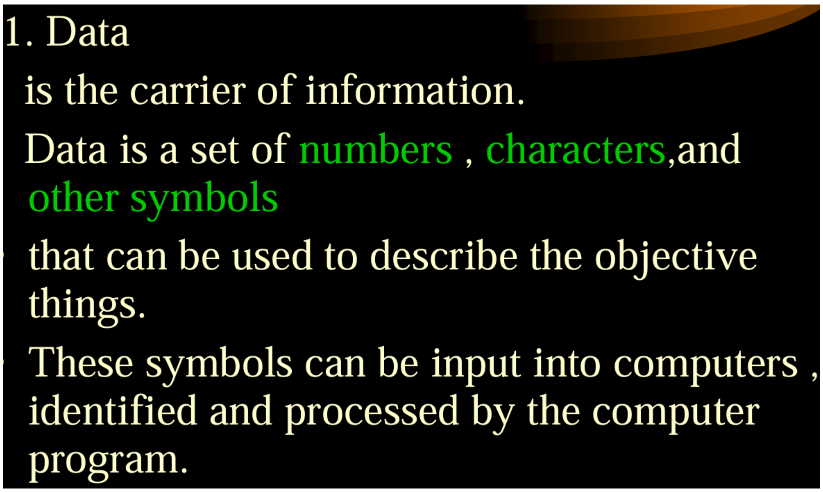
一、　　第一章

数据结构定义、数据定义:



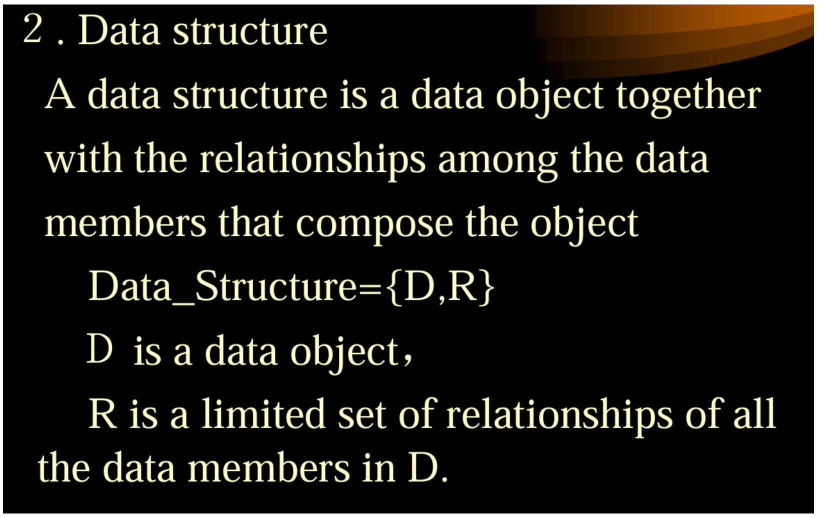
1. Data

is the carrier of information.

Data is a set of numbers , characters,and other symbols

that can be used to describe the objective things.

These symbols can be input into computers , identified and processed by the computer program.



2 . Data structure

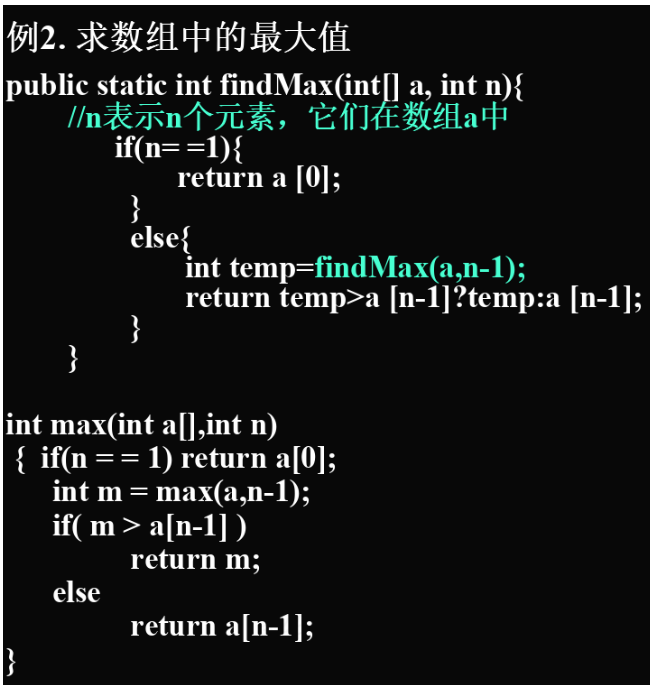
A data structure is a data object together with the relationships among the data members that compose the object

Data_Structure={D,R}

D is a data object，

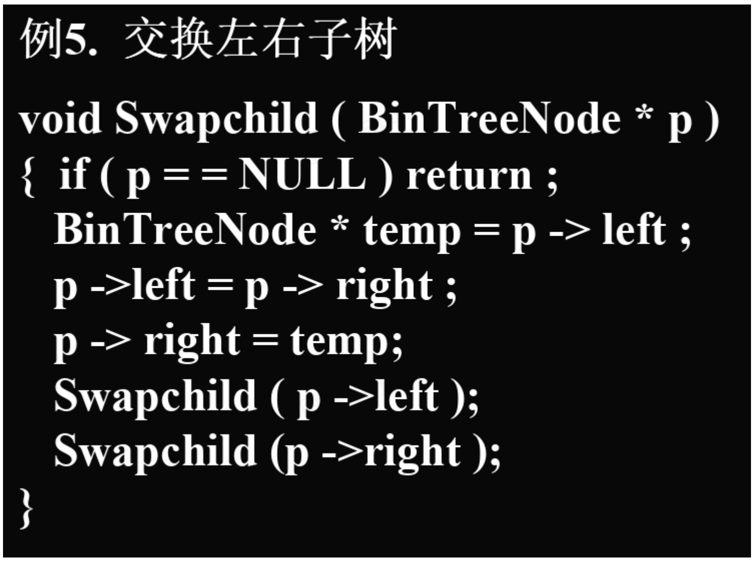
R is a limited set of relationships of all the data members in D.

递归的概念与实现



例2. 求数组中的最大值

```
public static int findMax(int[] a, int n){
      //n表示n个元素，它们在数组a中
           if(n= =1){
                 return a [0];
            }
            else{
                  int temp=findMax(a,n-1);
                  return temp>a [n-1]?temp:a [n-1];
            }
      }

int max(int a[],int n)
 {  if(n = = 1) return a[0];
     int m = max(a,n-1);
     if( m > a[n-1] )
            return m;
     else
            return a[n-1];
}
```



例5. 交换左右子树

```
void Swapchild ( BinTreeNode * p )
{  if ( p = = NULL ) return ;
   BinTreeNode * temp = p -> left ;
   p ->left = p -> right ;
   p -> right = temp;
   Swapchild ( p ->left );
   Swapchild (p ->right );
}
```

面向对象部分定义不会考

二、　　第二章：算法分析

时间复杂度和空间复杂度

几个表示法，四种表示法

给一个算法计算对应的复杂度

计算明确的操作次数，会告知所有的相应操作的操作个数

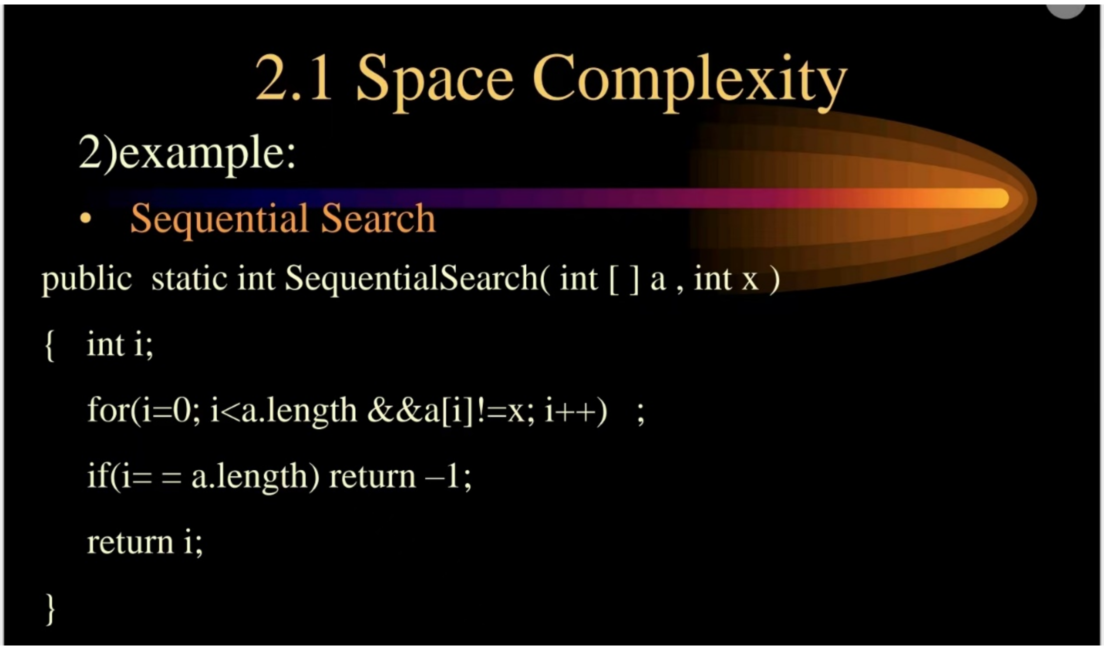
最佳、最差和平均情况下的复杂度差异;

