



中國人民大學  
RENMIN UNIVERSITY OF CHINA

# 第5讲 数组

余力

buaayuli@ruc.edu.cn

# 输入格式

输入样例↵

3-4↵

3-8-9-10↵

2-5--3-5↵

7-0--1-4↵

```
int main()↵
{↵
    --- int n, m, sum = 0, x;↵
    --- scanf("%d%d", &n, &m);↵
    --- for (int i = 0; i < n; i++)↵
    ---     for (int j = 0; j < m; j++)↵
    ---     {↵
    ---         --- scanf("%d", &x);↵
    ---         --- ....↵
    ---     }↵
    --- return 0;↵
}↵
```

输入样式↵

3-5↵

2017101000-**10**-2.1-1.2-4.3-2.4-2.5-5.1-5.2-1.6-4.2-4.4↵

2017101001-**9**-2.1-2.2-2.5-3.2-1.1-1.3-4.3-4.4-5.2↵

2017101002-**10**-1.4-1.5-2.1-2.2-3.4-3.4-4.1-4.6-5.4-5.5↵

```
--- scanf("%d%d", &n, &k);↵
--- for (s = 0; i = 0; i < n; i++)↵
--- {--- scanf("%s", &id[i]);↵
---     --- scanf("%d", &num[i]);↵
---     --- for (j = 0; j < num[i]; j++)↵
---     {--- scanf("%d.%d", &date[s], &class1[s]);↵
---         --- s += 1;--- }↵
--- }↵
```

## #293. 外侧元素求和

```
#include <stdio.h>
int main() {
    int n,m,x,s,i,a=0;
    scanf("%d %d",&n,&m);
    for(i=1;i<=n*m;i++){
        if(i<=m){
            scanf("%d",&x);
            s=s+x;}
        else if(i>=m*n-m){
            scanf("%d",&x);
            s=s+x;}
        else if(i%m==1||i%m==0){
            scanf("%d",&x);
            s=s+x;}
        else scanf("%d",&x);
    }
    printf("%d",s);
    return 0;
}
```

原因：**s没有初始化**，但在本地机上，没有初始化就默认为0，友学网上不会

在本地运行可以，但在友学网上不通过

# #462 统计字符

```
#include <stdio.h>
```

```
int main() {
```

```
    char c;
```

```
    int i;
```

```
    int cnt[26] = {0};
```

```
    while ((c = getchar()) != '\n') {
```

```
        if (c >= 'A' && c <= 'Z') {
```

```
            cnt[c - 'A']++;
```

```
        }
```

```
    }
```

```
    for (i = 0; i < 26; i++) {
```

```
        char s;
```

```
        s = 'A' + i;
```

```
        printf("%c:%d\n", s, cnt[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

#379149

#462 统计字符

Time Limit Exceeded

0

3068 ms

384 KB

cpp / 288 B

余力(yuli)

2021/10/22 下午7:05

测试点 #0

2

得分: 0

用时: 1007 ms

内存: 384 KiB

测试点 #1

2

得分: 0

用时: 1009 ms

内存: 384 KiB

测试点 #2

2

得分: 0

用时: 1052 ms

内存: 264 KiB

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main() {
```

```
    char s[10000];
```

```
    int a[26] = {0};
```

```
    gets(s);
```

```
    for (int i = 0; i < strlen(s); i++)
```

```
        if (s[i] >= 65 && s[i] <= 90)
```

```
            a[s[i] - 'A']++;
```

```
    for (int i = 0; i < 26; i++)
```

```
        printf("%c:%d\n", 'A' + i, a[i]);
```

```
    return 0;
```

```
}
```

# 内容提要

---

- 5.1 数组的概念、定义和初始化
- 5.2 二维数组
- 5.3 数组的排序问题
- 5.4 筛法求素数



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

# 01. 数组概念定义初始化

---

# 数组

---

- 一组类型相同的顺序存储的数据（变量）
- 数组名、下标、元素
- 方便对一组数据进行命名和访问
  - 数组名+下标 唯一确定数组中的一个元素
  - 通过数组名+下标可以访问数组中的任意元素
- 应用：
  - 对一组数求最值、平均值
  - 对一组数据排序

# 一维数组的定义

## ■ 定义形式

- 类型说明符 数组名[常量]
- 例： `float sheep[10]; int a2001[1000];`

## ■ 数组的命名规则

- 数组名的第一个字符应为英文字母;
- 用方括号将常量表达式括起;
- 常量表达式定义了数组元素的个数;
- 数组的下标从0开始，如果定义了5个元素，是从第0个元素到第4个元素
- 常量表达式中不允许含有变量



# 一维数组的数组组织方式

---

`int boo[4]` (note: 2 bytes per int)

1980	46	4816	3
<code>boo[0]</code>	<code>boo[1]</code>	<code>boo[2]</code>	<code>boo[3]</code>

`char foo[4]` (note: 1-byte char)

h	e	l	p
<code>foo[0]</code>	<code>foo[1]</code>	<code>foo[2]</code>	<code>foo[3]</code>

# 数组初始化

## ■ 直接声明时初始化

➤ 例如: `int a[5] = { 3, 5, 4, 1, 2 };`

➤ 效果

a	3	5	4	1	2
下标	0	1	2	3	4

## ■ 思考：在声明之后，这样初始化可以吗？

`a[0] = 3; a[1] = 5; a[2] = 4; a[3] = 1; a[4] = 2;`

# 数组元素的访问

- 访问一维数组中元素的形式：

数组名[下标]

- 例如：

➤  $a[0] = a[1] + a[2];$

- 其中：

- 下标写在一个方括号中；
- 下标是整型表达式，如果为浮点型数据，C截去小数部分，自动取整。
- 引用时下标不能超界，否则编译程序检查不出错误，但执行时出现不可知结果。

# 一维数组的访问

---

// List A of n integer elements has already been set

int i;

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

printf( "%d ", A[i]);

printf( "\n" );

# 思考：如何输出每月天数

---

```
/* prints the days for each month */  
#include <stdio.h>  
#define MONTHS 12  
int main() {  
    int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};  
    int index;  
    for (index = 0; index < MONTHS; index++)  
        printf("Month %d has %2d days.\n", index + 1, days[index]);  
    return 0;  
}
```

**Month\_days.cpp**

# 思考：如果数组没初始化呢？

---

```
/* no_data.c -- uninitialized array */
#include <stdio.h>
#define SIZE 4
int main(void) {
    int no_data[SIZE];
    /* uninitialized array */
    int i;
    printf("%2s%14s\n", "i", "no_data[i]");
    for (i = 0; i < SIZE; i++)
        printf("%2d%14d\n", i, no_data[i]);
    return 0;
}
```

No\_data.cpp

# 字符数组与字符串

---

character array but not a string

y	o	u		c	a	n		s	e	e		i	t	.
---	---	---	--	---	---	---	--	---	---	---	--	---	---	---

character array and a string

y	o	u		c	a	n		s	e	e		i	t	.	\0
---	---	---	--	---	---	---	--	---	---	---	--	---	---	---	----

▲  
null character

# 程序举例：哪只羊最重？

---

- 中秋佳节，有贵客来到草原，主人要从羊群中选一只肥羊宴请宾客，当然要选最重者。
  - 要记录每只羊的重量，如果有成千上万只羊，不可能用一般变量来记录。可以用带有下标的变量，即数组
  - 将羊的重量读入存放到数组中
  - 声明一个变量保存最大的重量，不断更新之



# 程序框图

bigsheep = 0.0f; 将记录最重的羊的重量置 0  
bigsheepNo = 0; 记录最重的羊的编号

for ( i=0; i<10; i=i+1 )

提示输入第 i 只羊的重量;  
键入第 i 只羊的重量 sheep[i];

bigsheep < sheep[i]

是

否

bigsheep = sheep[i];  
bigsheepNo = i;  
存重者, 记录第 i 只。

输出 bigsheep ( 最重的羊的重量 )

输出 bigsheepNo ( 最重的羊的编号 )

```

#include <stdio.h> // 预编译命令
int main()          // 主函数
{
    float sheep[10] = {0}; // 用于存10只羊每一只的重量
    float bigsheep=0;       // 浮点类型变量, 存放最肥羊的重量
    int i=0, bigsheepNo=0;  // 整型变量, i 用于计数循环,
                           // bigsheepNo用于记录最肥羊的号

    for ( i=0; i<10; i=i+1 )
    {
        printf("请输入羊的重量sheep[%d]=", i);
        scanf("%f", &sheep[i]); // 输入第i只羊的重量
        if ( bigsheep < sheep[i] ) // 如果第i只羊比当前最肥羊大
        {
            bigsheep = sheep[i]; // 让第i只羊为当前最肥羊
            bigsheepNo = i;       // 纪录第i只羊的编号
        }
    }

    // 循环结束

    printf("最肥羊的重量为%f\n", bigsheep);
    printf("最肥羊的编号为%d\n", bigsheepNo);
    return 0;
}

```

Big\_sheep.cpp

## 思考

- 1#include <stdio.h>  
int main()  
{  
    int a[4];      // 声明项  
    int i=0;  
    for(i=0; i<4; i++)  
        printf("%d\n", a[i]);  
    return 0;  
}
- 2.其他不变, 改变声明项为  
    int a[4] = { 0, 1, 2, 3 };

