

程序设计基础

教学团队：徐明星，兴军亮，任炬

renju@tsinghua.edu.cn

2024秋，每周一第2节，三教2301

本讲提纲

- 问题解答（程序调试方法）
- 6.1 字符串
- 6.2 二维数组
- 6.3 动态内存分配
- 6.4 函数指针
- 6.5 结构体
- 6.6 环形数组

【调试任务】求30以内的素数

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

int main() {
    int prime[30];
    int i, k;
    //用筛法求30以内素数
    for (i = 2; i < 30; i++) {
        if (prime[i] == 0) {
            k = i;
            do {
                k = k + i;
                prime[k] = 1;
            } while (k <= 30);
        }
        cout << i << ' ' << prime[i] << endl;
    }
}
```

以上代码在Dev-C++ 5.11上运行后显示:

```
1 0
2 40
1 0
2 40
.....
```

如果将 `int prime[30];` 或者 `int i;` 移至主函数之前则无上述问题。
如果将 `int prime[30];` 改为 `int prime[30] = {};` 则显示变为:

```
1 0
1 0
1 0
.....
```

持续输出，无法停止！



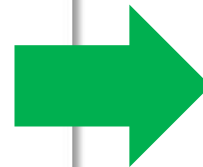
“勿以恶小而为之，勿以善小而不为”



《三国志·蜀书·先主传》

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

int main() {
    int prime[30];
    int i, k;
    //用筛法求30以内素数
    for (i = 2; i < 30; i++) {
        if (prime[i] == 0) {
            k = i;
            do {
                k = k + i;
                prime[k] = 1;
            } while (k <= 30);
        }
        cout << i << ' ' << prime[i] << endl;
    }
}
```

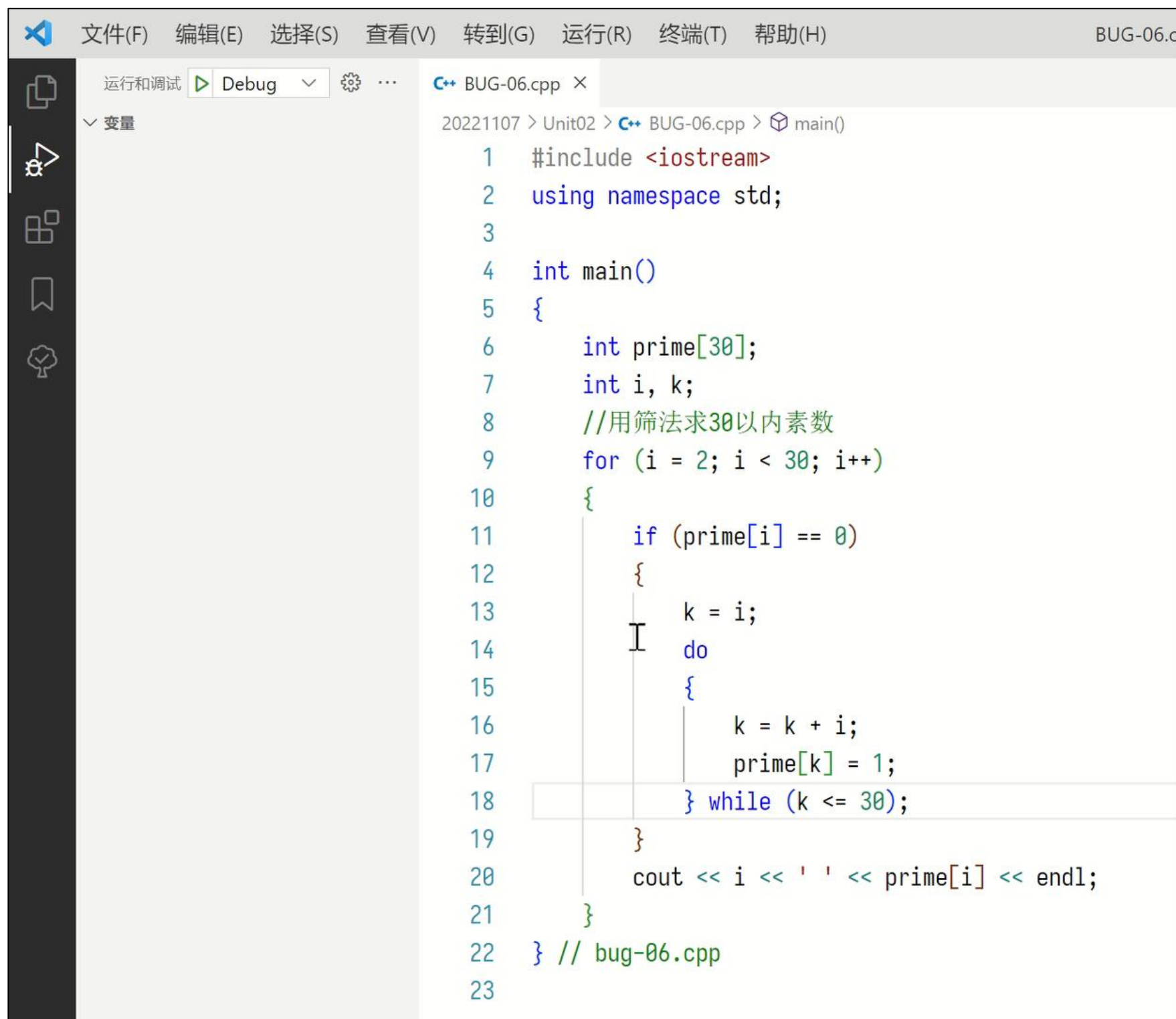


```
#include <iostream>
using namespace std;

int main()
{
    int prime[30];
    int i, k;
    //用筛法求30以内素数
    for (i = 2; i < 30; i++)
    {
        if (prime[i] == 0)
        {
            k = i;
            do {
                k = k + i;
                prime[k] = 1;
            } while (k <= 30);
        }
        cout << i << ' ' << prime[i] << endl;
    }
} // bug-06.cpp
```