大 O、Ω和 θ 符号

1）分析某个语句的执行次数（频度）

2）分析某个程序段执行的时间复杂度（用大 O 表示，要求写出推导过程）

```
例 2.    x = 0; y = 0;
        for (int i = 1; i <= n; i++)
                for (int j = 1; j <= i; j++)
                        for (int k = 1; k <= j; k++)
                                x = x+y;
           次数为:     n*(n+1)*(n+2)/6
例 3.   int x = 91;   int y = 100;
           while(y>0)
          {     if(x>100) { x -= 10;   y--; }
                else x++;
          }
            1100 次
```

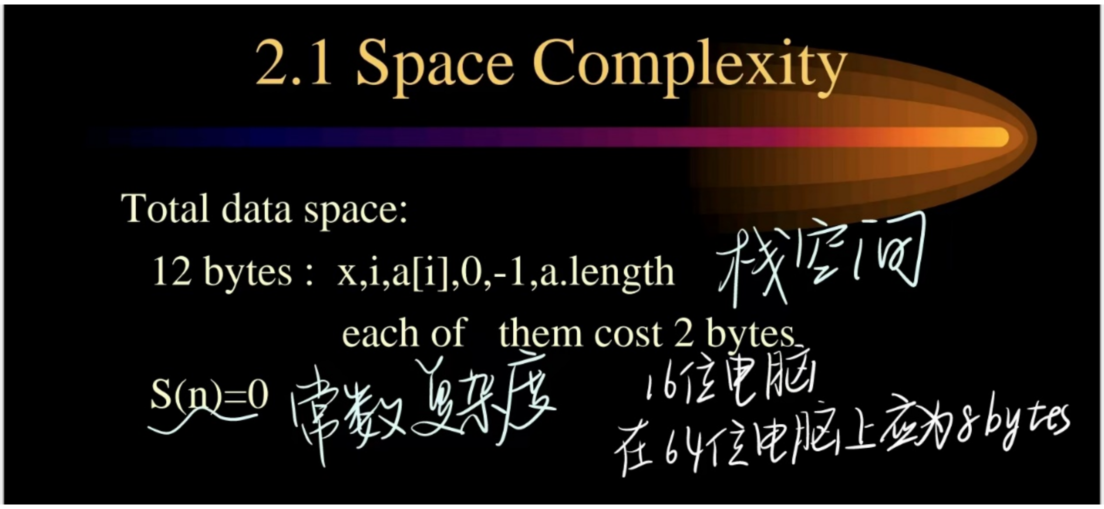


2.1 Space Complexity

2)example:

- Sequential Search

```
public  static int SequentialSearch( int [ ] a , int x )
{   int i;
    for(i=0; i<a.length &&a[i]!=x; i++)   ;
    if(i= = a.length) return –1;
    return i;
}
```



2.1 Space Complexity

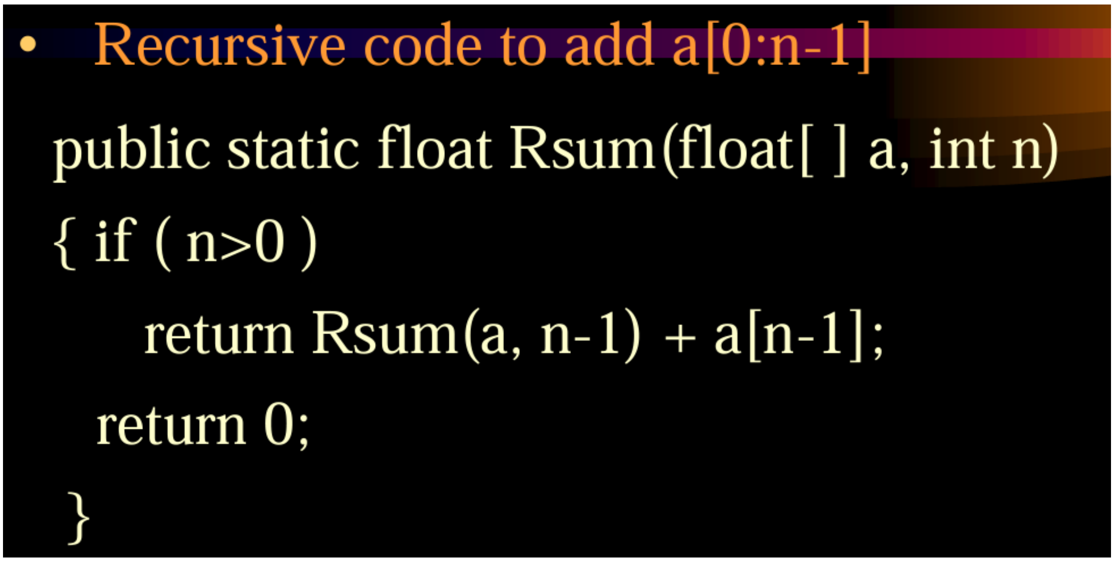
Total data space:

12 bytes : x,i,a[i],0,-1,a.length 栈空间

each of them cost 2 bytes 16位电脑

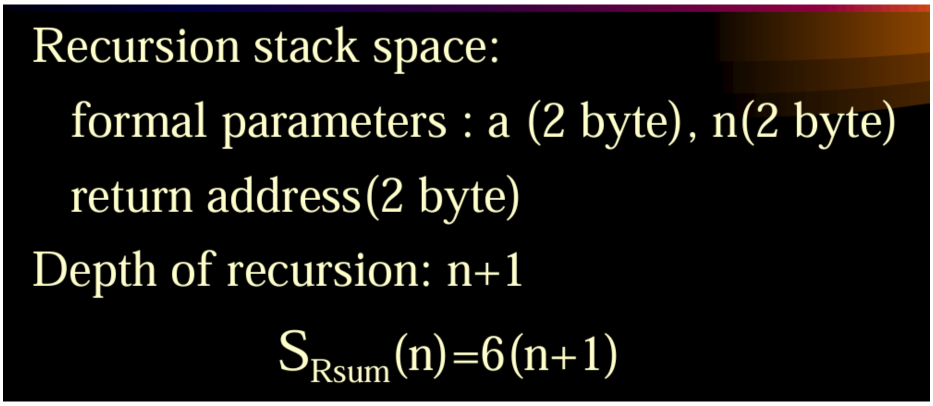
在64位电脑上应为8bytes

S(n)=0 常数复杂度



- Recursive code to add a[0:n-1]

```
public static float Rsum(float[ ] a, int n)
{ if ( n>0 )
      return Rsum(a, n-1) + a[n-1];
   return 0;
 }
```



Recursion stack space:

formal parameters : a (2 byte), n(2 byte)

return address(2 byte)

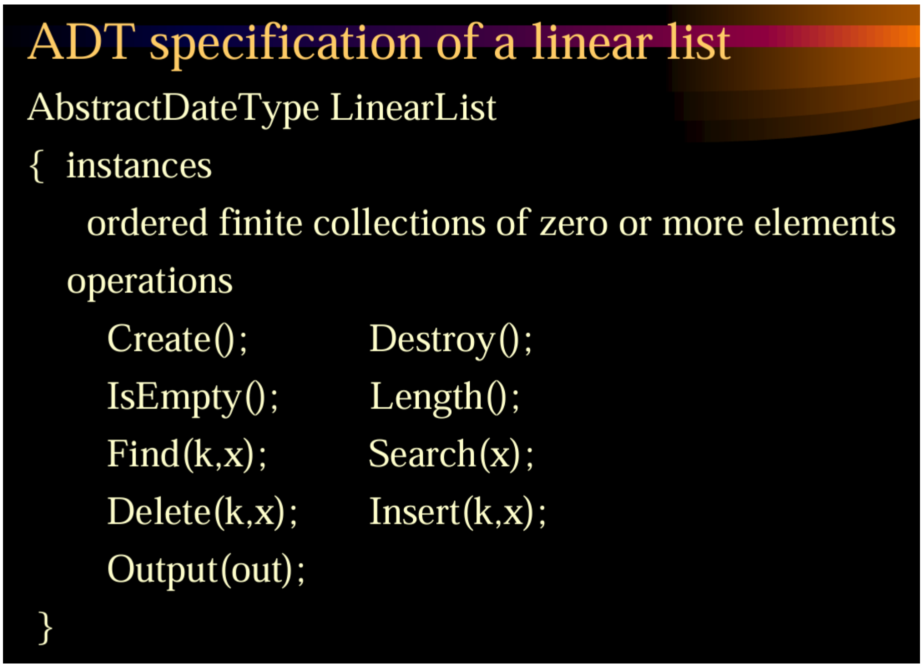
Depth of recursion: n+1

$$S_{Rsum}(n)=6(n+1)$$

一、第三章

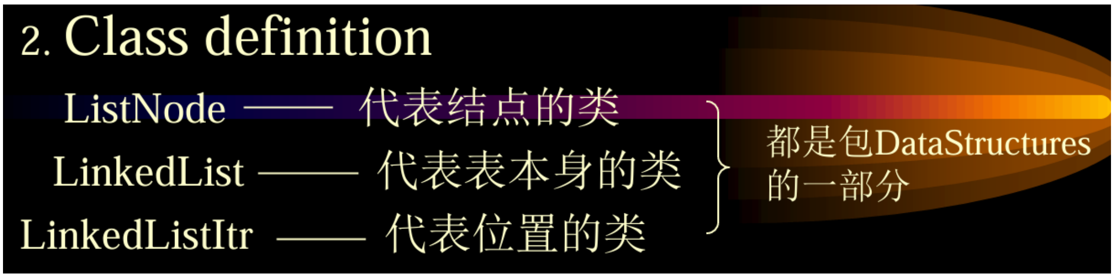
线性表的定义

线性表的代码以及各种代码的变体



ADT specification of a linear list

```
AbstractDateType LinearList
{  instances
      ordered finite collections of zero or more elements
   operations
      Create();          Destroy();
      IsEmpty();         Length();
      Find(k,x);         Search(x);
      Delete(k,x);       Insert(k,x);
      Output(out);
 }
```



