Chapter 3.1

栈 & 队列

Stack and Queue

线性表的部分操作是线性复杂度的

且无法改变.

解决方法：限制部分操作

这就是栈和队列的理解方式之一.

—— 改变了逻辑层.

Stack

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.

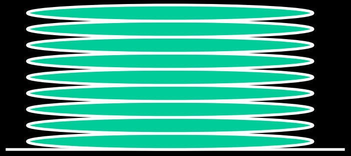
栈顶

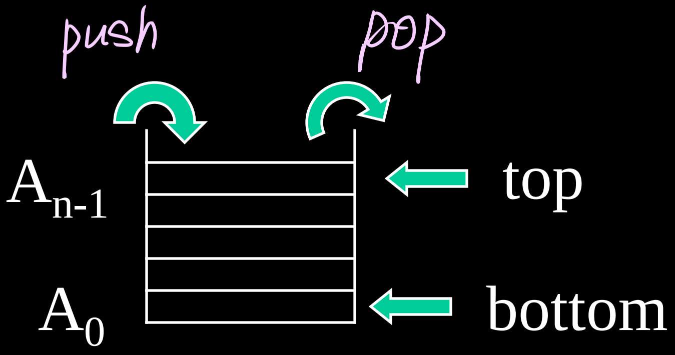
This end is called the top. The other end of the list is called the bottom. 栈底.

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

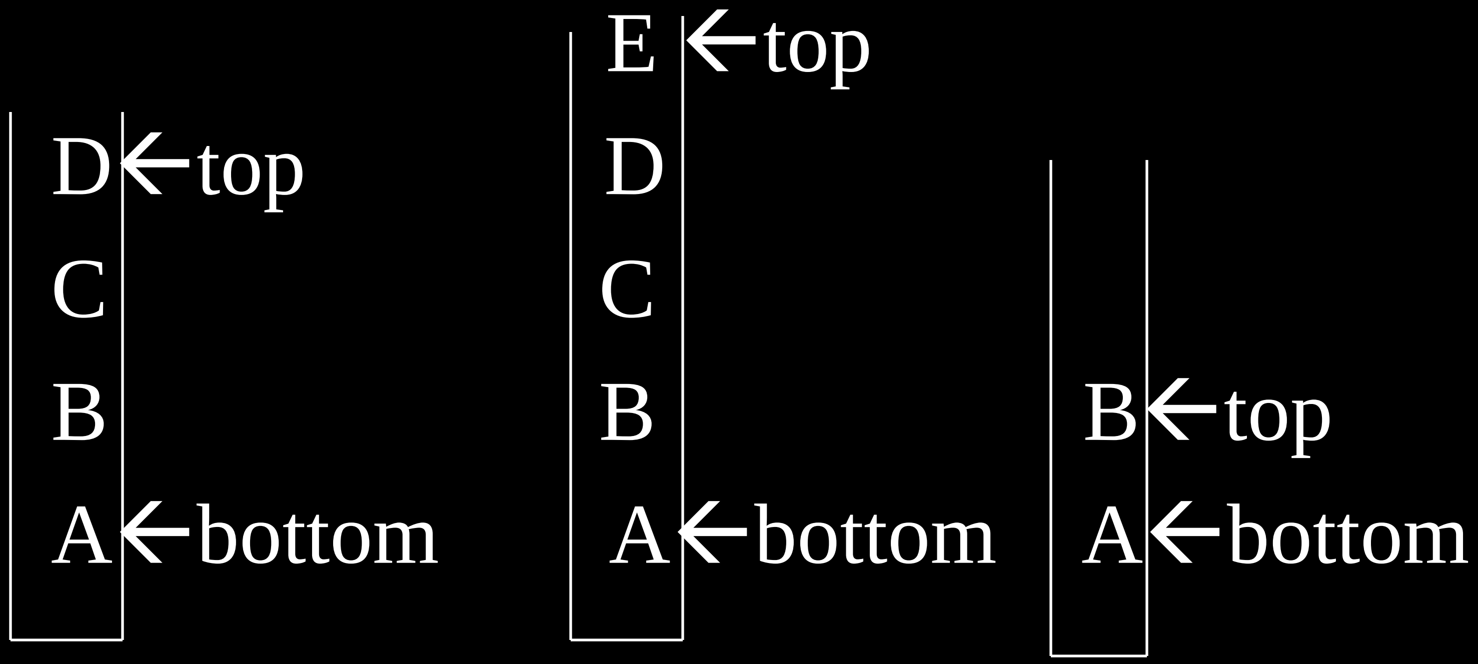
后进先出.

