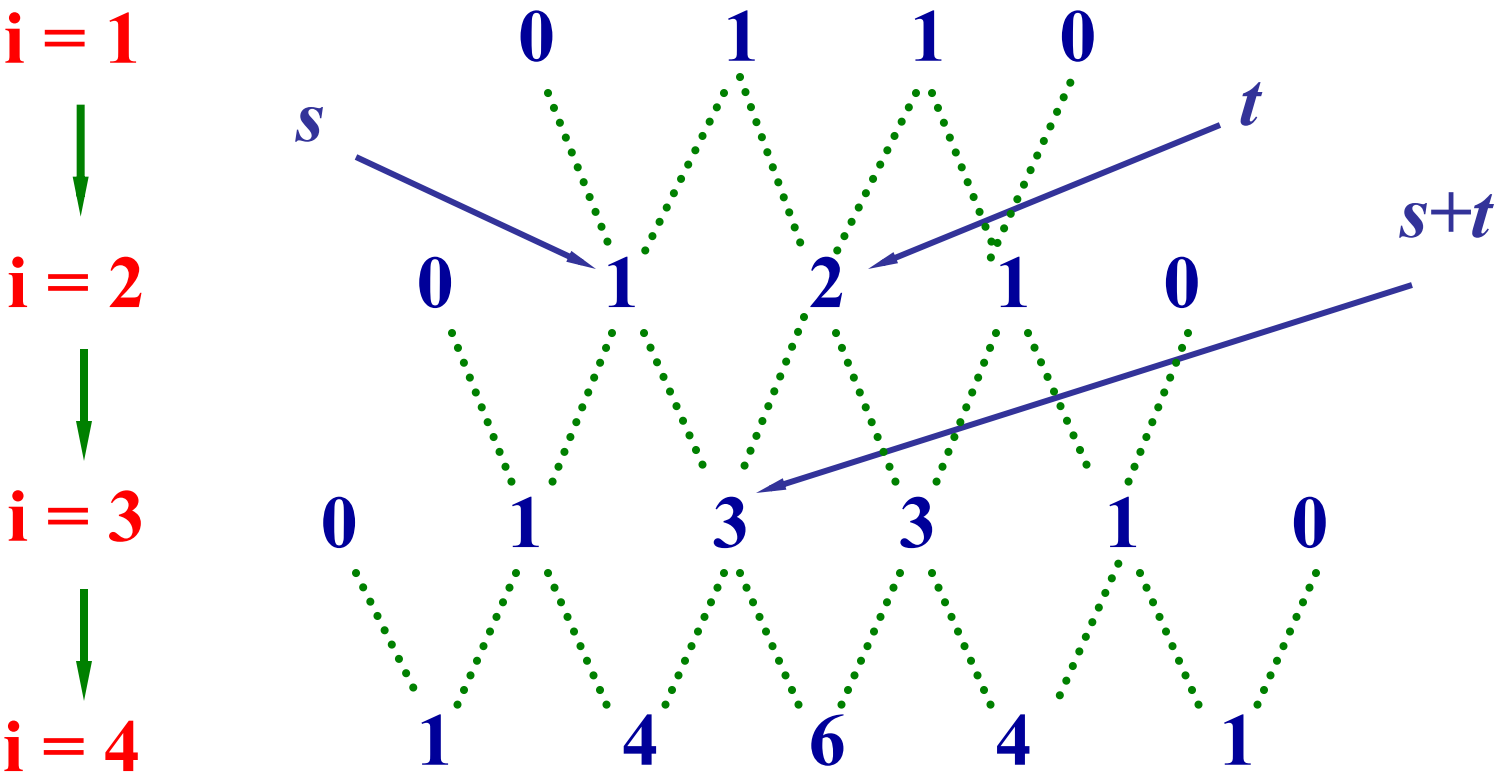


思考：如何输出杨辉三角形前n行？

逐排扫描处理问题

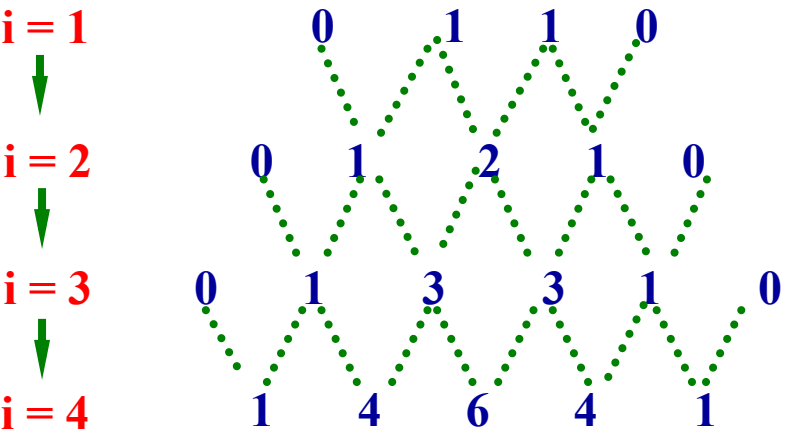
每行数据都需要被计算和存储

分析第 *i* 行数据与第 *i*+1行数据的关系



从第i行数据可以计算第i+1行数据。为了计算i+1行数据，在得到第i行数据的同时，需要存储第i行的数据。那么，高效的存储结构是什么？

存储结构设计



在第i行的数据打印完，并且第i+1行的数据也全部计算出来和存储妥当，第i行的数据才算被全部处理完毕。

第i行数据需要被存储。每行数据都需要被存储。但是可以不在一个时刻每行数据都被存储。

存储结构设计1

（n行，每行i+1列）

1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

存储结构设计2

（n行，每行n+1列）

