



计算机程序设计与Visual C++

第4章 数组与字符串的应用



主要内容

- 数组的概念
- 数组的定义、初始化和存储数据
- 对一维数组操作
- 对二维数组操作
- 字符数组和字符串类(string)的应用

问题的引出

编程实现：在5位同学中找出“计算机程序设计”成绩最好的同学。

//找出5个数中的最大值

```
void main( )
```

```
{
```

```
    int a1,a2,a3,a4,a5;
```

```
    cout<<"Input the number:";    当 n=10  n=100  n=1000
```

```
    cin>>a1>>a2>>a3>>a4>>a5;
```

```
    if(a1>a2&&a1>a3 && a1>a4 && a1>a5)
```

```
        cout<<"The max_number is"<<a1;
```

```
    else if(a2>a3 && a2>a4 && a2>a5)
```

```
        cout<<"The max_number is"<<a2;
```

```
    else if(a3>a4 && a3>a5)
```

```
        cout<<"The max_number is"<<a3;
```

```
    else if(a4>a5)
```

```
        cout<<"The max_number is"<<a4;
```

```
    else cout<<"The max_number is"<<a5;
```

```
}
```

问题的引出

实际应用的程序设计中，只用几个变量的情况是极少的；更多的情况是处理大批量的相同类型或不同类型的数据。

用什么样的数据结构来描述这类应用更简洁？

具有一定顺序关系的若干相同类型数据的集合体

学号	成绩
20122824	88
20122825	91
20122830	81
20122831	84
20122845	84
20122847	88
20122853	90
20122854	85

4.1 数组的概念

P₁₁₈

数组是具有一定顺序关系的若干相同类型数据的集合体，数组属于复合类型。

数组用数组名来标识。一个数组名用来表示一组同类型的数据，这批同类型的数据被称为数组元素或分量。

成绩	
score	88
元素(分量)	91
	81
	84
	84
	88
	90
	85

4.2 数组的定义 P₁₂₀

定义的格式：

类型说明符 **数组名** [常数表达式][.....][,.....];

说明：

- 类型说明符通常为int、char、double等
- 数组可以是一维、二维或多维数组，是几维数组，数组名右边就有几对方括号。
- 常数表达式即该表达式的值是一个确定的值，它的值确定各维的长度（元素个数）。

```
int score[10];  
int a[3][4];
```

4.3 一维数组 P₁₂₀

● 一维数组的声明

元素在数组中的位置与其下标相差1。

类型说明符 数组名[常量表达式];

score[0]	88
score[1]	91
score[2]	81

Microsoft Visual C++ Debug Library



Debug Error!

Program: c:\Documents and Settings\lq\桌面\20160001-张三\Debug\test.exe
Module: c:\Documents and Settings\lq\桌面\20160001-张三\Debug\test.exe
File:

Run-Time Check Failure #2 - Stack around the variable 'a' was corrupted.

(Press Retry to debug the application)

终止(A)

重试(R)

忽略(I)

● 一维数组必须先定义，后使用。

1. 一维数组的初始化 P₁₂₁

- 在声明数组的同时可以对数组元素赋以初始值

例如：int a[5]={0,1,2,3,4};

- 可以只给一部分元素赋初值（至少一个），当没有为数组中每个元素提供初始值时，C++将未初始化的元素赋值为默认值。

例如：int a[5]={0,1,2}; (a[3]=0,a[4]=0)

- 如果没有赋任何一个初值，C++不能自动初始化元素，数组元素将包含垃圾数据。
- 在对全部数组元素赋初值时，可以不指定数组长度。

例如：int a[]={1,2,3,4,5} （数组长度为5）

P₁₁₁ int a[5]={0,1,2,3,4}; int i=1,j=1,x=1;
a[i] a[j++] a[2*x+1] a[5] (×)

静态一维数组 P₁₂₁

声明数组时，也可在前面加上关键字**static**

```
static int a[5];
```

- 若未对静态数组进行初始化，则数组的所有元素将自动获取初始值0。如上例中定义的数组a的5个元素均为0(**整型数组**)。
- 静态数组的初始化是在编译时完成的，以后不用再初始化，直接使用系统赋予的初值。

2. 一维数组的存储顺序 P₁₂₀

- 数组元素在内存中顺次存放，数组元素在内存中是从低地址开始顺序存放的，各元素的存储单元占用内存大小相同(由其数据类型决定)，它们的地址是连续的。
- 各元素的存储单元之间没有空隙，可以从数组第一个元素存储单元的起始地址计算出任意一个元素存储单元的起始地址。
- 数组名字score是数组首元素的内存地址。

```
int score[10];
```

score[0]	0x2000	← score
score[1]	0x2004	
score[2]	0x2008	
.....	
score[9]		

注意

● C++数组第一个元素的下标为0，而不是1，最后一个元素的下标为n-1，且下标式是固定的。

● 元素在数组中之间相差1。

● 数组是一种复合类型，是不能作为一个整体进行访问和处理的，只能按元素进行个别的访问和处理。

请思考：该如何对数组元素进行访问？

下标访问：只需变化元素的下标，即可遍历所有数组元素。下标i从0~n-1

score[0] 88

score[1] 91

score[2] 81

score[3] 84

score[4] 84

score[5] 88

score[8] ...

score[n-1]

数组在内存中的存储

示例：一维数组的声明与引用

```
#include <iostream>
using namespace std;
void main()
{ int A[10],B[10];
  int i;
  for(i=0;i<10;i++)
  /*利用循环语句给数组元素逐个赋值*/
    A[i]=i*2+1;
    B[10-i-1]=A[i];
}
```

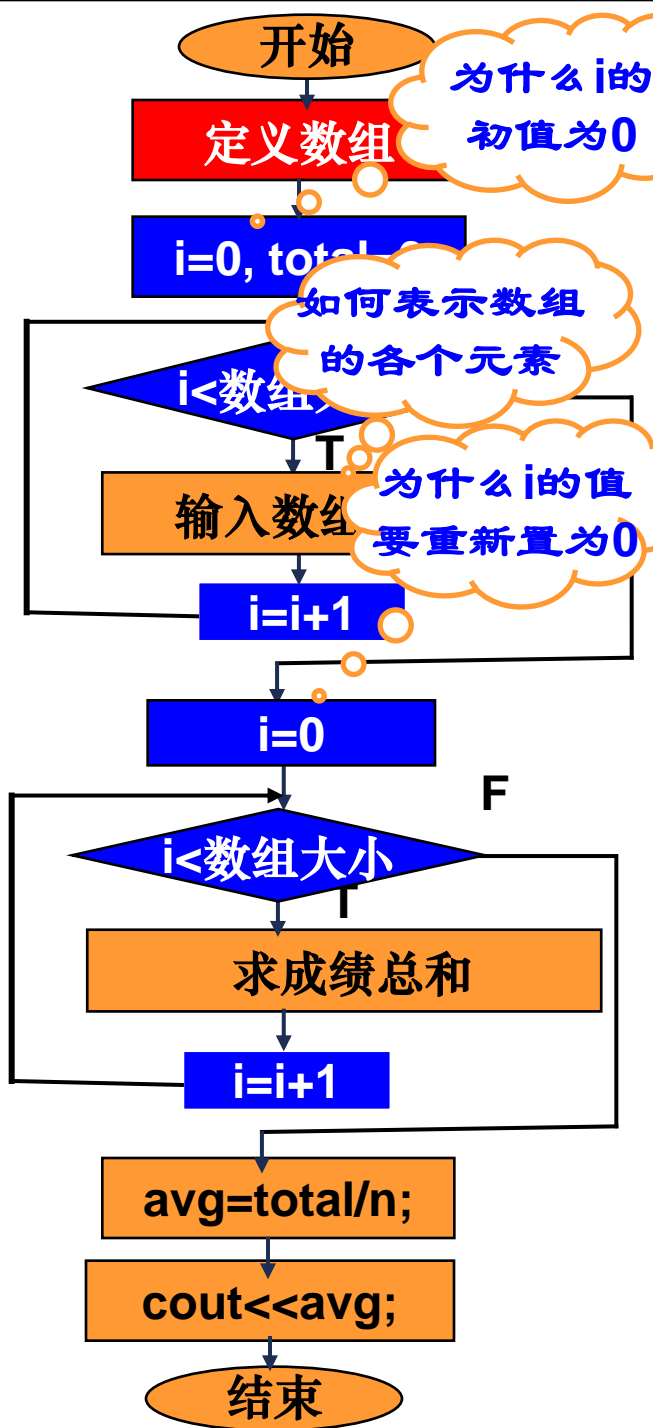
数组是复合类型，不能作为一个整体进行访问和处理的，只能按元素进行个别的访问和处理。

```
for(i=0;i<10;i++)
  cout<<"A["<<i
    <<"]="<<A[i];
  cout<<" B["<<i
    <<"]="
    <<B[i]<<endl;
}
```

一维数组基本操作：

计算数组中所有元素的平均值

- **问题：**某教师需要一个程序，用来计算和显示其学生考试的平均成绩。
- **分析：**可以把所有学生的**成绩保存在一个一维数组中**，然后计算所有数组元素之和，再除以学生总人数即可得到学生的平均成绩。



```
#include <iostream>
using namespace std;
void main( )
```

```
{
    const int n=10;
    int scores[n];
```

```
    int n,i,total=0;
    float avg=0;
    cout<<"请输入数组长度:"<<endl;
    cin>>n;
```

```
    for(i=0;i<n;i++)
```

```
    {
        cin>>scores[i];
        total=total+scores[i];
    }
```

```
    avg=float(total)/n; //计算平均成绩
    cout<<"Average:"<<avg<<endl;
}
```

错误1:

```
int n;
int scores[n];
cout<<"请输入数组长度:"<<endl;
cin>>n; (x)
```

原因: C++编译系统编译时还不知道n的值(n的值是运行时才从键盘输入的), 故无法为数组在内存开辟存储空间!

错误2:

```
int n=10;
int scores[n];
```

原因: 数组长度不能为变量, 必须是常量表达式

关于数组定义时的长度问题

```
#include <iostream>
void main()
{
    const int N=30;
    int a[N], b[N], c[N], n, i;
    cout<<"请输入数组的实际长度："<<endl;
    cin>>n;    //数组实际长度
    for(i=0; i<n; i++)
        cin>>a[i]>>b[i]>>c[i];
    for(i=0; i<n; i++)
        cout<<a[i]<<b[i]<c[i];
    for(i=0; i<n; i++)
    {
        sum=sum+a[i];
        sum=sum+b[i];
        sum=sum+c[i];
    }
    .....
}
```

预定义数组的长度，C++编译系统在编译时预先为数组分配空间。**注意：数组的实际长度不能超过该长度！即： $N \geq n$ 。**如果预留空间不够可增加N的值。

数组的实际长度通过变量n从键盘接收，好处：

- 1、**便于调试。**可先输入较小的n值，在较少的数据情况下进行代码的调试，**正确后，再输入要求的长度值。**
- 2、**避免修改多处代码。**

一维数组基本操作：

求一维数组中的最大值及其位置

- 问题：某人需要一个程序，用来显示一周中花费的最大金额。
- 分析：可以将一周中每天的花销保存在一维数组中，然后通过两两比较数组元素可找出数组中值最大的元素及其所在数组中的位置。

默认第1个元素为最大值,
j记录最大值所在位置

从键盘接收

从第二个元素
(其下标为1)
开始比较

high=dollars[0], j=0;

i=1

i<数组长度 记录新的最大值
及其所在位置

T

F

high=dollars[i]

j=i;

i=i+1

最大值为第j+1个数, 值为high

程序

```
#include <iostream>
using namespace std;
void main( )
{
```

```
    const int N=10;
```

//声明数组

```
    int dollars[N];
```

```
    int n,i,j,high;
```

```
    cout<<"请输入数组实际长度:";
```

```
    cin>>n;
```

```
    for(i=0;i<n;i++) cin>>dollars[i];
```

```
    high=dollars[0]; //默认第一个元素为最大值
```

```
    j=0;
```

//记录最大元素的下标

```
    for( i=1;i<n;i++ ) //从第二个元素开始比较
```

```
    {
```

```
        if(dollars[i]>high)
```

```
        {
```

```
            high=
```

元素下标与其

大值

```
            j=i;
```

在数组中的位置相差1

```
        }
```

```
    }
```

```
    cout<<"最大元素为第"<<j+1<<"个元素,
    它的值为:"<< dollars[j] <<endl;
```

```
}
```

例：Fibonacci数列 P_{152}

1202年，意大利数学家斐波那契出版了他的《算盘全书》。他在书中提出了一个关于兔子繁殖的问题：

如果一对兔子每月能生一对小兔（一雄一雌），而每对小兔在它出生后的第三个月里，又能开始生一对小兔，假定在不发生死亡的情况下，由一对出生的小兔开始，10个月后会会有多少对兔子？

分析

在第一个月时，只有一对小兔子，过了一个月，这对兔子成熟了，在第三个月时便生下一对小兔子，这时有两对兔子。再过一个月，成熟的兔子再生一对小兔子，而另一对小兔子长大，有三对小兔子。如此推算下去，便发现一个规律：

时间(月)	初生兔子(对)	成熟兔子(对)	兔子总数(对)
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
9	13	21	34
10	21	34	55

Fibonacci数列

由此可知，从第一个月开始以后每个月的兔子总数是：

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

若把上述数列继续写下去，得到的数列便称为斐波那契数列。数列中每个数便是前两个数之和，而数列的最初两个数都是1。

若设 $F_0=1$, $F_1=1$, $F_2=2$, $F_3=3$, $F_4=5$, $F_5=8$, ...

则：当 $n > 1$ 时， $F_{n+2} = F_{n+1} + F_n$ ，且 $F_0 = F_1 = 1$ 。

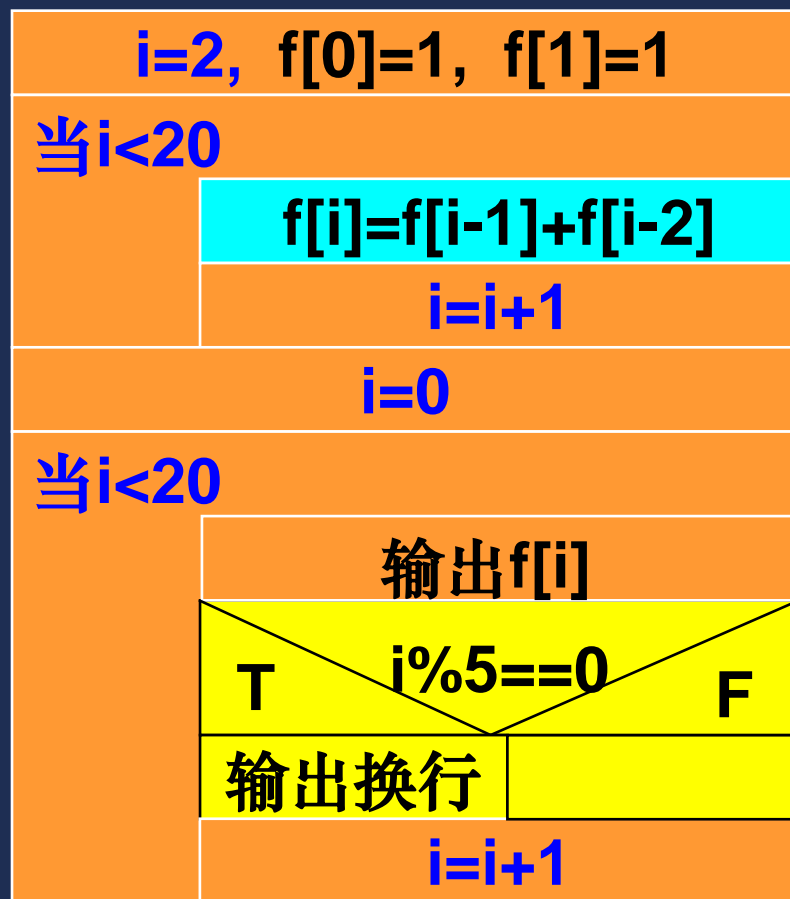
(递归的算法设计思想)

用数组来处理求Fibonacci数列前20项

```
#include <iostream>
#include <iomanip>
using namespace std;
void main( )
{
    int i,f[20]={1,1};
    for(i=2;i<20;i++)
        f[i]=f[i-2]+f[i-1];
    for(i=0;i<20;i++)
    {
        if(i%5==0) cout<<endl;
        cout<<setw(5)<<f[i];
    }
}
```

定义数组的同时初始化，只给出第1个和第2个数组元素的值，其余数组元素的值为0

每行输出5个值



自学P₁₅₃例4.16
自动生成分数序列

在数组中插入新元素

P₁₃₇

把一个整数插入到一个由小到大的有序数列中，并仍然保持由小到大的顺序。

算法分析： 设由10个整数按由小到大的顺序存放在a数组中，待插入的数存放在变量x中。

10个有序数依次为：2, 4, 6, 8, 12, 16, 17, 20, 30, 40，待插入的数x=11。

插入

```
const int N=11;  
int a[N],n; //n为数组实际长度, n<=N  
x=11
```

a[0] a[1] a[2] a[3] a[4] a[5]

2	4	6	8	12	
---	---	---	---	----	--

↑ ↑ ↑ ↑
p=1 p=2 p=3 p=4

$x > a[p]$ $x > a[p]$ $x > a[p]$ $x < a[p]$

算法关键:

1. 如何确定正确的插入位置?
2. 如何把插入位置“空出来”?
3. 如何“插入”?

