# Chapter 3.1

Stack and Queue

## 3.3 The Stack ADT

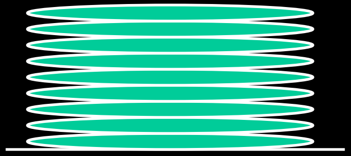
### 3.3.1. Stack Model

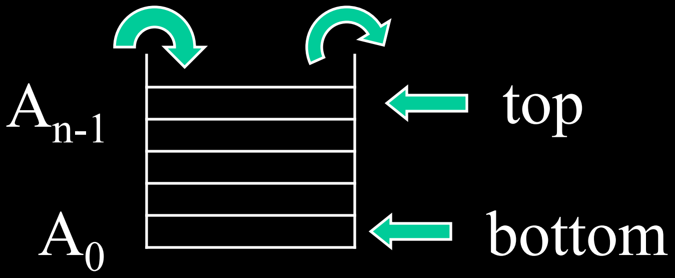
A stack is a list in which insertions and deletions take place at the same end.

This end is called the top. The other end of the list is called the bottom.

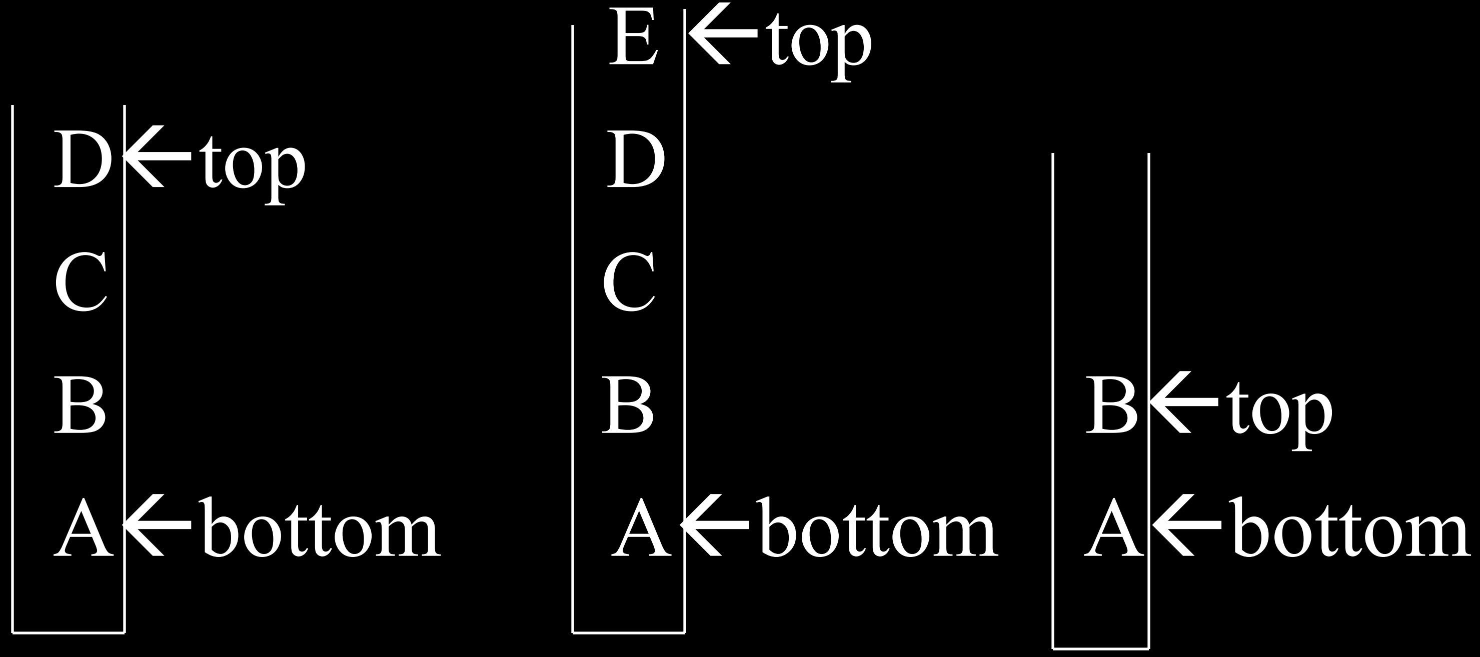
It is also called  a LIFO(last-in-first-out) list.

# Stack Model





## Stack Model

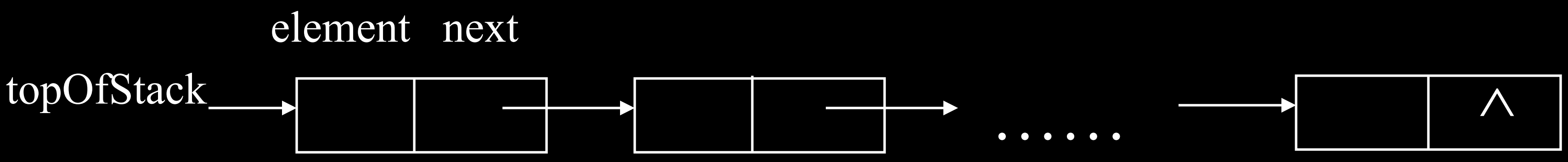


## Stack Model

```
AbstractDataType Stack{
   instances
      list of elements;  one end is called the  bottom;  the other is the top;
   operations
      Create():Create an empty stack;
      IsEmpty():Return true if stack is empty,return  false otherwise
      IsFull ():Return true if stack if full,return false otherwise;
      Top():return top element of the stack;
      Add(x): add element x to the stack;
      Delete(x):Delete top element from stack and put it in x;
   }
```

## 3.3.2. Implementation of Stack

1. Linked List Implementation of Stacks



when topOfStack= = null is empty stack

## 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;
}
```

Class  skeleton for linked list implementation of the stack ADT

## Linked List Implementation of  Stacks

Some Routine:

```
public void push( object x )
{  topOfStack = new ListNode( x, topOfStack );
}

public object top( )
{  if( isEmpty( ) )
        return null;
   return topOfStack.element;
}
```

## Linked List Implementation of Stacks

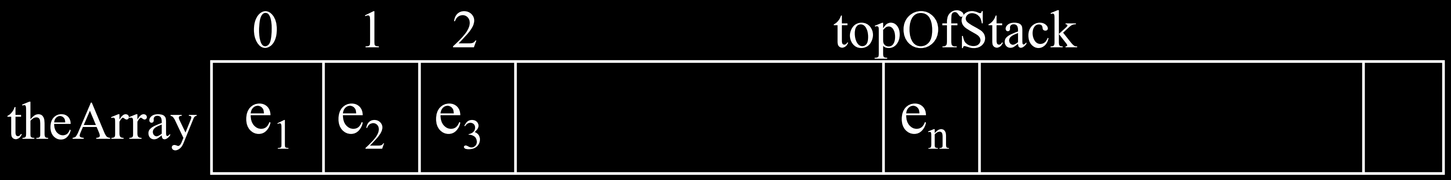
```
public void pop( ) throws Underflow
{  if( isEmpty( ) )
      throw new Underflow( );
     topOfStack = topOfStack.next;
}

public object topAndPop( )
{  if( isEmpty( ) )
      return null;

    object topItem = topOfstack.element;
    topOfStack = topOfStack .next;
    return topItem;
}
```

# 3.3.2. Implementation of Stack

2. Array Implementation of Stacks



when topOfStack= = -1 is empty stack

# 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;
}
```

Stack class skeleton---array implementation

## Array Implementation of Stacks

Some routine:

```
public StackAr( )
{   this( DEFAULT_CAPACITY );
}

public StackAr( int capacity )
{   theArray = new object [capacity ];
    topOfStack = -1;
}
```

Stack construction---array implementation

# Array Implementation of Stacks

```
 public void push( object x ) throws Overflow
{   if ( isfull( ) )  throw new Overflow( );
    theArray[ ++topOfStack ] = x;
}

 public object top( )
{   if( isEmpty( )
        return null;
    return theArray[ topOfStack ];
}
```

## Array Implementation of Stacks

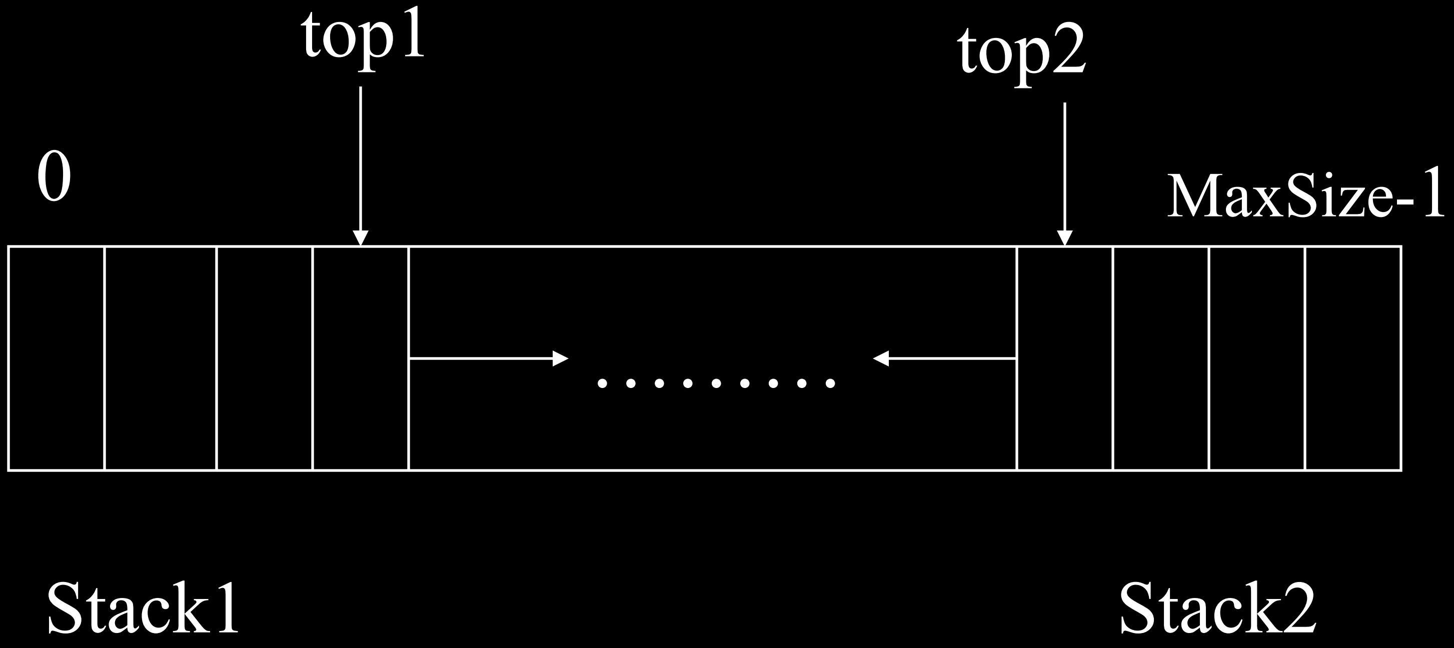
```
 public void pop( )  throws Underflow
{   if( isEmpty( ) )
       throw new Underflow( );
    theArray[ topOfStack-- ] = null;
}

public object topAndPop( )
{   if( isEmpty( ) )
       return null;
    object topItem = top( );
    theArray[ topOfStack-- ] = null;
    reurn topItem;
}
```

## Array Implementation of Stacks

- It is wasteful of space when multiple stacks are to coexist
- When there’s only two stacks, we can maintain space and time efficiency by pegging  the bottom of one stack at position 0 and the bottom of the other at position MaxSize-1. The two stacks grow towards the middle of the array.