1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

存储结构设计3

（1行，n+1列）

4	1	1	4	6
1	1	0	1	2
1	0	1	3	3
1	0	1	4	6
4	1			

逐排扫描处理问题的逻辑或框架：

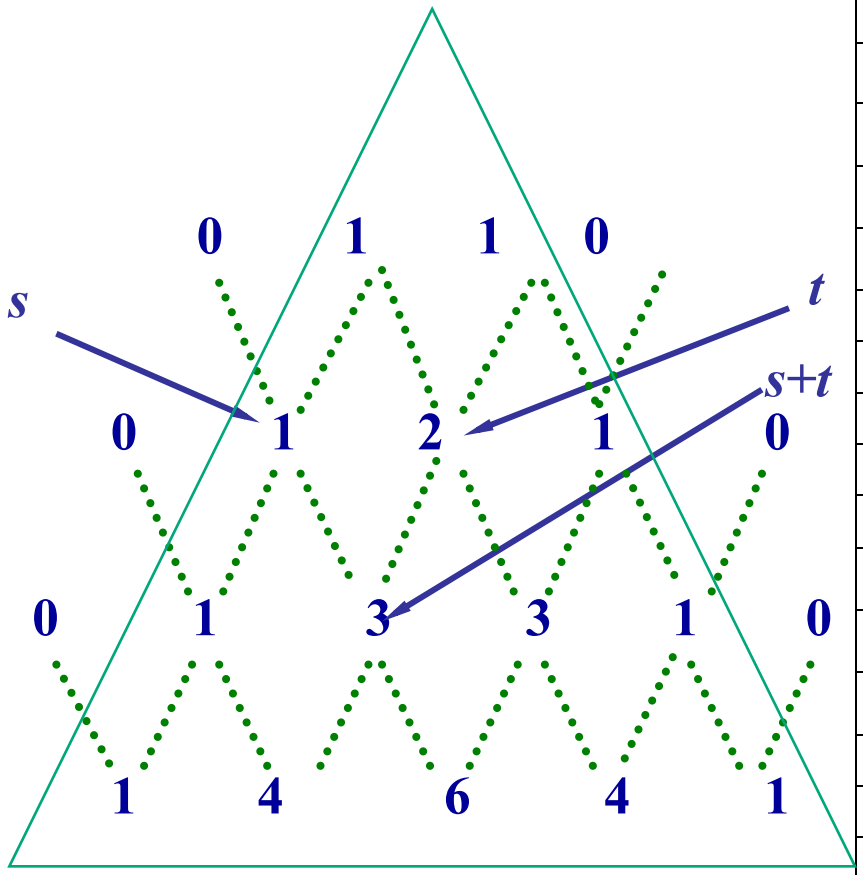
- **自上到下自左到右**扫描操作对象集合，在扫描的同时访问（处理）各个对象。当扫描结束时，完成所有对象处理，得到输出。
- 当正在访问（处理）的对象当前**不能完成所有**对它的处理时，需要**缓存该对象**。当等待到合适时机，再**从缓存取出**该对象并完成所有对它的处理。
- 如果**最先**访问（处理）的对象**最先**完成所有对它的处理，或者，**最后**访问（处理）的对象**最后**完成所有对它的处理，则**采用队列**作为未完成访问（处理）的对象的**缓存数据结构**。