2	4	6	8		12	16	17	20	30	40
---	---	---	---	--	----	----	----	----	----	----

2	4	6	8	X	12	16	17	20	30	40
---	---	---	---	---	----	----	----	----	----	----

算法分析

完成插入操作的关键是确定插入的位置。

- 设一个下标(位置)变量 p ，令其初始值为0，将 x 与 $a[p]$ 进行比较，只要 $x > a[p]$ ，就表示 x 所要插入的位置是在 $a[p]$ 之后，此时应使 p 后移一个位置，即 $p = p + 1$ ，接着再进行下一个元素的比较。
- 不断重复以上过程(循环处理)，一旦 $x \leq a[p]$ ，此时 p 值就是 x 要插入的位置。
- 如果 x 比所有元素都要大，则需要在数组最后插入，此时 $p = n$ ， p 停止移动(循环结束)。

在没有找到 x 所要插入的位置之前，循环次数无法确定，判断循环条件可用逻辑条件：

$(x > a[p] \ \&\& \ p < n)$

未到达数组尾部
(最后一个元素)

算法分析

完成插入操作的第二步是“空出”插入位置。

2	4	6	8	12	16	17	20	30	40	
---	---	---	---	----	----	----	----	----	----	--



$p \quad (x < a[p])$

2	4	6	8		12	16	17	20	30	40
---	---	---	---	--	----	----	----	----	----	----

```
for( i=n-1;i>=p;i-- )  
    a[i+1]=a[i];
```

C++源代码 P138

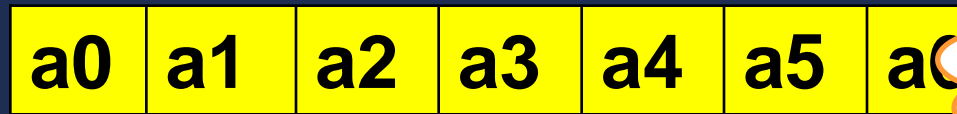
```
#include <iostream>
#include <iomanip>
using namespace std;
void main( )
{
    const int N=11;
    int a[N],n,p=0,x,i;
    cout<<"输入a数组长度(n的大小不能超过10): "<<endl;
    cin>>n; //数组a 实际长度
    cout<<"输入a数组: "<<endl;
    for(i=0;i<n;i++)
        cin>>a[i];
```

多一个元素位置用于
存放新插入的元素

```
cout<<"输入待插入的数x: "<<endl;
    cin>>x;
    while(x>a[p]&&p<n)
        p++;          //找到x应插入的正确位置
    for(i=n-1;i>=p;i--)
        a[i+1]=a[i];   //将a[p]~a[n-1]后移
    a[p]=x;           //x插入正确位置
    for(i=0;i<=n;i++)  //插入新元素后，长度增加1
        cout<<setw(3)<<a[i]<<endl;
}
```

一维数组基本操作: P₁₁₂

在指定位置插入新元素



注意:

插入位置和元素下标
相差1

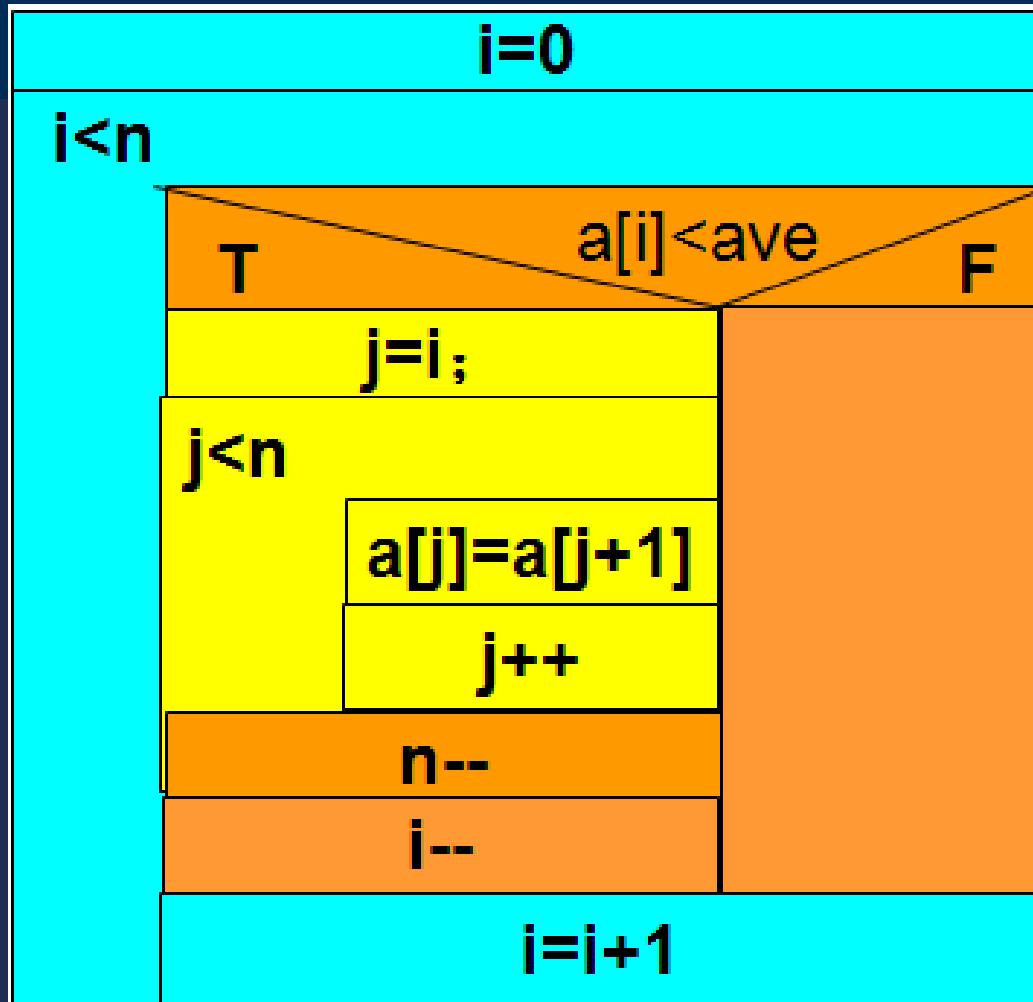
数组长度为 n , 插入位置为 S , 插入的元素值为 X (X , S 均从键盘输入值), 则插入操作:

```
for (j=n-1;j>=S-1;j--) a[j+1]=a[j]; //元素后移, 空出插入位置  
a[S-1]=X; //插入新元素, 注意插入位置和元素下标的关系
```

一维数组基本操作：

删除数组中所有小于平均值的元素

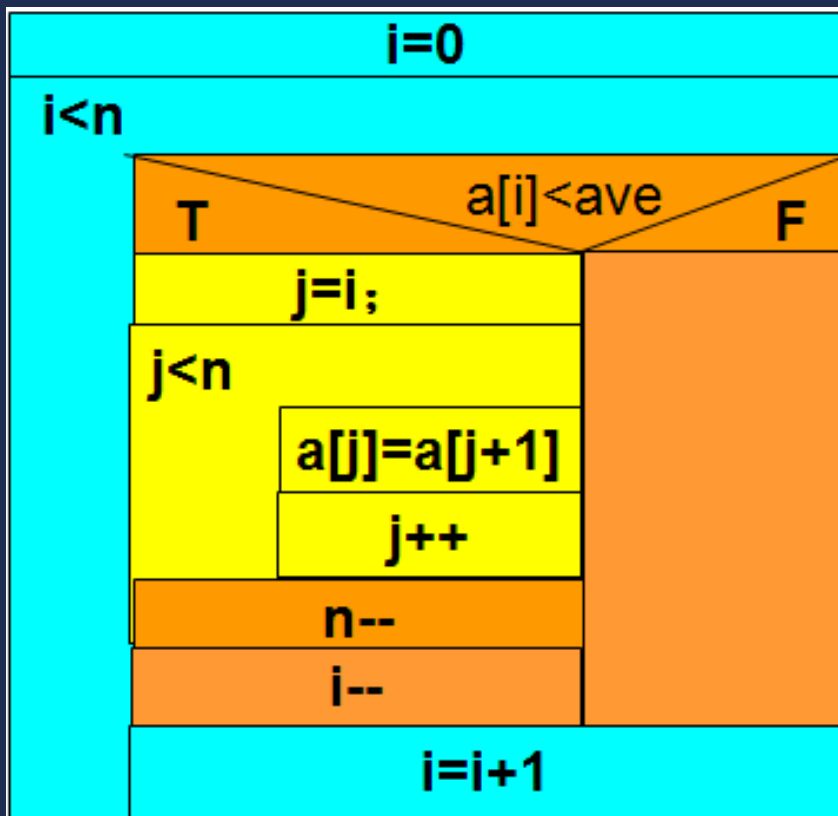
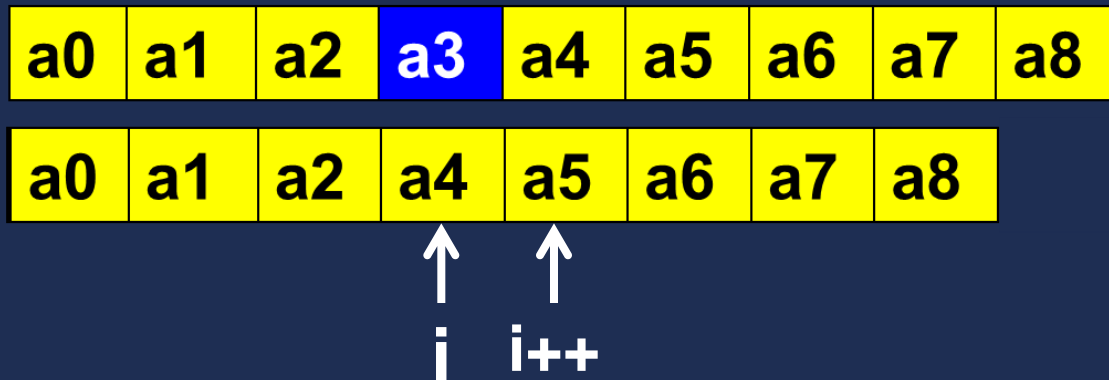
删除元素 $a[i]$,即将 $a[i+1]$ 移至 $a[i]$; $a[i+2]$ 移至 $a[i+1]$;以此类推,直到将 $a[n-1]$ 移至 $a[n-2]$ 为止。



注意：

- 算法描述中, $n--$, 表示删除小于平均值的元素后, 数组的实际长度。
- 为什么还需要 $i--$?

删除数组中所有小于平均值的元素



- 一次循环处理完成后会删除比平均值小的元素 $a[i]$ ($a[3]$), 然后进行下一次处理($i++$), 则下一次处理会从 $a[5]$ 开始。
- 实际上, 原来的 $a[4]$ 已经移动至 $a[i]$ ($a[3]$)的位置, 为了不漏掉元素, 需要“回溯”下标($i--$), 则在进行 $i++$ 操作后, 刚好是正确的处理位置。

实验八注意事项

- 本实验，每个实验任务的子任务较多，要求所有子任务均在一个主函数中完成，不要创建多个项目！！！！即8-1编写一个程序，程序里面包含三个子任务的实现；8-2编写一个程序。
- 建议同学在每个子任务前给出简单的功能注释，完成一个子任务的编码、调试、运行后再继续后面子任务的编写，这样方便代码的调试和查错，也是模块化设计的思想。

编码示例

```
#include<iostream>
#include<iomanip>
void main( )
{
    const int N=10;
    int a[N];
    int i,j,n,sum,avg;
    cout<<"请输入数组实际长度: "<<endl;
    cin>>n;
    //输入数组元素
    for(i=0;i<n;i++)
        cin>>a[i];
    //输出数组元素
    for(i=0;i<n;i++)
        cout<<setw(6)<<a[i];
    //求所有元素之和
    .....
}
```


调整元素位置

将大于平均值的元素放在数组前面，小于平均值的放在后面。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
13.5	20	11	67	3.3	41.2	3	5.2	29	21.47
29	41.2	67	11	3.3	20	3	5.2	13.5	21.47

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
13.5	20	11	67	3.3	41.2	3	5.2	29	21.47

↑
i

↑
j

29	20	11	67	3.3	41.2	3	5.2	13.5	21.47
----	----	----	----	-----	------	---	-----	------	-------

↑
i

↑
j

29	20	11	67	3.3	41.2	3	5.2	13.5	21.47
----	----	----	----	-----	------	---	-----	------	-------

↑
i

↑
j

29	20	11	67	3.3	41.2	3	5.2	13.5	21.47
----	----	----	----	-----	------	---	-----	------	-------

↑
i

↑
j

调整元素位置

将大于平均值的元素放在数组前面，小于平均值的放在后面。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
13.5	20	11	67	3.3	41.2	3	5.2	29	21.47
29	41.2	67	11	3.3	20	3	5.2	13.5	21.47

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
29	41.2	11	67	3.3	20	3	5.2	13.5	21.47
	↑ i				↑ j				
29	41.2	11	67	3.3	20	3	5.2	13.5	21.47
		↑ i		↑ j					
29	41.2	11	67	3.3	20	3	5.2	13.5	21.47
		↑ i	↑ j						
29	41.2	67	11	3.3	20	3	5.2	13.5	21.47
		↑ i	↑ i	↑ j					

调整元素位置

将大于平均值的元素放在数组前面，小于平均值的放在后面。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
13.5	20	11	67	3.3	41.2	3	5.2	29	21.47
29	41.2	67	11	3.3	20	3	5.2	13.5	21.47

处理流程：

设置两个下标变量 i (初值为0), j (初值为 $n-2$), 分别“指向”数组的第1个元素和最后一个元素(不包括平均数), 比较两个元素与平均数 ($a[n-1]$) 的大小关系, 并进行下列处理, 直到 $i \geq j$ 时停止:

如果 $a[i] < a[n-1] \&\& a[j] > a[n-1]$, 则交换 $a[i]$ 和 $a[j]$, 同时 $i++$, $j--$, 即位置“指针”分别往数组后方移动和往前方移动。

否则, 判断: 如果 $a[i] < a[n-1]$, 则 $j--$ 。(下标 j 前移, 在数组后部寻找大于平均值的元素)
否则, 判断: 如果 $a[j] > a[n-1]$, 则 $i++$ 。(下标 i 后移, 在前部寻找小于平均数的元素)

否则, $i++$ 、 $j--$ (此时, 数据均处于正确位置, 不需要交换, 直接移动前、后“指针”)。

补充案例 P₁₄₂

编写程序：只用一个数组把所有相同的数删到只剩下一个

1	2	1	7	3	4	3	5	2	7	1	2	7	3	4	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

处理流程：把一个数组分成两部分，前面一部分为已经处理好的数据；后面为待处理的数据。

把第1个数组元素默认视为已经处理好的结果，则从第2个到最后一个（下标 $1 \sim n-1$ ）为剩下的需处理的元素。同时用变量pos（初值为1）记录处理好的元素的存放位置。

	pos ↓									
i=0	1	2	1	7	3	4	3	5	2	7
i=1	1	2	1	7	3	4	3	5	2	7
i=2	1	2	1	7	3	4	3	5	2	7
i=3	1	2	7	7	3	4	3	5	2	7
i=4	1	2	7	3	3	4	3	5	2	7

补充案例

编写程序：把一个数列中所有系统的数删到只剩下一个

1	2	1	7	3	4	3	5	2	7
---	---	---	---	---	---	---	---	---	---

1	2	7	3	4	5
---	---	---	---	---	---

算法思想：

第1个元素默认为已经处理好的结果，从第2个元素开始，依次和前面已经处理好的元素作比较，判断是否相同，若不相同，则将其存入pos指定的位置，同时将pos指向下一个存放位置(pos+1)，之后继续处理数组中下一个未处理的元素。若相同，则直接跳过该元素，继续处理下一个未处理的元素，直到所有元素都处理完。

i=0:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=1:	<u>1</u>	<u>2</u>	1	7	3	4	3	5	2	7
i=2:	<u>1</u>	<u>2</u>	1	<u>7</u>	3	4	3	5	2	7
i=3:	<u>1</u>	<u>2</u>	<u>7</u>	7	<u>3</u>	4	3	5	2	7
i=4:	<u>1</u>	<u>2</u>	<u>7</u>	<u>3</u>	3	<u>4</u>	3	5	2	7
i=5:	<u>1</u>	<u>2</u>	<u>7</u>	<u>3</u>	<u>4</u>	4	<u>3</u>	5	2	7
i=6:	<u>1</u>	<u>2</u>	<u>7</u>	<u>3</u>	<u>4</u>	4	3	<u>5</u>	2	7
i=7:	<u>1</u>	<u>2</u>	<u>7</u>	<u>3</u>	<u>4</u>	<u>5</u>	3	5	<u>2</u>	7
i=8:	<u>1</u>	<u>2</u>	<u>7</u>	<u>3</u>	<u>4</u>	<u>5</u>	3	5	2	<u>7</u>
i=9:	<u>1</u>	<u>2</u>	<u>7</u>	<u>3</u>	<u>4</u>	<u>5</u>	3	5	2	7

```
const int N=10;    int a[N];
```

```
int i,j,pos=1;
```

输入数组a的所有元素

```
i=1 //从第二个元素开始处理
```

```
i<N
```

```
j=0 //与已经处理好的元素比较是否相等
```

```
j<pos
```

T $a[i]==a[j]$ N

```
break;
```

```
j++;
```

T $j \geq pos$ N

```
a[pos]=a[i];
```

```
pos++;
```

```
i++;
```

输出a[0]~a[pos-1] //删除重复数据的结果

pos=1



i=0:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=1:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=2:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=3:	<u>1</u>	2	7	7	3	4	3	5	2	7
i=4:	<u>1</u>	2	7	3	3	4	3	5	2	7
i=5:	<u>1</u>	2	7	3	4	4	3	5	2	7
i=6:	<u>1</u>	2	7	3	4	4	3	5	2	7
i=7:	<u>1</u>	2	7	3	4	5	3	5	2	7
i=8:	<u>1</u>	2	7	3	4	5	3	5	2	7
i=9:	<u>1</u>	2	7	3	4	5	3	5	2	7

```

#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    const int N=10;
    int a[N];
    int i,j,pos=1;
    for(i=0;i<N;i++)
        cin>>a[i];
    for(i=1;i<10;i++) //从第2个元素开始处理
    {
        for(j=0;j<pos;j++) //与结果集中元素比较
            if(a[i]==a[j]) //与结果集中元素相同
                break;
        if(j<pos) //若与结果集中所有元素均不相同
        {
            a[pos]=a[i]; //将元素写入结果集(pos指示写入的位置)
            pos++; //pos指向下一个写入位置
        }
    }
    for(i=0;i<pos;i++) //输出结果集中所有元素, 下标从0~pos-1
        cout<<a[i]<<" ";
    cout<<endl;
}

```

pos
↓

i=0:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=1:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=2:	<u>1</u>	2	1	7	3	4	3	5	2	7
i=3:	<u>1</u>	2	7	7	3	4	3	5	2	7
i=4:	<u>1</u>	2	7	3	3	4	3	5	2	7
i=5:	<u>1</u>	2	7	3	4	4	3	5	2	7
i=6:	<u>1</u>	2	7	3	4	4	3	5	2	7
i=7:	<u>1</u>	2	7	3	4	5	3	5	2	7
i=8:	<u>1</u>	2	7	3	4	5	3	5	2	7
i=9:	<u>1</u>	2	7	3	4	5	3	5	2	7