Stack Model





Stack Model



D ← top
C
B
A ← bottom

E ← top
D
C
B
A ← bottom

B ← top
A ← bottom

逻辑层.

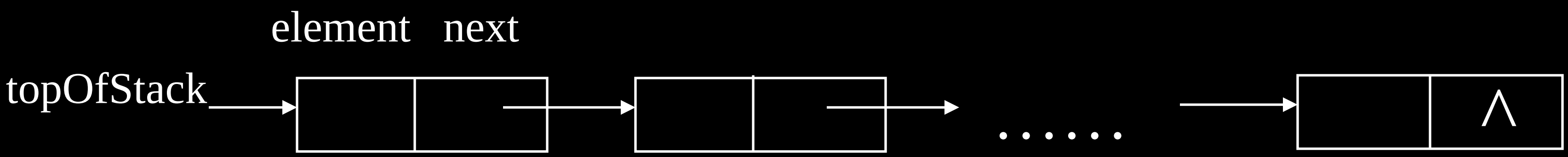
Stack Model

```
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;
    push  Add(x): add element x to the stack;
    pop   Delete(x):Delete top element from stack and put it in x;
    }
```

3.3.2. Implementation of Stack

物理层.

1. Linked List Implementation of Stacks



when topOfStack= = null is empty stack

Linked List Implementation of Stacks

Linked List

```
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;
}
```

加入?

一行解决!!!

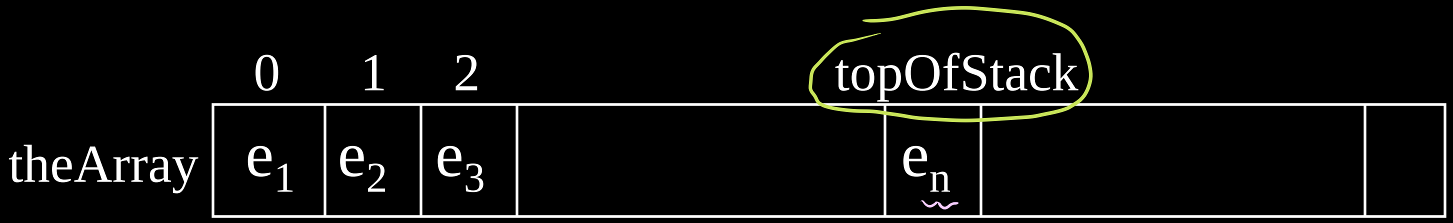
Linked List Implementation of Stacks

```
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;
}
```

删除? 一行解决!!!

3.3.2. Implementation of Stack

2. Array Implementation of Stacks



0 1 2 topOfStack

theArray e_1 e_2 e_3 e_n

代表空.

when topOfStack= = -1 is empty stack

记录栈顶的下标

Array Implementation of Stacks

```
                    Array
 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 );
}
```

调用 ↓ 这个构造函数 默认数组大小

```
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;
}
```