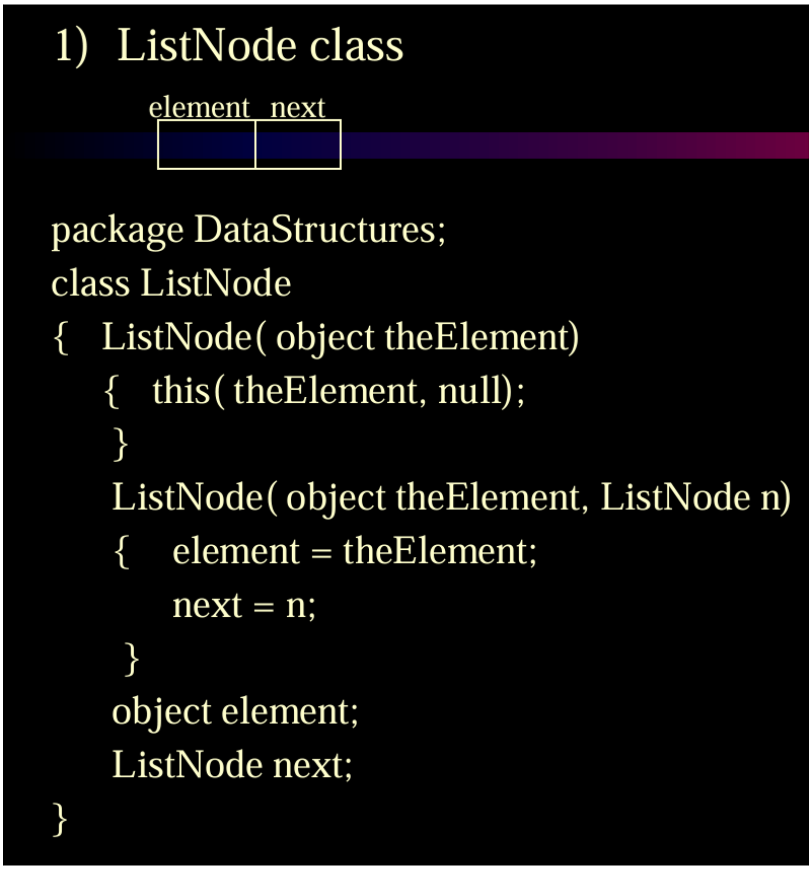
2. Class definition

ListNode —— 代表结点的类

LinkedList —— 代表表本身的类

LinkedListItr —— 代表位置的类

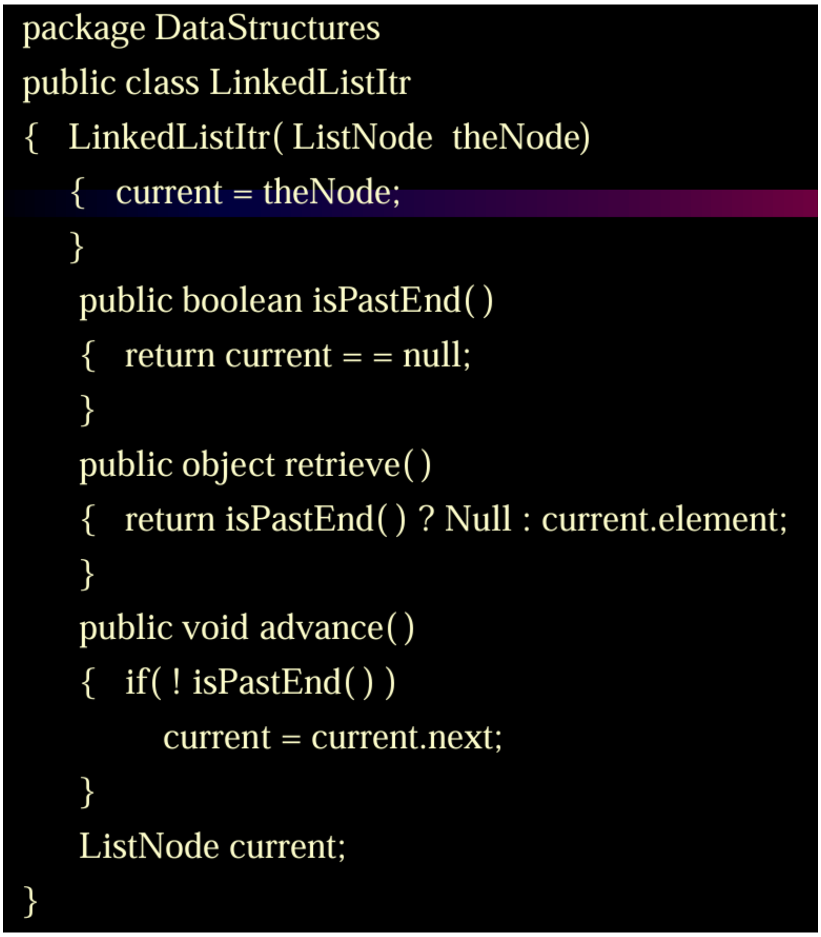
都是包DataStructures的一部分



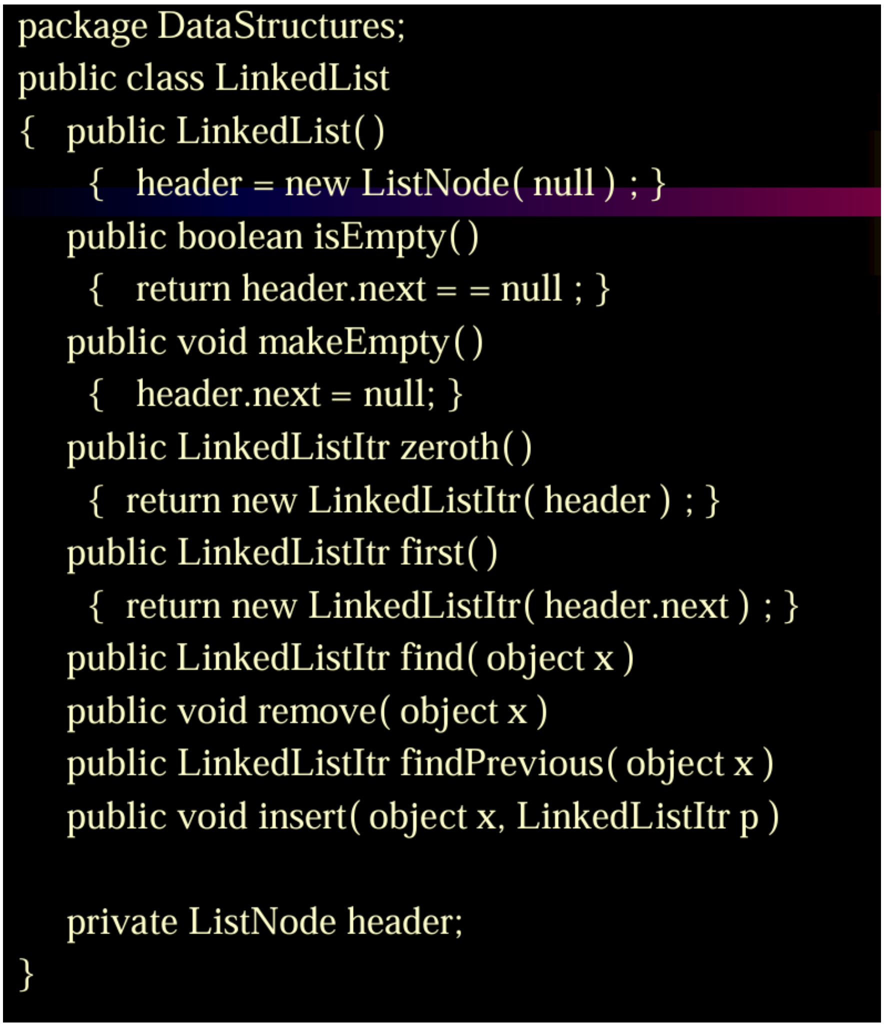
1) ListNode class

element
next

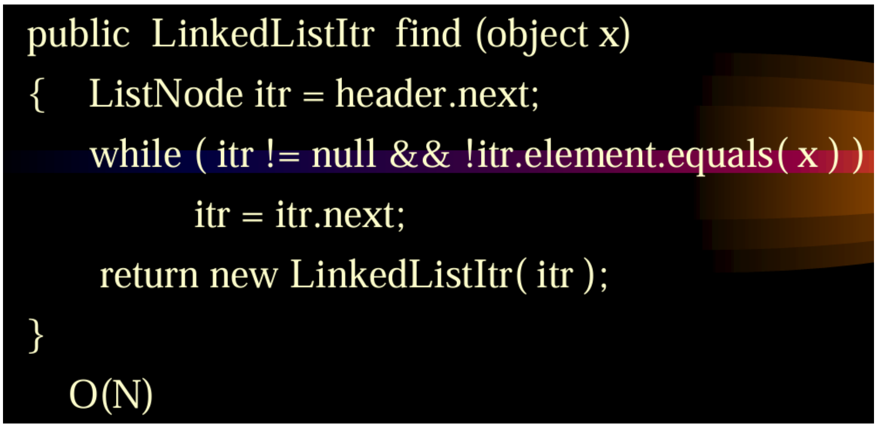
```
package DataStructures;
class ListNode
{   ListNode( object theElement)
    {   this( theElement, null);
     }
     ListNode( object theElement, ListNode n)
     {    element = theElement;
          next = n;
      }
     object element;
     ListNode next;
}
```