i = 1

i = 2

i = 3

i = 4



i　　i+1个值

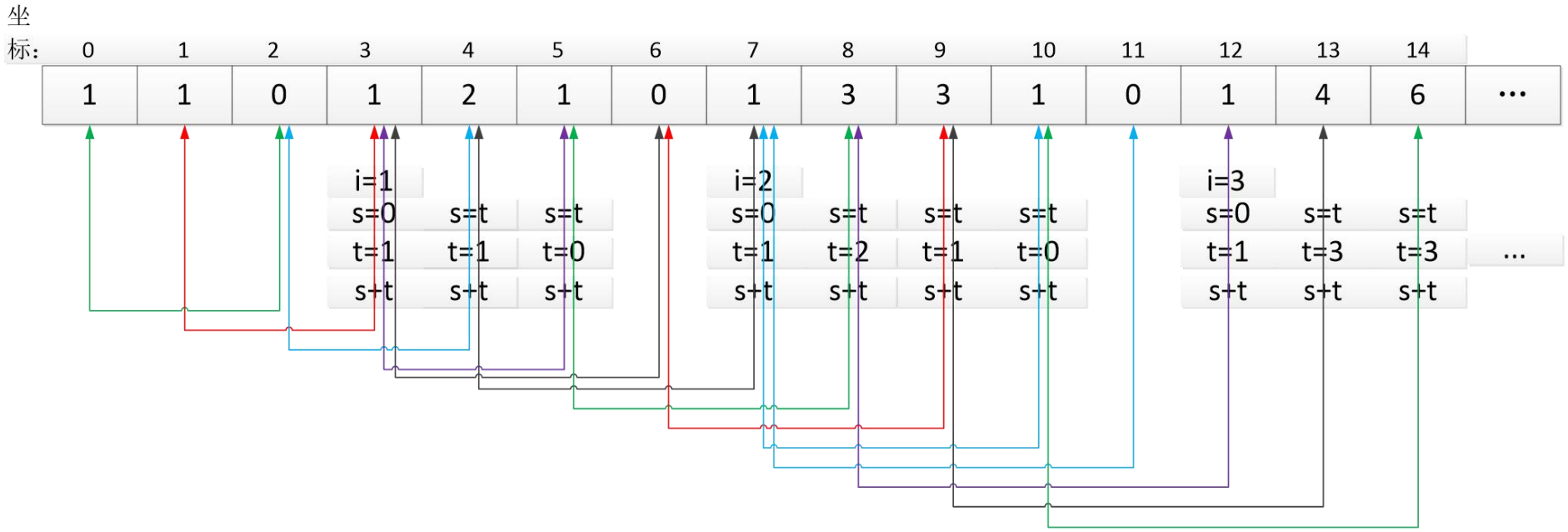
.....

i = n　　n+1个值

共n行　　合计 $\sum_{i=1}^{n}(i+1)=(n^2+3n)/2$ 个值

i	s	t	s+t	队列(队头->队尾)
				1 1
1				1 1 0
	0	1	1	1 0 1
	1	1	2	0 1 2
	1	0	1	1 2 1
2				1 2 1 0
	0	1	1	2 1 0 1
	1	**2**	**3**	**1 0 1 3**
	2	**1**	**3**	**0 1 3 3**
	1	0	1	1 3 3 1
3				1 3 3 1 0
	0	1	1	3 3 1 0 1
	1	3	4	3 1 0 1 4
	3	3	6	1 0 1 4 6
	3	1	4	0 1 4 6 4
	1	0	1	1 4 6 4 1
4				1 4 6 4 1 0
	0	1	1	4 6 4 1 0 1
	1	4	5	6 4 1 0 1 5
	4	6	10	4,1,0,1,5,10
	6	4	10	1,0,1,5,10,10
	4	1	5	0,1,5,10,10,5
	1	0	1	1,5,10,10,5,1