游戏时间

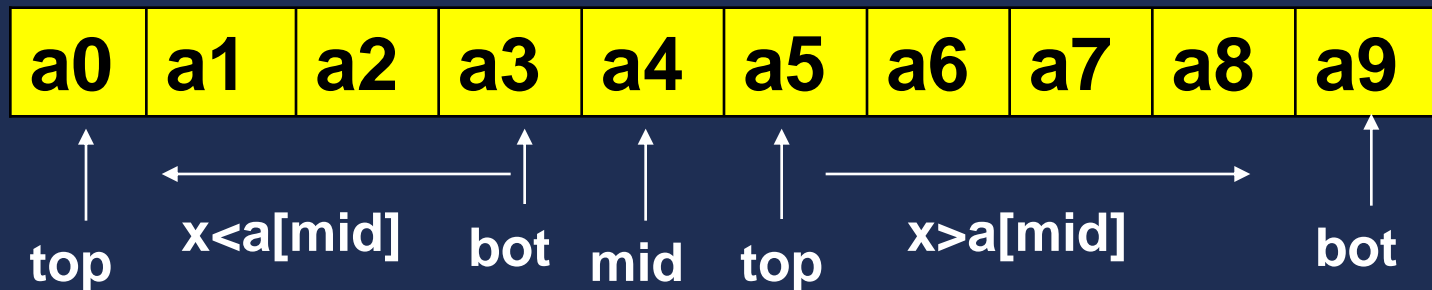


折半查找

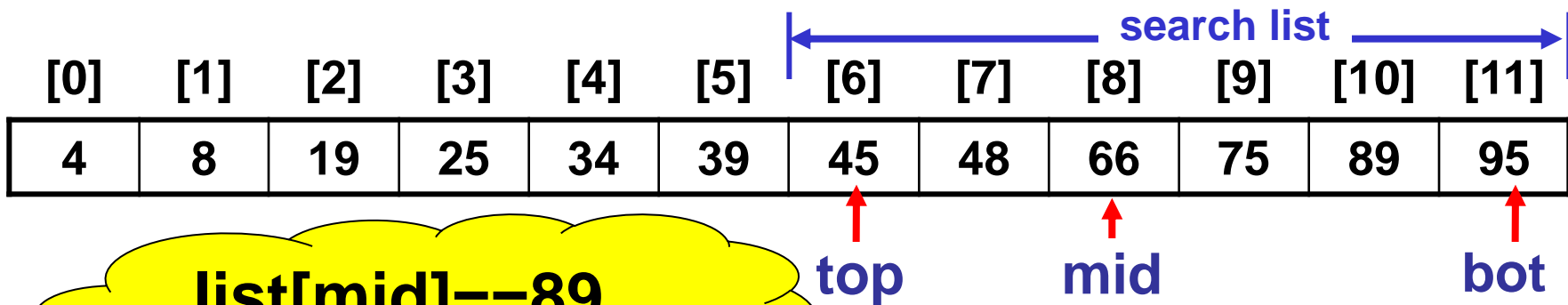
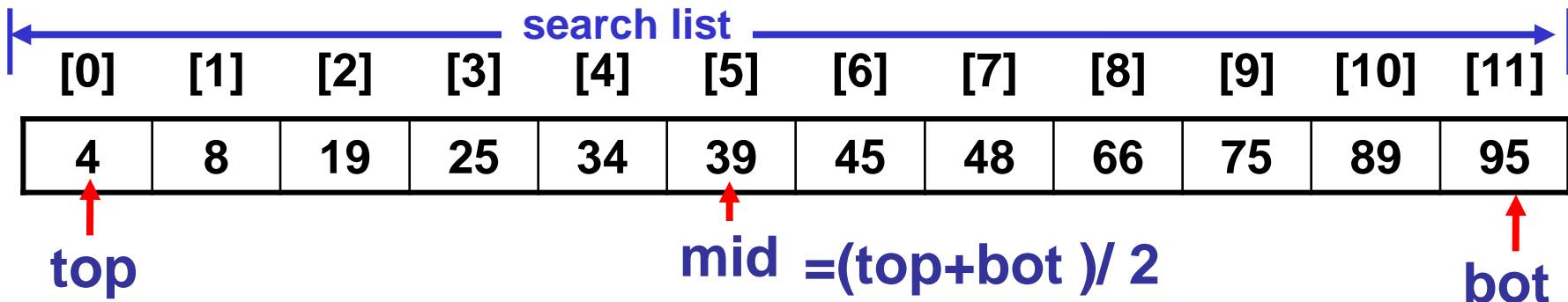
P₁₅₀

将数列有序化(递增或递减)排列，查找过程中采用折半方式查找，即先以有序数列的中点位置为比较对象，如果要找的元素X的值小于该中点元素，则将待查序列缩小为数列的左半部分，否则为右半部分。通过一次比较，将查找区间缩小一半。

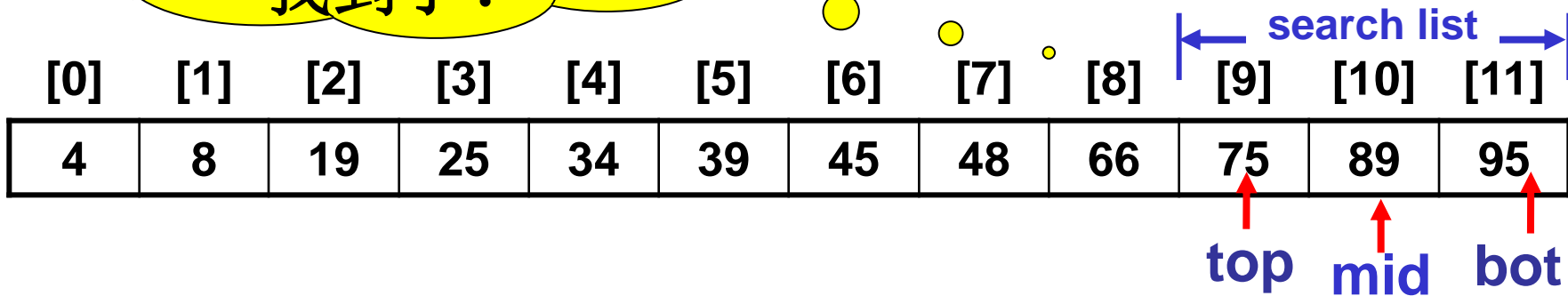
重复上述操作，直到找到元素为止或是超出处理范围为止。



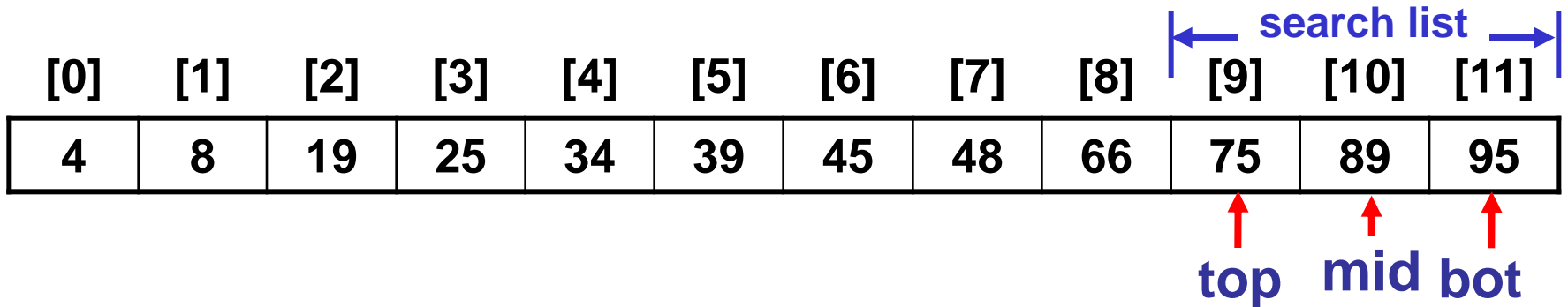
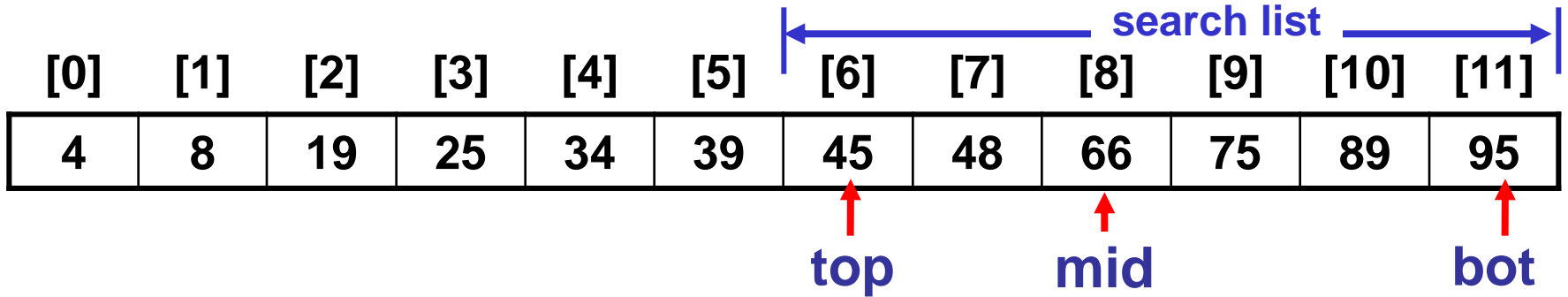
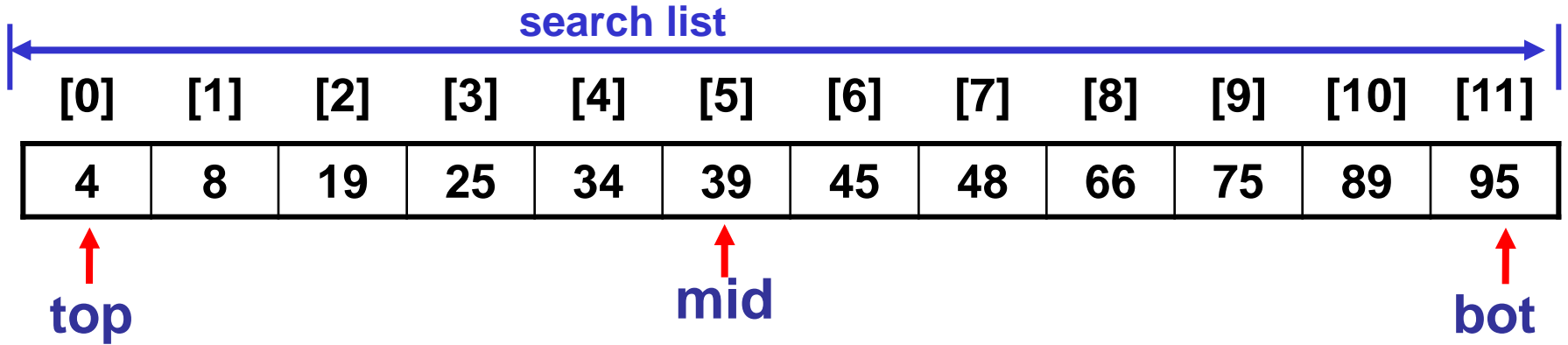
折半查找示例1： 搜索89是否在如下顺序表中



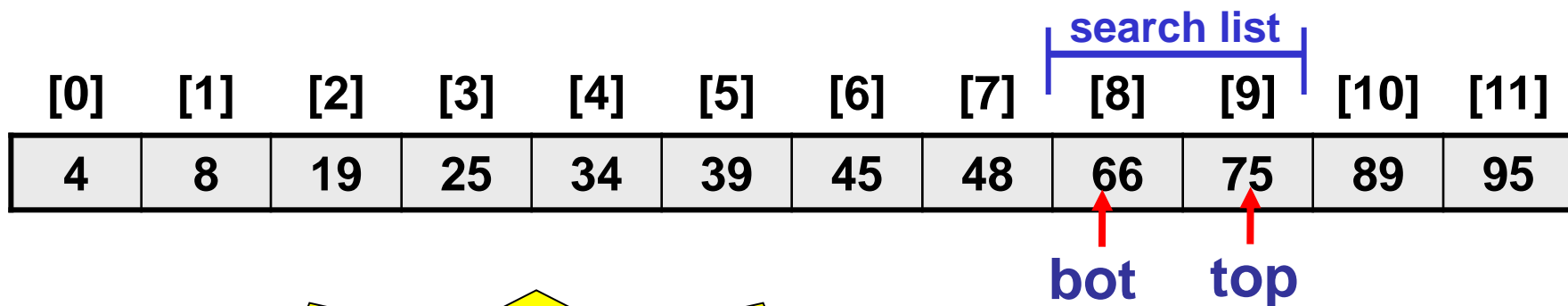
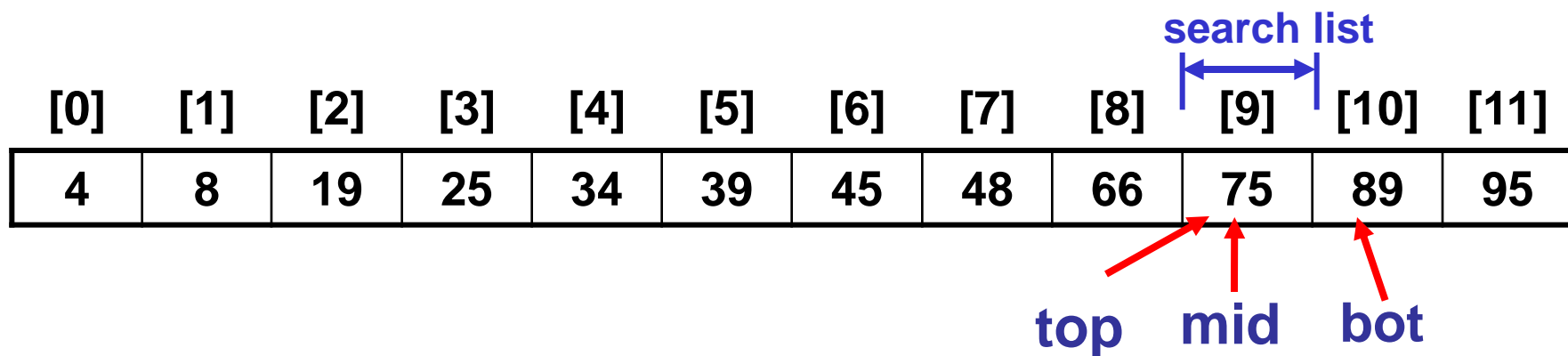
list[mid]==89
找到了！