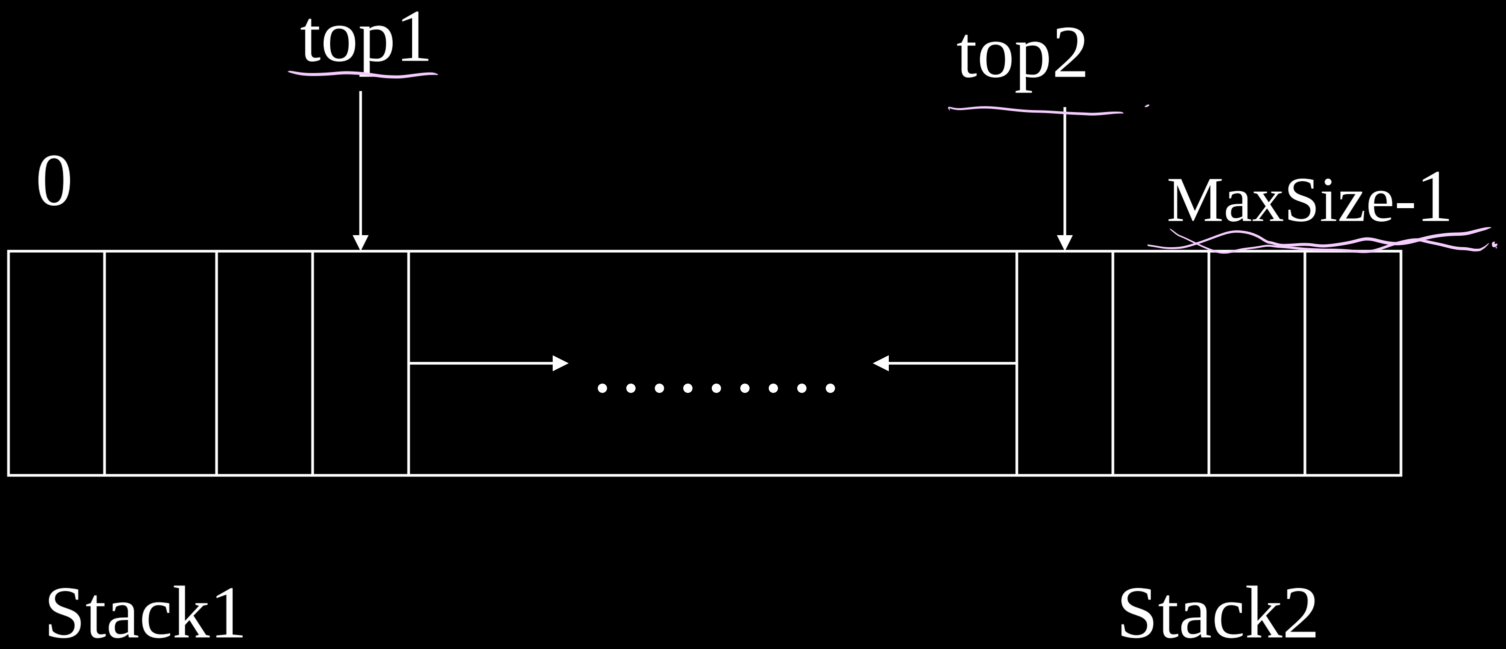
Array Implementation of Stacks

- It is wasteful of space when multiple stacks are to coexist

空间浪费

- 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

1. Parenthesis Matching 扩号匹配

(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) → (char数组) string 的首地址.
{   Stack<int> s(Maxlength);
     int j,  length = strlen(expr);
     for ( int i = l; i <= length; i++)
     {    if  ( expr[i-1]= =‘(‘) s.Add(i);   j=i ？ 我觉少点东西. 关于j的
         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)

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

eg

进	出	now
abcd		abcd
	d	abc
	c	ab
e		abe
	e	ab
	b	a
f		af
	f	a
	a	a

3.4 . The Queue ADT 队列

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

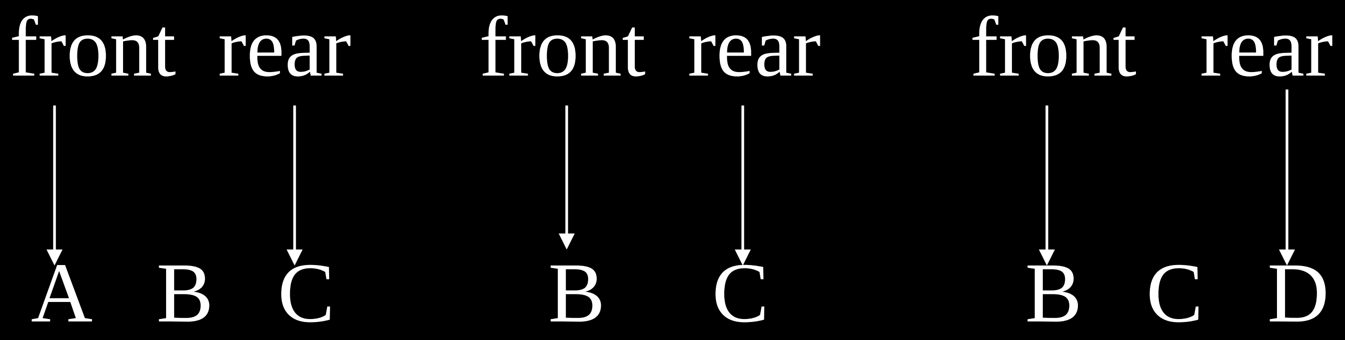
It is also called a first-in-first-out list.