从第 *i* 行数据计算并存放第 *i*+1 行数据 （设n>=12）



带箭头的线条代表队列

【利用队列打印杨辉三角形】 P121 程序3.31

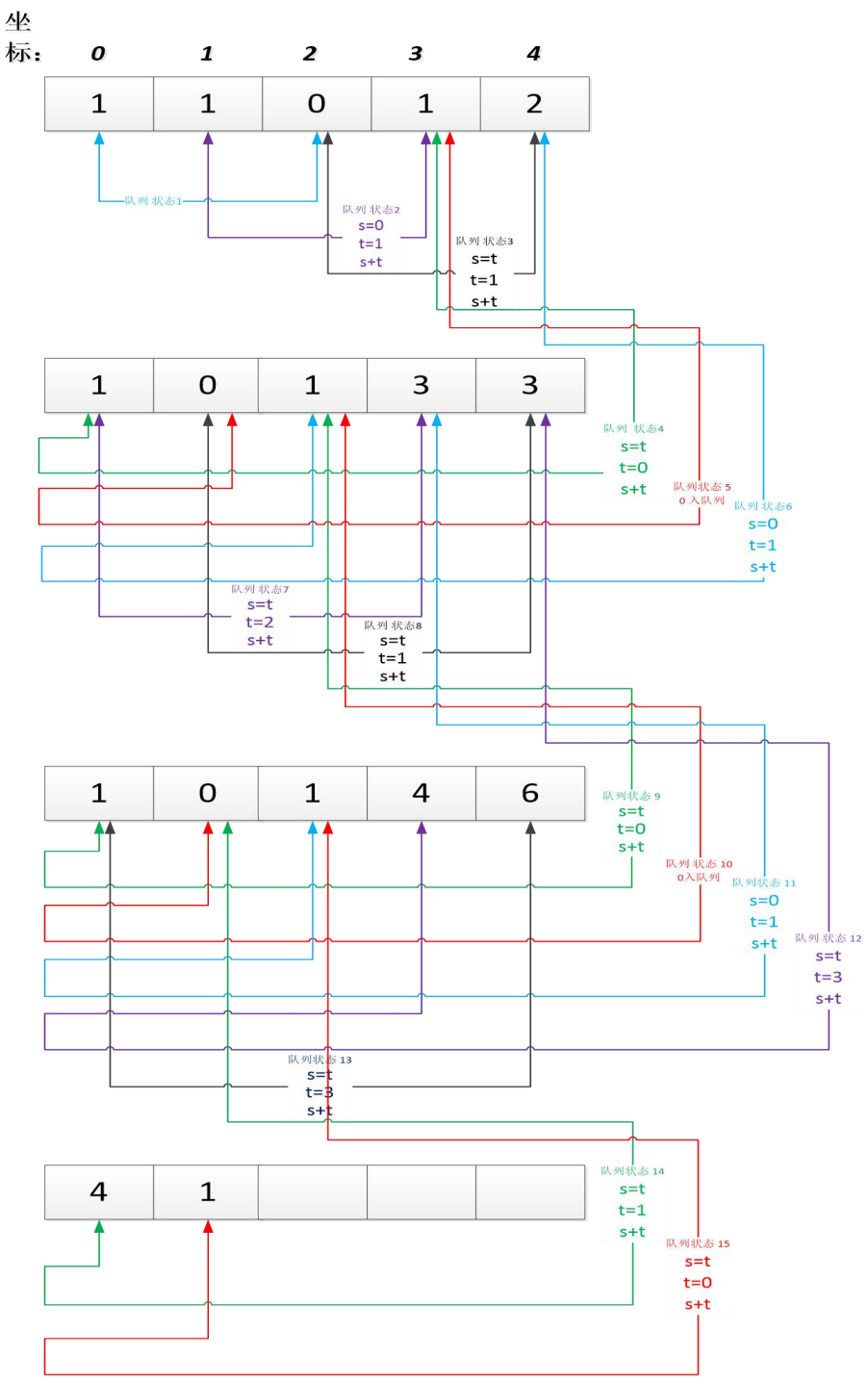
```
#include <stdio.h>
#include <iostream.h>
#include "SeqQueue.h"
void YANGVI ( int n ) {
    Queue q(n+2);   int s=0,t,u;//队列初始化
    q.EnQueue (1);        //1,1入队列
    q.EnQueue (1);
    for ( int i = 1; i <= n; i++ ) {//逐行计算
        cout << endl;
        q.EnQueue (0);     //0入队列
        for ( int j = 1; j <= i+2; j++ )
        {//计算第i+1行并存到队列，输出第i行
            q.DeQueue (t );
            u=s+t;
            q.EnQueue (u);
            s = t;       //得到并入队列第i+1行
            if ( j != i+2 ) cout << s << ' ';
              //出队列时输出第i行
    }
```