Array\_initial.cpp

- 3.其他不变, 改变声明项为  
int a[4] = { 3, 8 };
- 4.其他不变, 改变声明项为  
int a[4] = { 2, 4, 6, 8, 10 };
- 5.其他不变, 改变声明项为  
int a[4] = { 2, 4, 6, d };
- 6.其他不变, 改变声明项为  
int d;  
int a[4] = { 2, 4, 6, d };
- 7.其他不变, 改变声明项为  
int n=4;  
int a[n] = { 0, 1, 2, 3 };

# 练习

---

- 1. 给定一组整数，找到其中最小的整数，并按照输入的逆序输出这组整数
- 2. 用数组的方式来处理Fibonacci数列问题



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

## 02. 二维数组

---

## 二维应用场景

- 有 $n$ 个学生，每个学生学 $m$ 门课，已知所有学生的各门课的成绩，分别求每门课的平均成绩和每个学生的平均成绩。设各学生成绩如下：

课程 姓名	课程		
	课程 1	课程 2	课程 3
学生 1	89	78	56
学生 2	88	99	100
学生 3	72	80	61
学生 4	60	70	75

# #195 平均成绩排序

有  $n$  位学生，每位学生修读的科目数不尽相同，已知所有学生的各科成绩，要求按学生平均成绩由高到低输出学生的学号、平均成绩；当平均成绩同时，按学号从低到高排序。对平均成绩，只取小数点后前 2 位，从第 3 位开始舍弃（无需舍入）。↵

输入格式↵

□□输入为  $n+1$  行，第一行为  $n$  表示学生人数。↵

□□从第二行开始的  $n$  行，每行为一名学生的成绩信息，包括：学号、科目数，各科成绩。其中  $n$ 、学号、成绩均为整数，它们的值域为： $0 \leq n \leq 10000$ ， $1 \leq \text{学号} \leq 1000000$ ， $0 \leq \text{成绩} \leq 100$ 。学生的科目数都不超过 100 门。↵

输出格式↵

□□最多  $n$  行，每行两个数，学号在前，后为平均成绩，空格分隔。若  $n$  为 0，输出 NO；若某学生所修科目不到 2 门，则不纳入排序，若无人修满 2 门，也输出 NO。↵

输入样例↵

```
5↵
1001 2 89 78↵
2003 4 88 99 100 88↵
4004 3 72 80 66↵
1004 3 70 66 82↵
3001 1 100↵
```

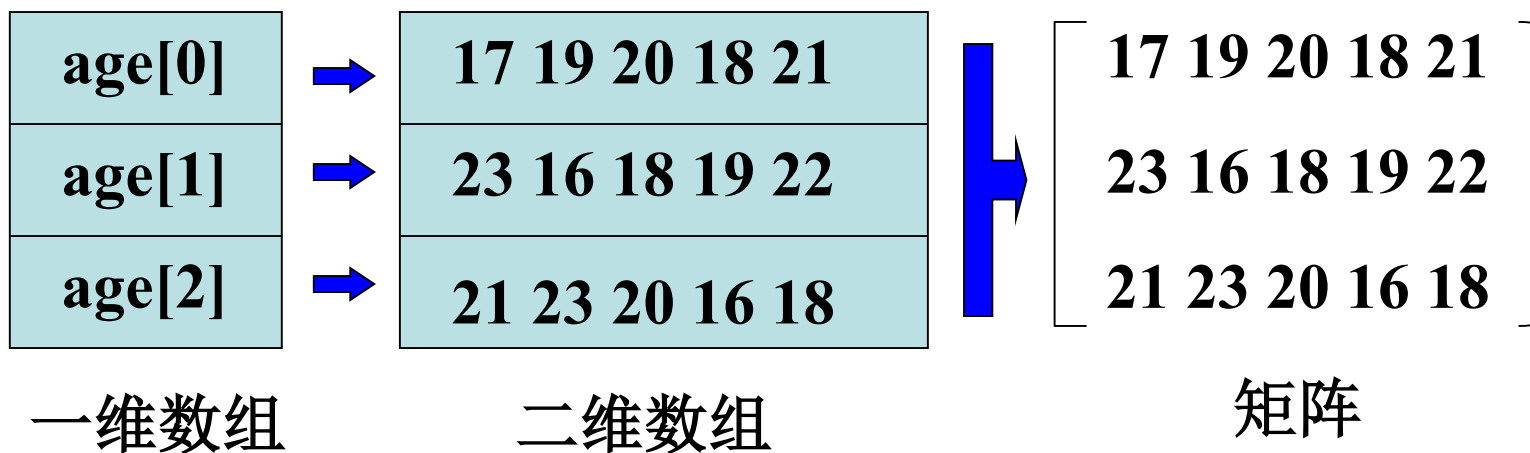
输出样例↵

```
2003 93.75↵
1001 83.50↵
┌───────────┐
1004 72.66↵
4004 72.66↵
```



# 二维数组的概念及其定义

- 当一维数组的每个元素是一个一维数组时，就构成了二维数组。
- 二维数组与数学中的矩阵概念相对应。



# 二维数组的定义

## ■ 定义方式

- 类型标识符 数组名[ 常量表达式1 ] [ 常量表达式2]
- 其中：
  - 类型标识符：数组中每个元素的数据类型。可以是C语言中所有的数据类型。
  - 数组名：合法的标识符，数组名就是变量名。
  - 常量表达式1：又称行下标，指出二维数组中一维数组元素的个数。
  - 常量表达式2：又称行列标，表明每个一维数组中的元素个数。

# 多维数组的定义

---

- 二维数组的每一个元素又是相同的类型的一维数，就构成了三维数组，……依此类推，就可构成四维或更高维数组
- 定义一个n维数组：

类型标识符 数组名[常量表达式1] [常量表达式2] .....[常量表达式n]

# 三维数组的排列顺序

- 说明一个三维数组：

```
int a[2][3][2];
```

- 12个元素在内存中排列顺序如右图：

a[0][0][0]
a[0][0][1]
a[0][1][0]
a[0][1][1]
a[0][2][0]
a[0][2][1]
a[1][0][0]
a[1][0][1]
a[1][1][0]
a[1][1][1]
a[1][2][0]
a[1][2][1]

## 2.2 访问二维数组和多维数组

---

- 访问二维数组中元素的形式：

**数组名[下标][下标]**

- 其中：

- 每一个下标写在一个方括号中；
- 下标是整型表达式，如果为浮点型数据，C截去小数部分，自动取整。
- 引用时下标不能超界，否则编译程序检查不出错误，但执行时出现不可知结果。

# 二维数组和 multidimensional 数组的初始化

- 二维数组和 multidimensional 数组都可以初始化，与一维数组初始化的差别是由于维数增多，初始化时特别注意元素的排列顺序。

➤ 例:

二维数组的初始化

```
int a[2][3]={{1,2},{4,5,6}};
```

或写成

```
int a[2][3]={1,2,4,5,6};
```

# 矩阵转置

```
#include <stdio.h>
int main()
{ int a[2][3]={{1,2,3},{4,5,6}};
  int b[3][2],i,j;
  printf("array a:\n");
  for (i=0;i<=1;i++)
  { for (j=0;j<=2;j++)
    { printf("%5d",a[i][j]);
      b[j][i]=a[i][j];    }
    printf("\n");
  }
```

矩阵转置.cpp

```
printf("array b:\n");
for (i=0;i<=2;i++)
{ for(j=0;j<=1;j++)
  {printf("%5d",b[i][j]);
   printf("\n");}
}
return 0;
```

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Rightarrow b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

## 03. 数组排序问题

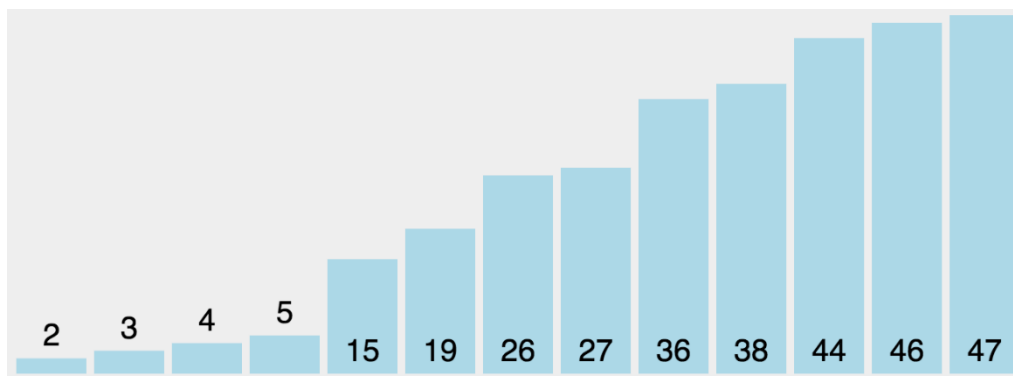
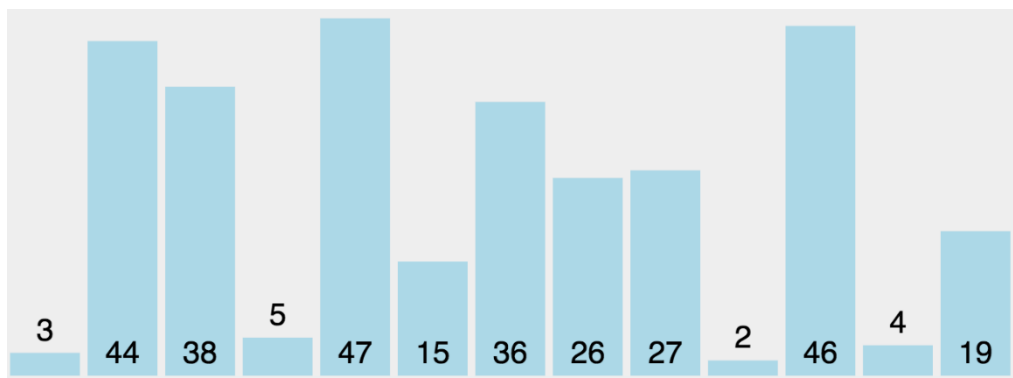
---