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

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

3.4.1. Queue Model

Sample queues



front rear front rear front rear

A B C B C B C D

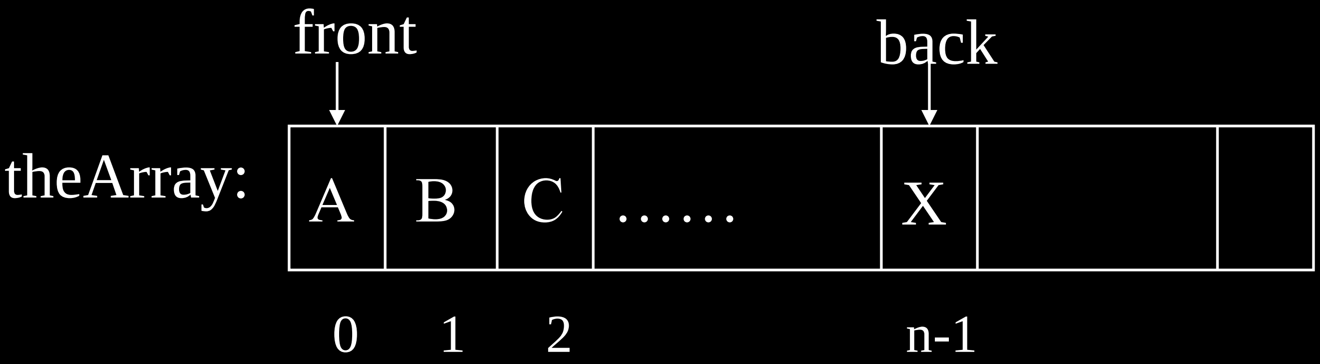
Delete A

Add D

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



front

back

theArray:

A B C …… X

0 1 2 n-1

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:

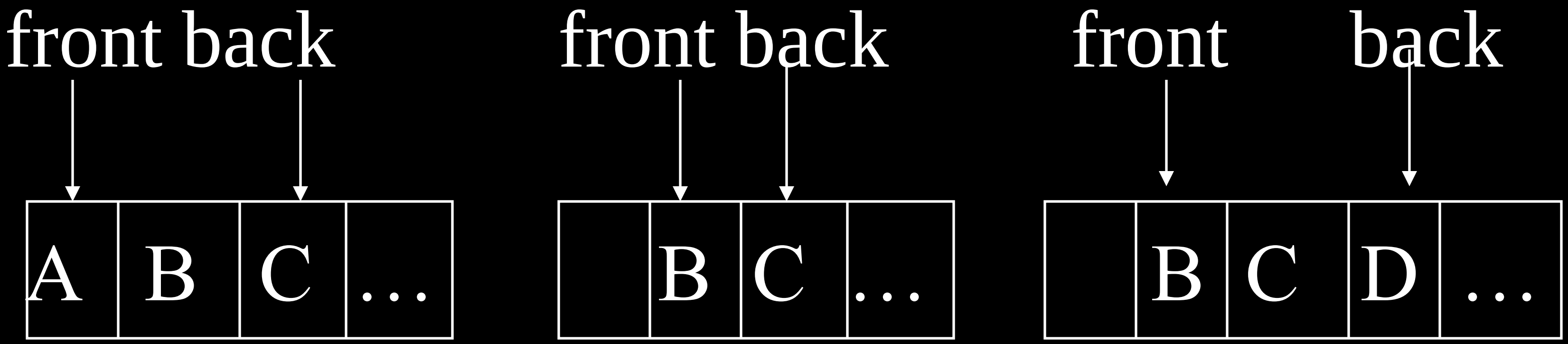
back=back+1; theArray[back]=x; O(1)

To delete an element: two methods:

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

2) shift the queue one position left. $\underline{O(n)}$ (개선)