i	j	s	t	u=s+t	队列	循环队列 头-尾				
					1 1					
1					1 1 0	1	1	0		
	1	0	1	1	1 0 1		1	0	1	
	2	1	1	2	0 1 2			0	1	2
	3	1	0	1	1 2 1	1			1	2
2					1 2 1 0	1	0		1	2
	1	0	1	1	2 1 0 1	1	0	1		2
	2	1	2	3	1 0 1 3	1	0	1	3	
	3	2	1	3	0 1 3 3		0	1	3	3
	4	1	0	1	1 3 3 1	1		1	3	3
3					1 3 3 1 0	1	0	1	3	3
	1	0	1	1	3 3 1 0 1	1	0	1	3	3
	2	1	3	4	3 1 0 1 4	1	0	1	4	3
	3	3	3	6	1 0 1 4 6	1	0	1	4	6
	4	3	1	4	0 1 4 6 4	4	0	1	4	6
	5	1	0	1	1 4 6 4 1	4	1	1	4	6

循环队列的变化（设n=3）

带箭头的线条代表队列

i	j	队　列（队头->尾）				
1		**1**	1	0		
	1		**1**	0	1	
	2			**0**	1	2
	3	1			**1**	2
2		1	0		**1**	2
	1	1	0	1		**2**
	2	**1**	0	1	3	
	3		**0**	1	3	3
	4	1		**1**	3	3
3		1	0	**1**	3	3
	1	1	0	1	**3**	3
	2	1	0	1	4	**3**
	3	**1**	0	1	4	6
	4	4	**0**	1	4	6
	5	4	1	**1**	4	6



队列的应用

- 操作系统中用到队列
  - 1、对CPU的分配管理：进程排队、作业排队

一般计算机只有一个CPU，采用一个就绪队列管理CPU的分配。 当多个进程需要CPU运行时，它的进程名就被插入到就绪队列的队尾。CPU 总是首先执行排在队头的进程。一个进程分配到CPU的一段时间执行完了，又将它插入到队尾等待，CPU转而为执行下一个 队头进程。如此，按“先进先出”的原则一直进行下 去，直到执行完的进程从队列中逐个删除掉。

队列的应用（续）

◆ 操作系统中用到队列

▪ 2、主机与外部设备之间通信

由于外部设备传输速度远远低于主机传输速度，可以设定一个“输出数据队列缓冲区”。当主机要输出数据时，将数据按块（例如每块512B）逐个添加到“队列缓冲区”的尾端，写满后就暂停输出，继而去做其它的事情。而外部设备则依其输出速度按照先进先出的原则依次从队首逐个取出数据块输出，打印完后再向主机发出请求，主机接到请求后再向缓冲区写入打印数据，这样利用队列既保证了输出的数据有完全相同的次序，不致发生输出次序的混乱或数据的丢失，同时保证了主机的效率。

队列的应用（续）

◆事件驱动模拟

- 银行业务模拟

  每个来到的顾客发一个号码，如果哪个柜台空闲了，就叫号码最靠前的顾客来办理业务；如果同时几个柜台空闲，就按照一种法则来决定这几个柜台叫号的顺序（最简单的是按柜台号码顺序）。这样，就能保证顾客按照先来后到的顺序接受服务——因为大家排在一个队里。

- 航空客运订票系统

  对于预约客户名单数据来说，由于要遵循先到先服务的原则，因此采用队列结构。由于预约人数无法预计，因此队列应以链表作为存储结构。

- 电梯调度模拟……

3.4 优先级队列（自学）

- 任务的执行顺序与任务的优先权的关系：任务的优先权越高，越先被执行。
- **优先级队列（Priority Queue）** 每次从队列中取出的是具有最高优先权的元素的队列