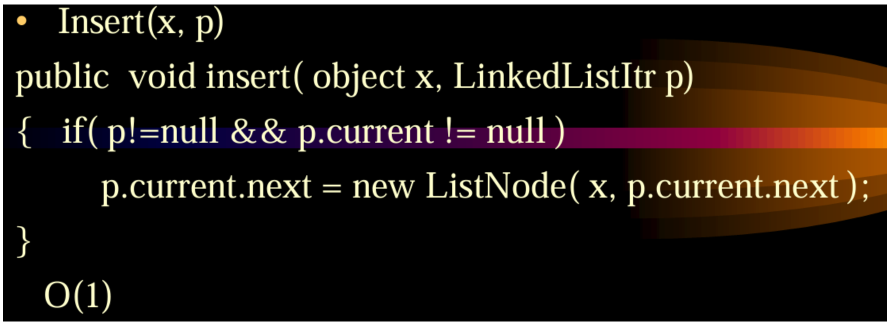
```
package DataStructures
public class LinkedListItr
{   LinkedListItr( ListNode  theNode)
    {   current = theNode;
    }
     public boolean isPastEnd( )
     {   return current = = null;
     }
     public object retrieve( )
     {   return isPastEnd( ) ? Null : current.element;
     }
     public void advance( )
     {   if( ! isPastEnd( ) )
             current = current.next;
     }
     ListNode current;
}
```



```
package DataStructures;
public class LinkedList
{   public LinkedList( )
      {   header = new ListNode( null ) ; }
    public boolean isEmpty( )
      {   return header.next = = null ; }
    public void makeEmpty( )
      {   header.next = null; }
    public LinkedListItr zeroth( )
      {  return new LinkedListItr( header ) ; }
    public LinkedListItr first( )
      {  return new LinkedListItr( header.next ) ; }
    public LinkedListItr find( object x )
    public void remove( object x )
    public LinkedListItr findPrevious( object x )
    public void insert( object x, LinkedListItr p )

    private ListNode header;
}
```



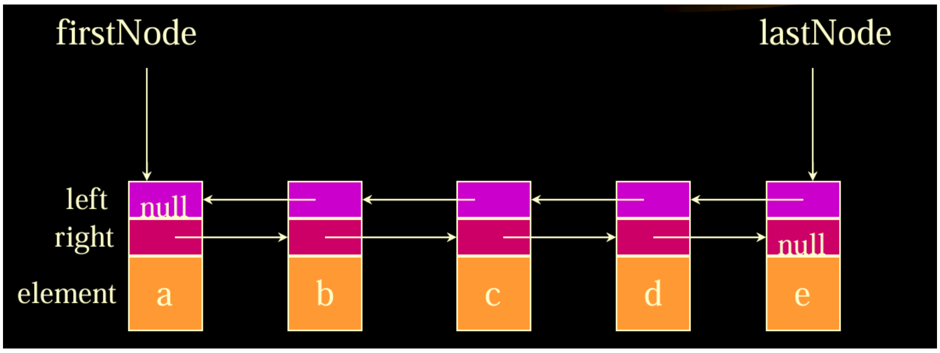
```
public  LinkedListItr  find (object x)
{    ListNode itr = header.next;
     while ( itr != null && !itr.element.equals( x ) )
               itr = itr.next;
      return new LinkedListItr( itr );
}
   O(N)
```



- Insert(x, p)

```
public  void insert( object x, LinkedListItr p)
{   if( p!=null && p.current != null )
        p.current.next = new ListNode( x, p.current.next );
}
  O(1)
```

双向链表



firstNode

lastNode

left

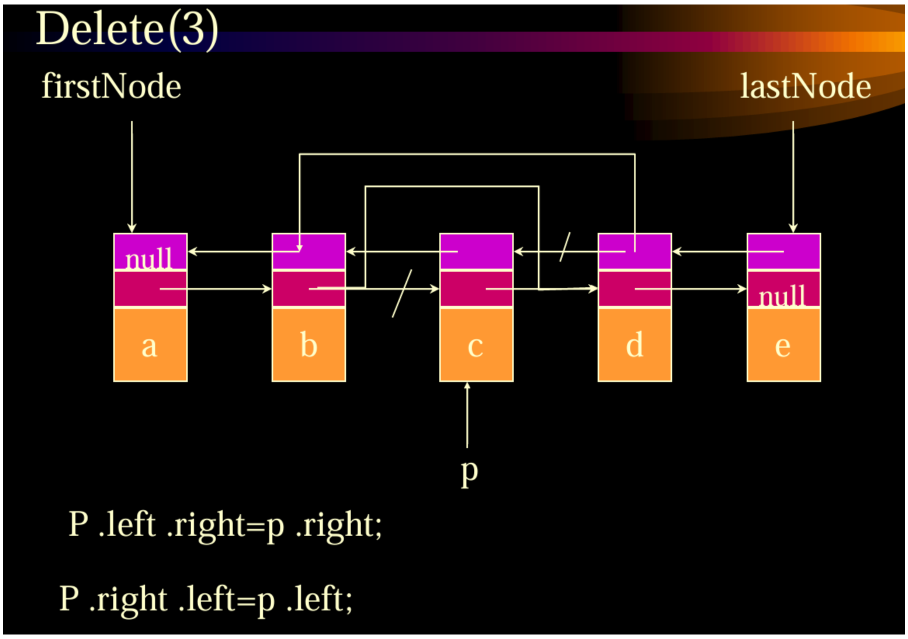
right

element

a	b	c	d	e

null

null



Delete(3)

firstNode

lastNode

null

null

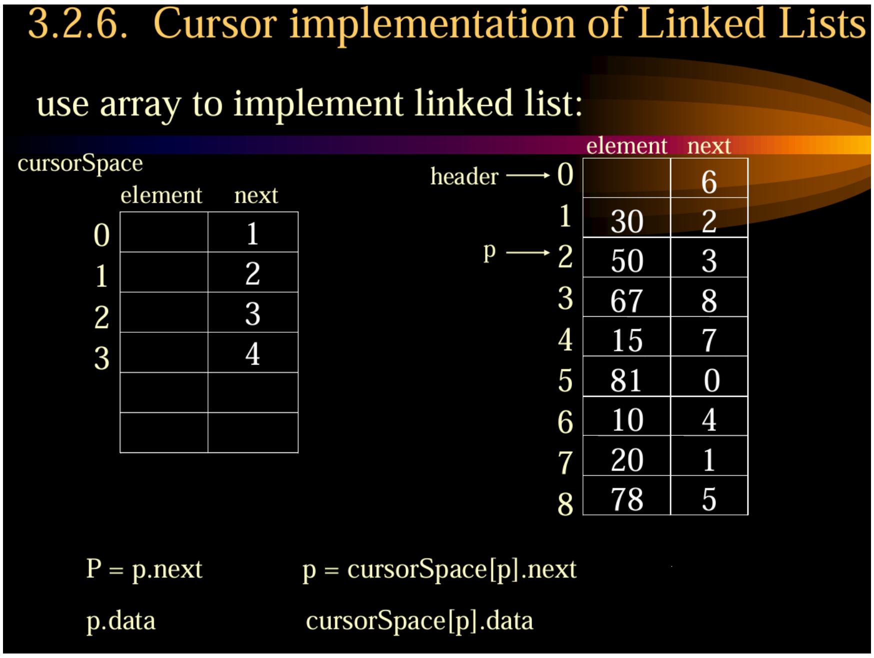
a	b	c	d	e

p

P .left .right=p .right;

P .right .left=p .left;

静态链表和含有游标的链表



3.2.6. Cursor implementation of Linked Lists

use array to implement linked list:

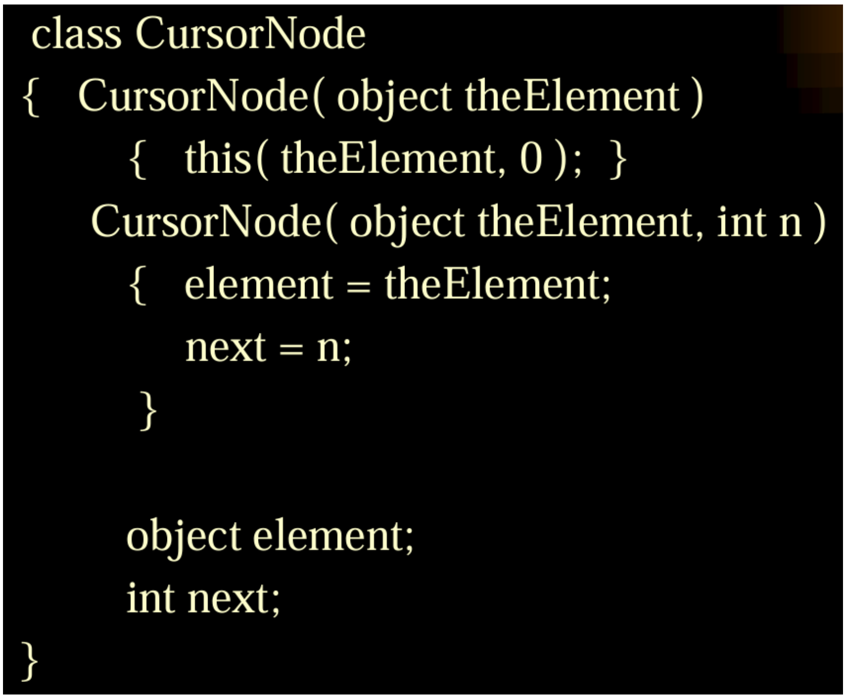
cursorSpace

	element	next
0		1
1		2
2		3
3		4

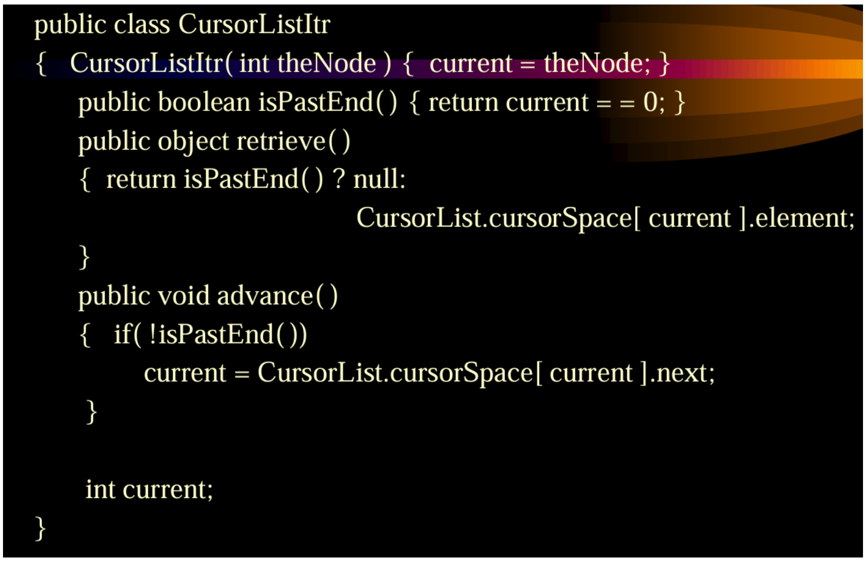
		element	next
header →	0		6
	1	30	2
p →	2	50	3
	3	67	8
	4	15	7
	5	81	0
	6	10	4
	7	20	1
	8	78	5

P = p.next　　　p = cursorSpace[p].next

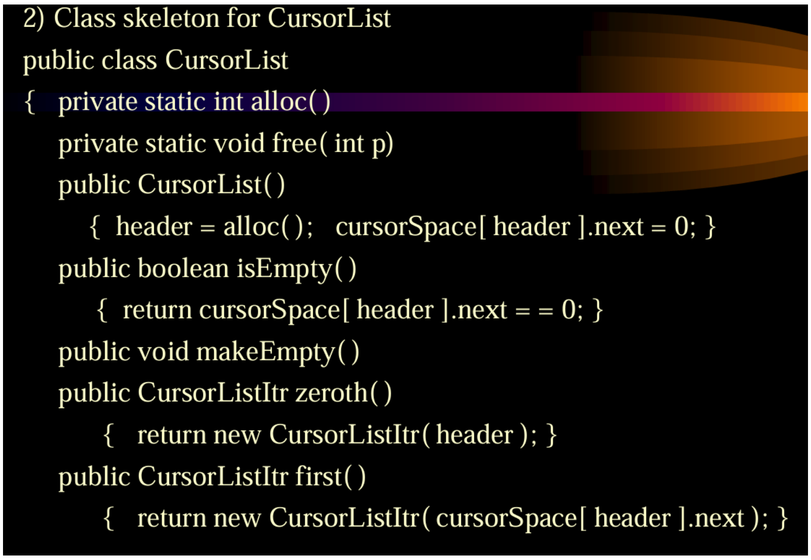
p.data　　　cursorSpace[p].data



```
class CursorNode
{   CursorNode( object theElement )
        {   this( theElement, 0 );  }
     CursorNode( object theElement, int n )
        {   element = theElement;
            next = n;
         }

        object element;
        int next;
}
```

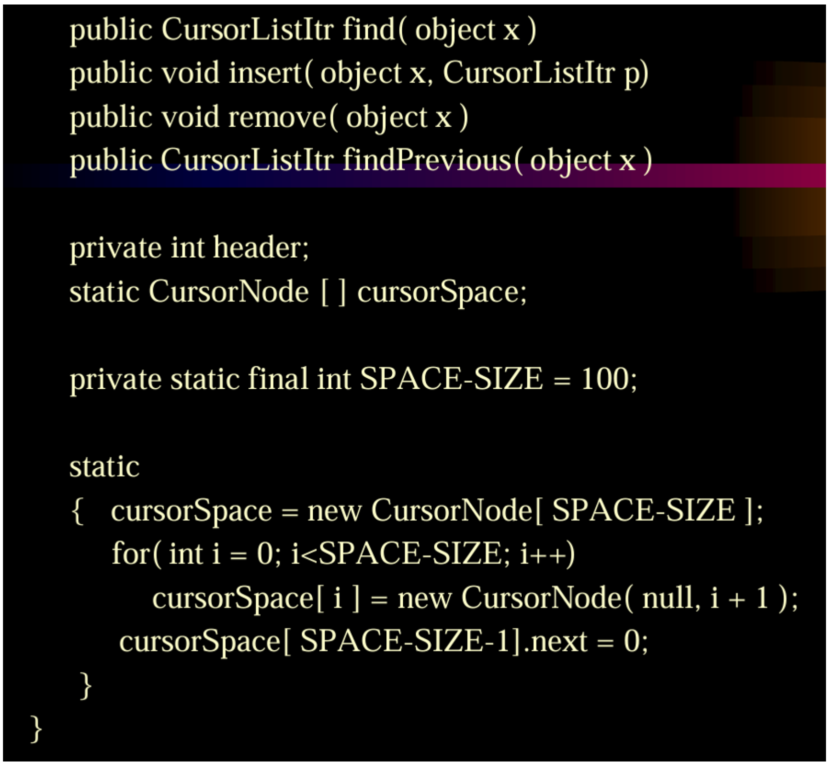


```
 public class CursorListItr
 {   CursorListItr( int theNode ) {  current = theNode; }
       public boolean isPastEnd( ) { return current = = 0; }
       public object retrieve( )
       {  return isPastEnd( ) ? null:
                                    CursorList.cursorSpace[ current ].element;
       }
       public void advance( )
       {   if( !isPastEnd( ))
                current = CursorList.cursorSpace[ current ].next;
        }

        int current;
 }
```