# 排序问题 (1)

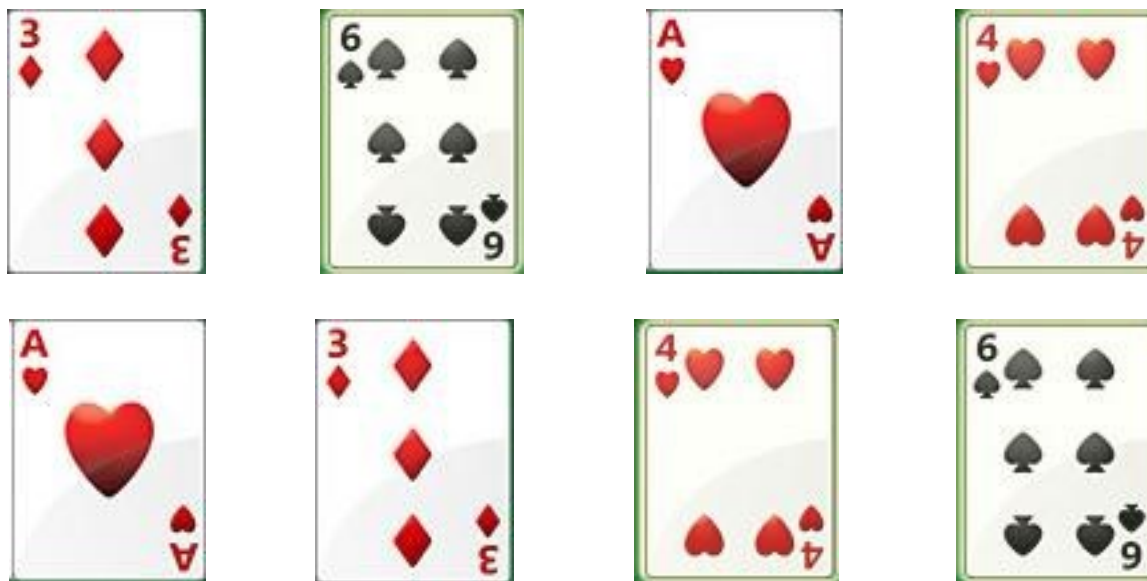
## ■ 问题定义

- 将数组中的元素按照一定顺序重新排列



# 排序问题 (2)

- 排序问题例子



- 还有哪些排序的例子？

# 排序算法 – 冒泡排序 (1)

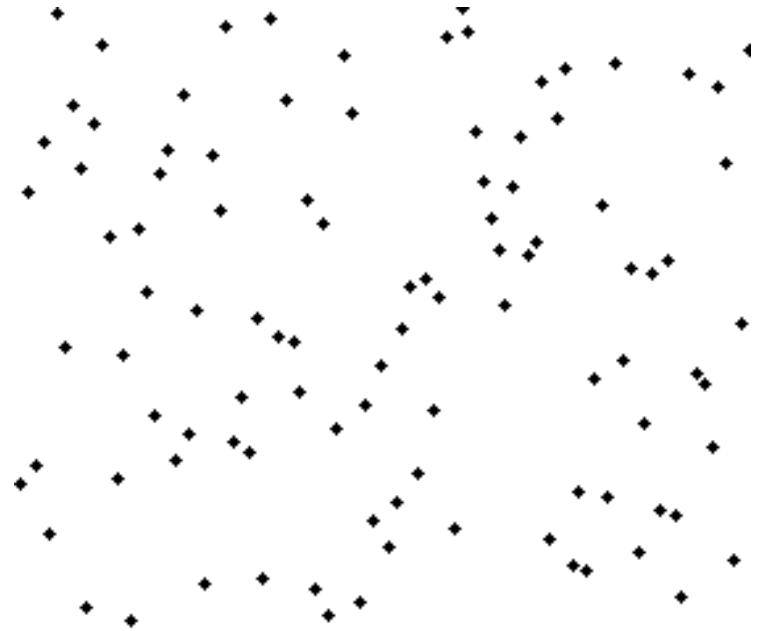
---

## ■ 基本思路

- 从头到尾依次访问数组
- 如果发现顺序错误就交换过来
- 直到没有再需要交换的元素

算法运行详解

<https://visualgo.net/sorting>



# 排序算法 – 冒泡排序 (2)

---

- 如何将上述思想用程序实现?
- 定义三个变量
  - $n$  – 数组中待排序元素的个数
  - $i$  – 当前是第几趟扫描  $i = 0, 1, 2, \dots, n - 2$
  - $j$  -- 第 $i$ 遍扫描待比较元素的下标  $j = 1, 2, \dots, n - i$

## 排序算法 – 冒泡排序 (3)

---

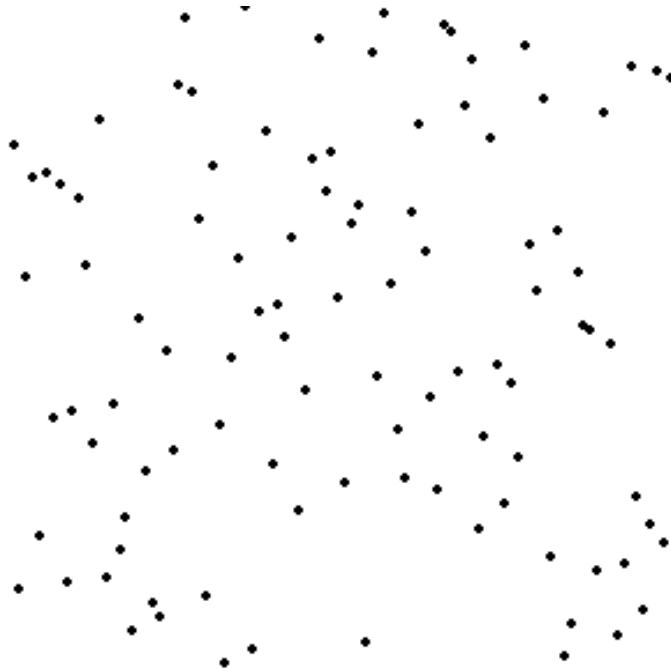
```
for (int i = 0; i < count - 1; i ++) {  
    int swap = 0;  
    for (int j = 0; j < count - 1 - i; j ++) {  
        if (arr[j] > arr[j+1]) {  
            int tmp = arr[j];  
            arr[j] = arr[j+1];  
            arr[j+1] = tmp;  
            swap = 1;  
        }  
    }  
    if (swap == 0) break;  
}
```

Bubble\_sort.cpp

# 排序算法 – 选择排序 (1)

## ■ 基本思路

- 把数组分为已排序和未排序两部分
- 每次遍历未排序部分，选择出其中最小元素，放到已排序部分
- 直到所有元素都位于已排序部分



算法运行详解

<https://visualgo.net/sorting>

# 排序算法 – 选择排序 (2)

---

- 如何将上述思想用程序实现?
- 定义三个变量
  - $n$  – 数组中待排序元素的个数
  - $i$  – 已/未排序分界  $i = 0, 1, 2, \dots, n-2$
  - $j$  – 当前访问未排序元素  $j = i+1, i+2, \dots, n-1$
  - $\text{min\_index}$  – 未排序中最小元素的下标

## 排序算法 – 选择排序 (3)

---

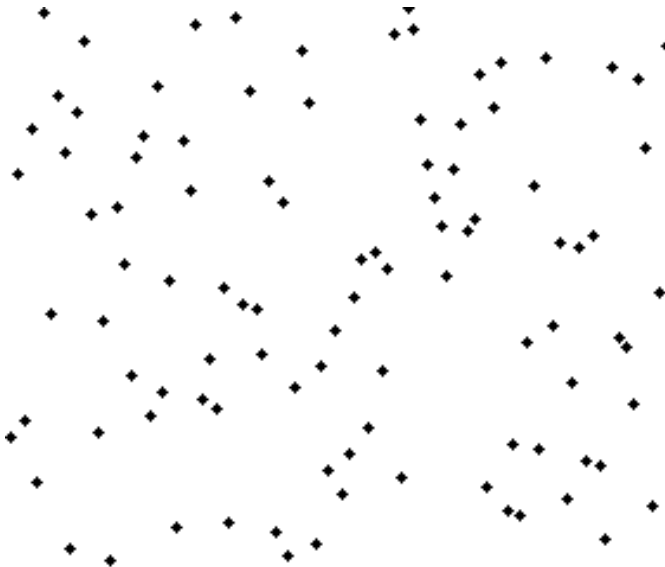
```
for (int i = 0; i < count - 1; i ++) {  
    int min_index = i;  
    for (int j = i + 1; j < count; j ++) {  
        if (arr[min_index] > arr[j]) {  
            min_index = j;  
        }  
    }  
    if (min_index != i) {  
        int tmp = arr[min_index];  
        arr[min_index] = arr[i];  
        arr[i] = tmp;  
    }  
}
```