折半查找示例2： 查找74是否在表中？

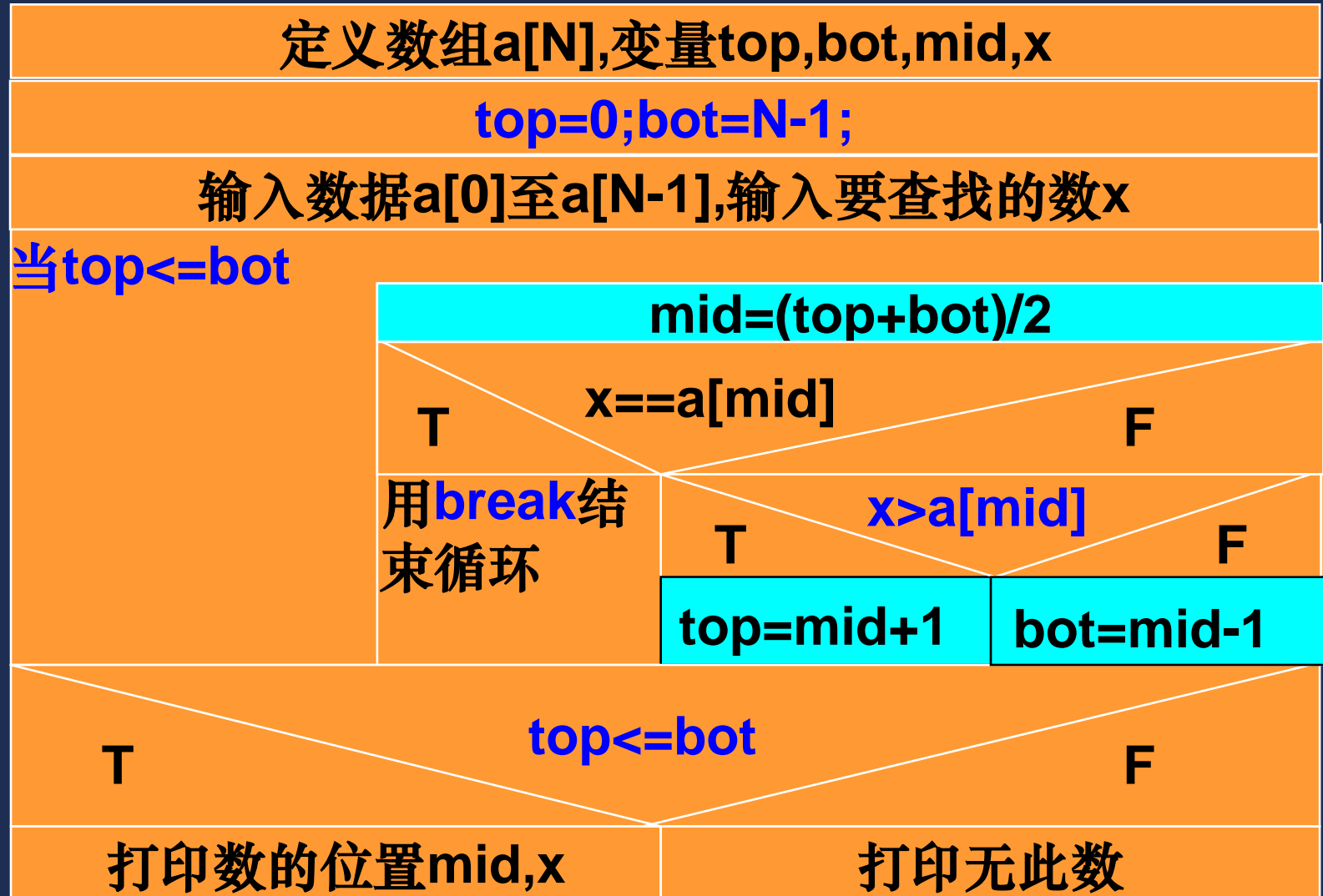


折半查找示例2：查找74是否在表中？



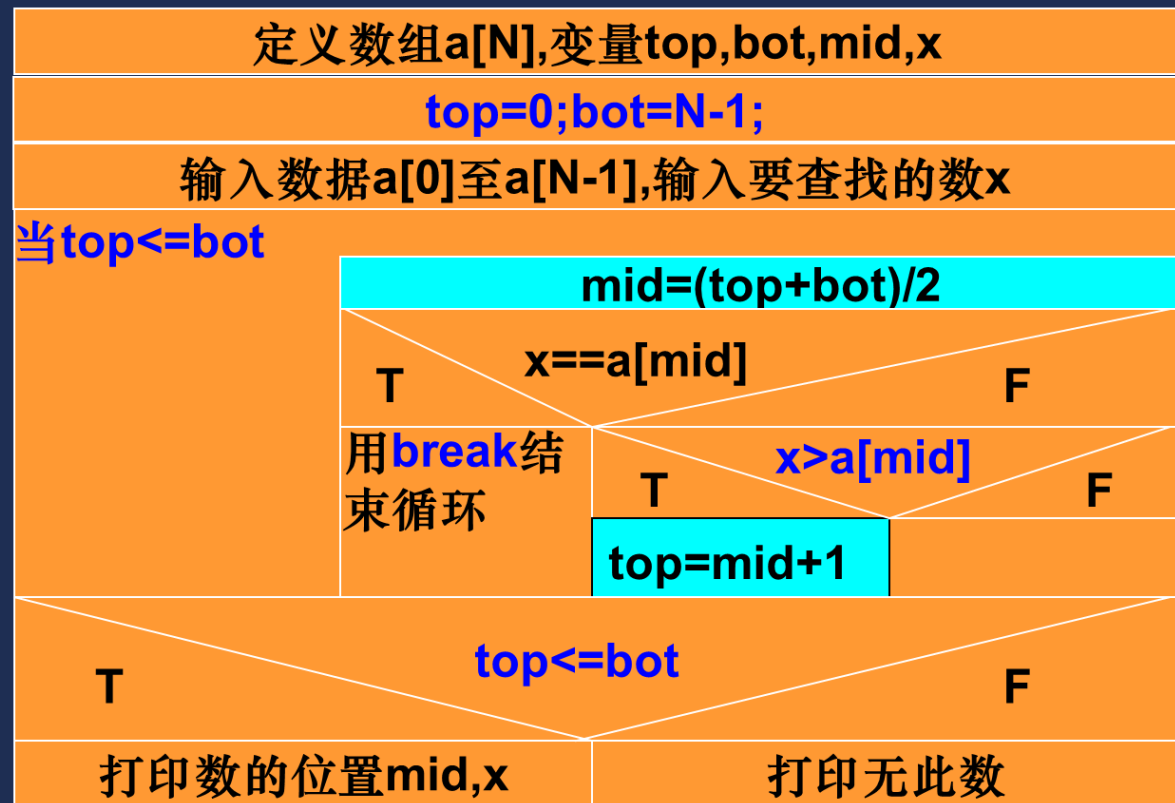
此时 $bot < top$, 结束
查找

折半查找算法的N-S流程图



```
#include <iostream>
using namespace std;
void main( )
{
```

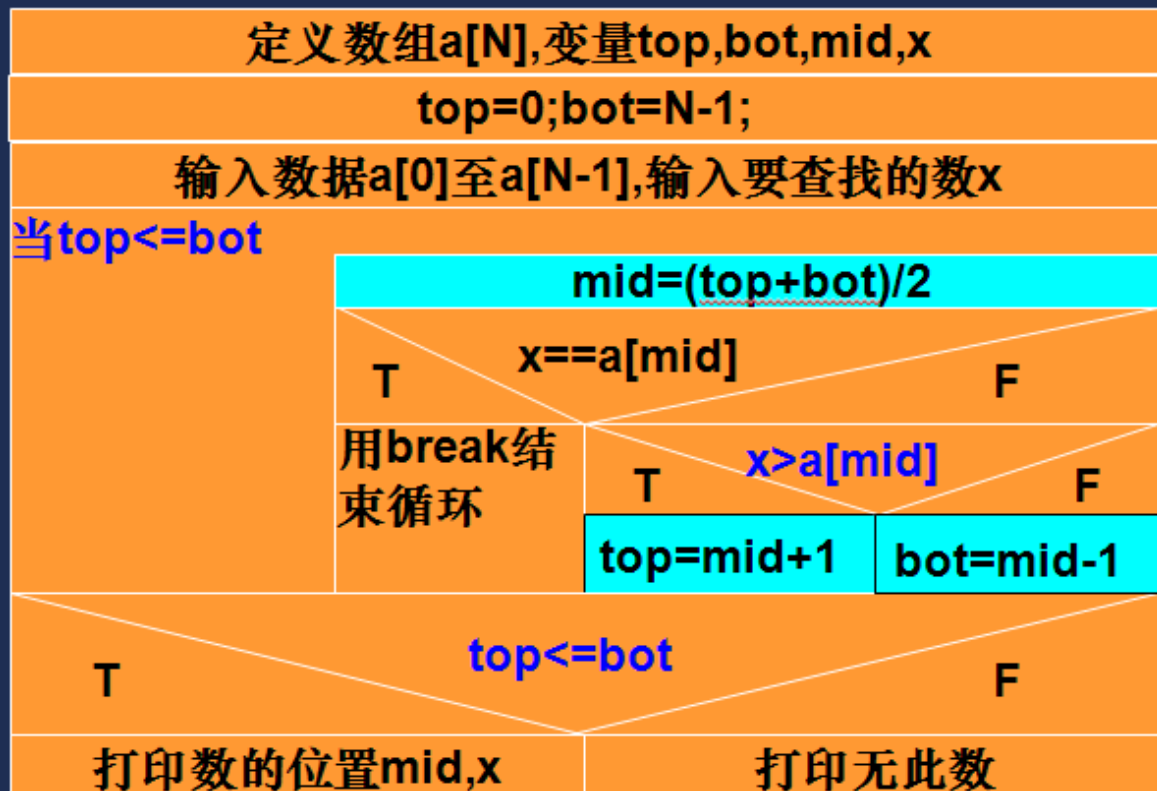
```
    const int N=10;
    int a[N],i, top=0,bot=9,mid, x;
    cout<<"Please input ten numbers(from small to big):"<<endl;
    for(i=0;i<N;i++)    //利用循环输入有序数列
        cin>>a[i];
    cout<<"Please input the number to search:"<<endl;
    cin>>x;    //从键盘接收需要查找的数
```



```

while( top<=bot )
{
    mid=(bot+top)/2;
    if(x==a[mid])
        break;
    //结束最近的封闭循环体
else
    if(x>a[mid])
        top=mid+1;
    else
        bot=mid-1;
}
if( top<=bot )
    cout<<x<<" is at the "<<mid+1<<" position.";
else
    cout<<"No this number.";
}

```



折半查找

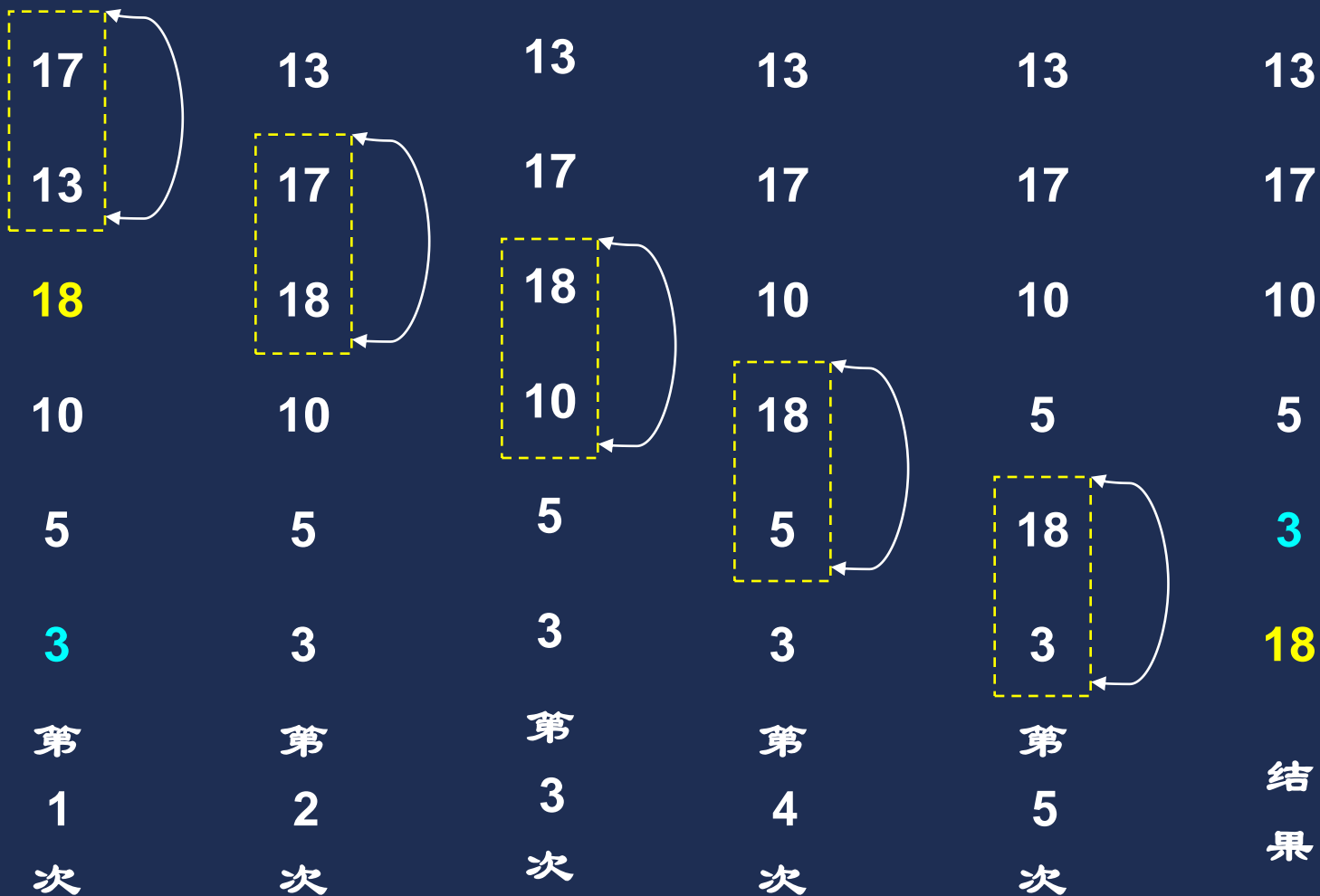
折半查找是一种高效的查找方法(算法效率为 $\log_2 N$)。它可以明显减少比较次数，提高查找效率。但是，折半查找的先决条件是查找表中的数据元素必须有序。折半查找的存储结构采用一维数组存放。

数据排序是数据处理中经常使用的一种重要的操作，它的功能是将无序的数据序列调整为有序的。排序的方法很多，比如选择法，冒泡法，希尔排序法等。不论采用何种排序方法，要排序的原始数据均可用一维数组表示。

1.冒泡排序算法 P₁₄₇

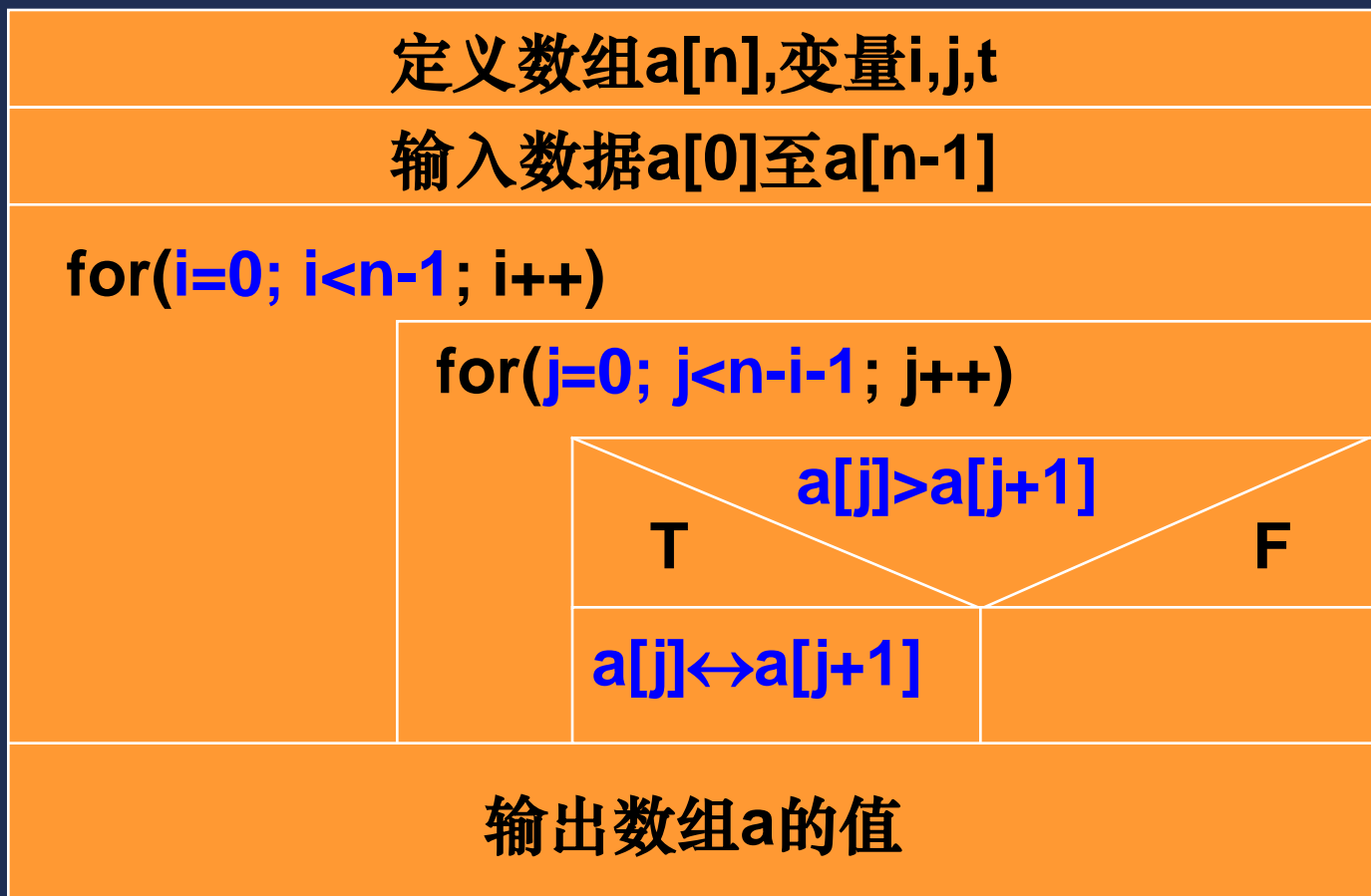
起泡法算法又称为冒泡法。就是先将 n 个数进行**第一轮比较**，**每次比较相邻的两个数**，若不满足顺序条件，就将两者对调，**经过 $n-1$ 次两两比较后**，最大的数“沉底”($a[n-1]$)，而最小的数“上升”了一个位置；然后在剩余的数中进行**第二轮比较**，**经过 $n-2$ 次两两比较**，最大的数“沉底”($a[n-2]$)，而最小的数“上升”了一个位置；.....；如此进行下去，直到全部数排序完成。

冒泡法示意图(第i轮)



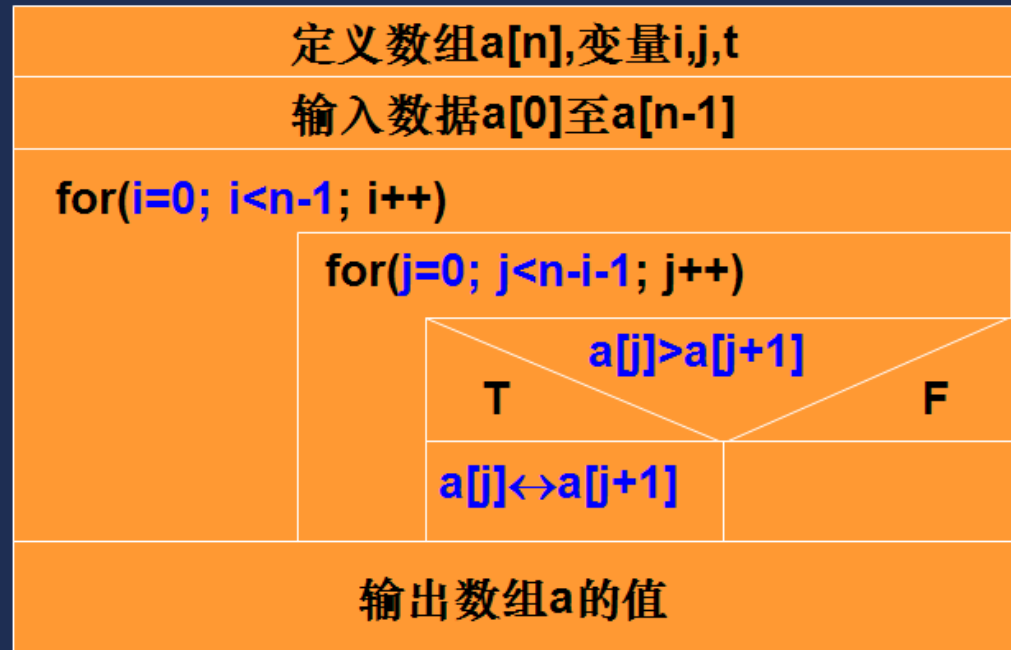
用冒泡法对n个数排序的轮数是 $n-1$ ，而在第i轮中，所要进行两两比较的次数是 $n-i-1$ 次(i的初值为0)。

冒泡法N-S流程图



冒泡法C++源代码

```
#include <iostream>
using namespace std;
void main( )
{
    const int N=20;
    int a[N];
    int i,j,t,n;
    cout<<"Input n:";
    cin>>n;
    cout<<"Input numbers:"<<endl;
    for(i=0; i<n; i++)
        cin>>a[i];    //输入数组元素
```



```

for( i=0; i<n-1; i++)    //对n个数排序的轮数是n-1
    for( j=0; j<n-i-1; j++) //第i轮中,进行两两比较的次数是n-i-1次
        if(a[j]>a[j+1])
            { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }

```

```

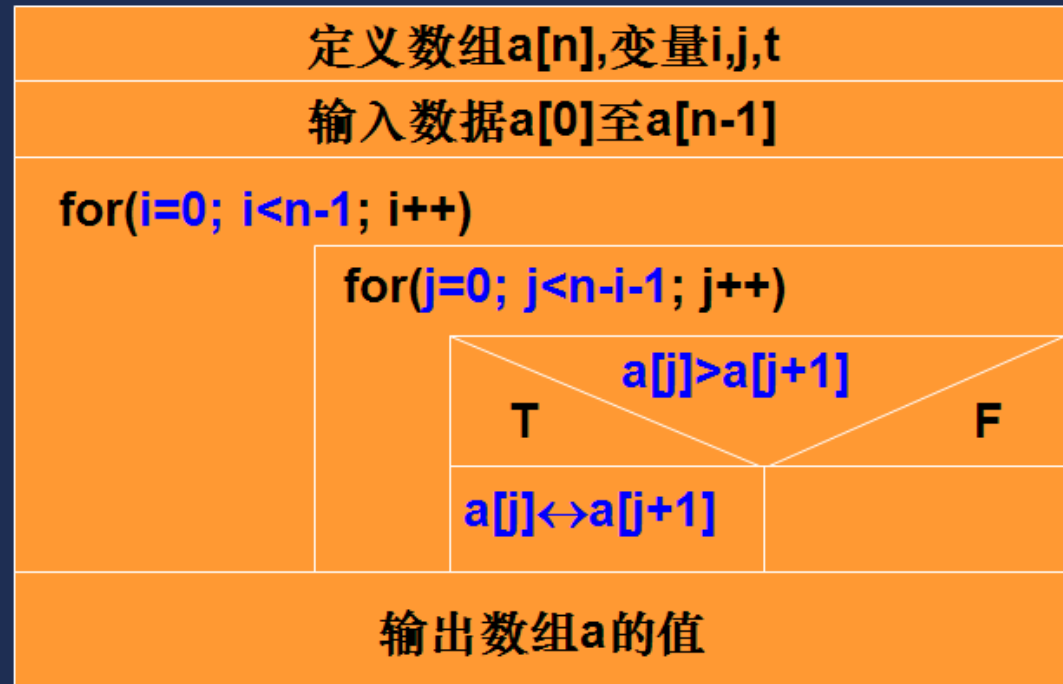
cout<<"The sorted numbers is:"<<endl;

```

```

for(i=0; i<n; i++)
{
    cout.width(5);
    cout<<a[i];
}
cout<<endl;
}

```



2. 选择排序算法

P₁₄₅

第一轮将n个数中的最小数与第1个数($a[0]$)对调，第二轮将剩余的n-1个数中的最小数与第2个数($a[1]$)对调，……，每比较一轮，找出一个未经排序的数中的最小数，与第i个数的数对调。n个待排的数总共需要

算法关键问题：

- 如何在每轮中找到最小数(位置)？
- 如何实现两数值的交换？



选择排序算法描述

```
const int N=100;
int a[N],n;
cin>>n; //n为数组长度
for(i=0;i<n;i++) cin>>a[i];
for(i=0;i<n-1;i++) //n-1轮处理
{
    每一轮处理:
    找出最小数和a[i]做交换;
}
```

```
if(min!=i)
    // a[i]↔a[min]
    {
        t=a[i];
        a[i]=a[min];
        a[min]=t;
    }
```

```
//找出数组中最小值及其位置
min=i;
int min=0; //
int j;
for(j=i+1;j<n;j++) //从默认值的后一个元素开始比较
{
    if(a[j]<a[min])
        min=j;
}
//交换a[min]和a[i]
```

第1轮: $a[\text{min}] \leftrightarrow a[0]$ /* 默认最小值为第1个元素, 即: $i=0$, $\text{min}=i$, 从第2个($j=i+1$)元素开始比较*/

第2轮: $a[\text{min}] \leftrightarrow a[1]$ /* 默认最小值为第2个元素, 即: $i=1$, $\text{min}=i$, 从第3个($j=i+1$)元素开始比较*/

.....