## Array Implementation of Stacks



Two stacks in an array

## 3.3.3.Applications

1. Balancing  Symbols(Parenthesis Matching)
2. Expression  Evaluation

## 1. Parenthesis Matching

(a*(b+c)+d)

(a+b))(

1 2 3 4 5 6 7 8 9 10 11 121314 151617 181920212223

( d + ( a + b ) * c * ( d + e ) – f ) ) ( ( )

| | |
|---|---|
| 4 | 8 |
| 12 | 16 |
| 1 | 19 |
| 20 | 位置不匹配 |
| 22 | 23 |
| 21 | 位置不匹配 |

```
#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;
  }}
```

```
void static main(void)
{    char expr[MaxLength];
     cout<< “type an expression of length at most”
            <<MaxLength<<endl;
     cin.getline(expr, MaxLength);
     cout<<“the pairs of matching parentheses in “
           <<endl;
     puts(expr);
     cout<<“are”<<endl;
     printMatcnedPairs(expr);
}
```

O(n)

## 2.Expression Evaluation

- compiler:

  infix Expression $\longrightarrow$ postfix Expression $\longrightarrow$ postfix Expression Evaluation

  A*B-C*D# $\longrightarrow$ AB*CD*-#

  (A+B)*((C-D)*E+F)# $\longrightarrow$ AB+CD-E*F+*#

  无括号； 运算分量的次序不变； 运算符的次序就是具体执行的次序。

- postfix Expression Evaluation

  AB*CD*-#

- infix Expression $\longrightarrow$ postfix Expression

  A*B-C*D# $\longrightarrow$ AB*CD*-#

- postfix Expression Evaluation

**AB*CD*-#**

开辟一个运算分量栈

**1) 遇分量进栈;**

**2) 遇 运算符：取栈顶两个分量进行运算,栈中退了两个分量,并将结果进栈.**

```
enum Boolean {False, True};
#include<math.h>
#include <stack.h>
#include<iostream.h>
class calculator
{ public:
    calculator(int sz) :s (sz) { }
    void Run( );   void clear( );
  private:
    void AddOperand (double value) ;
    Boolean Get2Operands(double & left, double & right) ;
    void DoOperator (char op) ;
    stack <double> s ;}
```

```
void calculator:: AddOperand(double value)
{  s.Push(value);
}
Boolean calculator::Get2Operands(double &left,
                                 double &right)
{  if (s.IsEmpty())
     {cerr<<“Missing Operand!”<<endl; return false;}
   right = s.Pop();
   if (s.IsEmpty())
     {cerr<<“Missing Operand!”<<endl; return false;}
   left = s.Pop();
   return true;
 }
```

```
void calculator :: DoOperator(char op)
{  double left, right;     Boolean  result;
   result = Get2Operands(left, right);
   if (result= =true)
      Switch (op)
      {  case ‘+’: s.Push (left + right) ; break;
         case ‘-’ : s.Push (left - right) ; break;
         case ‘*’ : s.Push (left * right) ; break;
         case ‘/’ : if (right = = 0.0)
                    {cerr << “divide by 0!” << endl; s.Clear ( ); }
                    eles s.Push(left / right) ; break;
          case ‘^’ : s.Push (power (left, right) ); break;
      }
   else s.Clear ( );
}
```

```
void Calculator:: Run ()
{  char ch;   double newoperand;
   while ( cin >>ch ,   ch != ‘#‘)
    {  switch (ch)
        { case ‘+’: case ‘-’: case ‘*’: case ‘/’: case ‘^’:
              DoOperator (ch);  break;
            default:  cin . Putback(ch);
                      cin >> newoperand;
                      AddOperand(newoperand);
                      break;
        }
    }
}
```

```
void Calculator::clear()
   {   s.MakeEmpty();
   }

#include <Calculator.h>
 void main ()
    {  Calculator CALC(10);
       CALC.Run();
    }
```

## • infix Expression $\longrightarrow$ postfix Expression

A+B*C#--------→ ABC*+#

1) 遇运算分量(操作数)直接输出

2) 遇运算符：比较当前运算符与栈顶运算符的优先级. 若当前运算符的优先级<=栈顶运算符的优先级，则不断取出运算符栈顶输出；否则进栈. 因此一开始栈中要放一个优先级最低的运算符，假设为“#”， 例子： A+B+C； A*B-C

(A+B)*(C-D)#------→ AB+CD-*#

3）遇 ‘（’ ：压栈，

每个运算符有双重优先级.

4）遇 ‘）’ : 不断退栈输出，直到遇到’(’

5）遇 ‘#’ : 将栈中的所有运算符出栈打印，过程终止。

## infix Expression $\longrightarrow$ postfix Expression

```
void postfix (expression e)
 {  Stack <char> s ; char ch,  y ;
    s.MakeEmpty ( ) ;  s.Push (‘#’);
    while (cin.get ( ch ) , ch != ‘#’)
    {   if (isdigit ( ch )) cout << ch;
        else  if (ch = = ‘)’)
                 for (y=s.Pop( );  y!= ‘(‘;  y=s.Pop( )) cout << y ;
              else
                 { for (y = s.Pop ( ); isp (y) > icp (ch); y=s.Pop( ))
                                                  cout<<y ;
                   s.Push (y) ; s.Push (ch); }
    }
    while (! s.IsEmpty ( ))  { y = s.Pop ( ) ;  cout <<y ; }
 }
```