# 排序算法 – 插入排序 (1)

## ■ 基本思路

- 把数组分为已排序和未排序两部分
- 依次访问未排序元素，将它插入到已排序部分的相应位置
- 为了支持插入，需要将元素向后移位
- 直到所有元素都位于已排序部分



算法运行详解

<https://visualgo.net/sorting>

# 排序算法 – 插入排序 (2)

---

- 如何将上述思想用程序实现?
- 定义三个变量
  - $n$  – 数组中待排序元素的个数
  - $i$  – 已/未排序分界  $i = 0, 1, 2, \dots, n-1$
  - $j$  – 当前访问以排序元素  $j = i-1, i-2, \dots, 0$
  - $\text{arr}[j+1] = \text{arr}[j]$  – 数组元素移位操作

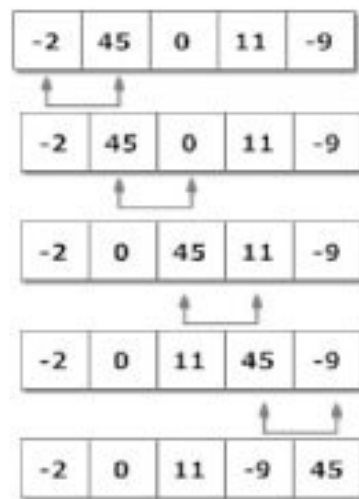
## 排序算法 – 插入排序 (3)

---

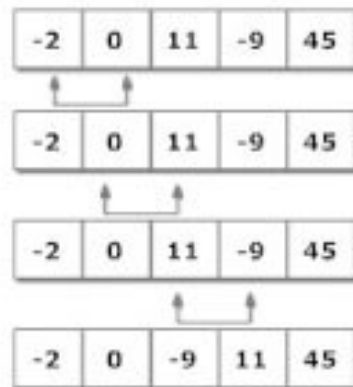
```
for (int i = 0; i < count; i++) {  
    int tmp = arr[i];  
    int j = i - 1;  
    for (; j >= 0; j--) {  
        if (arr[j] <= tmp) break;  
        arr[j+1] = arr[j];  
    }  
    arr[j+1] = tmp;  
}
```

Insert\_sort.cpp

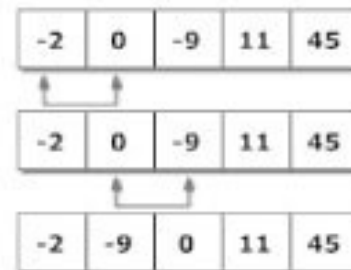
# 排序算法 – 冒泡排序



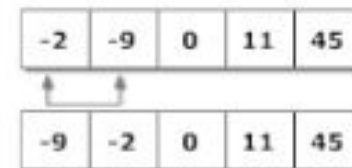
Step 1



Step 2

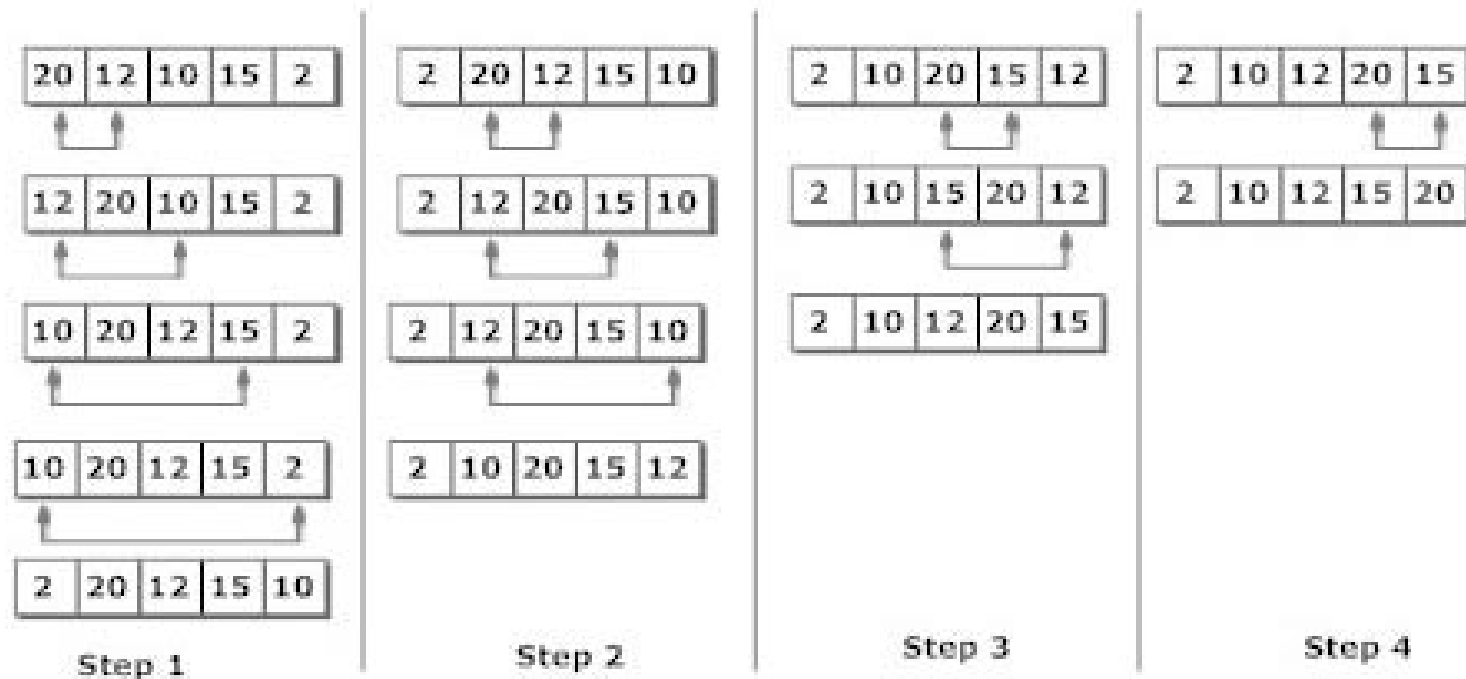


Step 3

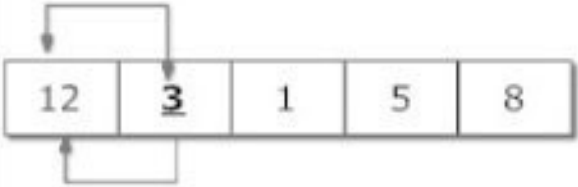
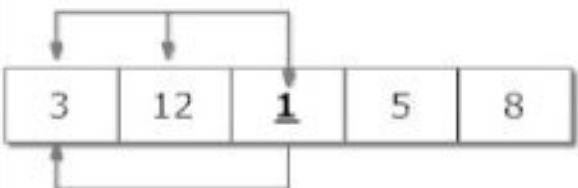
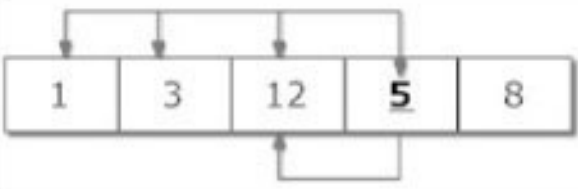
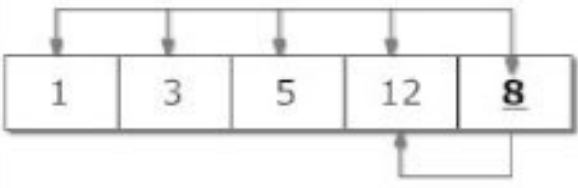



Step 4

# 排序算法 – 选择排序



# 排序算法 – 插入排序

Step 1		Checking second element of array with element before it and inserting it in proper position. In this case, 3 is inserted in position of 12.
Step 2		Checking third element of array with elements before it and inserting it in proper position. In this case, 1 is inserted in position of 3.
Step 3		Checking fourth element of array with elements before it and inserting it in proper position. In this case, 5 is inserted in position of 12.
Step 4		Checking fifth element of array with elements before it and inserting it in proper position. In this case, 8 is inserted in position of 12.
		Sorted Array in Ascending Order

# 排序算法 – 代码比较

```
for (int i = 0; i < count - 1; i++) {  
    int swap = 0;  
    for (int j = 0; j < count - 1 - i; j++) {  
        if (arr[j] > arr[j+1]) {  
            int tmp = arr[j];  
            arr[j] = arr[j+1];  
            arr[j+1] = tmp;  
            swap = 1;  
        }  
    }  
    if (swap == 0) break;  
}
```

```
for (int i = 0; i < count; i++) {  
    int tmp = arr[i];  
    int j = i - 1;  
    for (; j >= 0; j--) {  
        if (arr[j] <= tmp) break;  
        arr[j+1] = arr[j];  
    }  
    arr[j+1] = tmp;  
}
```

```
for (int i = 0; i < count - 1; i++) {  
    int min_index = i;  
    for (int j = i + 1; j < count; j++) {  
        if (arr[min_index] > arr[j]) {  
            min_index = j;  
        }  
    }  
    if (min_index != i) {  
        int tmp = arr[min_index];  
        arr[min_index] = arr[i];  
        arr[i] = tmp;  
    }  
}
```