第n-1轮: $a[\text{min}] \leftrightarrow a[n-2]$ /* 默认最小值为第n-2个元素, 即: $i=n-2$, $\text{min}=i$, 从第n-1个($j=i+1$)元素开始比较*/

选择法算法的N-S流程图



选择排序算法C++代码

```
#include<iostream>
using namespace std;
void main( )
{
    const int N=20;
    //定义符号常量N
    int a[N];
    //定义整型数组,等价于int a[20]
    int i,j,t,n,min;
    cout<<"Input n:";
    cin>>n;
    cout<<"Input numbers:"<<endl;
    for(i=0;i<n;i++)  cin>>a[i];  //利用循环输入待排序的一组数
```

定义数组a[n],变量i,j,t,min(记录最小值位置)

输入数据a[0]至a[n-1]

for(i=0;i<n-1;i++) //n个元素一个进行n-1轮比较

min=i //记录本轮中最小值所在位置

for(j=i+1;j<n;j++) //从默认最小值的下一个元素比较

a[j]<a[min]

T

F

min=j //记录最小值下标

min!=i

T

F

a[i]↔a[min]

输出排序后数组a的所有元素

```
for(i=0;i<n-1;i++) //n个数比较n-1轮
```

```
{
    min = i; //初始化min的值
    for(j=i+1; j<n; j++)
    {
        if (a[j] < a[min])
            min = j;
        //记录最小值所在位置
    }
    if (min != i )
    {
        //交换a[i]和a[min]
        t = a[i];    a[i] = a[min];    a[min] = t;
    }
}
```

```
cout<<"The sorted numbers is:"<<endl;
```

```
for(i=0;i<n;i++)
```

```
{
    cout.width(5); //输出格式控制
    cout<<a[i];
}
```

定义数组a[n],变量i,j,t,min(记录最小值位置)

输入数据a[0]至a[n-1]

for(i=0;i<n-1;i++) //n个元素一个进行n-1轮比较

min=i //记录本轮中最小值所在位置

for(j=i+1;j<n;j++) //从默认最小值的下一个元素比较

a[j]<a[min]

T

F

min=j //记录最小值下标

T

min!=i

F

a[i]↔a[min]

输出排序后数组a的所有元素

```
c:\Documents and Settings...
Input n:5
Input numbers:
12 45 36 6 3
```

解决方案资... 类视图

监视 1

名称	值	类型
j	1	int
i	0	int
a[0]	12	int
a[1]	45	int
a[2]	36	int
a[3]	6	int
a[4]	3	int
min	0	int

监视 1

名称	值	类型
j	4	int
i	0	int
a[0]	12	int
a[1]	45	int
a[2]	36	int
a[3]	6	int
a[4]	3	int
min	4	int

解决方案资... 类视图

监视 1

名称	值	类型
j	5	int
i	0	int
a[0]	3	int
a[1]	45	int
a[2]	36	int
a[3]	6	int
a[4]	3	int
min	4	int

解决方案资... 类视图

监视 1

名称	值	类型
j	5	int
i	0	int
a[0]	3	int
a[1]	45	int
a[2]	36	int
a[3]	6	int
a[4]	12	int
min	4	int

改进的选择法算法

定义数组a[n],变量i,j,t

输入数据a[0]至a[n-1]

for(i=0;i<n-1;i++)

for(j=i+1;j<n;j++)

T

$a[i] > a[j]$

F

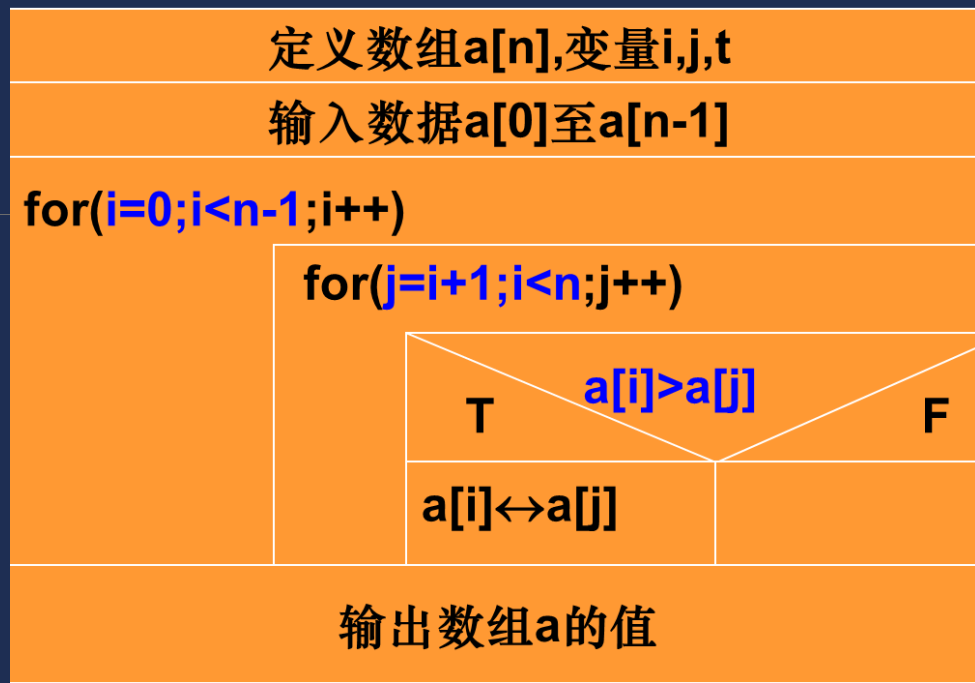
$a[i] \leftrightarrow a[j]$

输出数组a的值

```

#include<iostream>
using namespace std;
void main( )
{
    const int N=20;
    int a[N];
    int i,j,t,n;
    cout<<"Input n:";
    cin>>n;
    cout<<"Input numbers:"<<endl;
    for(i=0;i<n;i++)
        cin>>a[i];        //输入数组元素
    for(i=0;i<n-1;i++)
        for(j=i+1; j<n; j++)
            if(a[i]>a[j])
                { t=a[i]; a[i]=a[j]; a[j]=t; }    //排序
    cout<<"The sorted numbers is:"<<endl;
    for(i=0;i<n;i++)
    {
        cout.width(5);
        cout<<a[i];
    }
    //输出
}

```



```
c:\Documents and Set...
input 5 numbers:
12 45 36 6 3
```

解决方案资... 类视图

监视 1

名称	值	类型
j	1	int
i	0	int
a[0]	12	int
a[1]	45	int
a[2]	36	int
a[3]	6	int
a[4]	3	int

解决方案资... 类视图

监视 1

名称	值	类型
j	3	int
i	0	int
a[0]	6	int
a[1]	45	int
a[2]	36	int
a[3]	6	int
a[4]	3	int

解决方案资... 类视图

监视 1

名称	值	类型
j	3	int
i	0	int
a[0]	6	int
a[1]	45	int
a[2]	36	int
a[3]	12	int
a[4]	3	int

解决方案资... 类视图

监视 1

名称	值	类型
j	4	int
i	0	int
a[0]	3	int
a[1]	45	int
a[2]	36	int
a[3]	12	int
a[4]	3	int

解决方案资... 类视图

监视 1

名称	值	类型
j	4	int
i	0	int
a[0]	3	int
a[1]	45	int
a[2]	36	int
a[3]	12	int
a[4]	6	int

补充：用数组名作为函数参数

利用改进的选择法排序，将键盘输入的10个整数按从小到大的顺序排序。

思路分析：用一个子函数实现选择法排序，主函数调用该子函数实现排序。

数组的定义，变量的定义

数组的输入

调用子函数进行排序(选择法)

输出排序后的数组

主函数算法描述

如何定义子函数？

需要对一组数进行排序，故形参应为数组。排序的结果由数组带回(传地址，可以返回多个结果)，故不需要返回值。

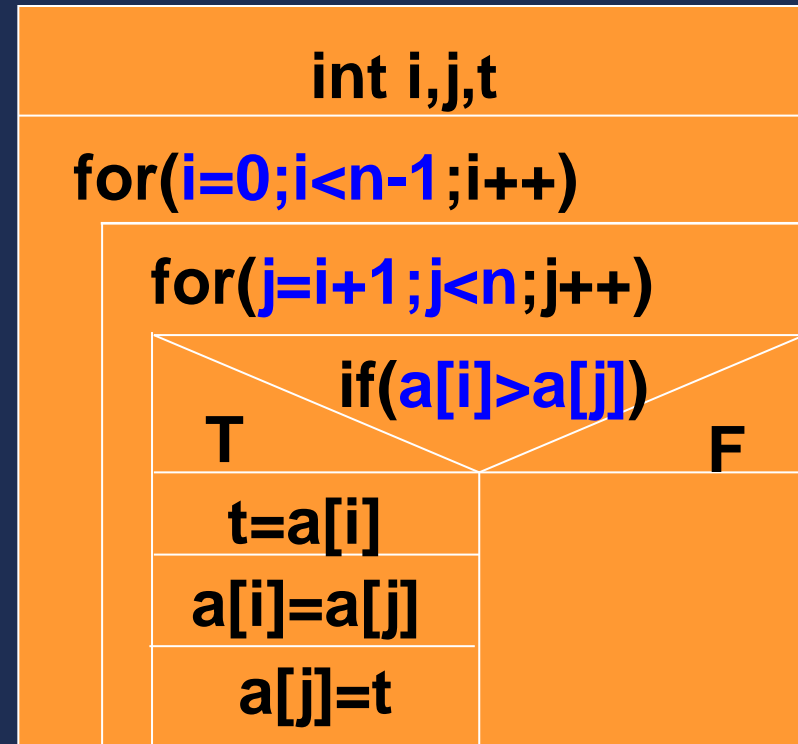
```
void sort(int a[ ], int n)
{.....}
```



```
void sort(int a[ ], int n)
{.....}
```

```
#include<iostream>
using namespace std;
```

```
void sort (int a[ ],int n)
{
    int t=0,i,j;
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
            if(a[i]>a[j])
            {
                t=a[i]; a[i]=a[j]; a[j]=t;
            }
}
```



子函数算法描述

```
void main( )
```

```
{
```

```
    const int N=10;
```

```
    int a[N],i;
```

```
    cout<<"Input 10
```

```
    for(i=0;i<N;i++)
```

```
        cin>>a[i];
```

```
    sort (a,N); //函数调用实现排序
```

```
    cout<<"The sorted numbers is:"<<endl;
```

```
    for(i=0;i<N;i++)
```

```
        cout<<a[i]<<endl;
```

```
}
```

补充练习：

用函数的形式实现插入

void insert(int a[],int n, int x);

折半查找

bool find(int a[], int n, int x);

数组名作为函数参数时，应注意以下几点：

- **实参与形参都是数组名(形参数组名后的方括号不能省略掉!!!)。**

```
void sort(int a[ ], int n) {.....}
```

- **实参数组与形参数组类型应一致，如不一致，结果将出错。**

4.4 二维数组 P₁₂₅

1. 二维数组的定义

类型说明符 数组名[常数表达式][常数表达式];

例: `int a[3][4];`

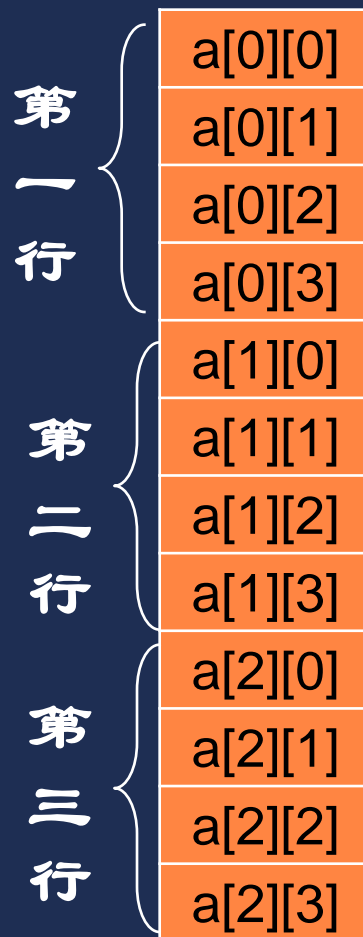
可以理解为:

a	[<code>a[0]</code>	——	<code>a₀₀</code>	<code>a₀₁</code>	<code>a₀₂</code>	<code>a₀₃</code>
		<code>a[1]</code>	——	<code>a₁₀</code>	<code>a₁₁</code>	<code>a₁₂</code>	<code>a₁₃</code>
		<code>a[2]</code>	——	<code>a₂₀</code>	<code>a₂₁</code>	<code>a₂₂</code>	<code>a₂₃</code>

二维数组a可以看成是由3个元素组成的一维数组，而每个元素又是一个一维数组，即可以看成由一维数组构成的数组。

2. 二维数组的存储 P₁₂₅

二维数组在内存中存放时是按行存放的，如数组int a[3][4]的存储顺序为：



3. 二维数组的引用

访问二维数组的元素时，也是采用下标法，需要包含数组名、行下标(0~n-1)和列下标(0~m-1)。

例如定义二维数组：

```
float a[5][4],b[3][4];  
..... //给a,b数组赋值  
b[1][2]=a[2][4]/2;
```

下标不要越界

4. 二维数组的初始化 P₁₂₅

- 分行给二维数组赋初值

例如：int a[3][4]={1,2,3,4},{5,6,7,8},{9,10,11,12};

- 将所有数据写在一个{}内，按顺序赋值

例如：int a[3][4]={1,2,3,4,5,

此时表示每行数据的
花括号不能省略，否
则赋值结果就改变了！

- 可以对部分元素赋初值

例如：int a[3][4]={0,0,0,1},{0,6},{0,0,11};

- 如果对全部元素都赋初值，则定义数组时对第一维的大小可以忽略，但第二维的大小不能省。

例如：int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12}

5. 二维数组元素的访问

- 回忆对一维数组元素的访问
- 数组是一种**复合类型**，是不能作为一个整体进行访问和处理的，只能按元素进行个别的访问和处理。

```
for(i=0;i<n;i++)  
    cin>>a[i];
```

- 对二维数组元素的赋值(**行和列**)

```
for(i=0;i<n;i++)  
    for (j=0;j<m;j++)  
        cin>>a[i][j];
```

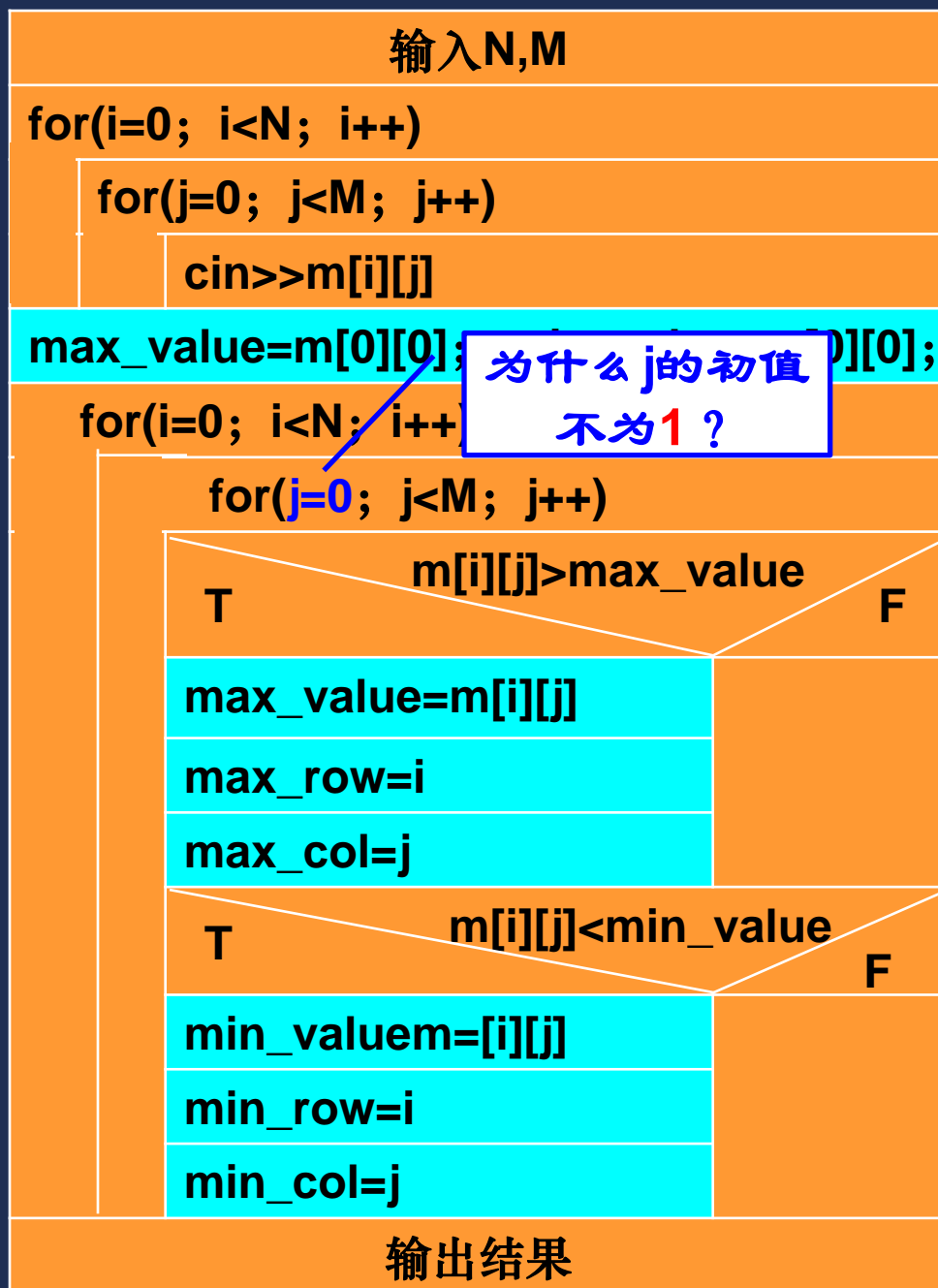
例：二维数组应用举例

问题：求一个整型数值二维表中的最大值和最小值及其相应的行列号。

分析：

- 可以用二维数组表示二维表，分别用max_value、min_value表示要求的最大值和最小值，max_row、min_row表示要求的最大值和最小值的行号，max_col、min_col表示要求的最大值和最小值的列号。
- 将max_value、min_value的初值设置为m[0][0]，然后重复用max_value(或min_value)与其余的元素进行比较，始终保持max_value(或min_value)中存放比较结果的最大值(或最小值)