3.4.2. Array Implementation of Queue



front back

A B C …

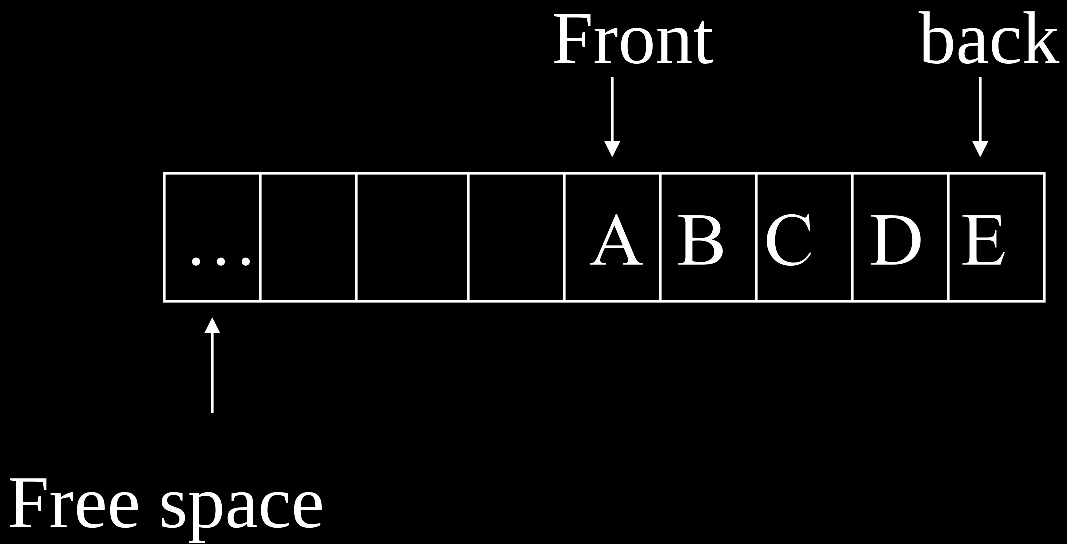
front back

B C …

front back

B C D …

3.4.2. Array Implementation of Queue



Front

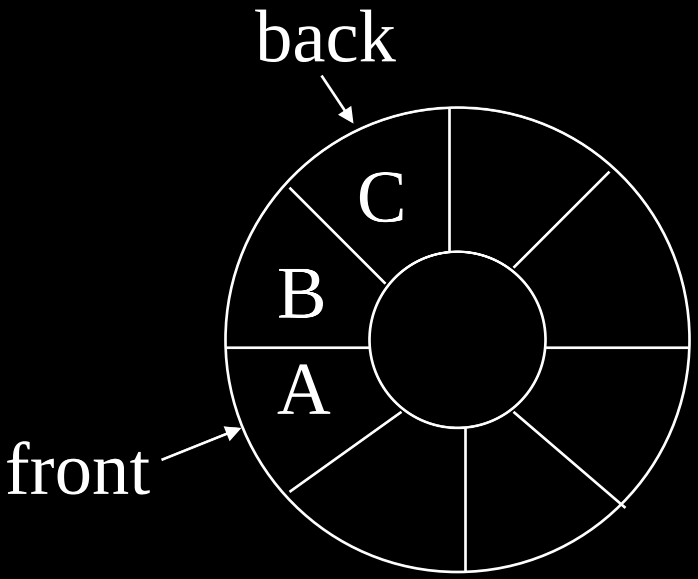
back

…				A	B	C	D	E

Free space

3.4.2. Array Implementation of Queue

to use a circular array to represent a queue



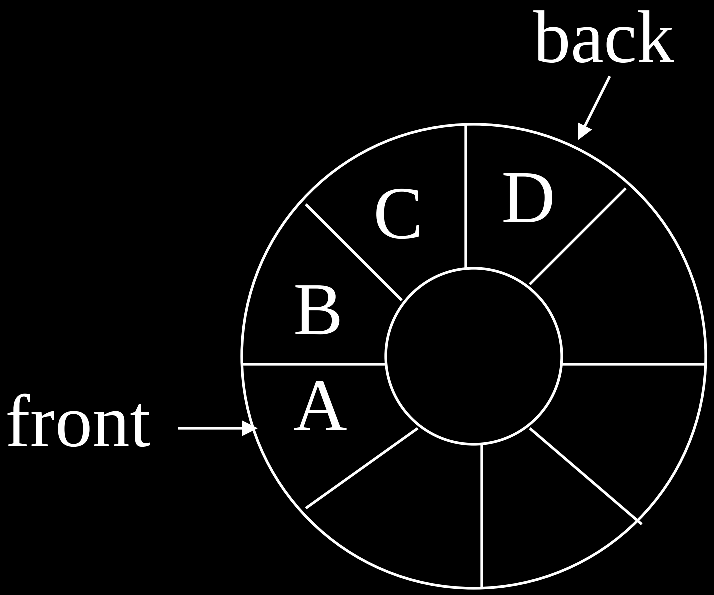
back

C

B

A

front



back

C D

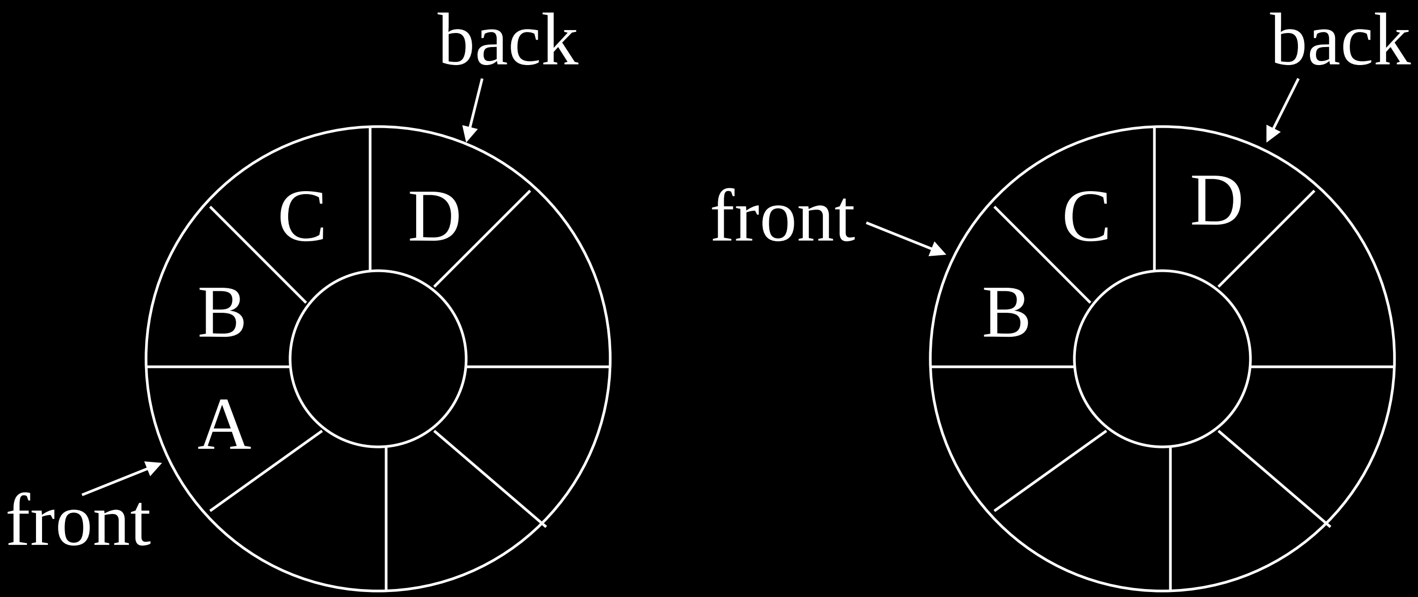
B

front A

Addtion

3.4.2. Array Implementation of Queue

deletion



back

C D

B

A

front

back

front

C D

B

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                                    所有操作都是O(1)