# 练习：选择排序跟踪

3	1	5	2
---	---	---	---

*i*



*j*

因为 $a[1] < a[0]$ ,  
更新min\_index为1

```
24 for (int i = 0; i < count - 1; i ++) {  
25     int min_index = i;  
26     for (int j = i + 1; j < count; j ++) {  
27         if (arr[min_index] > arr[j]) {  
28             min_index = j;  
29         }  
30     }  
31     if (min_index != i) {  
32         int tmp = arr[min_index];  
33         arr[min_index] = arr[i];  
34         arr[i] = tmp;  
35     }  
36 }
```



# 练习：选择排序跟踪

3	1	5	2
---	---	---	---

*i*



*j*

因为 $a[2] > a[1]$   
不必更新min\_index

```
24     for (int i = 0; i < count - 1; i++) {
25         int min_index = i;
26         ➡ for (int j = i + 1; j < count; j++) {
27             if (arr[min_index] > arr[j]) {
28                 min_index = j;
29             }
30         }
31         if (min_index != i) {
32             int tmp = arr[min_index];
33             arr[min_index] = arr[i];
34             arr[i] = tmp;
35         }
36     }
```

# 练习：选择排序跟踪

3	1	5	2
---	---	---	---

*i*

因为 $a[3] > a[1]$   
不必更新min\_index

*j*

```
24     for (int i = 0; i < count - 1; i++) {  
25         int min_index = i;  
26         ➡ for (int j = i + 1; j < count; j++) {  
27             if (arr[min_index] > arr[j]) {  
28                 min_index = j;  
29             }  
30         }  
31         if (min_index != i) {  
32             int tmp = arr[min_index];  
33             arr[min_index] = arr[i];  
34             arr[i] = tmp;  
35         }  
36     }
```

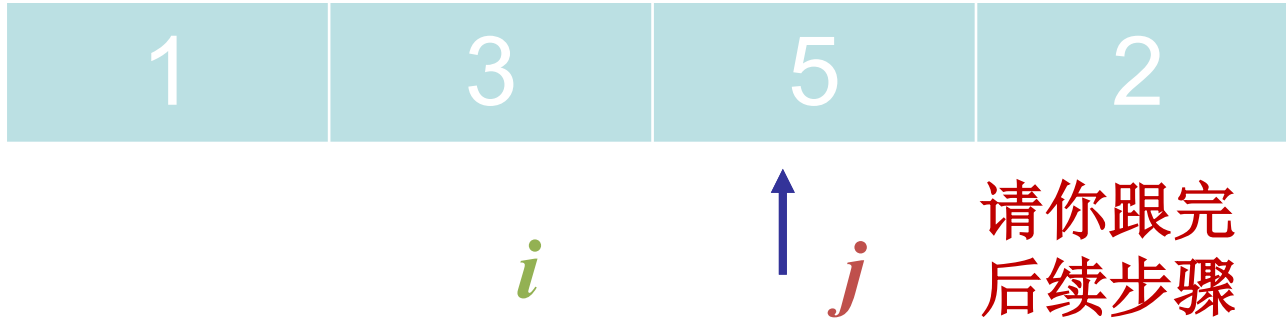
# 练习：选择排序跟踪



元素交换

```
24     for (int i = 0; i < count - 1; i++) {  
25         int min_index = i;  
26         for (int j = i + 1; j < count; j++) {  
27             if (arr[min_index] > arr[j]) {  
28                 min_index = j;  
29             }  
30         }  
31         if (min_index != i) {  
32             int tmp = arr[min_index];  
33             arr[min_index] = arr[i];  
34             arr[i] = tmp;  
35         }  
36     }
```

# 练习：选择排序跟踪



```
24     for (int i = 0; i < count - 1; i++) {  
25         int min_index = i;  
26         ➔ for (int j = i + 1; j < count; j++) {  
27             if (arr[min_index] > arr[j]) {  
28                 min_index = j;  
29             }  
30         }  
31         if (min_index != i) {  
32             int tmp = arr[min_index];  
33             arr[min_index] = arr[i];  
34             arr[i] = tmp;  
35         }  
36     }
```

# 练习：插入排序跟踪

3	1	5	2
---	---	---	---



*j*

*i*

此时无需操作

注：可以从*i*=1开始循环

```
28     for (int i = 0; i < count; i ++){
29         int tmp = arr[i];
30         int j = i - 1;
31         ➡ for (; j >= 0; j --) {
32             if (arr[j] <= tmp) break;
33             arr[j+1] = arr[j];
34         }
35         arr[j+1] = tmp;
36     }
```

# 练习：插入排序跟踪

tmp

1

3

1

5

2



*j*

*i*

```
28     for (int i = 0; i < count; i++) {
29         int tmp = arr[i];
30         int j = i - 1;
31         → for (; j >= 0; j--) {
32             if (arr[j] <= tmp) break;
33             arr[j+1] = arr[j];
34         }
35         arr[j+1] = tmp;
36     }
```

# 练习：插入排序跟踪

tmp

1

3

3

5

2



*j*

*i*

```
28     for (int i = 0; i < count; i ++){
29         int tmp = arr[i];
30         int j = i - 1;
31         for (; j >= 0; j --){
32             if (arr[j] <= tmp) break;
33             arr[j+1] = arr[j];
34         }
35         arr[j+1] = tmp;
36     }
```

# 练习：插入排序跟踪

tmp

1

1

3

5

2



*j*

*i*

```
28     for (int i = 0; i < count; i++) {  
29         int tmp = arr[i];  
30         int j = i - 1;  
31         for (; j >= 0; j--) {  
32             if (arr[j] <= tmp) break;  
33             arr[j+1] = arr[j];  
34         }  
35         arr[j+1] = tmp;  
36     }
```



# 练习：插入排序跟踪

tmp

5

1

3

5

2

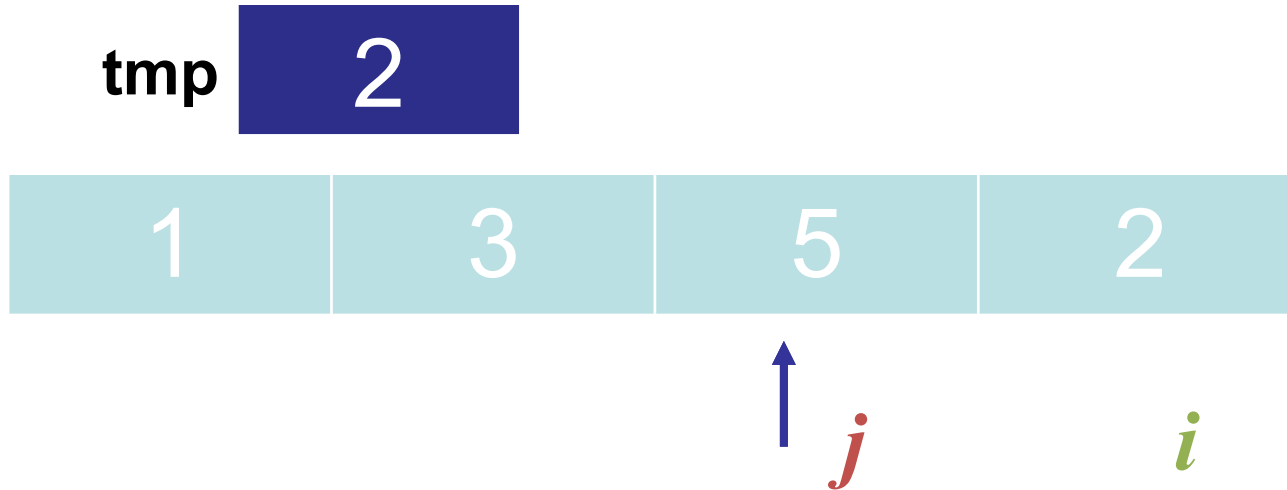


*j*

*i*

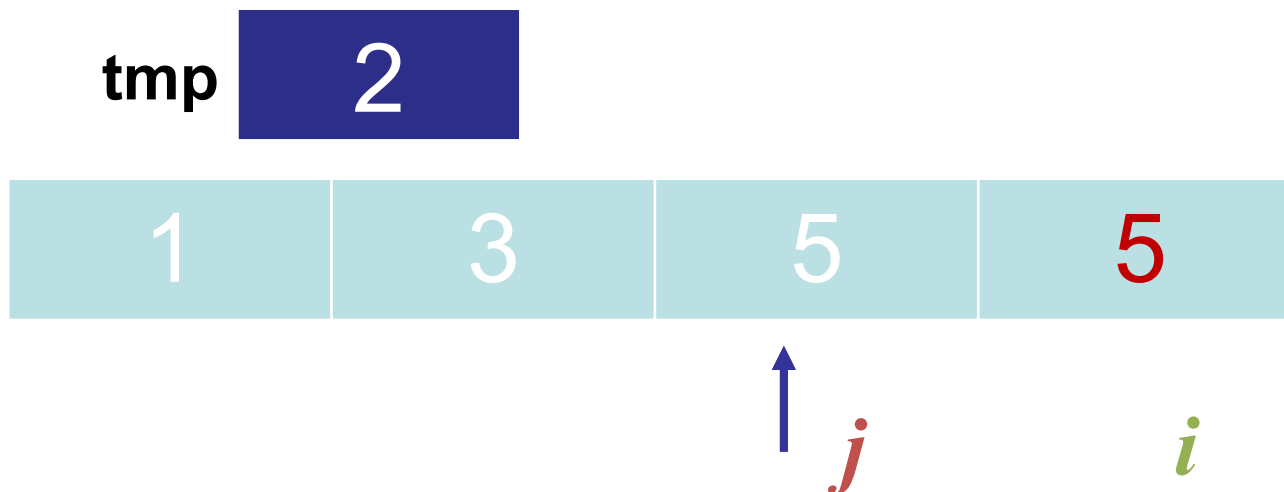
```
28     for (int i = 0; i < count; i++) {
29         int tmp = arr[i];
30     ➔   int j = i - 1;
31         for (; j >= 0; j--) {
32             if (arr[j] <= tmp) break;
33             arr[j+1] = arr[j];
34         }
35         arr[j+1] = tmp;
36     }
```