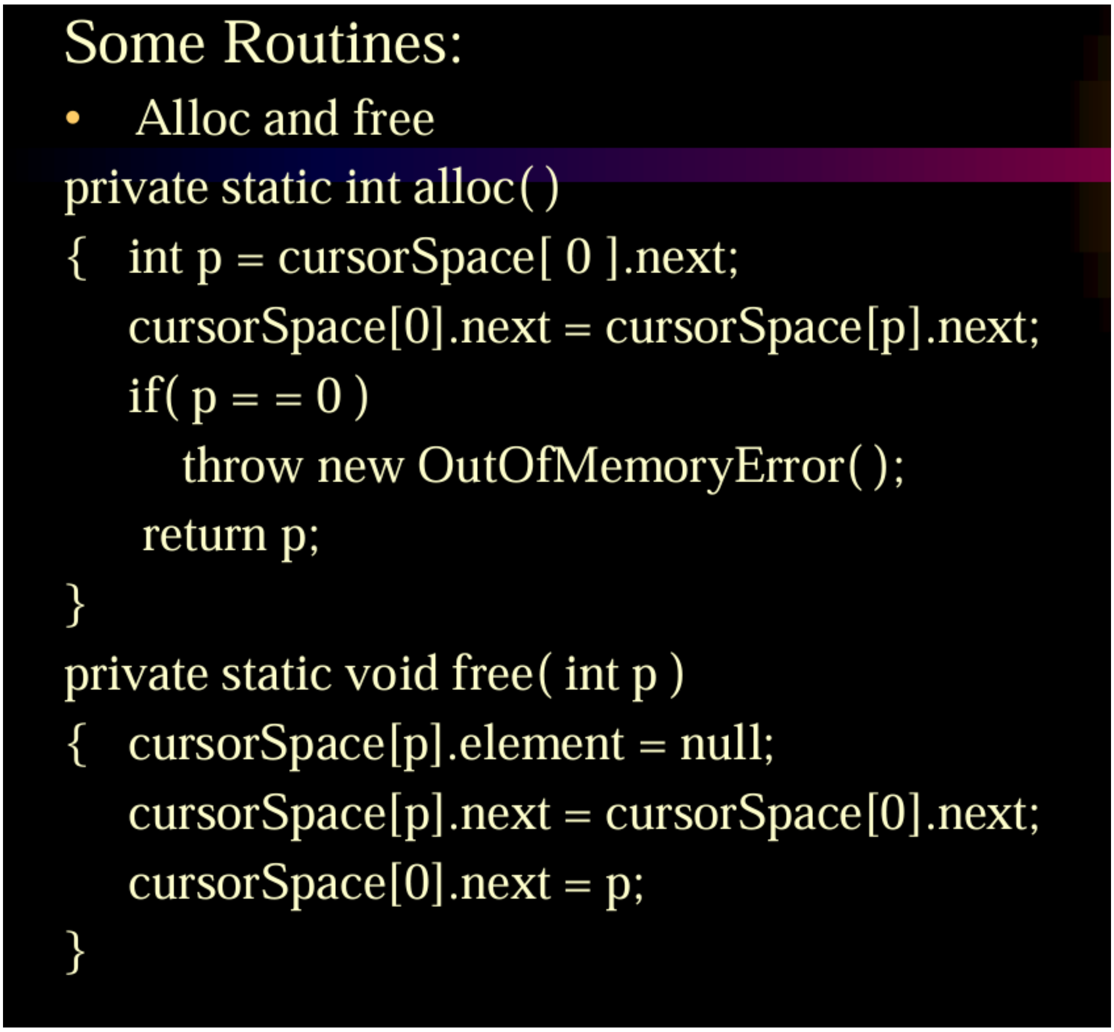


2) Class skeleton for CursorList

```
public class CursorList
{   private static int alloc( )
    private static void free( int p)
    public CursorList( )
       {  header = alloc( );   cursorSpace[ header ].next = 0; }
    public boolean isEmpty( )
       {  return cursorSpace[ header ].next = = 0; }
    public void makeEmpty( )
    public CursorListItr zeroth( )
        {   return new CursorListItr( header ); }
    public CursorListItr first( )
        {   return new CursorListItr( cursorSpace[ header ].next ); }
```



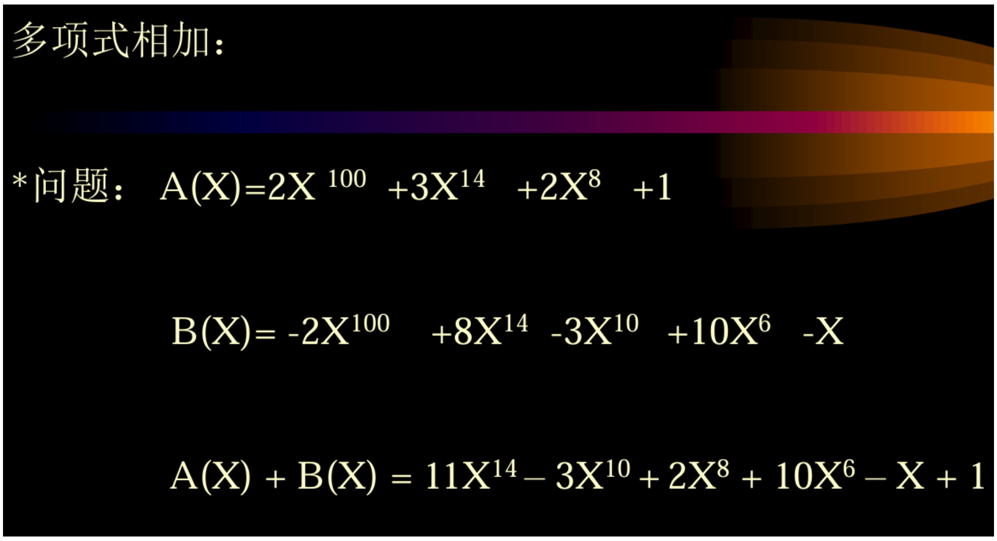
```
    public CursorListItr find( object x )
    public void insert( object x, CursorListItr p)
    public void remove( object x )
    public CursorListItr findPrevious( object x )

    private int header;
    static CursorNode [ ] cursorSpace;

    private static final int SPACE-SIZE = 100;

    static
    {   cursorSpace = new CursorNode[ SPACE-SIZE ];
        for( int i = 0; i<SPACE-SIZE; i++)
            cursorSpace[ i ] = new CursorNode( null, i + 1 );
         cursorSpace[ SPACE-SIZE-1].next = 0;
     }
}
```



Some Routines:

- Alloc and free

```
private static int alloc( )
{   int p = cursorSpace[ 0 ].next;
    cursorSpace[0].next = cursorSpace[p].next;
    if( p = = 0 )
        throw new OutOfMemoryError( );
     return p;
}
private static void free( int p )
{   cursorSpace[p].element = null;
    cursorSpace[p].next = cursorSpace[0].next;
    cursorSpace[0].next = p;
}
```