务必保持良好的代码缩进风格！

条件断点设置的操作录屏频



The screenshot shows the Visual Studio Code interface with a C++ file named `BUG-06.cpp` open. The file contains a program to find prime numbers up to 30 using the Sieve of Eratosthenes. A conditional breakpoint is set on line 18, with the condition `k <= 30`. The breakpoint is represented by a red dot on the left margin, and a small box next to it contains the condition `k <= 30`. The code is as follows:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int prime[30];
7      int i, k;
8      //用筛法求30以内素数
9      for (i = 2; i < 30; i++)
10     {
11         if (prime[i] == 0)
12         {
13             k = i;
14             do
15             {
16                 k = k + i;
17                 prime[k] = 1;
18             } while (k <= 30);
19         }
20         cout << i << ' ' << prime[i] << endl;
21     }
22 } // bug-06.cpp
23
```

条件断点：满足条件时触发的断点

The screenshot shows a C++ IDE with a program for finding prime numbers up to 30. The program is as follows:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int prime[30];
7      int i, k;
8      //用筛法求30以内素数
9      for (i = 2; i < 30; i++)
10     {
11         if (prime[i] == 0)
12         {
13             k = i;
14             do
15             {
16                 k = k + i;
17                 prime[k] = 1;
18                 while (k <= 30);
19             }
20             cout << i << ' ' << prime[i] << endl;
21         }
22     } // bug-06.cpp
```

The IDE interface includes a **VARIABLES** panel on the left showing local variables: `prime: [30]`, `i: 0`, and `k: 16`. Below it, the **WATCH** panel shows the expression `i == 2: false`.

A conditional breakpoint is set on line 18 of the code. The breakpoint is represented by a red dot with a minus sign. The expression for the breakpoint is `i != 2`, which is circled in red. A context menu is open over the breakpoint, showing options: **删除断点** (Delete), **编辑断点...** (Edit Breakpoint...), and **禁用断点** (Disable Breakpoint).