N-S 流程图



为什么j的初值
不为1？

C++源代码

```
#include <iostream>
using namespace std;
void main( )
{
    const int K=10;
    int m[K][K],i,j,M,N;
    cout<<"请输入数组的行列值："<<endl;
    cin>>N>>M; //实际行、列长度
    int max_value, max_row=0,
        max_col=0;
    int min_value, min_row=0,
        min_col=0;
    for(i=0; i<N; i++)
        for(j=0; j<M; j++)
            cin>>m[i][j]; //初始化二维数组
```

int m[K][K],m,n;
error C2040: "m": "int"
与 "int [3][10]"的
间接寻址级别不同

```
max_value=m[0][0]; min_value=m[0][0]; //设置默认最大、最小值
```

输入N,M

```
i=0; i<N; i++)
```

```
for(j=0; j<M; j++)
```

```
cin>>m[i][j]
```

```
max_value=m[0][0]; min_value=m[0][0];
```

```
for(i=0; i<N; i++)
```

```
for(j=0; j<M; j++)
```

```
m[i][j]>max_value
```

```
max_value=m[i][j]
```

```
max_row=i
```

```
max_col=j
```

```
m[i][j]<min_value
```

```
min_value=m[i][j]
```

```
min_row=i
```

```
min_col=j
```

输出结果

```

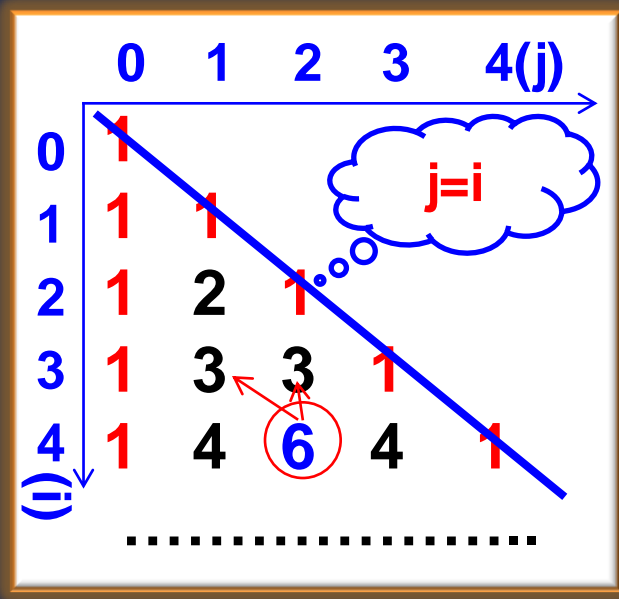
for(i=0;i<N;i++)
    for(j=0;j<M;j++)
    {
        if(m[i][j]>max_value)
        {
            max_value=m[i][j];
            max_row=i; max_col=j;
        } //找出最大值，并记录行列号
        if(m[i][j]<min_value)
        {
            min_value=m[i][j];
            min_row=i; min_col=j;
        } //找出最小值，并记录行列号
    }
//按要求输出最大、最小值
cout<<"The max is m["<< max_row
    <<"["<<max_col <<"]="
    << max_value <<endl;
cout<<"The min is m["<< min_row
    <<"["<< min_col <<"]="
    <<min_value <<endl;
} //主函数结束

```

输入N,M			
for(i=0; i<N; i++)			
	for(j=0; j<M; j++)		
		cin>>m[i][j]	
max_value=m[0][0]; min_value=m[0][0];			
for(i=0; i<N; i++)			
for(j=0; j<M; j++)			
	m[i][j]>max_value		
	max_value=m[i][j]		
	max_row=i		
	max_col=j		
	m[i][j]<min_value		
	min_valuem=[i][j]		
	min_row=i		
	min_col=j		
	输出结果		

例 二维数组应用举例 P₁₅₄

打印以下的扬辉三角形。



杨辉三角最本质的特征是：它的两条斜边都是由数字1组成的，而其余的数则是等于它肩上的两个数之和。

所有值为1的元素该如何表示？

```
for (i=0;i<n;i++)
    {    a[i][0]=1;    a[i][i]=1;    }
```

三角形中，其余位置上的数字等于它肩上的两个数之和：

```
for (i=2;i<n;i++)
```

```
for (j=1;j<i;i++)
```

$$a[i][j] = a[i-1][j-1] + a[i-1][j]$$

```
for(i=0;i<10;i++) //在数组中写入所有为1的数
```

```
a[i][0]=1
```

```
a[i][i]=1
```

```
for(i=2;i<10;i++)
```

```
for(j=1;j<i;j++) //第3行第2列开始
```

```
a[i][j]=a[i-1][j-1]+a[i-1][j]
```

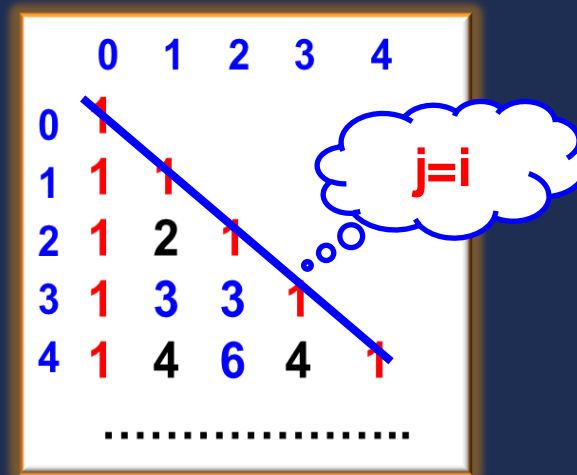
```
for(i=0;i<10;i++) //输出10行
```

```
for(j=0;j<=i;j++) //控制每行输出元素
```

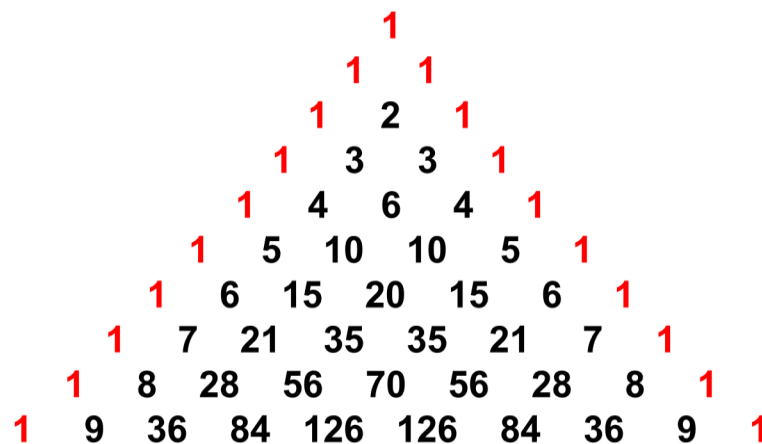
```
输出a[i][j]
```

```
输出换行
```

N-S流程图



杨辉三角输出为一个下三角。



杨辉三角若要输出为一个等腰三角形如何处理？

```
#include <iostream>
#include <iomanip>
using namespace std;
void main( )
```

```
{
    const int max=10;
    int a[max][max];
    int n,i,j;
    cout<<"请输入杨辉三角的排数 (<10)"<<endl;
    cin>>n;

    for(i=0;i<n;i++)
        { a[i][0]=1; a[i][i]=1; }
```

```
for(i=0;i<10;i++) //在数组中写入所有为1的数
```

```
a[i][0]=1
```

```
a[i][i]=1
```

```
for(i=2;i<10;i++)
```

```
for(j=1;j<i;j++) //第3行第2列开始
```

```
a[i][j]=a[i-1][j-1]+a[i-1][j]
```

```
for(i=0;i<10;i++) //输出10行
```

```
for(j=0; j++) //控制每行输出元素
```

```
输出a[i][j]
```

```
输出换行
```

//初始化两条斜边上元素的值

```
for(i=2;i<n;i++)
```

```
for(j=1;j<=i-1;j++)
```

a[i][j]=a[i-1][j-1]+a[i-1][j]; //计算杨辉三角中其它位置的元素值

```
cout<<"YAN HUI Triangle:"<<endl;
```

```
for(i=0;i<n;i++) //控制输出行数
```

{

```
for(j=0;j<=i;j++) //控制每行输出元素
```

```
cout<<setw(4)<<a[i][j];
```

```
cout<<endl; //换行输出下一行
```

杨辉三角若要输出为一个等腰三角形如何实现？

杨辉三角若要输出为一个等腰三角形如何实现？

```

      1
     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
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1

```

```
for(i=0,i<10;i++)
```

a[i][0]=1

a[i][i]=1

```
for(i=2;i<10;i++)
```

```
for(j=1;j<i;j++) //第3行第2列开始
```

$$a[i][j] = a[i-1][j-1] + a[i-1][j]$$

```
for(i=0;i<10;i++) //输出10行
```

```
for(j=0; j<=i; j++) //控制每行输出元素
```

输出a[i][j]

输出换行

例：求矩阵的转置 P₁₆₁

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \xrightarrow{\text{转置}} B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

数据分析： 矩阵是由若干行若干列组成的，因此对矩阵可采用二维数组表示。

$$b[j][i] = a[i][j]$$

C++源代码

```
#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    const int N=2,M=3;
    float a[N][M],b[M][N];
    int i,j;
    cout<<"Input array a"<<endl;
    for(i=0;i<N;i++)
        for(j=0;j<M;j++)
            cin>>a[i][j];
```

```
for(i=0;i<N;i++)
```

```
{
```

```
    for(j=0;j<M;j++)
```

```
    {
```

```
        cout<<setw(4)<<a[i][j]; //输出数组a，并进行格式控制  
        b[j][i]=a[i][j]; //转置
```

```
    }
```

```
    cout<<endl; //输完一行后，换行再继续输出下一行
```

```
}
```

```
cout<<"output Array b:"<<endl;
```

```
for(i=0;i<M;i++)
```

```
{
```

```
    for(j=0;j<N;j++)
```

```
        cout<<setw(4)<<b[i][j]; //输出数组b，并进行格式控制  
    cout<<endl; //输完一行后，换行再继续输出下一行
```

```
}
```

```
}
```

如何用一个二维数组实现方阵的转置？

示例

自动产生 $N \times N$ 数据（ N 取值最大不超过100）存入a数组，数据形成规律如下图示（呈S形），并取出a的下三角形区域数据输出，输出形式如下图所示（呈等腰三角形）。

```
C:\WINDOWS\system32\cmd.exe
请输入方阵的阶数(n):
6
  1   2   3   4   5   6
12  11  10   9   8   7
13  14  15  16  17  18
24  23  22  21  20  19
25  26  27  28  29  30
36  35  34  33  32  31

      1
    12  11
  13  14  15
    24  23  22  21
      25  26  27  28  29
        36  35  34  33  32  31
请按任意键继续. . .
```

请输入方阵的阶数<n>:

6

1	2	3	4	5	6
12	11	10	9	8	7
13	14	15	16	17	18
24	23	22	21	20	19
25	26	27	28	29	30
36	35	34	33	32	31

故处理输入的方法如下:

```
for(i=0;i<n;i++)
{
    if(第1、3、5行)
    { value=i*n+1;
      for(j=0;j<n;j++)
          a[i][j]=value++;
    }
    else //同样处理偶数行
}
```

(1) 数据输入问题

输入的二维数组元素有一定规律:

第1($i=0$)、3($i=2$)、5($i=4$)的元素递增,

起始值为 $value=(i*n+1)$;

该行后续元素值为: $value++$;

第2($i=1$)、4($i=3$)、6($i=5$)的元素递减,

起始值为 $value=((i+1)*n)$;

该行后续元素值为: $value--$;

(2) 数据输出格式控制(图案输出)

```
for(i=0;i<n;i++) //输出n行数据
{
    for(j=0;j<3*(n-i);j++) //j与i的关系式
        cout<<' '; //输出空格
    for(j=0;j<=i;j++) //j与i的关系式，输出下三角的元素
        cout<<setw(6)<<a[i][j]; //输出数据
    cout<<endl;
}
```

1	2	3	4	5	6
12	11	10	9	8	7
13	14	15	16	17	18
24	23	22	21	20	19
25	26	27	28	29	30
36	35	34	33	32	31

					1
				12	11
		13	14	15	
	24	23	22	21	
25	26	27	28	29	
36	35	34	33	32	31

二维数组典型问题

- 1、求一个整型数值二维表中的最大值和最小值及其相应的行号；交换最大、最小值。
- 2、主对角线($i=j$)、次对角线($i+j=N-1$)、上三角、下三角。

```
#include<iostream>
#include<iomanip>
#include<cstdlib>
#include<ctime>
using namespace std;
void main()
{
    const int N=5;
    int a[N][N];
    int i,j;
    srand(time(NULL)); //随机产生二维数组元素
    for(i=0;i<N;i++)
        for (j=0;j<N;j++)
            a[i][j]=10+rand()%20; //随机产生数组元素
    for(i=0;i<N;i++)
    {
        for (j=0;j<N;j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl; //输出一行数据后，输出回车！
    }
}
```

```
cout<<"输出左下三角"<<endl;
for(i=0;i<N;i++) //输出N行
{
    for (j=0;j<=i;j++)
        cout<<setw(4)<<a[i][j]; //输出数据
    cout<<endl;
}
cout<<"输出右上三角"<<endl;
for(i=0;i<N;i++)
{
    for(j=0;j<i;j++)
        cout<<setw(4)<<" "; //输出每行空格
    for (;j<N;j++)
        cout<<setw(4)<<a[i][j]; //输出数据
    cout<<endl;
}
```

输出左下三角

14	23	18	14	12
19	22	18	12	15
12	30	13	17	15
21	20	29	11	28
23	22	22	29	19

14				
19	22			
12	30	13		
21	20	29	11	
23	22	22	29	19

10	17	13	28	25
25	14	14	24	27
27	14	29	20	13
23	22	21	20	25
29	15	20	19	14

输出右上三角

10	17	13	28	25
	14	14	24	27
		29	20	13
			20	25
				14

二维数组典型问题

- 1、求一个整型数值二维表中的最大值和最小值及其相应的行号；交换最大、最小值。
- 2、主对角线($i==j$)、次对角线($i+j==N-1$)、上三角、下三角。

```
#include<iostream>
#include<iomanip>
#include<cstdlib>
#include<ctime>
using namespace std;
void main()
{
    const int N=5;
    int a[N][N];
    int i,j;
    srand(time(NULL)); //随机产生二维数组元素
    for(i=0;i<N;i++)
        for (j=0;j<N;j++)
            a[i][j]=10+rand()%20; //随机产生数组元素
    for(i=0;i<N;i++)
    {
        for (j=0;j<N;j++)
            cout<<setw(4)<<a[i][j];
        cout<<endl; //输出一行数据后，输出回车！
    }
}
```

```
cout<<"输出左上三角"<<endl;
for(i=0;i<N;i++) //输出N行
{
    for (j=0;j<=N-1-i;j++)
        cout<<setw(4)<<a[i][j]; //输出数据
    cout<<endl;
}
cout<<"输出右下三角"<<endl;
for(i=0;i<N;i++)
{
    for(j=0;j<N-1-i;j++)
        cout<<setw(4)<<" "; //输出每行空格
    for (;j<N;j++)
        cout<<setw(4)<<a[i][j]; //输出数据
    cout<<endl;
}
```

输出左上三角

16	29	20	15	17
21	16	17	24	21
19	17	30	30	14
18	19	24	25	12
30	11	10	22	29
16	29	20	15	17
21	16	17	24	
19	17	30		
18	19			
30				

输出右下三角

27	26	11	11	13
12	24	17	28	19
22	11	10	22	28
14	20	30	19	10
15	17	19	17	23
				13
			28	19
		10	22	28
	20	30	19	10
15	17	19	17	23

- 3、杨辉三角及各种输出图案(等腰、下三角)
- 4、矩阵转置、用一个二维数组实现方阵的转置
- 5、产生数据具有某种规律的方阵 (S方阵、回型方阵-实验10-3)
- 6、利用二维数组统计学生的各门成绩及学科成绩包括求总成绩、平均成绩等 (实验10-4)

```
for(i=0;i<n;i++) //在数组中写入所有为1的数
```

```
a[i][0]=1
```

```
a[i][i]=1
```

```
for(i=2;i<n;i++)
```

```
for(j=1;j<i;j++) //第3行第2列开始
```

```
a[i][j]=a[i-1][j-1]+a[i-1][j]
```

```
for(i=0;i<n;i++) //输出n行
```

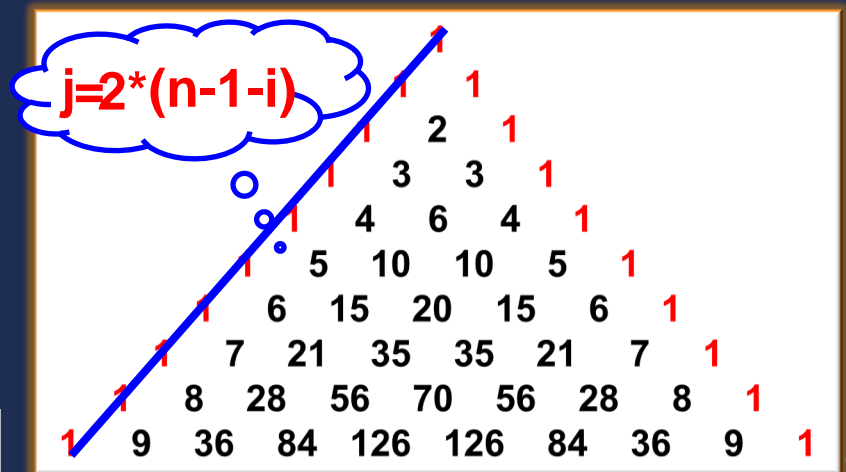
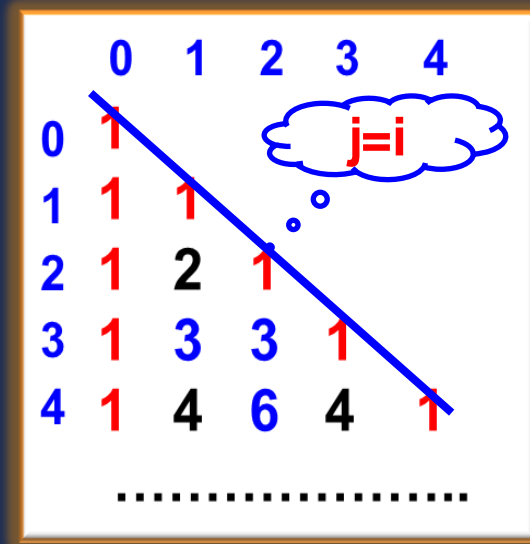
```
for(j=0;j<=2*(n-1-i); j++) //控制每行输出空
```

```
cout<<" ";
```

```
for(j=0; j<=i; j++) //控制每行输出元素
```

```
cout<<setw( 4 )<<a[i][j];
```

输出换行



4.6 字符数组 P₁₂₆

回顾1：如何定义一维数组？

类型说明符 数组名[常量表达式];

如果数组的“类型”为char，则该一维数组为字符数组，即每个数组元素均为字符。

```
const int N=10;    char s[N];
```

s	t	r	i	n	g	i	s	a	b
---	---	---	---	---	---	---	---	---	---

回顾2：如何访问一维数组的所有元素？

在数组中才有下标法访问数组元素，下标从0~N-1

```
const int N=10;  
char s[N];    int i;  
for(i=0;i<=N-1;i++)  
    cin>>s[i];
```

一维字符数组的输入和输出

- 逐个字符的输入、输出

```
#include <iostream>
using namespace std;
void main()
{
    char str[10];
    cout<<"输入十个字符:";
    for(int i=0; i<10; i++)
        cin>>str[i];
    for(int i=0; i<10; i++)
        cout<<str[i];
}
```

当输入为：string is abc

输出为：stringisab

字符数组用cin逐个输入元素时不接收空格，回车表示输入结束！

s	t	r	i	n	g	i	s	a	b
---	---	---	---	---	---	---	---	---	---

数组是一种复合类型，不能作为一个整体进行访问和处理的，只能按元素进行个别的访问和处理，故只能逐个输入字符存储在字符数组中，逐个输出数组元素(字符)！

通过上例我们发现：在字符数组中存储的所有字符元素，其实就是一串字符（字符串）的形式！

问题：能否用字符数组表示和处理字符串呢？

4.6 字符数组 P₁₂₆

回顾3:

C++中, 字符串常量 (用双引号括起来), 例如:
“china”。没有字符串变量!