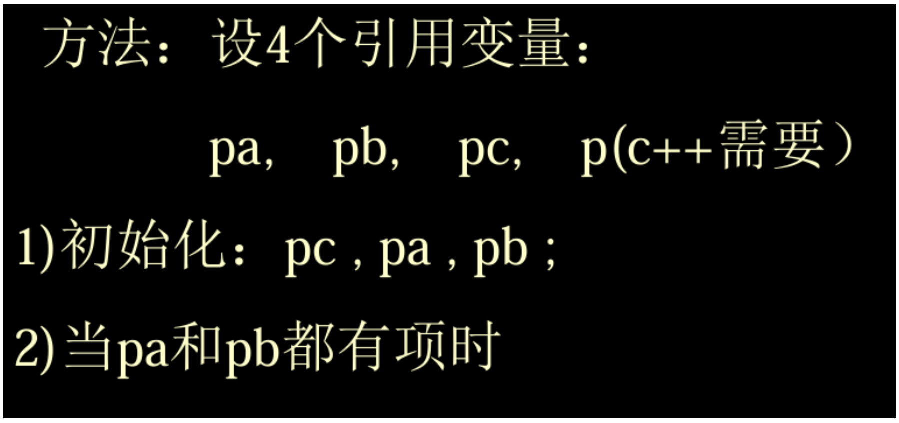


多项式相加：

*问题： $A(X)=2X^{100}+3X^{14}+2X^{8}+1$

$B(X)=-2X^{100}+8X^{14}-3X^{10}+10X^{6}-X$

$A(X)+B(X)=11X^{14}-3X^{10}+2X^{8}+10X^{6}-X+1$

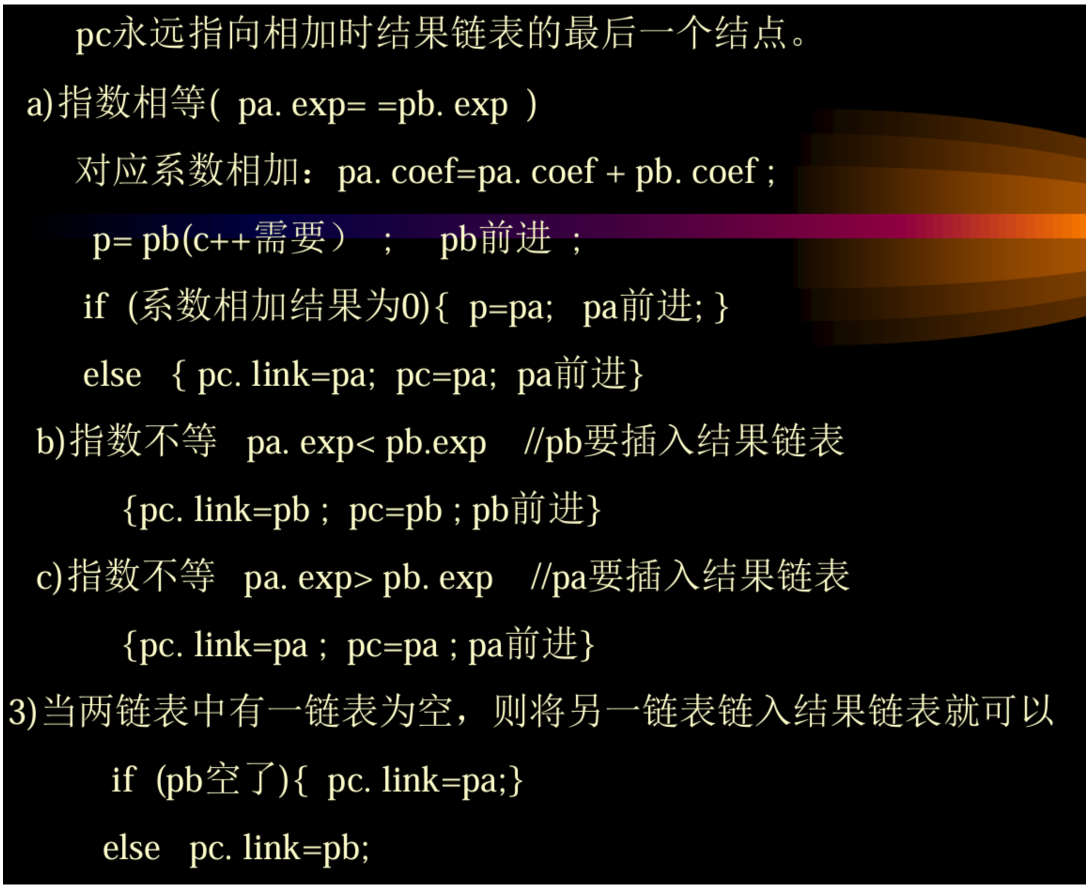


方法：设4个引用变量：

pa， pb， pc， p(c++需要）

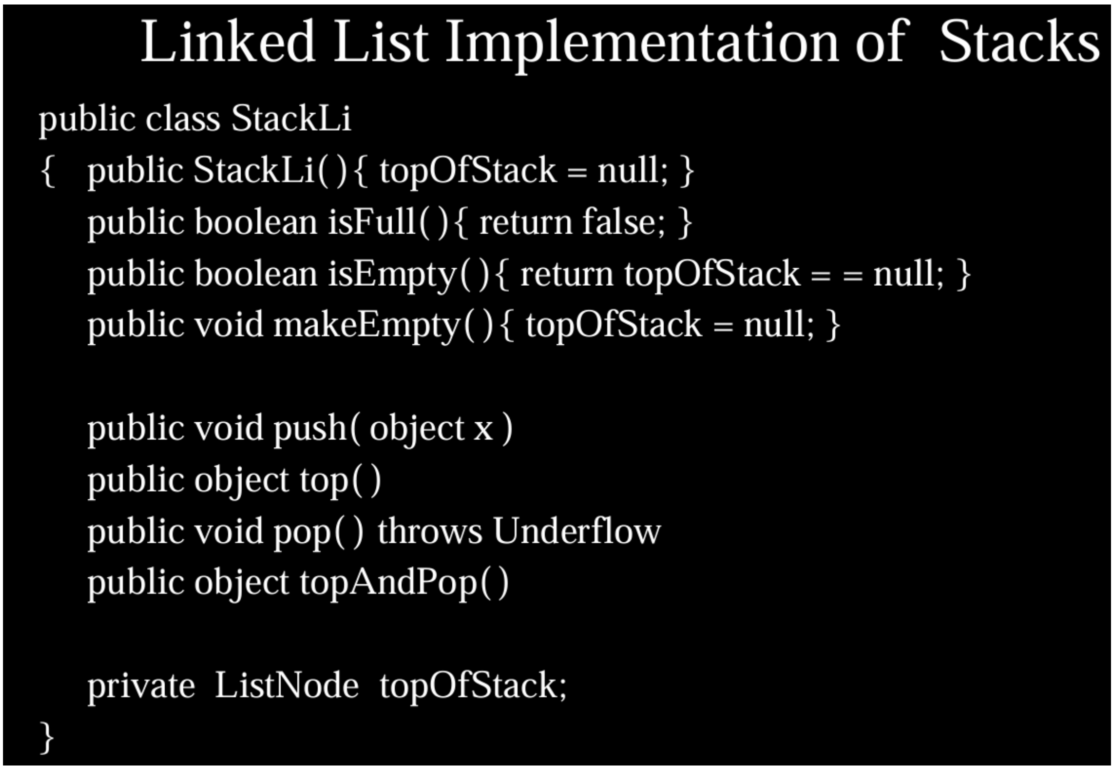
1)初始化：pc , pa , pb ;

2)当pa和pb都有项时



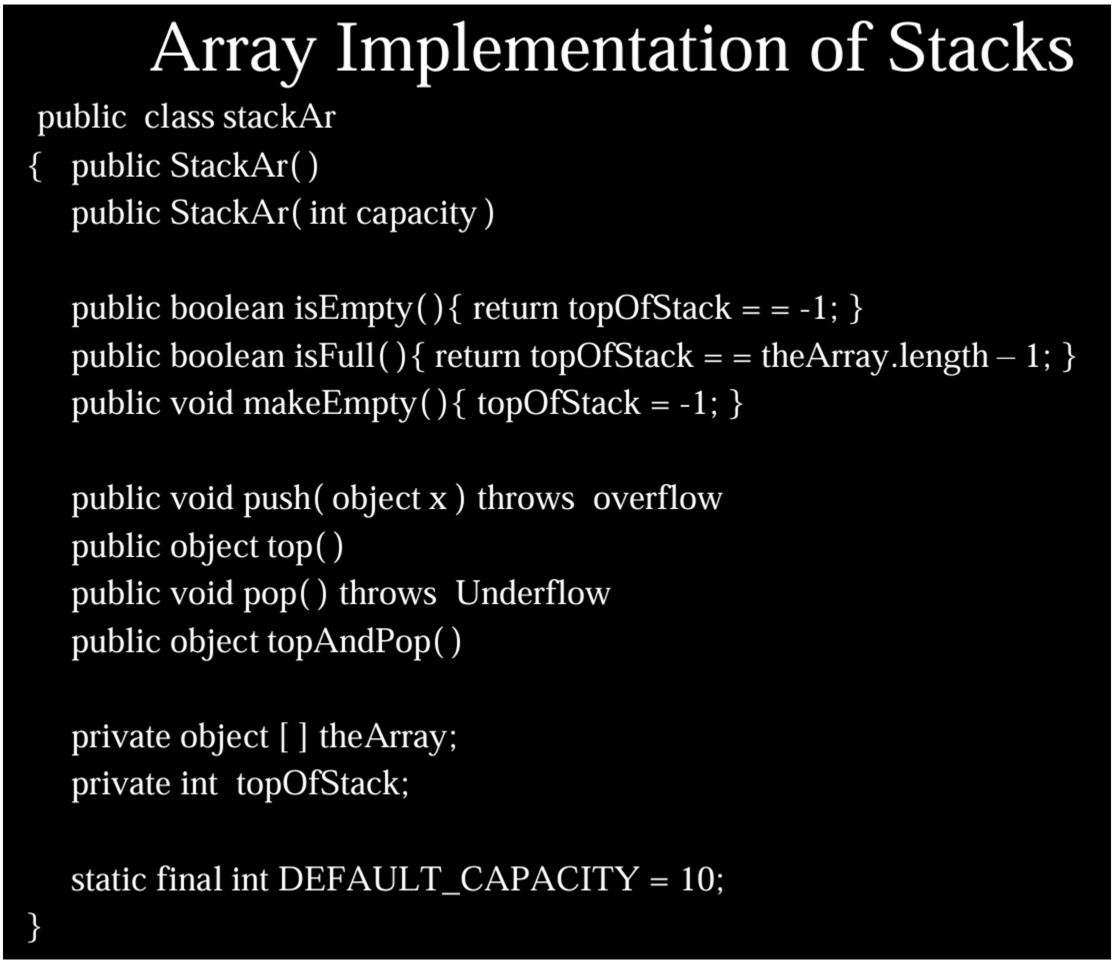
```
   pc永远指向相加时结果链表的最后一个结点。
 a)指数相等( pa. exp= =pb. exp ）
   对应系数相加：pa. coef=pa. coef + pb. coef ;
    p= pb(c++需要）  ；  pb前进 ；
    if (系数相加结果为0){ p=pa;  pa前进; }
    else  { pc. link=pa; pc=pa; pa前进}
 b)指数不等  pa. exp< pb.exp   //pb要插入结果链表
      {pc. link=pb ; pc=pb ; pb前进}
 c)指数不等  pa. exp> pb. exp   //pa要插入结果链表
      {pc. link=pa ; pc=pa ; pa前进}
3)当两链表中有一链表为空，则将另一链表链入结果链表就可以
     if (pb空了){ pc. link=pa;}
     else  pc. link=pb;
```