如果合为一步怎么做？实习题。

# Chapter 3----stack

2010年全国统考题

1、若元素a,b,c,d,e,f依次进栈，允许进栈、退栈操作交替进行。但不允许连续三次进行退栈工作，则不可能得到的出栈序列是（ ）

A：dcebfa B：cbdaef C：bcaefd D：afedcb

## 实习题：

5. 中缀表达式 $\longrightarrow$ 后缀表达式 $\longrightarrow$ 对后缀表达式求值, 合为一趟来做。

## 3.4 . The Queue ADT

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

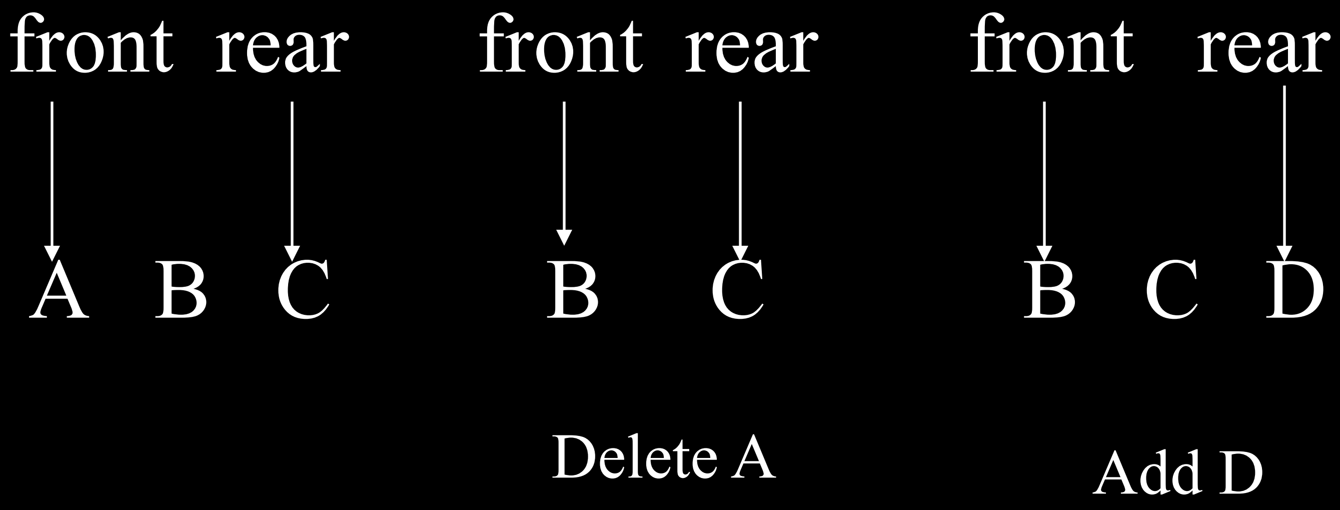
It is also called a first-in-first-out list.

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

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

## 3.4.1. Queue Model

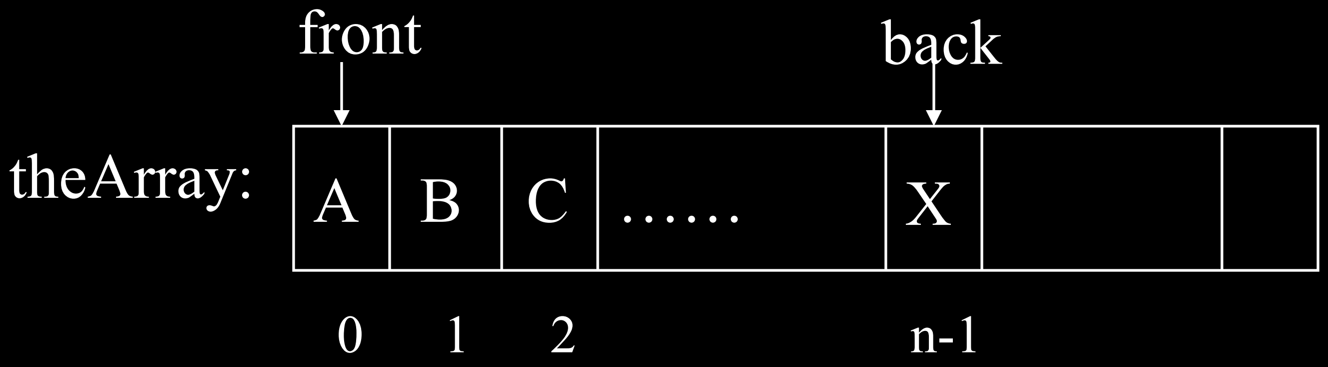
Sample queues



## Queue Model

```
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;
}
```

## 3.4.2. Array Implementation of Queue



currentSize

the queue size : currentSize;

an empty queue has currentSize= = 0;

an full queue has currentSize= = theArray.length;

## 3.4.2. Array Implementation of Queue

To add an element:

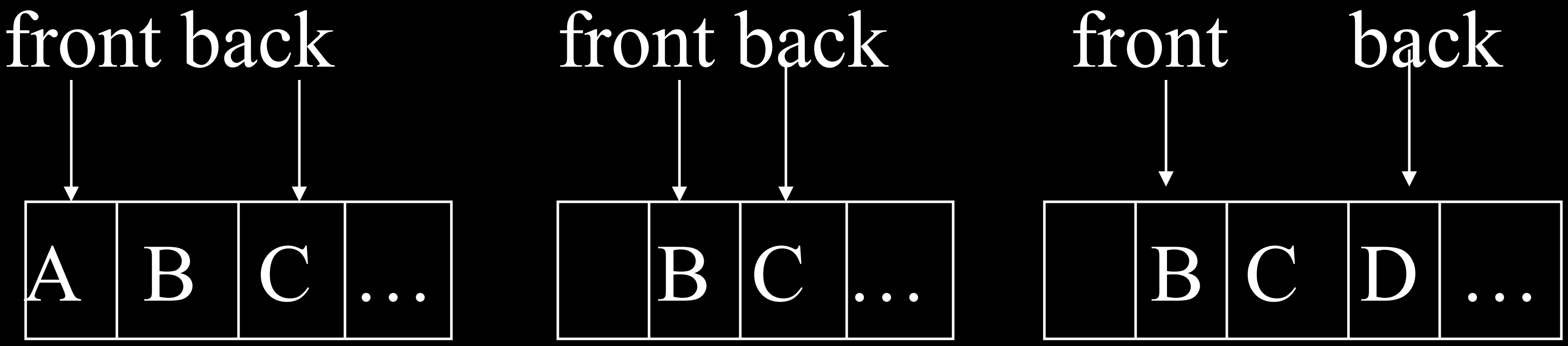
```
back=back+1;  theArray[back]=x;
```

To delete an element: two methods:

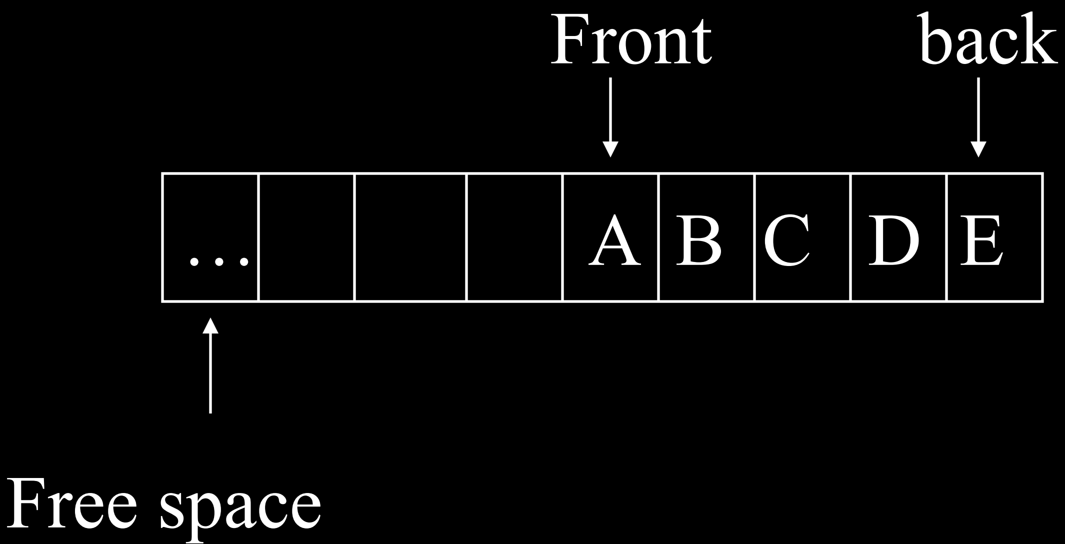
1) front=front+1;    O(1)

2) shift the queue one position left.   O(n)