C++ 中两种处理字符串的方法:

(1) **字符数组** 用字符数组来存放字符串。

- 用一维字符数组来存放一个字符串。
- 用二维字符数组来存放多个字符串。

注意: 1、使用字符数组处理字符串有结束符'\0'才表示一个“字符串”, 否则仅为存储在字符数组中的一个一个的字符元素!!!

2、必须注意结束符'\0'的处理!

(2) **字符串类-string**

1. 二维字符数组 P₁₃₀

可以用一个二维字符数组表示若干个字符串数组。

```
char cc[3][9]={"shanghai","aba","beijing"};
```

cc[0][]	s	h	a	n	g	h	a	i	\0
cc[1][]	a	b	a	\0					
cc[2][]	b	e	i	j	i	n	g	\0	

2. 一维字符数组 P₁₂₈

- **str1中存储若干个字符类**
`char str1[]={'p','r','o'}`
- **str2中存储一个字符串类**
`char str2[]={'p','r','o','\0'}`
- **str3中存储一个字符串数据**
`char str3[]="program";`

1、str1、str2和str3均在**定义时赋初值**，故可省略数组长度。

2、字符数组str3的实际长度为**8**个字节，因为数组**最后一个元素为字符串结束符'\0'**。

分析：

- str1中数据只能以元素为单位进行操作。
无结束标志'\0'，故不是字符串，只是字符数组，即每个元素为字符的数组，只能按元素逐个访问，不能整体操作。
- str2和str3中的数据，既可以元素为单位进行操作，也可以作为一个整体(字符串)为单位进行操作。

需注意问题

```
#include<iostream>
#include<string>
using namespace std;
void main()
{
    char s1[10];
    int i;
    s1="program";
    for(i=0;i<9;i++)
        cout<<s1[i];
}
```

s1为**字符数组**，不能整体操作，只能逐个元素访问，故这种整体赋值形式不正确！

修改方法：在定义的同时赋初值。

char s1[]="program";

error C2440: “=”: 无法从“const char [8]”转换为“char [10]”

示例

A、
`char s[3]="C++";`
`cout<<s<<endl;`

B、
`char s[]="C++";`
`cout<<s<<endl;`

C、
`char s[3]={'C','+','+','\0'};`
`cout<<s<<endl;`

B

D、
`char s[3]={'C','+','+'};`
`cout<<s<<endl;`

用字符数组处理字符串的输入和输出

- 逐个字符的输入、整体输出

```
#include <iostream>
using namespace std;
void main()
{
    char str[11];
    cout<<"输入十个字符:";
    for(int i=0; i<10; i++)
        cin>>str[i];
    str[10]='\0'; //写入结束符, 构成一个字符串, 可整体处理
    cout<<str;    //整体输出
}
```

字符数组用cin逐个输入元素时, 不接收空格, 以回车表示输入结束

当输入为: string is abc
输出为: stringisab

s	t	r	i	n	g	i	s	a	b	'\0'
---	---	---	---	---	---	---	---	---	---	------

用字符数组处理字符串的输入和输出

- 用字符数组整体操作字符串的输入、输出

```
#include <iostream>
using namespace std;
void main()
{
    char s1[50],s2[60];
    cout<<"输入两个字符串:"<<endl;
    cin>>s1;    //整体操作
    cin>>s2;
    cout<<" s1="<<s1<<endl;
    cout<<"s2="<<s2<<endl;
}
```

当输入为：string is abc
输出为：

s1=string
s2=is

- 在输入字符串时遇到空格或换行，认为一个字符串结束，接着的非空格字符作为一个新的字符串开始。
- 当把一个字符数组中的字符作为字符串输出时，遇到'\0'时认为字符串结束。

s	t	r	i	n	g	'\0'
---	---	---	---	---	---	------

i	s	'\0'
---	---	------

示例

P₁₇₅

在遇到‘#’之前，**一直从键盘接收输入字符**。故可用**循环**实行输入。

```
char ch;
```

```
do //至少输入一次，故用do-while结构
```

```
{
```

```
    cin>>ch;
```

```
    .....
```

```
}while(ch!='#')
```

如何判断

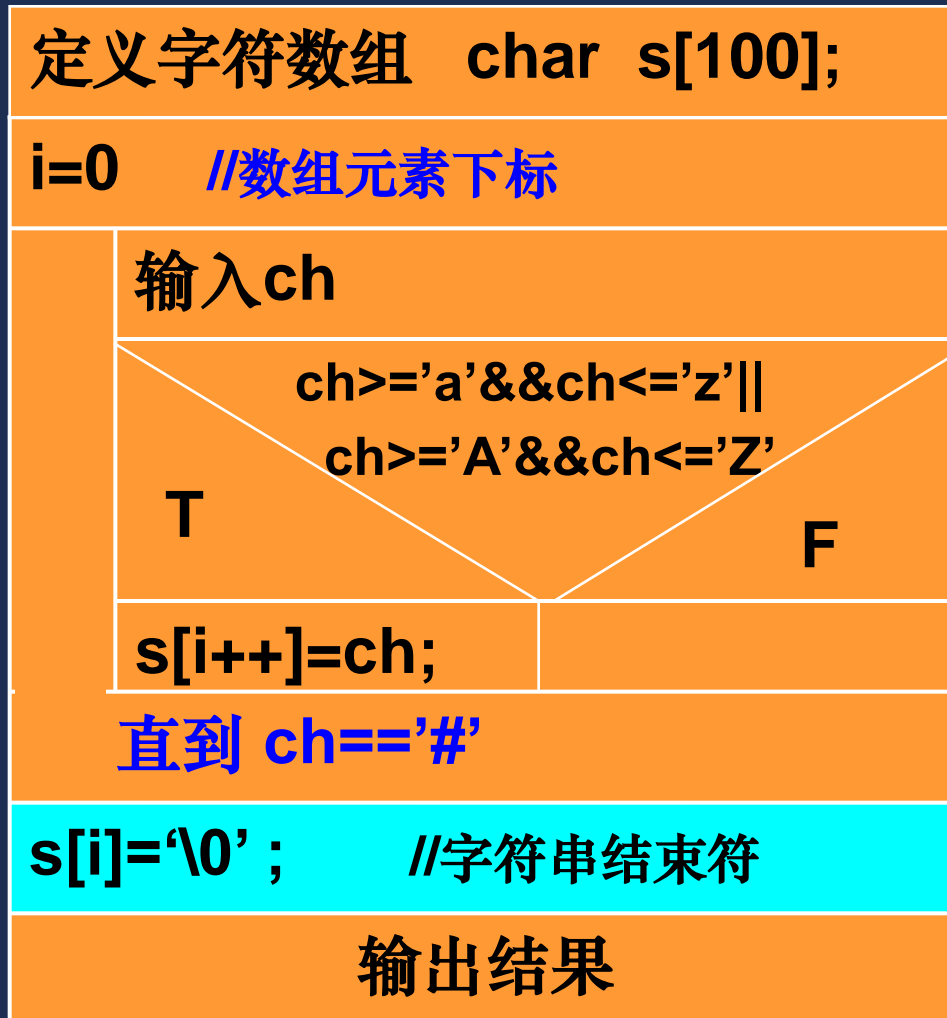
ch>

|| ch

输入一个字符串，以#号作为输入结束标志。
将字符串中的字母记录下来，然后依次显示
这些字母。

数据结构：对于一个字符串可以用一维字符
数组表示。注意处理字符串的结束符'\0'

算法的NS流程图



```

#include<iostream>
using namespace std;
void main( )
{
    const int N=100;
    char s[N],ch;
    int i=0;
    cout<<"please input string:";
    do
    {
        cin>>ch;
        if((ch>='a')&&(ch<='z')||(ch>='A')&&(ch<='Z'))
            s[i++]=ch; /*英文字符写入字符数组中*/
    } while ( ch!='#');
    s[i]='\0'; //写入结束符，构成字符串
    cout<<s; //整体输出字符串
}

```

定义字符数组 char s[100];

i=0 //数组元素下标

输入ch

ch>='a'&&ch<='z' ||
ch>='A'&&ch<='Z'

T

F

s[i++]=ch

直到 ch=='#'

s[i]='\0' //字符串结束符

输出结果

改为while结构

```
#include<iostream>
using namespace std;
void main( )
{
    const int N=100;
    char s[N],ch;
    int i=0;
    cout<<"please input string:";
    cin>>ch;    //输入字符后判断其是否为结束标志
    while (ch!='#')
    {
        if((ch>='a')&&(ch<='z')||(ch>='A')&&(ch<='Z'))
            s[i++]=ch;    /*英文字符写入字符数组中*/
        cin>>ch;
    }
    s[i]='\0';    //写入结束符构成字符串
    cout<<s;    //整体输出字符串
}
```

3. C风格字符串处理函数 P_{128~130}

在系统函数库中提供了一些字符串处理函数。

- puts/gets (输入输出)
- strcat (连接)
- strcpy (复制)
- strcmp (比较)
- strlen (求长度) 只统计字符数, 包括结束符'\0'
- strlwr (转换为小写)
- strupr (转换为大写)

上述字符串处理函数只能用于字符数组!!!

gets/puts函数

●gets函数

调用方法： `gets(字符数组)`

功能：从终端输入一个字符串到字符数组，并得到一个函数值，该函数值是字符数组在内存中的起始地址。

```
例如：char c[4];  
       gets(c);
```

说明：执行以上操作时，从键盘输入：xyz，则将字符串"xyz"赋给字符数组c，并得到字符数组元素c[0]在内存的起始地址。

gets/puts函数

● puts函数

调用方法： puts(字符数组)

功能： 将字符数组存储的字符串输出到终端。

例如： char c[]="China";

puts(c); // 屏幕显示： China

strcat / strcat_s 函数 P₁₂₉

调用方法：

strcat(字符数组1, 字符数组2或字符串常量)

功能：把字符数组2中的字符串或字符串常量连接到字符数组1中的字符串后面，连接的结果放在字符数组1中，函数调用后得到一个函数值——字符数组1的起始地址。

例如：`char c[20]={“abcd”};`
`strcat(c,“xyz”);`

说明：以上操作，将字符数组c表示的字符串“abcd”与字符串“xyz”连接成一个新的字符串“abcdxyz”赋给字符数组c，并得到字符数组元素c[0]在内存的起始地址。

strcpy / strcpy_s 函数 P₁₃₀

调用方法：

strcpy(字符数组1, 字符数组2或字符串常量)

功能：将字符数组2中的字符串或字符串常量拷贝到字符数组1中，并覆盖字符数组1中原有的内容。

```
例如：char c[]="abcdef";  
        strcpy(c,"xyz");  
        cout<<c;    // 输出为xyz
```

说明：执行以上操作，将字符串“xyz”赋给字符数组c。拷贝时连同字符串后面的'\0'一起拷贝到字符数组c中

strcmp函数 P₁₂₇ 例4.3

调用方法：

strcmp(字符数组1或字符串1, 字符数组2或字符串2)

例如：

```
char c1[20]={“abcdefg”};
```

```
char c2[20]={“abcdeffg”};
```

```
strcmp(c1,c2);
```

```
//执行以上操作，得到一个正整数。
```

功能：作用是将字符数组1或字符串1表示的字符串与字符数组2或字符串2表示的字符串进行比较。

字符串比较的规则是对两个字符串自左至右逐个字符相比较（按ASCII码值大小比较），直到出现不同的字符或遇到‘\0’为止。如全部字符相同，则认为相等；若出现不相同的字符，则以第一个不相同的字符的比较结果为准，比较结果由函数值带回。

- 如果字符串1==字符串2，函数值为0。
- 如果字符串1>字符串2，函数值为一正整数。
- 如果字符串1<字符串2，函数值为一负整数。

strlen函数

调用方法：

strlen(字符数组或字符串常量)

功能：测定字符串常量或字符串的长度，不包括结束标志'\0'。

```
例如：char c[]={'x','y','z','\0'};  
cout<<strlen(c); //输出为3
```

getline函数

P₁₂₈

getline是输入流对象cin的成员函数。

调用方法：

```
cin.getline(字符数组名St, 字符个数N, 结束符);
```

功能：

一次连续读入多个字符（可以包括空格），读入的字符串存放于字符数组St中，直到读满N-1个，或遇到指定的输入结束标志（默认为'\n'，换行符）。读取结束标志但不会在缓冲区存储，故不会影响下一次的输入操作。数组的最后一位存放字符串结束符'\0'。

get函数 P₁₂₈

get是输入流对象cin的成员函数。

调用方法：

```
cin.get(字符数组名St, 字符个数N, 结束符);
```

功能：

一次连续读入多个字符（可以包括空格），读入的字符串存放于字符数组St中，直到读满N-1个，或遇到指定的输入结束标志（默认为‘\n’，换行符）。输入结束标志会留在输入缓冲区中，将被下一次输入操作捕获，影响该输入处理。数组的最后一位存放字符串结束符‘\0’。

比较cin.getline和cin.get的效果

```
#include <iostream>
using namespace std;
void main ()
{   char city[20];
    char state[20];
    cin.getline(city,20,',');
    cin.getline(state,20);
    cout << "City:" <<city<<endl<< "State:"<<state<<endl;
}
```

```
cheng du shi,si chuan sheng
City:cheng du shi
State:si chuan sheng
```

//结束符为','

//结束符为换行符

```
#include <iostream>
using namespace std;
void main ()
{
    char city[20];
    char state[20];
    cin.get(city,20,',');
    cin.get(state,20);
    cout << "City:" <<city<<endl<< "State:"<<state<<endl;
}
```

```
cheng du shi, si chuan sheng
City:cheng du shi
State:., si chuan sheng
```

//结束符为','

//结束符为换行符

示例 P₁₇₇

- 从键盘输入一段文本，在该文本的最大值字符后，插入另外一段文本(子文本)
- **算法思想**：该问题需要解决的关键点有两个，一个找最大值；另一个就是在指定位置插入数据的问题。
- **注意**：用字符数组处理字符串，结束符'\0'需要处理！


```
const int N=30;
```

```
char s[N], b[ ]="xyz"; int i,j;
```

```
char max;
```

```
cin>>s; //gets(s); 或  
cin.getline(s,N); 或 cin.get(s,N);
```

```
max=s[0], j=0;
```

```
i=1
```

```
s[i]!='\0'
```

```
T
```

```
s[i]>max
```

```
F
```

```
max=s[i]
```

```
j=i;
```

```
i=i+1
```

```
i=strlen(s);
```

```
i>j
```

```
s[i+3]=s[i];
```

```
i=i-1
```

```
s[i+1]=b[0]; s[i+2]=b[1];s[i+3]=b[3];
```

```
cout<<s; //puts(s)
```

```
#include <iostream>  
using namespace std;  
void main ()
```

```
{
```

```
const int N=30;
```

```
char s[N],b[ ]="xyz",max;
```

```
int i,j;
```

```
cout<<"请输入字符串: "<<endl;
```

```
cin>>s; //gets(s); 或 cin.getline(s,N);
```

```
max=s[0]; cin.get(s,N);
```

```
j=0; //设置默认最大值及其位置
```

```
for(i=1;s[i]!='\0';i++)
```

```
if(s[i]>max)
```

```
{ max=s[i];
```

```
j=i; //找到最大值所在位置
```

```
}
```

```
for(i=strlen(s);i>j;i--)
```

```
s[i+3]=s[i]; //在最大值后空出3个存储位
```

```
s[i+1]=b[0]; s[i+2]=b[1]; s[i+3]=b[2];
```

```
//插入字符串
```

```
cout<<"插入字符串后的结果为:"<<s<<endl;
```

```
// puts(s);
```

```
}
```

4.7 标准的C++ string类 P₁₃₂

由于C风格字符串(以空字符结尾的字符数组)太过复杂难于掌握，不适合大程序的开发，所以C++标准库定义了string类，该类定义在头文件<string>。

```
#include<string>
using namespace std;
```

1. string类的三个构造函数

创建类的对象实例时，需要初始化对象的数据成员。为简化初始化对象的过程，C++使用一特殊函数，称为构造函数，程序每次创建对象实例时自动执行构造函数。

- **string str;**

//调用默认的构造函数，建立空串

- **string str("OK");**

//调用采用C字符串初始化的构造函数

- **string str(str1);**

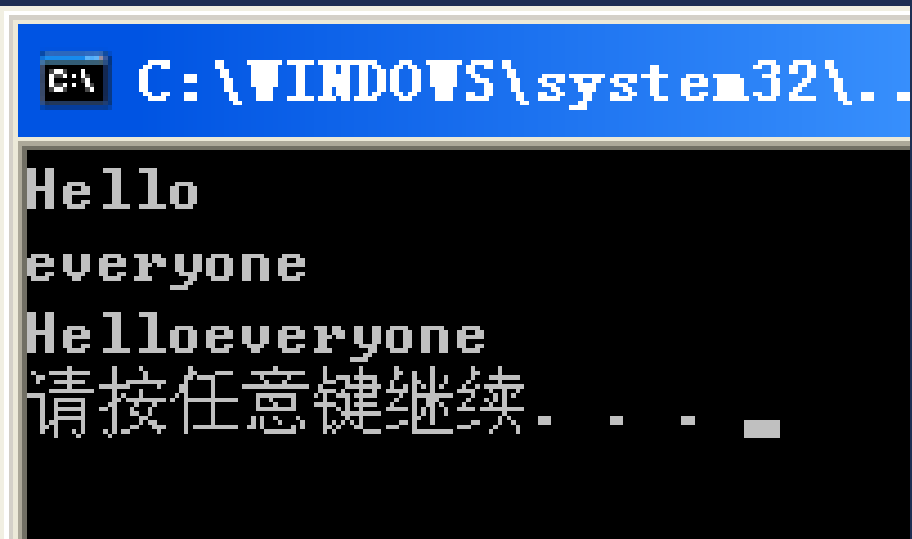
//调用复制构造函数，将str1的值复制给str

示例

- 字符串变量的定义

```
#include <iostream>
#include <string>
using namespace std;
void main()
```

```
{
    string s1("Hello");           //第二种构造方法
    string s2;                    //第一种构造方法
    s2="everyone";
    string s3(s1+s2);             //第三种构造方法
    cout<<s1<<endl<<s2<<endl<<s3<<endl;
}
```



```
C:\WINDOWS\system32\...
Hello
everyone
Helloeveryone
请按任意键继续. . .
```