队列、栈的概念



Linked List Implementation of Stacks

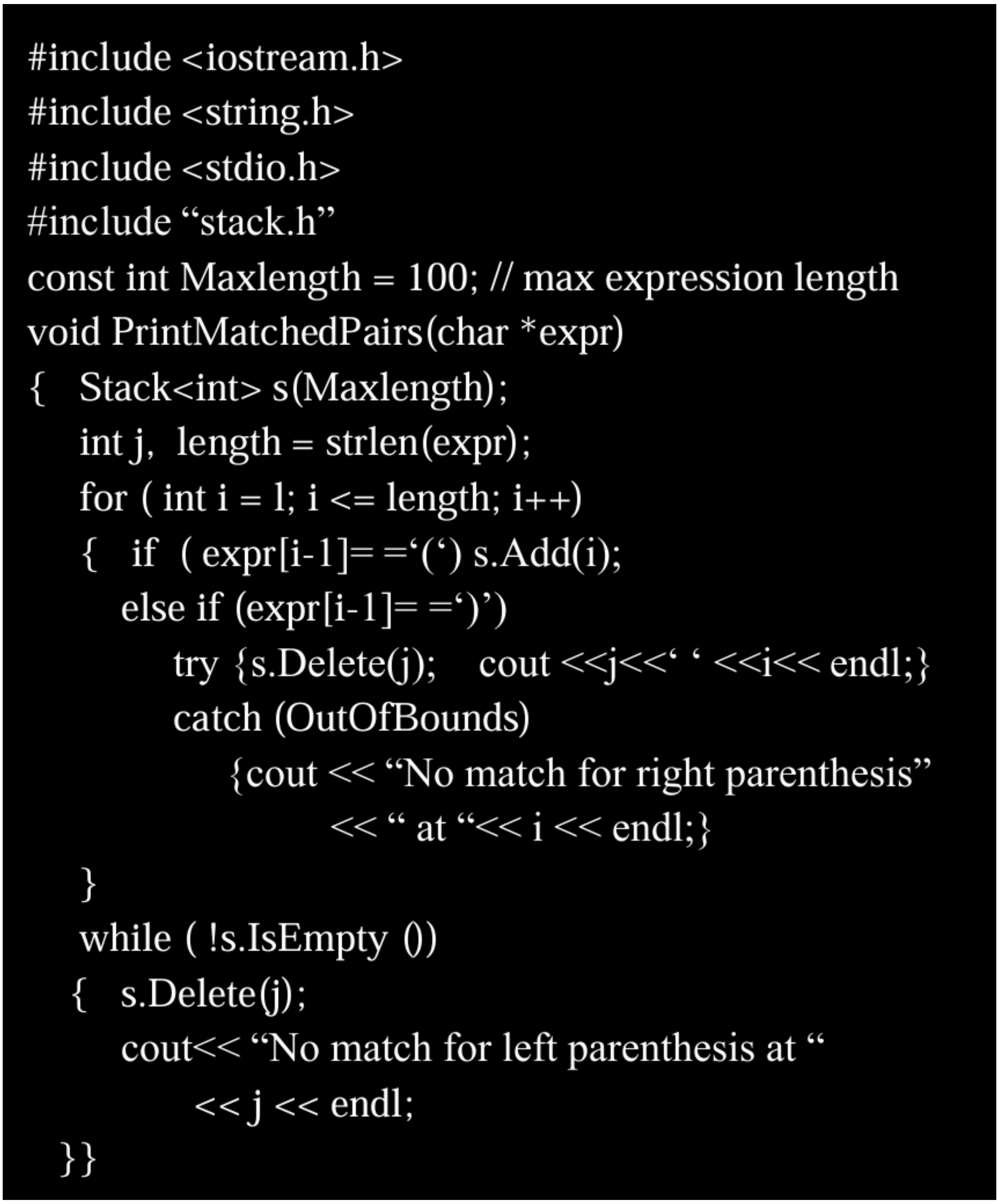
```
public class StackLi
{   public StackLi( ){ topOfStack = null; }
    public boolean isFull( ){ return false; }
    public boolean isEmpty( ){ return topOfStack = = null; }
    public void makeEmpty( ){ topOfStack = null; }

    public void push( object x )
    public object top( )
    public void pop( ) throws Underflow
    public object topAndPop( )

    private  ListNode  topOfStack;
}
```



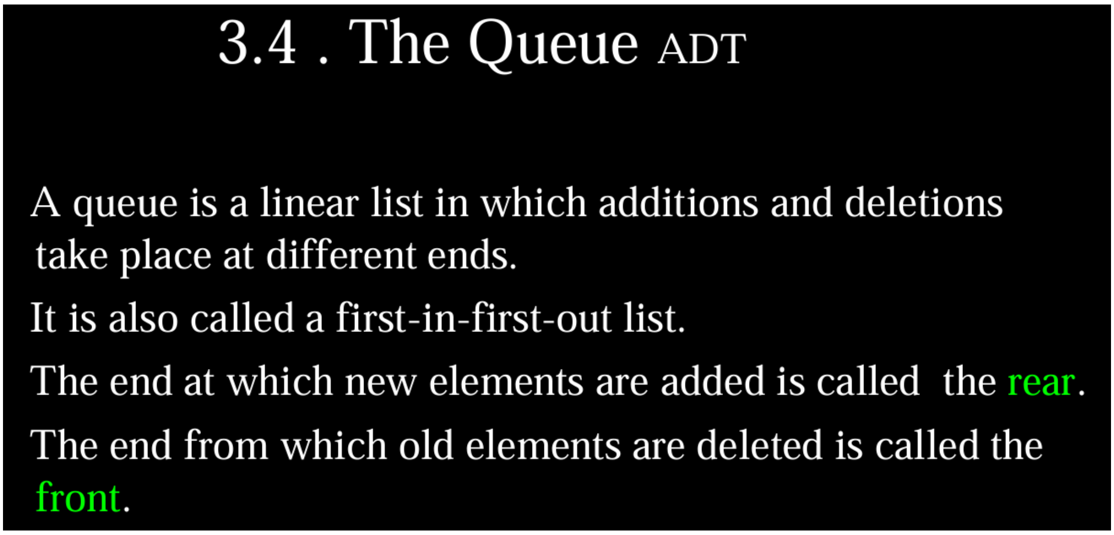
Array Implementation of Stacks

```
public  class stackAr
{   public StackAr( )
    public StackAr( int capacity )

    public boolean isEmpty( ){ return topOfStack = = -1; }
    public boolean isFull( ){ return topOfStack = = theArray.length – 1; }
    public void makeEmpty( ){ topOfStack = -1; }

    public void push( object x ) throws  overflow
    public object top( )
    public void pop( ) throws  Underflow
    public object topAndPop( )

    private object [ ] theArray;
    private int  topOfStack;

    static final int DEFAULT_CAPACITY = 10;
}
```

栈、队列的例子（重点看，括号匹配、表达式计算）



```
#include <iostream.h>
#include <string.h>
#include <stdio.h>
#include “stack.h”
const int Maxlength = 100; // max expression length
void PrintMatchedPairs(char *expr)
{   Stack<int> s(Maxlength);
    int j,  length = strlen(expr);
    for ( int i = l; i <= length; i++)
    {   if  ( expr[i-1]= =‘(‘) s.Add(i);
        else if (expr[i-1]= =‘)’)
            try {s.Delete(j);    cout <<j<<‘ ‘ <<i<< endl;}
            catch (OutOfBounds)
                {cout << “No match for right parenthesis”
                         << “ at “<< i << endl;}
    }
    while ( !s.IsEmpty ())
   {   s.Delete(j);
       cout<< “No match for left parenthesis at “
              << j << endl;
  }}
```



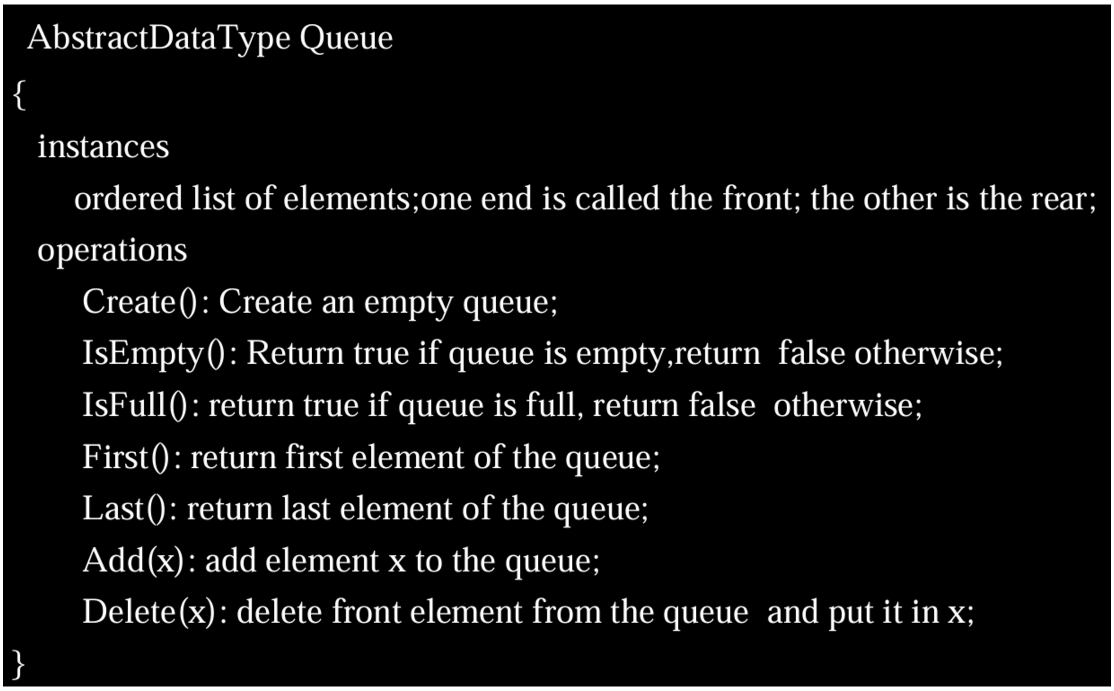
3.4 . The Queue ADT

A queue is a linear list in which additions and deletions take place at different ends.

It is also called a first-in-first-out list.

The end at which new elements are added is called the rear.

The end from which old elements are deleted is called the front.



```
 AbstractDataType Queue
{
  instances
      ordered list of elements;one end is called the front; the other is the rear;
  operations
       Create(): Create an empty queue;
       IsEmpty(): Return true if queue is empty,return  false otherwise;
       IsFull(): return true if queue is full, return false  otherwise;
       First(): return first element of the queue;
       Last(): return last element of the queue;
       Add(x): add element x to the queue;
       Delete(x): delete front element from the queue  and put it in x;
}
```

杨晖三角（不用重点看）