通过地址对比查找变量异常改动的原因

g++.exe - 生成

VARIABLES

Locals

> prime: [30]

i: 1

k: 31

WATCH

i == 2: false

> &prime[k]: 0x61fe1c

*&prime[k]: 1

> &i: 0x61fe1c

*&i: 1

> &k: 0x61fe18

*&k: 31

> &prime: 0x61fda0

> &prime[29]: 0x61fe14

> &prime[30]: 0x61fe18

> &prime[31]: 0x61fe1c

数组越界，破坏其他变量

C++ bug-06-A.cpp **C++ bug-06.cpp** X

C++ bug-06.cpp > main()

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int prime[30];
7      int i, k;
8      //用筛法求30以内素数
9      for (i = 2; i < 30; i++)
10     {
11         if (prime[i] == 0)
12         {
13             k = i;
14             do
15             {
16                 k = k + i;
17                 prime[k] = 1;
18             } while (k <= 30);
```

条件断点被触发时，k 等于 31

从数组越界访问举一反三

修改后的代码（仅供参考）

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

int main() {
    int prime[30] = {0};
    int i, k;
    //用筛法求30以内素数
    for (i = 2; i < 30; i++) {
        if (prime[i] == 0) {
            k = i;
            do {
                k = k + i;
                if (k < 30) prime[k] = 1;
            } while (k < 30);
        }
        cout << i << ' ' << prime[i] << endl;
    }
    return 0;
} // bug-06-A.cpp
```

检查边界条件和初始状态



针对以下代码，哪个选项是正确的？

```
int main()  
{  
    while (1) ;    // A  
    for (;;) ;     // B  
    do while (1);  // C  
    if (1) ;       // D  
}
```