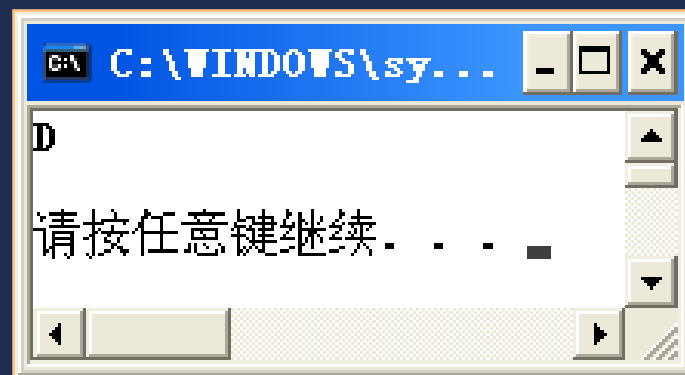
2. string类字符元素的访问

● str[i]

//返回str中下标为i的字符(**i的初值为0**), 不查是否出界

```
#include <iostream>
#include <string>
using namespace std;
void main()
{
    string s1("ABCDE");
    cout<<s1[3]<<endl<<s1[5]<<endl;
}
```

//第二种构造方法



3. string类的输入输出

- 输出使用插入运算符<<和cout。

```
string s1("hello");  
cout<<s1;
```

- 输入如用cin>>，读取的字符串可以包含空格，以回车表示输入结束。也可用getline函数输入。

```
string str;      //第一种构造方法  
cin>>str;       //换行符(回车)结束输入  
getline(cin,str); //串以'\n'(回车)结束  
getline(cin,str,ch); //串以ch字符结束
```

例如：getline(cin,str,'a');

输入：dsdfsfsf**a**dfsfsf

输出：dsdfsfsf

注意区别函数的使用

● 应用于字符数组输入、输出

```
char str[N];  
for(i=0;i<N-1;i++)  
    cin>>str[i];  
str[N-1]='\0'; //构成字符串  
cout<<str;
```

```
char str[N];  
cin>>str; ↔ cout<<str;  
// gets(str); ↔ puts(str);
```

```
cin.getline(字符数组名str, 字符个数N, [结束符]);  
cin.get(字符数组名str, 字符个数N, [结束符]); //均可接收空格  
cout<<str;
```

● 应用于字符串类输入、输出

```
string str1;  
cin>>str1;  
getline(cin,str1,[结束符]); //可接收空格  
cout>>str1;
```

4. string类的相关运算

- **str1=str2;** //把str2复制给str1,str1成为str2的副本
- **str1+=str2;** //str2的字符数据连接到str1的尾部
- **str1+str2;**
//返回一个字符串, 它将str2连接到str1的尾部
- **str1==str2; str1!=str2;**
//比较串是否相等, 返回布尔值
- **str1<str2; str1>str2; str1<=str2; str1>=str2;**
//基于字典序的比较, 返回布尔值

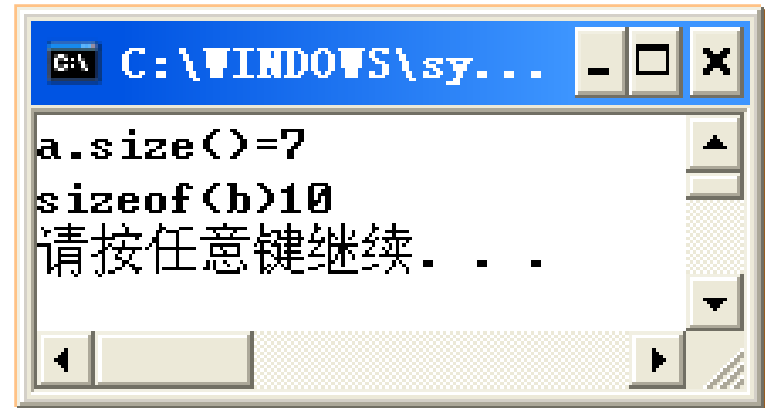
相关案例P₁₃₂ 例4.5

5.string类常用的字符串处理方法

- `str.length();` `str.size();` //返回串长度
- `str.substr(pos,length1);` //返回一个子串, 该子串从pos位置起, 长度为length1个字符
- `str.empty();` //查是否空串
- `str.insert(pos,str2);` //将str2插入pos位置处
- `str.remove(pos,length);` `str.erase(pos,length)`
//从pos处起, 删除长度为length的子串
- `str.find(str1);`
//返回str1首次(str1)在str中出现的位置,只能处理小写字母
- `str.find(str1,pos);`
//返回从pos处起str1首次(str1)在str中出现时的位置

示例

```
#include<iostream>
#include<string>
using namespace std;
void main( )
{
    string a="aabbccd";
    char b[10]="aabbccd";
    cout<<"a.size()="<<a.size()<<endl;
    cout<<"sizeof(b)"<<sizeof(b)<<endl;
}
```



有语句string S1="program"; 执行cout<<S1.size(); 则输出结果为 7。

字符串"program" 用字符数组存储, 则在内存中占用 8 个字节的空間。

示例

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ int n1,n2;
  string s1("Hello"),s2;
  cout<<"请输入字符串s2:";
  getline(cin,s2);
  n1=s1.size();  n2=s2.size();    //获取s1,s2的长度
  cout<<n1<<" " "<<n2<<endl;
  for(int i=0;i<n1;i++)
    cout<<s1[i];                //逐个访问字符元素并输出
  cout<<endl;
  cout<<s2<<endl;               //整体输出
  if(s1>=s2) cout<<s1<<endl;
  else cout<<s2<<endl;
  s2+=s1;
  cout<<"S2=S2+S1:"<<s2<<endl;
  cout<<"length of S2:"<<s2.length()<<endl;
}
```

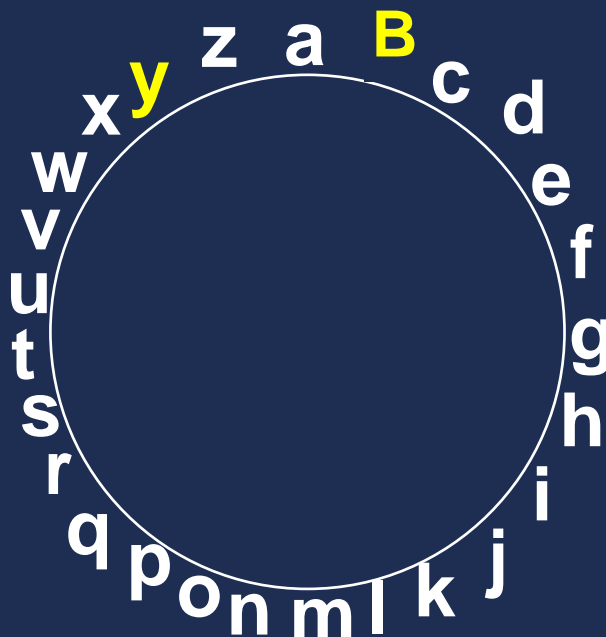
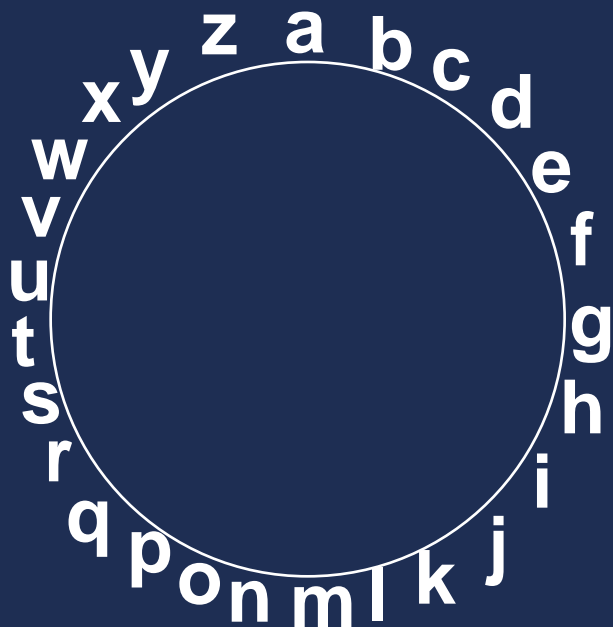
```
请输入字符串s2:Everyone
5      8
Hello
Everyone
Hello
S2=S2+S1:EveryoneHello
length of S2:13
请按任意键继续. . .
```

示例

理解 P_{172} 4.25

问题：对输入的一串信息（不超过40个字符）进行替换加密，加密规则如下：将字母表看成首尾衔接的闭合环，遇大写字母用该字母后面的第3个小写字母替换，遇小写字母用该字母后面的第3个大写字母替换.....

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

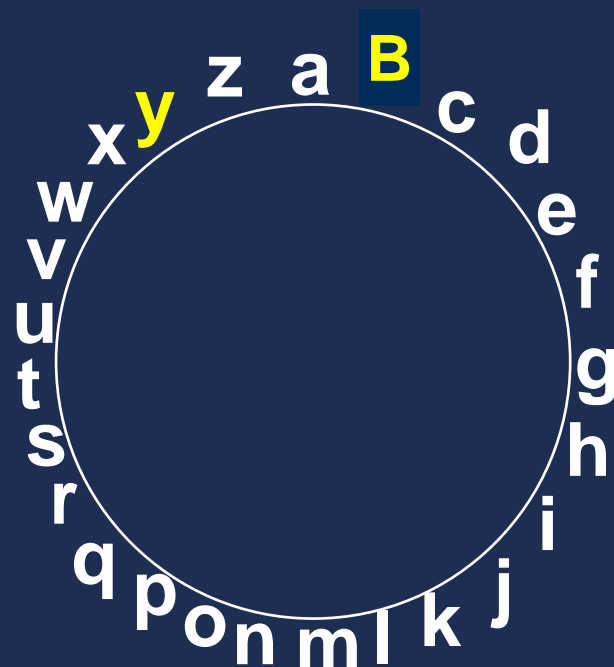
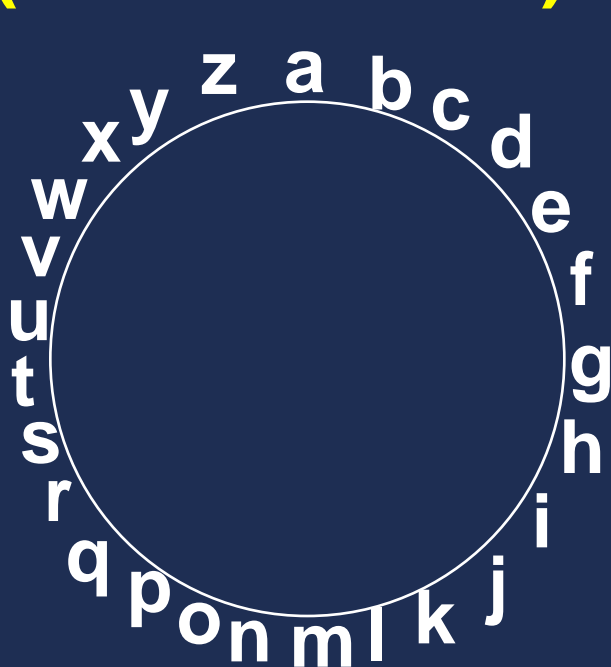


$a+3 \rightarrow d$
 $d-3 \rightarrow a$

$y ?$
 $y \rightarrow B$
 $(y+3-32)-26$

示例

分析：输入的原字符串可用string数据类型表示，在字符串的有效范围内，取出每一个字符，判断其大小写，如为大写，则加上32再加3；如为小写，则减去32再加3。处理后的值如果超出了字母范围，则应再“减”去26(26个英文字母)。



string s1; int i;

输入字符串

`l=s1.size()` //或者 `l=s1.length();`

for (i=0;i<l;i++)

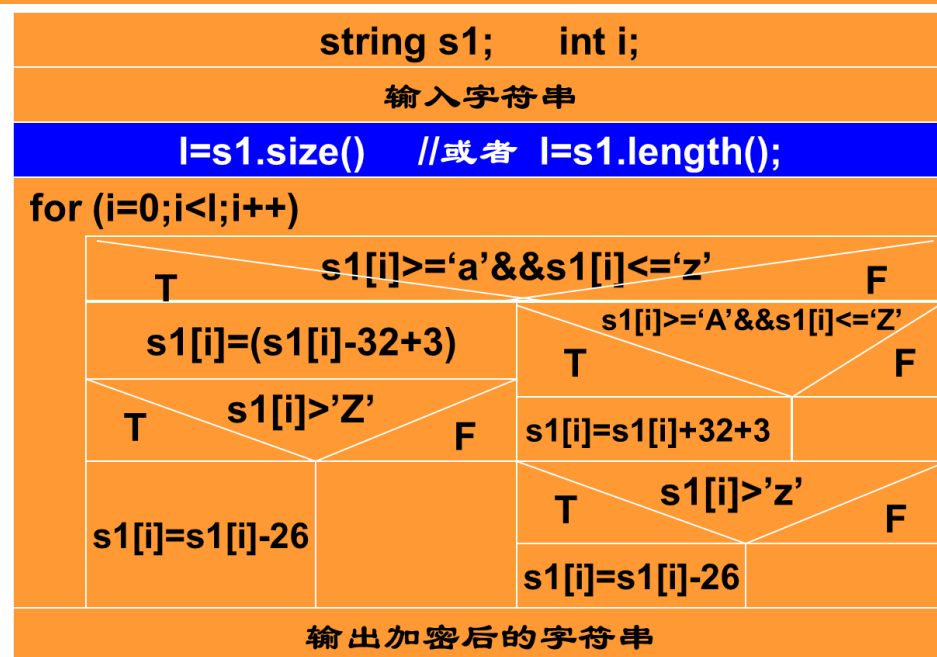
T		s1[i]>='a'&& s1[i]<='z'		F	
s1[i]=(s1[i]-32+3)			T		s1[i]>='A'&& s1[i]<='Z'
T			s1[i]>'Z'		F
F			s1[i]=s1[i]+32+3		
s1[i]=s1[i]-26			T		s1[i]>'z'
			F		
			s1[i]=s1[i]-26		

输出加密后的字符串

```
#include <iostream>
#include <string>
using namespace std;
void main( )
{
```

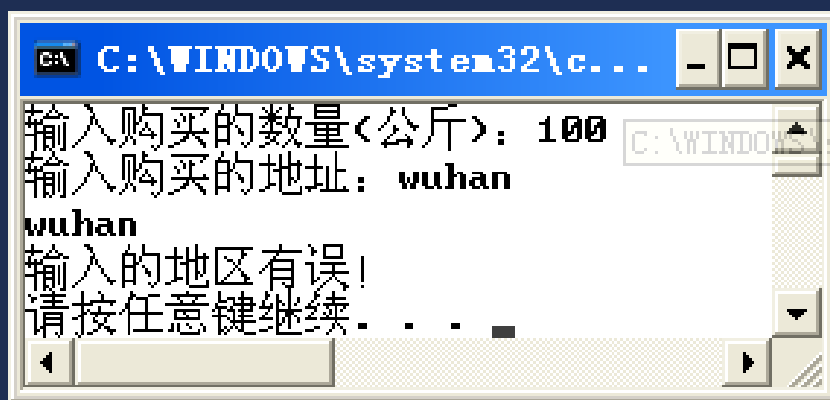
```
    string s1;    int i, l;
    cout<<"input a sentence:"<<endl;
    getline(cin,s1); //从键盘接收字符串
    l=s1.size( );    //获取字符串的长度
    for( i=0;i<l;i++)
```

```
    {
        if(s1[i]>= 'a' && s1[i]<='z') //若为小写字母
        { s1[i]=s1[i]-32+3; //转换成大写字母并移位
          if (s1[i]>'Z') s1[i]=s1[i]-26; //若超出字符范围
        }
        else
        if(s1[i] >='A' && s1[i]<='Z') //若为大写字母
        { s1[i]=s1[i]+32+3; //转换成小写字母并移位
          if (s1[i]>'z') s1[i]=s1[i]-26; //若超出字符范围
        }
    }
    cout<<s1<<endl;
}
```

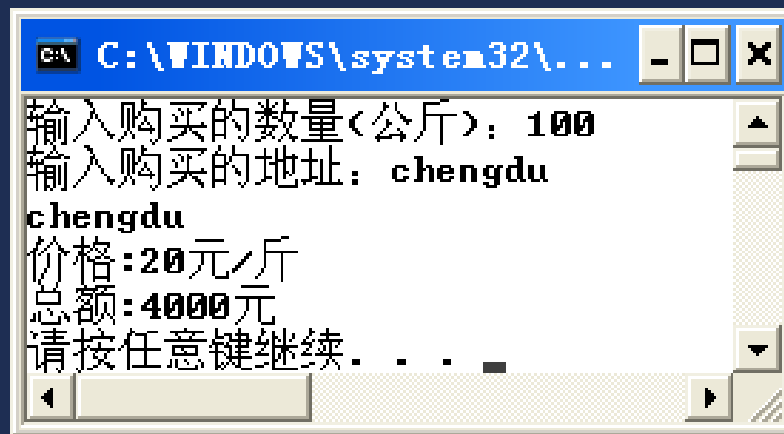


示例

- 编程实现：计算运费。不同地区单位运价不同，“chengdu”地区运费单价为20元/斤，“chongqing”地区运费单价为15元/斤。请根据输入的地区及货物重量（公斤），计算得到相应的运费。



```
C:\WINDOWS\system32\cmd.exe
输入购买的数量<公斤>: 100
输入购买的地址: wuhan
wuhan
输入的地区有误!
请按任意键继续. . .
```



```
C:\WINDOWS\system32\cmd.exe
输入购买的数量<公斤>: 100
输入购买的地址: chengdu
chengdu
价格:20元/斤
总额:4000元
请按任意键继续. . .
```

- 思路分析：输入地区（字符串）可用字符串类 string 处理。根据地区确定运费单价后，即可计算得出总运费。

算法流程图

string s;			
string s1="chengdu",s2="chongqing";			
int number,price,sum;			
输入number,s			
T	if(s!=s1&& s!=s2)		F
输出 “输入地区 有误!”	T	if(s==s1)	
	price=20		price=15
	sum=2*number*price;		
	cout<<sum;		

```
#include<iostream>
#include<string>
using namespace std;
void main()
{
```

```
    int number,price,sum;
    string s,s1="chengdu",s2="chongqing";
```

```
    cout<<"输入购买的数量(公斤): ";
    cin>>number;
    cin.ignore(1); //取消一个分隔符, 确保之后的getline正确执行
    cout<<"输入购买的地址: ";
    getline(cin,s);
    cout<<s<<endl;
```

- 用cin>>number; 是以回车作为输入的结束标志
- 用getline(cin,s); 接收数据时会受上次输入的影响。

P₁₃₄

string s; string s1="chengdu",s2="chongqing";			
int number,price,sum;			
输入number,s			
T	if(s!=s1&& s!=s2)		F
输出 “输入地区 有误!”	T	if(s==s1)	
		price=20	price=15
	sum=2*number*price;		
	cout<<sum;		

```
if(s!=s1&&s!=s2)
```

```
    cout<<"输入的地区有误！"<<endl;
```

```
else
```

```
{
```

```
    if(s==s1) price=20;
```

```
    else if(s==s2) price=15;
```

```
    sum=2*number*price;
```

```
    cout<<"价格:"<<price<<"元/斤"<<endl;
```

```
    cout<<"总额:"<<sum<<"元"<<endl;
```

```
}
```

```
}
```