## 3.4.2. Array Implementation of Queue

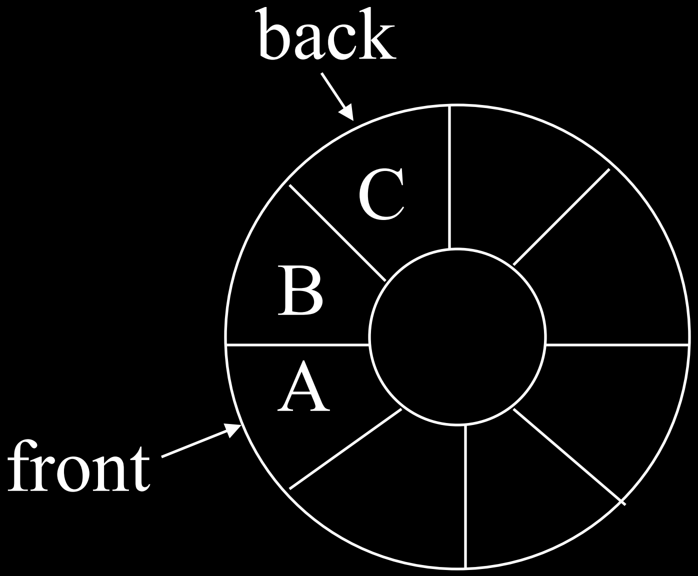


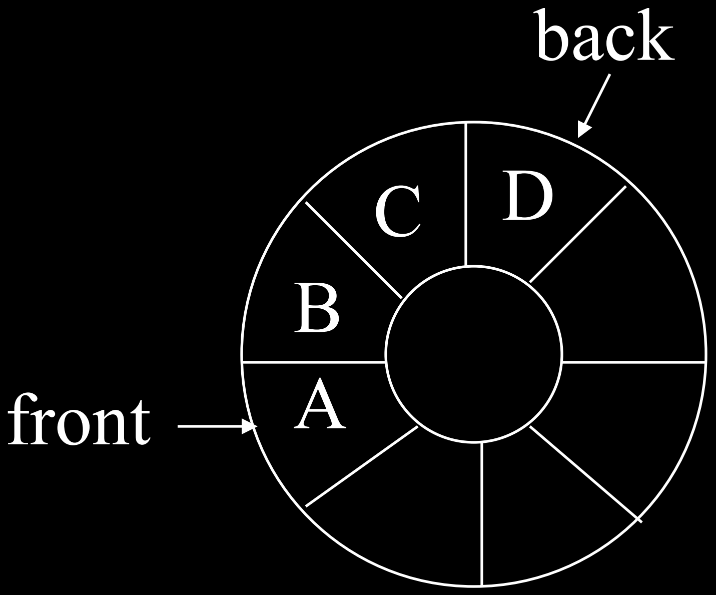
# 3.4.2. Array Implementation of Queue



## 3.4.2. Array Implementation of Queue

to use a circular array to represent a queue

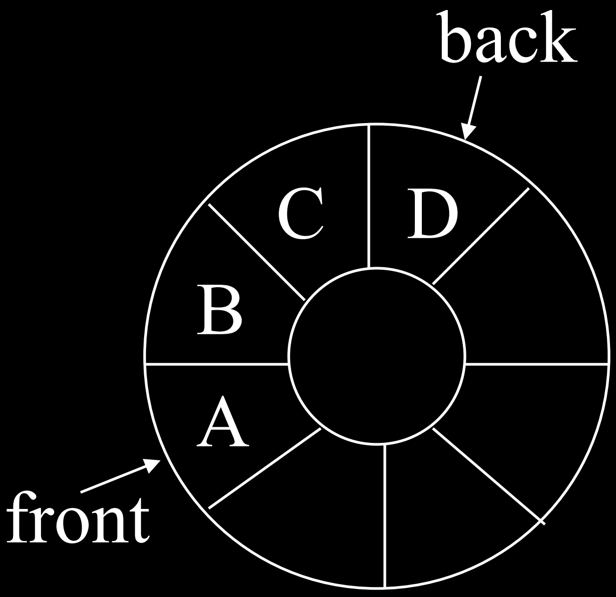


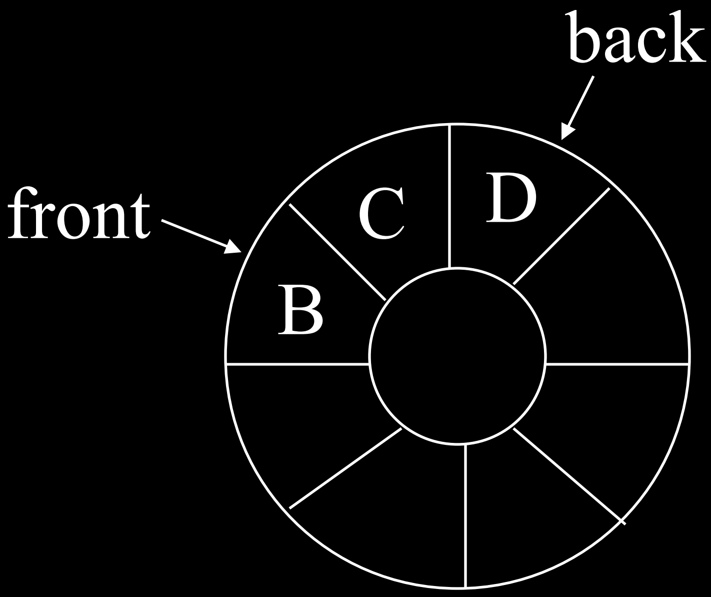


Addtion

## 3.4.2. Array Implementation of Queue

deletion





deletion

## 3.4.2. Array Implementation of Queue

How implementation a circular array:

1) When front or back reachs theArray.length-1, reset 0
2) back = (back+1) % theArray.length
   front = (front + 1) % theArray.length

# 3.4.2. Array Implementation of Queue

```
Public class QueueAr
{   public QueueAr( )
    public QueueAr( int capacity )
    public boolean isEmpty( ){ return currentsize = = 0; }
    public boolean isfull( ){ return currentSize = = theArray.length; }
    public void makeEmpty( )
    public Object getfront( )
    public void enqueue( Object x ) throw Overflow
    private int increment( int x )
    private Object dequeue( )

    private Object [ ] theArray;
    private int currentSize;
    private int front;
    private int back;

    static final int DEFAULT_CAPACITY = 10;
}
```

## 3.4.2. Array Implementation of Queue

```
 public QueueAr( )
{   this(  DEFAULT_CAPACITY );
}

 public QueueAr( int capacity )
{   theArray = new Object[ capacity ];
    makeEmpty( );
}

 public void makeEmpty(  )
{   currentSize = 0;
    front = 0;
    back = -1;
}
```

## 3.4.2. Array Implementation of Queue

```
public void enqueue( object x ) throw Overflow
{   if( isFull( ) )
        throw new Overflow( );
    back = increment( back );
    theArray[ back ] = x;
    currentSize++;
}

private  int increment( int x )
{   if( ++x = = theArray.length )
          x = 0;
    return x;
}
```

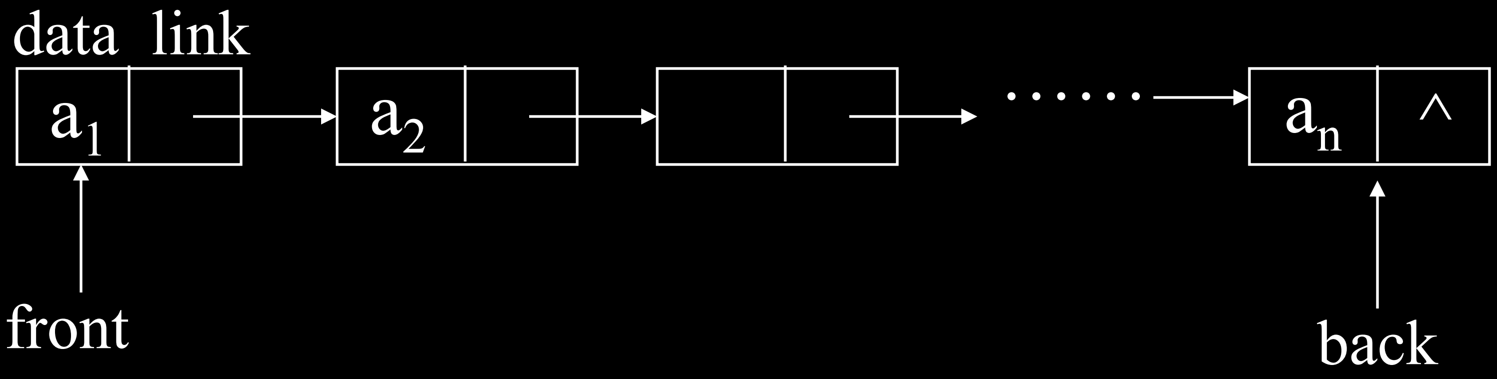
## 3.4.2. Array Implementation of Queue

```
public Object dequeue( )
{   if( isEmpty( ) )
        return null;
    currentSize--;

    Object fromtItem = theArray[ front ];
    theArray[ front ] = null;
    front = increment( front );
    return frontItem;
}
```

## 3.4.3 Linked Representation of queue

Linked queues



## 3.4.3 Linked Representation of queue

### Class definition for a linked queue

```
template<class T>class LinkedQueue
{  public:
      LinkedQueue(){front=back=0;}
      ~LinkedQueue();
      bool IsEmpty()const{return ((front)?false:true);}
      bool IsFull()const;
      T First()const;
      T Last()const;
      LinkedQueue<T>&Add(const T& x);
      LinkedQueue<T>& Delete(T& x);
  privarte:
     Node<T>*front;    Node<T>*back;
};
```

## 3.4.3 Linked Representation of queue

1)destructor

```
 template<class T>
 LinkedQueue<T>::~LinkedQueue()
 {  Node<T>*next;
    while(front)
      {
        next=front.link;
        delete front;
        front=next;
      }
}
```