# 练习：插入排序跟踪



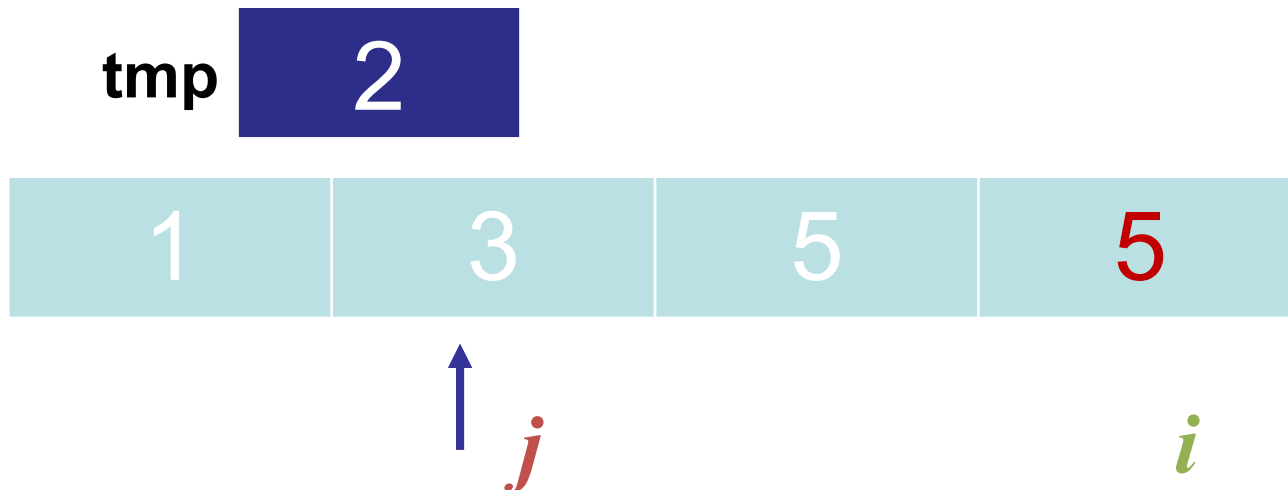
```
28     for (int i = 0; i < count; i++) {  
29         int tmp = arr[i];  
30         ➡ int j = i - 1;  
31         for (; j >= 0; j--) {  
32             if (arr[j] <= tmp) break;  
33             arr[j+1] = arr[j];  
34         }  
35         arr[j+1] = tmp;  
36     }
```

## 练习：插入排序跟踪



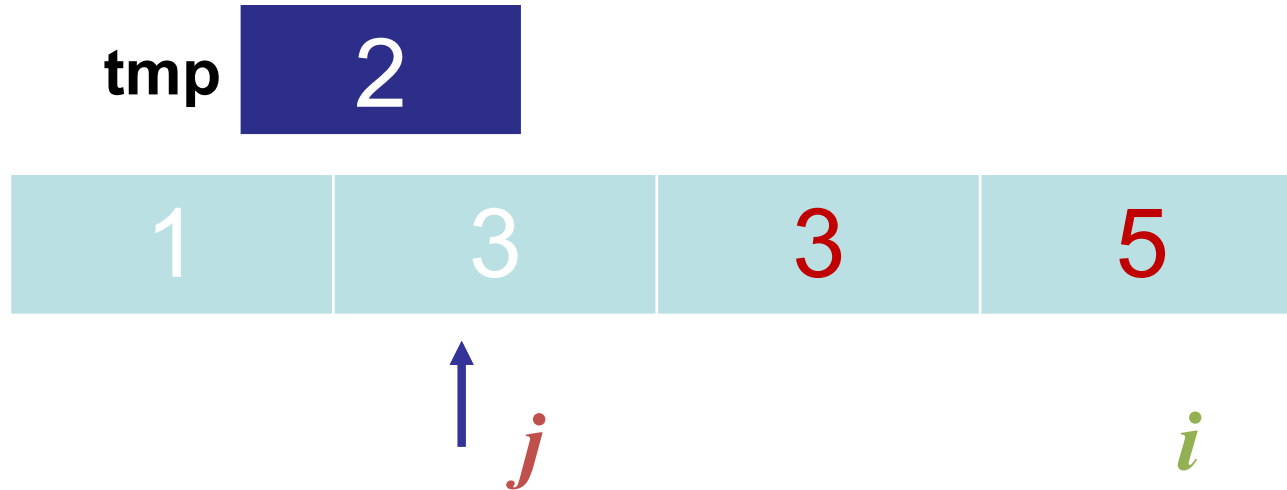
```
28     for (int i = 0; i < count; i++) {  
29         int tmp = arr[i];  
30         int j = i - 1;  
31         for (; j >= 0; j--) {  
32             if (arr[j] <= tmp) break;  
33             arr[j+1] = arr[j];  
34         }  
35         arr[j+1] = tmp;  
36     }
```

## 练习：插入排序跟踪



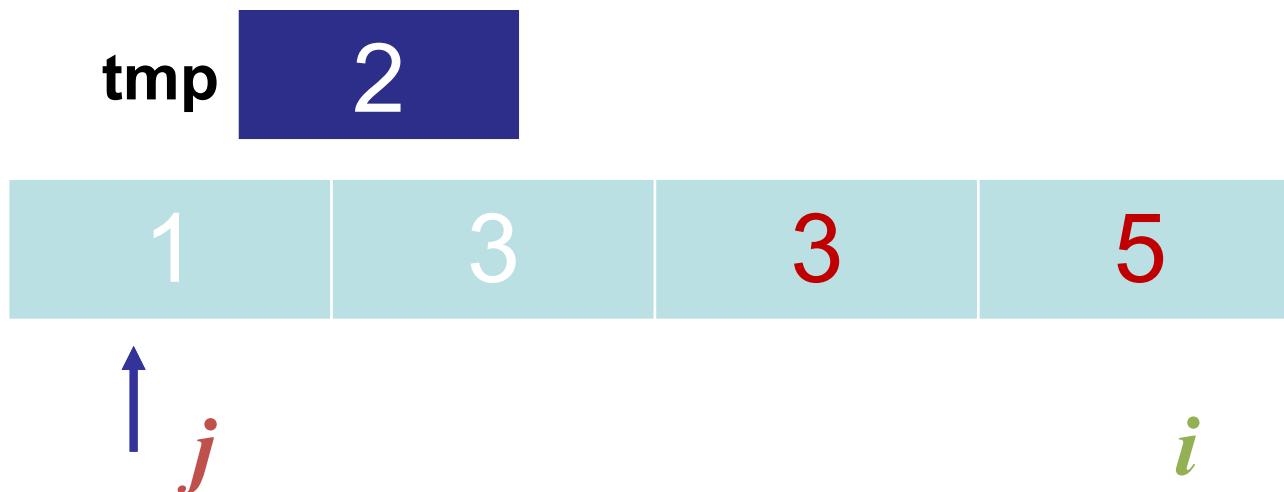
```
28     for (int i = 0; i < count; i++) {  
29         int tmp = arr[i];  
30         ➔ int j = i - 1;  
31         for (; j >= 0; j--) {  
32             if (arr[j] <= tmp) break;  
33             arr[j+1] = arr[j];  
34         }  
35         arr[j+1] = tmp;  
36     }
```

# 练习：插入排序跟踪



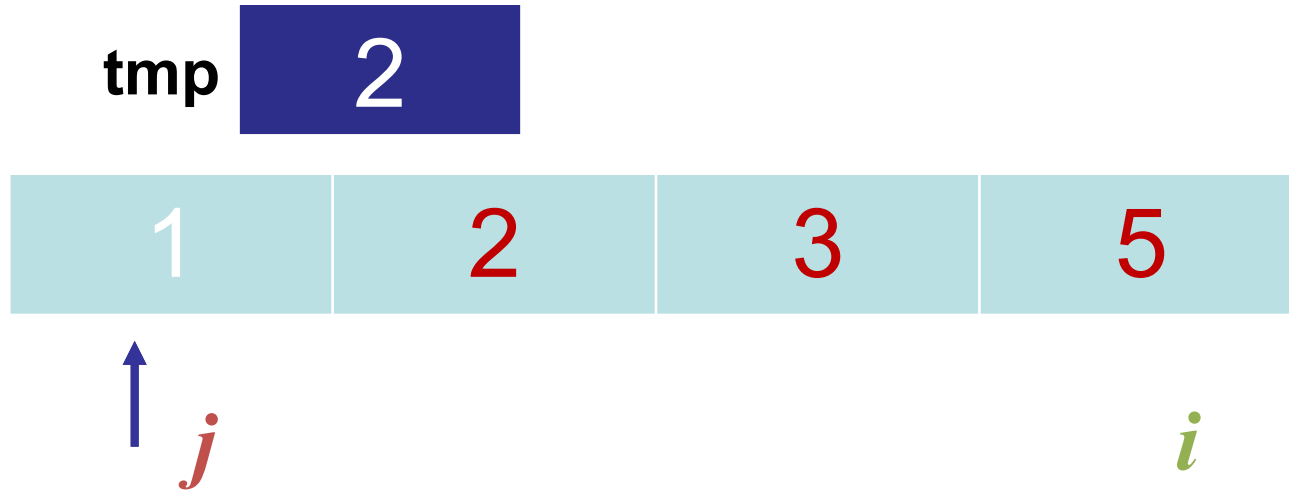
```
28     for (int i = 0; i < count; i++) {
29         int tmp = arr[i];
30         int j = i - 1;
31         for (; j >= 0; j--) {
32             if (arr[j] <= tmp) break;
33             arr[j+1] = arr[j];
34         }
35         arr[j+1] = tmp;
36     }
```