约定：数字越小，优先权越高

例如：

任务编号	1	2	3	4	5
优先权	20	0	40	30	10
执行顺序	3	1	5	4	2

最小优先级顺序队列的类定义 P124 程序3.33

```
#include <assert.h>
#include <iostream.h>
#include <stdlib.h>

template <class T>
class PQueue {
private:
    T *pqelements;        //存放数组
    int count;            /*队列元素计数，队头元
                素下标是0，队尾元素下标是count-1*/
    int maxPQSize;        //最大元素个数
    void adjust();        //调整
```

```
public:
    PQueue(int sz = 50);
   ~PQueue() { delete [ ] pqelements; }
   bool IsEmpty() const { return count == 0; }
   bool IsFull() const
        { return count == maxPQSize; }
   int GetSize const{return count;}
   void MakeEmpty() { count = 0; }
   int Length() const { return count; }
   bool Insert(T x);
   bool RemoveMin(T& x);
   bool GetFront(T& x);
};
```

返回

优先级队列部分成员函数的实现

```
template <class T>   //P124 程序3.34(1)
PQueue<T>::PQueue(int sz) {
    maxPQSize = sz;  count = 0;
    pqelements = new T[maxPQSize];
    assert (pqelements != NULL);
}

template <class T>  //P124 程序3.34(2)
bool PQueue<T>::Insert(T x) {
    if (IsFull() == true) return false;   //判队满断言
    pqelements[count++] = x;              //插入
    adjust();   return true;
}
```


10 20 40 50 70 90
插入60
10 20 40 50 70 90 60
缓存
60
10 20 40 50 60 70 90
缓存
60
10 20 40 50 60 70 90

```
template <class T>   //P124 程序3.34(3)
void PQueue<T>::adjust() {
    T temp = pqelements[count-1];
    //将最后元素暂存再从后向前找插入位置
    for (int j = count-2;  j >= 0;  j--)
        if (pqelements[j] <= temp) break;
        else pqelements[j+1] = pqelements[j];
    pqelements[j+1] = temp;
}//O(n)
```

```
template <class T>   //P124 程序3.34(4)
bool PQueue<T>::RemoveMin(T& x) {
    if (IsEmpty()) return false;
    x = pqelements[0];       //取出0号元素
    for (int i = 1; i < count; i++)
        pqelements[i-1] = pqelements[i];
        //从后向前逐个移动元素填补空位
    count--;
    return true;
} //O(n)
```

```
template <class T>  P124 程序3.34(5)
bool PQueue<T>::GetFront (T& x) {
    if (IsEmpty() == true) return false;
    x = pqelements[0];
    return true;
}
```

双端队列 (Double-ended queue) P126-131（略）

在队列的两端插入/删除

操作：

bool EnQueueHead(T &x)
bool EnQuereTail(T &x)
bool DeQuereHead(T &x)
bool DeQueueTail(T &x)
bool EnQueue(T &x)
bool DeQueue(T &x)

返回

自 学

1、P114 程序3.28 Class Queue<T>
2、P116 程序3.29 Class SeqQueue<T>、SeqQueue()、IsEmpty()、IsFull()、getSize()、makeEmpty()、EnQueue()、DeQueue()、getFront()
3、P118 程序3.30 (1,2,3,4,5,6,7)，重点 class LinkedQueue<T>、IsEmpty()、EnQueue()、DeQueue()、getFront()
4、P124-126 3.4章节 优先级队列（概念、存储表示和实现）

作业5

1、补充题

- **概念题：** 设数据元素序列{a,b,c,d,e,f,g}的进队列操作和出队列操作可任意进行，请罗列出所有的出队列序列。

本堂课要点总结

- 栈
  - 栈在函数调用中的作用
  - 栈在实现函数递归调用中所发挥的作用
    - n!递归算法
- 队列
  - 队列的定义、特征*、抽象基类
  - 顺序循环队列*的实现SeqQueue.h
  - 链队列的实现LinkedQueue.h
  - 队列的应用*
    - 打印杨辉三角形前n行

第二次上机——停车场管理

问题描述：

设停车场内只有一个可停放 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，一旦停车场内有车开走，则排在门外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，每辆停放在停车场的车在它离开停车场时必须按它停留的时间长短缴纳费用。试为停车场编制按上述要求进行管理的模拟程序。

用一个整数代表车辆的车牌号，设计找车函数返回被取车辆在停车场的位置，设计停车和取车函数完成按上述要求停车、取车缴纳费用功能。