A

注释为A的行有编译错误

B

注释为B的行有编译错误

C

注释为C的行有编译错误

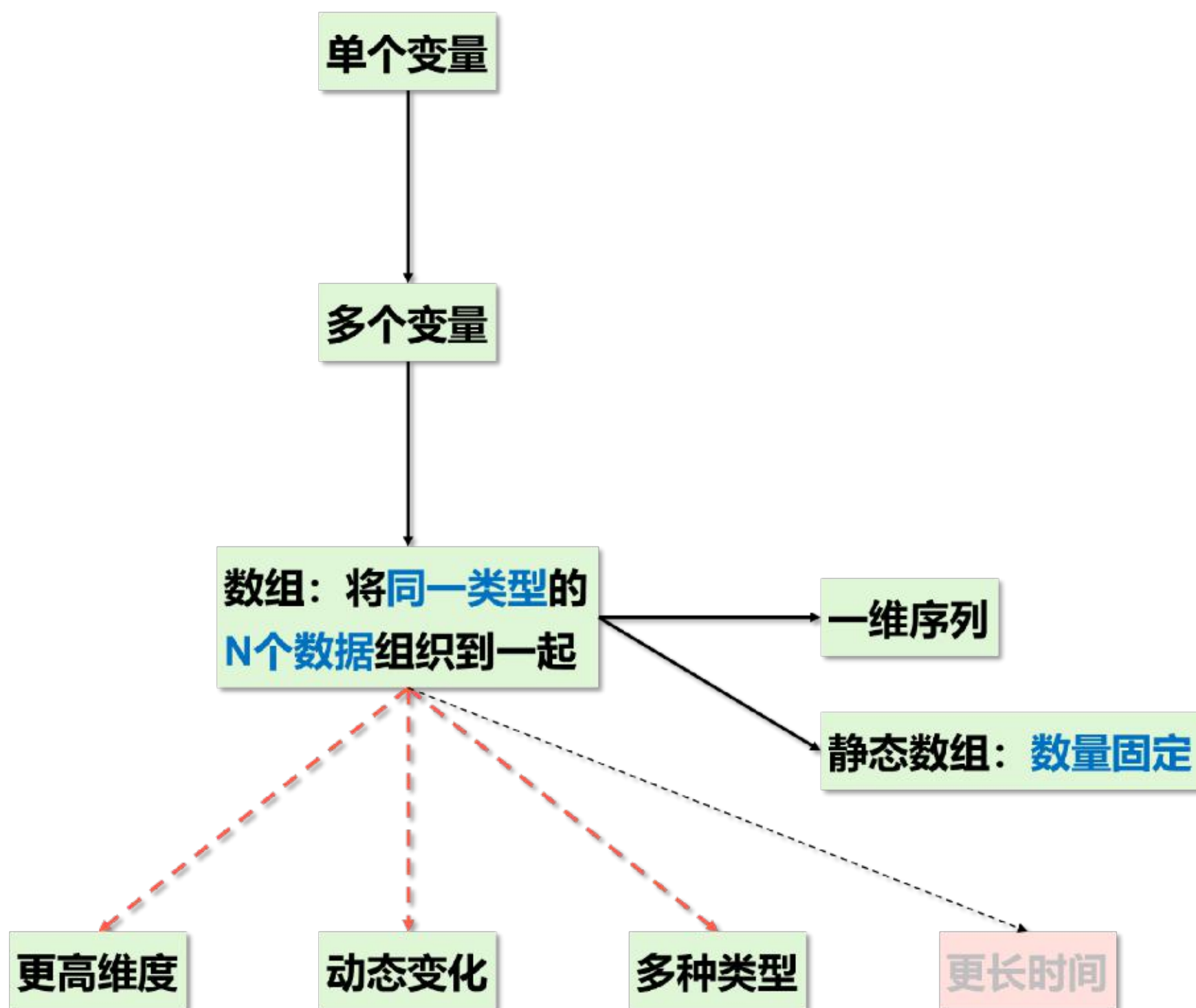
D

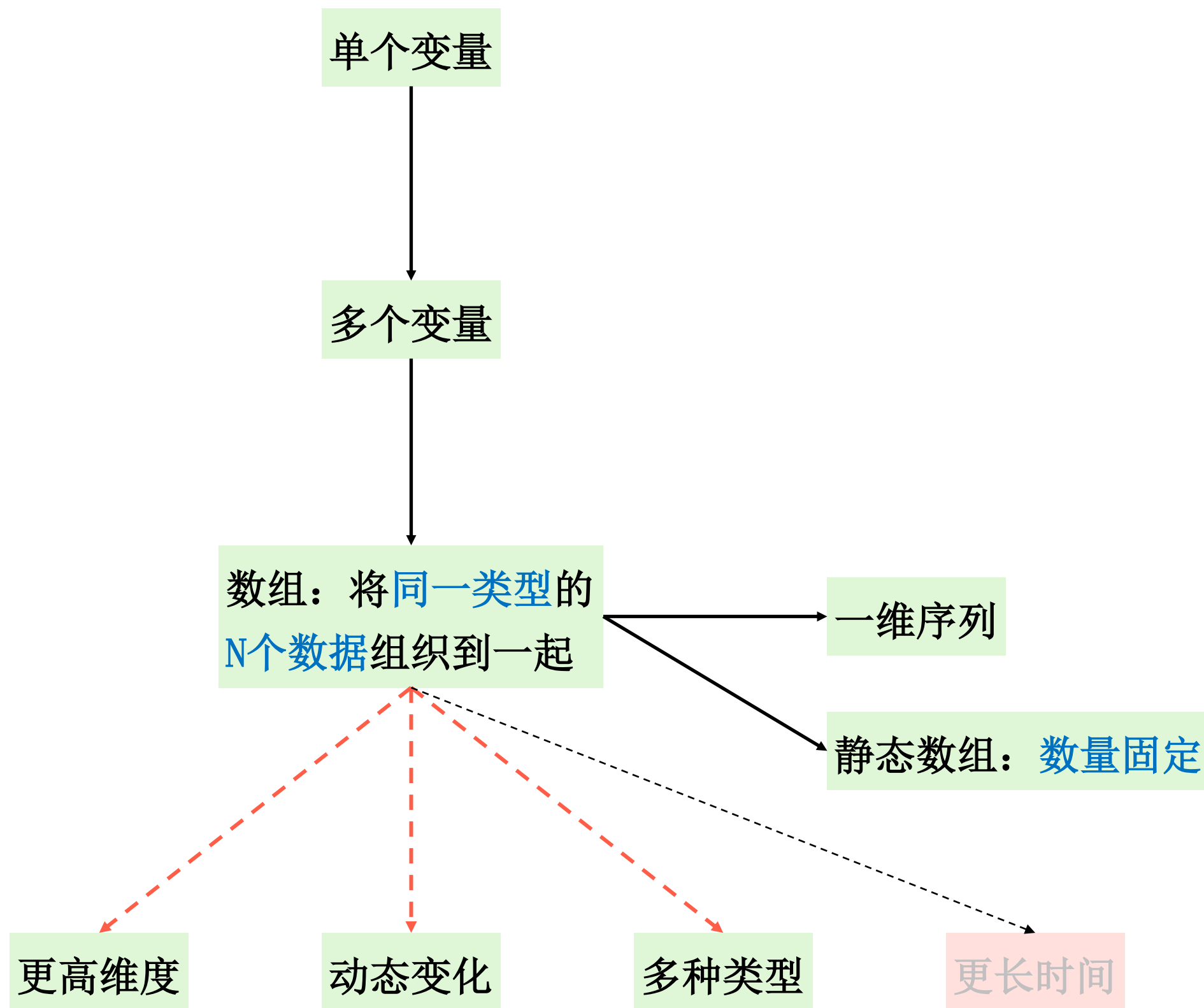
注释为D的行有编译错误

提交

第6讲 数据组织与处理（中）

程序中的数据（变量）组织方式





字符指针变量与字符串

1、单个字符的地址，也可以用字符指针变量保存

```
char ch = 'A';    char *pc = &ch;
```

2、cout在输出指针变量时，

2.1【通常】输出指针变量保存的地址值

```
int m; int *pm = &m; cout << pm << endl;
```

2.2【例外】对于字符指针变量，输出指针所指内存区域存放的字符内容：从字符指针存放的地址开始，一直输出，直到内存内容的值是0，即'\0'，输出才停止！

```
char ch = 'A';    char *pc = &ch;
```

```
cout << pc << endl;    /// 除输出'A'外，后面可能还会有乱七八糟的东西
```

```
char msg1[ ] = {'T', 'H', 'U'};
```

```
cout << msg1 << endl;  /// 除输出"THU"外，后面可能还会有乱七八糟的东西
```

```
char msg2[4] = {'T', 'H', 'U'};    /// 或 char msg2[] = "THU";
```

```
cout << msg2 << endl;  /// 输出"THU"，一切正常！
```

如何输出字符指针的地址？强制转换为非字符型指针 `static_cast<const void *>(pc)`

【编程经验】 字符数组变量的初始化

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

'B'	'e'	'i'	'J'	'i'	'n'	'g'	'\0'	'\0'	'\0'
-----	-----	-----	-----	-----	-----	-----	------	------	------

```
{
    char A[10] = {'B', 'e', 'i', 'J', 'i', 'n', 'g'};
    char B[10] = "BeiJing";
    char C[10] = {"BeiJing"};

    for (int i = 0; i < 10; i++)
        cout << A[i] << ' ' << setw(4) << int(A[i]) << " == "
              << B[i] << ' ' << setw(4) << int(B[i]) << " == "
              << C[i] << ' ' << setw(4) << int(C[i]) << endl;

    return 0;
}
```


试一试：输出字符数组变量

程序输出是什么？



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

int main()
{
    char str1[] = "123";
    char str2[] = {'1', '2', '3'};
    cout << sizeof(str1) << ' '
         << sizeof(str2) << endl;

    cout << str1 << '\n'
         << str2 << '\n';
    return 0;
}
```

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      char str1[] = "123";
6      char str2[] = {'1', '2', '3'};
7      cout << sizeof(str1) << ' '
8           << sizeof(str2) << endl;
9      cout << str1 << '\n'
10         << str2 << '\n';
11      return 0;
12 }
```

终端

```
4 3
123
123123
PS D:\FOP.VS.Projects> 
```

以下代码的输出结果是 [填空1]