## 练习：插入排序跟踪



```
28     for (int i = 0; i < count; i++) {  
29         int tmp = arr[i];  
30         ➡ int j = i - 1;  
31         for (; j >= 0; j--) {  
32             if (arr[j] <= tmp) break;  
33             arr[j+1] = arr[j];  
34         }  
35         arr[j+1] = tmp;  
36     }
```

# 练习：插入排序跟踪



```
28     for (int i = 0; i < count; i++) {  
29         int tmp = arr[i];  
30         int j = i - 1;  
31         for (; j >= 0; j--) {  
32             if (arr[j] <= tmp) break;  
33             arr[j+1] = arr[j];  
34         }  
35         arr[j+1] = tmp;  
36     }
```

# #91 相似乐曲

一个音乐搜索引擎，用户输入一首乐曲 Q、一个正整数 k。从音乐库中查找与 Q 最相似的 k 首乐曲。乐曲由一组正整数组成，每个正整数表示声音的频率。计算乐曲相似度的方法是**直方图方法**。下面是详细的计算过程。↵

第一步，将整个频率的取值区域  $[0, 255]$  均分为 16 个子区域，即  $[0, 15]$ 、 $[16, 31]$ 、.....、 $[240, 255]$ 。扫描一首乐曲，计算组成该乐曲的所有频率值出现在每个子区域的次数。例如一首乐曲 M1 由 4 个频率组成 10、12、245、245，则它的直方图就是↵

$[0, 15]: 2$ ↵

$[16, 31]: 0$ ↵

.....↵

$[240, 255]: 2$ ↵

即，该乐曲的频率在  $[0, 15]$  和  $[240, 255]$  这两个子区域各出现 2 次，其他子区域均出现 0 次。↵

第二步，分别得到两个乐曲 M1 和 M2 的直方图之后，将两个直方图看作两个向量，采用欧几里得公式计算两个直方图的距离。↵

欧几里得公式：给定两个向量  $v1 = [x1, x2, ..., xn]$ ，以及  $v2 = [y1, y2, ..., yn]$ ，它们的欧几里得距离为  $\sqrt{(x1-y1)^2 + (x2-y2)^2 + ..... + (xn-yn)^2}$ ↵

例如，若另一首乐曲 M2 的直方图为↵

$[0, 15]: 2$ ↵

$[16, 31]: 0$ ↵

.....↵

$[240, 255]: 0$ ↵

则 M1 和 M2 的欧几里得距离为  $\sqrt{(2-2)^2 + 0 + ... + (0-2)^2} = 2$ ↵



# #91 相似乐曲

所谓“与 Q 最相似的 k 首乐曲”，即与 Q 的欧几里得距离最小的前 k 首乐曲。

【输入格式】

- → 第 1 行，表示查询乐曲，一个正整数  $n_0$  ( $1 \leq n_0 \leq 100$ )，表示乐曲长度，后面有  $n_0$  个整数，每个整数在  $[0, 255]$  内，表示一个频率。
- → 第 2 行，两个整数  $n$  和  $k$ ，用空格隔开。表示有  $n$  首乐曲， $1 \leq n \leq 100$ ，查找最相似的  $k$  ( $1 \leq k \leq n$ ) 首乐曲。
- → 第 3 行到  $n+2$  行，表示编号从 0 到  $n-1$  的  $n$  首乐曲。每行一个正整数  $n_i$  ( $1 \leq n_i \leq 100$ )，表示该乐曲长度，后面  $n_i$  个整数，每个整数在  $[0, 255]$  内，表示一个频率。

【输出格式】

输出  $k$  个整数，与查询乐曲最相似的乐曲的编号，注意乐曲编号范围是  $[0, n-1]$ 。按相似度从高到低（即：欧式距离从小到大）的顺序输出。若两首乐曲与 Q 的距离相同，则编号小的排名靠前。

【输入样例】

4 10 12 245 245

3 1

6 2 4 2 250 250 250

1 189

4 10 12 245 245

【输出样例】

2

```
int main(){
    ... int n,k,n0,i,j,count0[16]={0},m;
    ... int a[100],b[100][3],d[100][100],count[100][16]={0}; //a 存曲子.
        float sim[100]; //c[i]存相似度.
    ...

```

```
    ... scanf("%d",&n0); //曲长.
    ... for(i=0;i<n0;i++){
        ... scanf("%d",&a[i]);
            scanf("%d %d",&n,&k); //n 首曲子找 k 个最相近.
    ...

```

```
    ... for(i=0;i<n;i++){
        ... b[i][2]=i; //编号.
    ...

```

```
    ... for(i=0;i<n;i++){
        ... scanf("%d",&b[i][1]);
        ... for(j=0;j<b[i][1];j++){
            ... scanf("%d",&d[i][j]);
        ... }
    ...

```

```

....for(i=0;i<16;i++){
.....for(j=0;j<n0;j++){
.....if(a[j]>=0+i*16&& a[j]<=15+i*16){
.....count0[i]++;//识别曲每个区间里的频率,
.....}
}

```

..

```

....for(i=0;i<n;i++){
.....for(j=0;j<16;j++){
.....for(m=0;m<b[i][1];m++){
.....if(d[i][m]>=0+j*16&& d[i][m]<=15+16*j){
.....count[i][j]++;//每首曲子每个区间的频率,
.....}
}
}

```

//上面的可以简化换成如下的,

```

....//for(i=0;i<n;i++){
....//....for(m=0;m<b[i][1];m++){
....//.....j=d[i][m]%16,
....//.....count[i][j]++;//每首曲子每个区间的频率,
....//.....}

```

```

- - - int sum;
- - - for(i=0;i<n;i++){
- - - - sum=0;
- - - - for(j=0;j<16;j++){
- - - - - sum=sum+(count0[j]-count[i][j])*(count0[j]-count[i][j]);
- - - - sim[i]=sqrt(sum);//第i首曲子的相似度
- - - }

```

```

- - - float max;int t;
- - - for(i=0;i<n-1;i++){
- - - - for(j=0;j<n-1-i;j++){
- - - - - if(sim[j]>sim[j+1]||(-(abs(sim[j+1]-sim[j])<1e-6)&&b[j][2]>b[j+1][2])){
- - - - - - max=sim[j];sim[j]=sim[j+1];sim[j+1]=max;
- - - - - - t=b[j][2];b[j][2]=b[j+1][2];b[j+1][2]=t;
- - - - - }

```

```

- - - for(i=0;i<k;i++){
- - - - printf("%d",b[i][2]);//要有空格
- - - return 0;
}

```

#91 相似乐曲.cpp

# #195 平均成绩排序

有  $n$  位学生，每位学生修读的科目数不尽相同，已知所有学生的各科成绩，要求按学生平均成绩由高到低输出学生的学号、平均成绩；当平均成绩同时，按学号从低到高排序。对平均成绩，只取小数点后前 2 位，从第 3 位开始舍弃（无需舍入）。↵

输入格式↵

□□输入为  $n+1$  行，第一行为  $n$  表示学生人数。↵

□□从第二行开始的  $n$  行，每行为一名学生的成绩信息，包括：学号、科目数，各科成绩。其中  $n$ 、学号、成绩均为整数，它们的值域为： $0 \leq n \leq 10000$ ， $1 \leq \text{学号} \leq 1000000$ ， $0 \leq \text{成绩} \leq 100$ 。学生的科目数都不超过 100 门。↵

输出格式↵

□□最多  $n$  行，每行两个数，学号在前，后为平均成绩，空格分隔。若  $n$  为 0，输出 NO；若某学生所修科目不到 2 门，则不纳入排序，若无人修满 2 门，也输出 NO。↵

输入样例↵

```
5↵
1001 2 89 78↵
2003 4 88 99 100 88↵
4004 3 72 80 66↵
1004 3 70 66 82↵
3001 1 100↵
```

输出样例↵

```
2003 93.75↵
1001 83.50↵
┌───────────┐
1004 72.66↵
4004 72.66↵
```

# #307 生辰八字

大富商杨家有一女儿到了出阁的年纪，杨老爷决定面向全城适龄男士征婚。杨老爷遵循传统，决定按生辰八字作为选女婿的依据。每个应征的男士须提供自己的生辰八字，亦即八个正整数，每个数的取值范围均在[1, 24]。杨老爷特意聘请了黄半仙来算命，选择和女儿最契合的前 k 个男士作为候选。黄半仙采用的算命方法是西洋传入的余弦相似度，具体做法如下：