{   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    进队
    private int increment( int x )  下标+1
    private Object dequeue( )       出队.

    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 );
}
```

调用了.

```
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

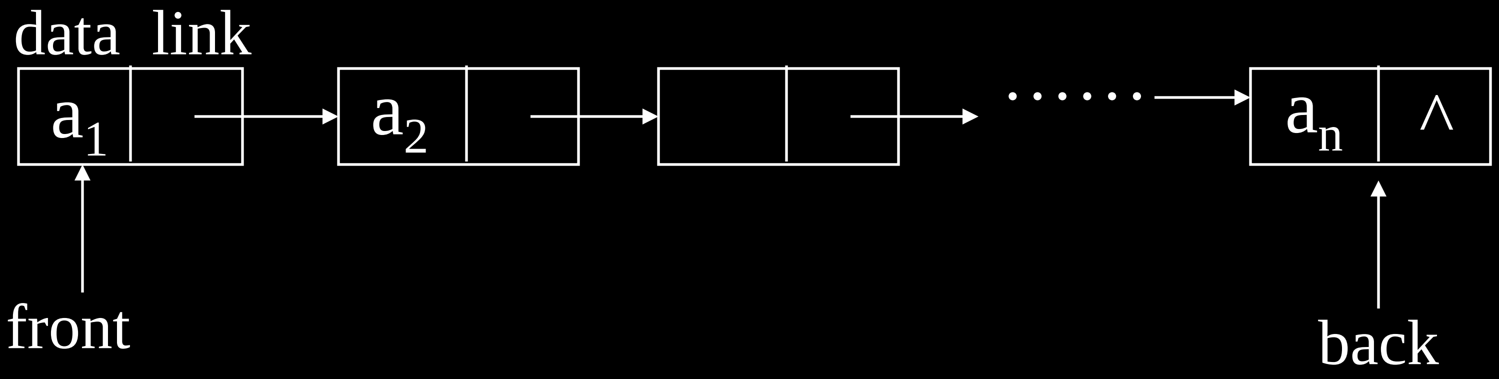
```
public void enqueue( object x ) throw Overflow
{   if( isFull( ) )
        throw new Overflow( );
    back = increment( back );
    theArray[ back ] = x;
    currentSize++;
}

private  int increment( int x )   ②.        ①取模.
{   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



链表的head 是队列的front.

3.4.3 Linked Representation of queue

Class definition for a linked queue