## 3.4.3 Linked Representation of queue

2)Add(x)

```
template<class T>
LinkedQueue<T>& LinkedQueue<T>::Add(const T&x)
{   Node<T>*p=new Node<T>;
    p. data=x;
    p. link=0;
    if(front) back. link=p;
    else front=p;
    back=p;
    return *this;
}
```

## 3.4.3 Linked Representation of queue

3)Delete(x)

```
template<class T>
LinkedQueue<T>& LinkedQueue<T>::Delete(T& x)
{  if(IsEmpty())throw OutOfBounds();
   x=front. data;
   Node<T>* p=front;
   front=front. link;
   delete p;
   return *this;
}
```

## 3.4.4 Application

1) Print the coefficients of the binomial expansion $(a+b)^i$ , i=1,2,3,…,n

```
            1   1
          1   2    1
        1   3    3    1
      1    4    6   4    1
    1   5   10    10   5   1
  1   6    15  20   15   6    1
```

## Print the coefficients of the binomial expansion

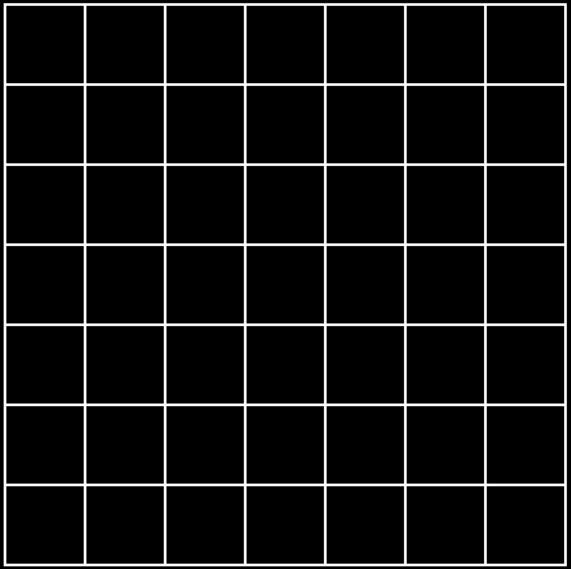
```
#include <stdio.h>
#include <iostream.h>
#include “queue.h”
void YANGHUI(int n)
{  Queue<int> q;   q.makeEmpty( );
   q.Enqueue(1);    q.Enqueue(1);
   int s=0;
   for (int i=1; i<=n;i++)
     { cout << endl;
       for (int k=1;k<=10-i;k++) cout<<‘ ‘;
       q.Enqueue(0);
       for (int j=1;j<=i+2;j++)
          {   int t=q.Dequeue( );
              q.Enqueue(s+t);
              s=t;
              if (j!=i+2) cout<< s <<‘ ‘;
          }
      }
}
```

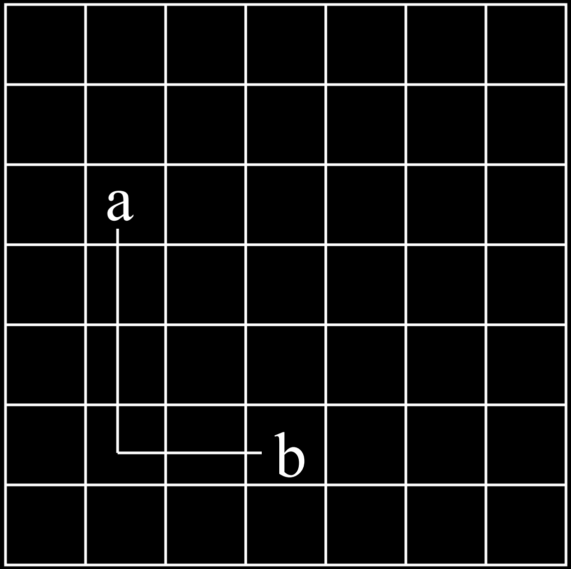
## Print the coefficients of the binomial expansion

用可变长度的二维数组来实现：

```
public class Yanghui
{   public static void main(String args[ ] )
    {    int n = 10;
         int mat[ ][ ] = new int [n ][ ];   //申请第一维的存储空间
         int i, j;
         for ( i = 0; i < n; i++)
         {   mat[i] = new int [i+1];   //申请第二维的存储空间，每次长度不同
             mat[i][0] = 1;     mat[i][i] = 1;
             for ( j = 1;j < i; j++)
                  mat[i][j] = mat[i-1][j-1] + mat[i-1][j];
          }
         for ( i = 0; i< mat.length; i++)
         {    for ( j = 0; j < n-i; j++) System.out.print("   ");
              for ( j = 0; j < mat[i].length; j++)
                   System.out.print("   " + mat[i][j]);
              System.out.println( );
         }
    }}
```