给定两个生辰八字  $A = [a_1, a_2, \dots, a_8]$ ,  $B = [b_1, b_2, \dots, b_8]$ , 则

【样例输出】

$$\text{Similarity}(A, B) = \sum_{i=1}^8 a_i * b_i / (\sqrt{\sum_{i=1}^8 (a_i)^2} * \sqrt{\sum_{i=1}^8 (b_i)^2})$$

1012345678

例如，若  $A = [10, 1, 1, 1, 1, 1, 1, 1]$ ， $B = [1, 1, 1, 1, 1, 1, 1, 1]$ ，则  $\text{Similarity}(A, B) = (10*1 + 1*1 + \dots + 1*1) / (\text{sqrt}(10^2 + 1^2 + \dots + 1^2) + \text{sqrt}(1^2 + 1^2 + \dots + 1^2)) = 1.29$

黄半仙认为一个男士和小姐两人的生辰八字的余弦相似度越大，两人就越契合。

【输入格式】

第 1 行两个整数，n 和 k，(1 ≤ n ≤ 100, 1 ≤ k ≤ n)，表示有 n 个男士应征，以及需要选择 k 人进入候选名单。

第 2 行 8 个整数，表示杨小姐的生辰八字。

后面 n 行，每行 9 个整数，第一个数字是某男士的身份证号，8 位整数；后面 8 个数字是该男士的生辰八字。整数之间均以空格隔开。

【样例输入】

2 1

1 1 1 1 1 1 1 1

1012345678 1 1 1 1 1 1 1 1

1087654321 1 1 1 1 1 8 8 8

【输出格式】

k 个整数，表示契合度最好的前 k 位男士的身份证号，按相似度从高到低排列。

注意：若两个男士和小姐的相似度相同，则身份证号码大的一个排在前面。当相似度相差小于 1e-10 时，认为相同。

# #110 回文判断

---

## ■ 回文

- Able was I ere I saw Elba
- 落败孤岛孤败落

## ■ 写一个程序，让用户任意输入一个字符串，判断是否是回文

## ■ 分析思路

- 判断位置 $i$ 的字符与 $n-1-i$ 的字符是否相等
- 循环开始和终止的边界？

# #110 回文判断

```
int main()
{
    int n, j, i;
    char a[1000];
    gets(a);
    n = strlen(a);
    for (i = 0, j = n - 1; i < (n + 1) / 2; i++, j--)
        if (a[i] != a[j])
            { printf("No"); break; }
    if (i == (n + 1) / 2) printf("Yes");
    return 0;
}
```

```
for (i = 0, j = n - 1; i < (n + 1) / 2; i++, j--)
    if (a[i] != a[j])
        { printf("No"); break; }
    if (i == (n + 1) / 2) printf("Yes");
```

#110 回文判断.cpp



# #260 二分查找

第一行一个数  $n$ ，表示需要输入的正整数个数。

第二行一个数  $m$  ( $1 \leq m \leq 2^{30}$ )，表示待查找的整数。

第三行包含  $n$  个正整数 ( $\leq 2^{30}$ )，每两个整数之间用一个空格隔开，是给定集合中的  $n$  个元素，输入数据保证这  $n$  个整数互不相等。

输出格式

输出共两行。

第一行包含一个整数  $k$ ，表示待查找的数在排序后的集合中的位置（位置从  $0 \sim n-1$  标号），如果没有找到则输出  $-1$ 。

第二行包含一个整数，表示查找成功前的比较次数。

输入样例 1

10

24

42 24 10 29 27 12 58 31 8 16

输出样例 1

4

1

特殊提示

【样例 1 说明】

排序后的集合  $a$  为：8 10 12 16 24 27 29 31 42 58

待查找的数 24，第一次与  $a[4]$  比较， $a[4]$  是 24，找到元素，算法结束，所以比较次数为 1。

# 二分查找

If searching for 23 in the 10-element array:

2	5	8	12	16	23	38	56	72	91
---	---	---	----	----	----	----	----	----	----

23 > 16,  
take 2<sup>nd</sup> half

L									H
2	5	8	12	16	23	38	56	72	91

23 < 56,  
take 1<sup>st</sup> half

					L					H
2	5	8	12	16	23	38	56	72	91	

Found 23,  
Return 5

					L	H			
2	5	8	12	16	23	38	56	72	91

# 二分查找

---

```
int L = 0, R = n-1;
int cmp = 0, found = -1;
while (L <= R) {
    int mid = (l + r) / 2;
    cmp ++;
    if (m < arr[mid] ) R = mid - 1;
    else if (m > arr[mid] ) L = mid + 1;
    else {
        found = mid;
        break;
    }
}
```

# 10-29上机

---

- #161 数据加密 (循环)
- #310 同构数 (循环)
- #195 平均成绩排序 (数组、排序)
- #307 生辰八字 (数组、排序)
- #260 二分查找 (数组、排序、循环)

# #161 数据加密

某个公司采用公用电话传递数据，数据是四位的整数，数据在传递过程中是加密的：每位数字都加上 5，得到的结果除以 10 的余数代替该数字，再将第一位和第四位交换，第二位和第三位交换。请你编写程序按照上述规则加密数据。

输入格式

□□输入只有一行，包括一个 4 位数的正整数  $d$  ( $1000 \leq d \leq 9999$ )，表示加密前的数据。

输出格式

□□输出只有一行，也是一个 4 位数的正整数，表示加密后的数据。

输入样例

1235

输出样例

876

特殊提示

□□【样例 1 说明】

□□1235 每位上数字加 5 后模 10 得到的新数字是 6780，按照要求第一位第四位交换，第二位第三位交换后是 876（先导 0 不输出）。

读取每一位

# #310 同构数

【问题描述】↵

计算正整数[a,b]之间的全部“同构数”之和。所谓“同构数”，是指一个正整数 n 是它平方数的尾部，则称 n 为同构数。如 6 的平方是 36，6 出现在 36 的右端，6 就是同构数。76 的平方数是 5776，76 是同构数。↵

【输入格式】↵

输入只有一行，输入两个正整数 a 和 b，中间由一个空格分隔，其中：1≤a≤b≤10000。↵

【输出格式】↵

输出一行，一个正整数，为 a、b 之间同构数之和。↵

【输入样例 1】↵ 1 100↵	【输入样例 2】↵ 80 100↵
【输出样例 1】↵ 113↵	【输出样例 2】↵ 0↵

【数据规模说明】↵

1≤a≤b≤10000。↵

76的平方数是5776，76是同构数



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

## 04. 筛法求素数

---

# 求素数

---

```
1 int prime(int x)
2 {
3     for(int i=2;i*i<=x;i++)
4     {
5         if(x%i==0)
6             return 0;
7     }
8     return 1;
9 }
```





# 筛法

例：使用筛法求100以内的所有素数。

## 思路

1. 想象将100个数看作沙子和小石头子，让小石头子权称素数；让沙子当作非素数。弄一个筛子，只要将沙子筛走，剩下的就是素数了。
2. 非素数一定是 2、3、4 ..... 的倍数。
3. 使用数组，让下标就是100以内的数，让数组元素的值作为筛去与否的标志。比如筛去以后让元素值为1。

		0	0	1	0	1	0	...	1	1
0	1	2	3	4	5	6	7		99	100

# 方法的依据

---

- 1至100这些自然数可以分为三类：
  - **单位数**：仅有一个数1。
  - **素数**：是这样一个数，它大于1，且只有1 和它自身这样两个正因数。
  - **合数**：除了1和自身以外，还有其他正因数。

**1不是素数，除1以外的自然数，当然只有素数与合数。  
筛法实际上是筛去合数，留下素数。**

# 该题目的筛法思路

- 第一块是一个计数型的循环语句，功能是将prime数组清零。

`prime[c] = 0;            c = 2, 3, ..., 100`

- 第二块是正因数d初始化为  $d = 2$

➤ 所有d的倍数将会被筛掉

- 第三块是循环筛数。这里用了一个 do while 语句，属于一种直到型循环，其一般形式为：

```
do
{
    循环体语句块
}
while ( 表达式 )
```

# 代码

```
int main() {
    int n, prime[10000];
    scanf("%d", &n);
    prime[1] = 0;
    for (int i = 2; i <= n; i++)
        prime[i] = 1;
    for (int i = 2; i <= n; i++) {
        if (prime[i] == 1)
            for (int j = 2; j * i <= n; j++)
                prime[i * j] = 0;
    }
    for (int i = 2; i <= n; i++)
        if (prime[i])
            printf("%d ", i);
    return 0;
}
```

筛法求素数.cpp

# 效率的考虑

---

- 令  $n$  为合数（这里是100）， $c$  为  $n$  的最小正因数

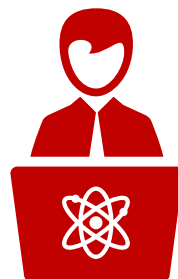
$$1 < c \leq \sqrt{n}$$

只要找到  $c$  就可以确认  $n$  为合数，将其筛去

- **注意：要进行“筛”的1—100的数字是与数组prime[101]的下标相对应的，而每个数组元素的取值只有2个：是0或1，分别代表（标志）与下标相对应的数字是素数或不是素数**



中國人民大學  
RENMIN UNIVERSITY OF CHINA



---

# 谢谢大家!

---