```
template<class T>class LinkedQueue
{  public:
      LinkedQueue(){front=back=0;}
   △  ~LinkedQueue();   析构函数
                                          是否是空指针
      bool IsEmpty()const{return ((front)?false:true);}
      bool IsFull()const;
      T First()const;
      T Last()const;                T类型的引用
      LinkedQueue<T>&Add(const T& x);    ?
      LinkedQueue<T>& Delete(T& x);
  private:
     Node<T>*front;    Node<T>*back;
};
```

若传x，函数改变的是x的副本

(值传递. copy)

若传x的引用 函数可以改变x

(传递引用) &

*: 传指针 ?

Java 只传引用.

3.4.3 Linked Representation of queue

1)destructor 把List全释放.

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

3.4.3 Linked Representation of queue

2)Add(x)

泛型 存放的数据类型

```
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

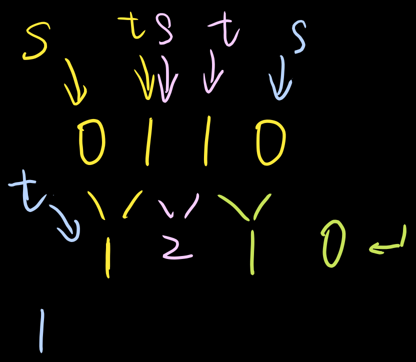
系数 杨辉三角. 又叫"二项式展开"

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++)   第i行
回车  { cout << endl;
空格     for (int k=1;k<=10-i;k++) cout<<‘ ‘;
         q.Enqueue(0); 三角帽行末尾添0
         for (int j=1;j<=i+2;j++)
           {   int t=q.Dequeue( );
               q.Enqueue(s+t);
               s=t;
不打印每行添的0  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： 不是队列 n阶

```
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( );
         }
    }}
```

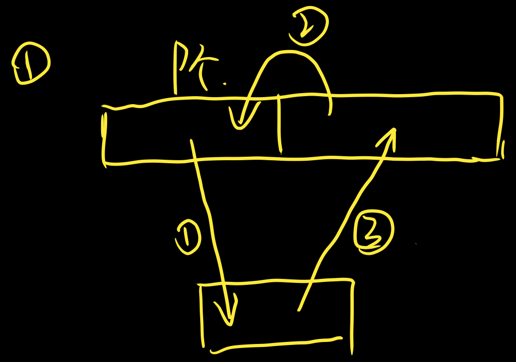
2010年全国考研统考题

（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)

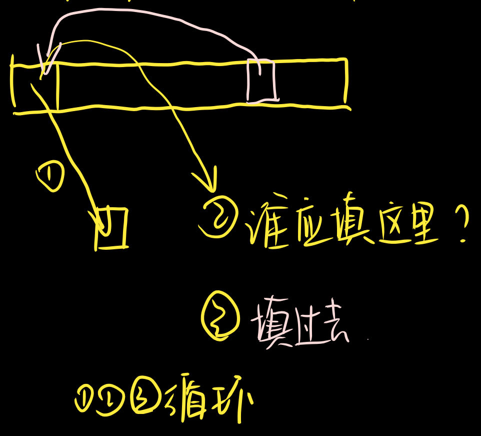
（1）给出算法的基本设计思想。 O(n) 的时间复杂度不能改变.

（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.



② 节省空间. 但算法更复杂



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 , p!=tail)

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。

出队要返回出队对象!!

队列的插入和删除（一定是在队尾和队头）