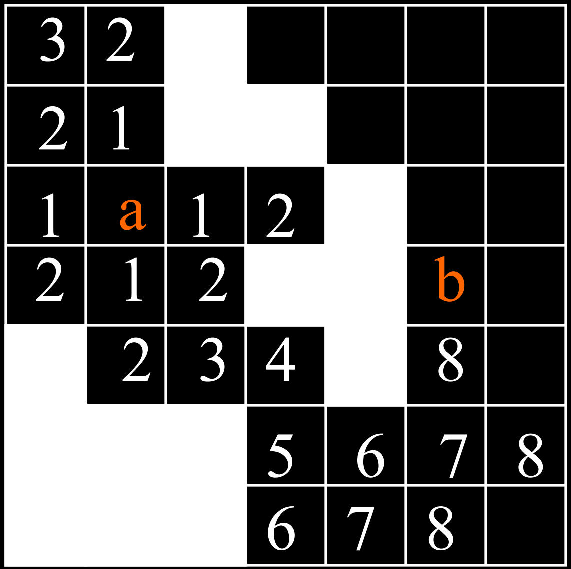
## 2)Wire Routing


A 7*7 grad

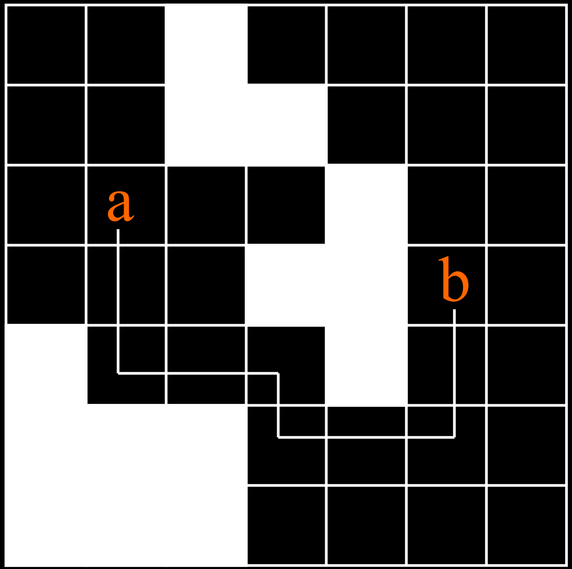


A wire between a and b

## 2)Wire Routing



Distance labeling



Wire path

步骤:

1. 搜索过程

* 先从位置a(3,2)开始, 把a可到达的相邻方格都表为1( 表示与a相距为1).
  注意: 具体实现时, 将a位置置为2, 其它相邻方格为a位置的值+1
* 然后把标记为1的方格可到达的相邻方格都标记为2( 表示与a相距为2).

  这里需要什么数据结构?

* 标记过程继续进行下去, 直至到达b或找不到可到达的相邻方格为止.
  本例中, 当到达b时, b上的表记为9(实现时为9+2=11)

2. 构造a---→b的路径. 从b回溯到a .

   这里需要什么数据结构?

* 从b出发, 并将b的位置保存在path中. 首先移动到比b的编号小1的相邻位置上(5,6)
* 接着再从当前位置继续移动到比当前标号小1的相邻位置上.
* 重复这一过程, 直至到达a.

## 2)Wire Routing

```
bool FindPath(Position start, Position finish, int & PathLen, Position * & path)
{  if (( start.row = = finish.row) &&(start.col = = finish.col))
      { PathLen = 0; return true;}
   for ( int i = 0; i<= m+1; i++)
      {  grid[0][i] = grid[m+1][i] = 1;
         grid[i][0] = grid[i][m+1] = 1;}
   Position offset[4];
   offset[0].row = 0;  offset[0].col = 1;
   offset[1].row = 1;  offset[1].col = 0;
   offset[2].row = 0;  offset[2].col = -1;
   offset[3].row = -1; offset[3].col = 0;
   int NumOfNbrs = 4;
   Position here, nbr;
   here.row = start.row;    here.col = start.col;    grid[start.row][start.col] = 2;
```

## 2)Wire Routing

```
LinkedQueue<Position>Q;
do{  //label neighbors of here
    for ( int i = 0; i< NumOfNbrs; i++)
      { nbr.row = here.row + offset[i].row;
        nbr.col = here.col + offset[i].col;
        if (grid[nbr.row][nbr.col] = = 0)
          {  grid[nbr.row][nbr.col] = grid[here.row][here.col]+1;
             if ((nbr.row = = finish.row) &&(nbr.col = = finish.col)) break;
             Q.Add(nbr);
          }// end of if
      }// end of for
    if (( nbr.row = = finish.row) && (nbr.col = = finish.col)) break;
    if (Q.IsEmpty()) return false;
    Q.Delete(here);
  } while(true);
```

## 2)Wire Routing

```
    PathLen = grid[finish.row][finish.col]-2;
    path = new Position [PathLen];
    //trace backwards from finish
    here = finish;
    for ( int j = PathLen-1; j >= 0; j--)
      {  path[j] = here;
         for ( int i = 0; i < NumOfNbrs; i++)
            {  nbr.row = here.row + offset[i].row;
               nbr.col = here.col + offset[i].col;
               if ( grid[nbr.row][nbr.col] = = j+2) break;
            }
          here = nbr;
      }
    return true;
}
```

## 2)Wire Routing

Time complexity:

1. 网格编号过程　O(m*m)
2. 重构过程　O(Pathlen)

## 2010年全国考研统考题

（13分）设将n(n>1)个整数存放到一维数组R中，试设计一个在时间和空间两方面尽可能有效的算法，将R中保有的序列循环左移P（0 < P < n）个位置，即将R中的数据由（X0 X1 ……Xn-1）变换为（Xp Xp+1 ……Xn-1  X0  X1 …Xp-1)

（1）给出算法的基本设计思想。

（2）根据设计思想，采用C或C++或JAVA语言表述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度

# Chapter 3----stack

实习题：

5. 中缀表达式 $\longrightarrow$ 后缀表达式 $\longrightarrow$
对后缀表达式求值, 合为一趟来做。

# queue exercises 作业题

1.2009年考研统考题:

1) 为解决计算机主机与打印机之间速度不匹配问题, 通常设置一个打印数据缓冲区,主机将要输出的数据依次写入该缓冲区, 而打印机则依次从该缓冲区中取出数据. 该缓冲区的逻辑结构应该是

A. 栈　　B. 队列　　C. 树　　D. 图

2) 设栈S和队列Q 的初始状态为空，元素 a,b,c,d,e,f,g 依次进入栈S. 若每个元素出栈后立即进入队列Q，且7个元素出队的顺序是 b,d,c,f,e,a,g，则栈S的容量至少是　　A. 1　B. 2　C. 3　D. 4

2. Suppose that a singly list is implemented with both a header and tail node.

Describe contant-time algorithms to

a. Insert item x before position p ( given by an iterator ).

b. Remove the item stored at position p ( given by an iterator )

3. 假设以数组Q[m]存放循环队列中的元素，同时以rear和length 分别指示环形队列中的队尾位置和队列中所含元素的个数：

1）求队列中第一个元素的实际位置。

2）给出该循环队列的队空条件和队满条件，并写出相应的插入(enqueue)和删除(dlqueue)元素的操作算法。