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

int main()
{
    char name[20] = "Tsinghua";
    int len = sizeof(name);
    name[len - 1] = 'A';
    name[len - 2] = 'A';
    cout << name;

    return 0;
}
```

作答

处理字符串的常用函数

strlen, strcat, strcmp, strcpy

```
#include <iostream>    // cout
#include <cstring>      // strlen, strcat, strcmp, strcpy
using namespace std;

int main() {
    char str1[50] = "BeiJing";
    cout << "str1 length: " << strlen(str1) << endl; // 字符串的长度
    strcat(str1, ", Tsinghua!");                       // 拼接两个字符串
    cout << "---> " << str1 << " ==> str1 length: " << strlen(str1) << endl;
    cout << "Tsinghua > Peking? "                     // 比较两个字符串的字典序大小
        << strcmp("Tsinghua", "Peking") << endl;      // 相等时返回0,大于返回1,小于返回-1
    char str2[100];                                     // 要注意有足够的空间
    strcpy(str2, "THIS YEAR, ");                       // 复制字符串
    strcat(str2, str1);
    cout << "str2 = " << str2 << endl;
    return 0;
}
```

用标准模板库STL中的string来存储“字符串”数据

```
#include <iostream> // cout
// #include <cstring> // strlen, strcat, strcmp, strcpy
#include <string> // string
using namespace std;

int main()
{
    // char str1[50] = "BeiJing";
    string str1 = "Beijing";
    // cout << "str1 length: " << strlen(str1) << endl; // 字符串的长度
    cout << "str1 length: " << str1.length() << endl; // 字符串的长度
    // strcat(str1, ", Tsinghua!"); // 拼接两个字符串
    str1 += ", Tsinghua!";
    // cout << "---> " << str1 << " ==> str1 length: " << strlen(str1) << endl;
    cout << "---> " << str1 << " ==> str1 length: " << str1.length() << endl;
    // cout << "Tsinghua > Peking? " // 比较两个字符串的字典序大小
    // << strcmp("Tsinghua", "Peking") << endl; // 相等时返回 0
    cout << "Tsinghua > Peking? " // 比较两个字符串的字典序大小
        << (string("Tsinghua") > string("Peking")) << endl; // 相等时返回 0
}
```

用标准模板库STL中的string来存储“字符串”数据

```
// char str2[100];           // 要注意有足够的空间
string str2;

// strcpy(str2, "THIS YEAR, "); // 复制字符串
str2 = "THIS YEAR, ";

// strcat(str2, str1);
str2 += str1;

cout << "str2 = " << str2 << endl;
return 0;
}
```

【任务6.1】寻找交通肇事车

某日某地，发生了一起严重交通事故，肇事司机逃跑了！

目击者对交警说：肇事汽车的号码为4位完全平方数，
且数字从左到右一个比一个大。

请你编写程序，帮交警算出肇事车的号码。

解题思路

令 n 为车号

n 为 4 位数

$n = i*i, \quad i = 32, 33, \dots, 99 \quad (31*31 < 1000 < 32*32)$

枚举思想要点:

- 1、范围（问题解空间）
- 2、有序（能依次访问）
- 3、评估（测试和推演）

对 i 枚举，得到不同的 n 。这样可以缩小枚举的范围，提高算法的运行效率（运行时间开销）。

对每个 n ，检查 n 的各位，

从高位到低位，是否一个比一个大？即，低位的数值最大。

设 n 的各位从高到低依次为 $n_3 \ n_2 \ n_1 \ n_0$ ，则依题意有：

$$n_3 < n_2 < n_1 < n_0$$

解题思路

- ① 四位数的千位数字 $n3 = n / 1000$
- ② 百位数字 $n2 = n / 100 \% 10$
- ③ 十位数字 $n1 = n / 10 \% 10$
- ④ 个位数字 $n0 = n \% 10$

```
if (((n/1000) < (n/100 % 10)) &&  
    ((n/100 % 10) < (n/10 % 10)) &&  
    ((n/10 % 10) < (n % 10)))  
    cout << "肇事汽车号码为" << n << endl;
```

按从左往右顺序，一个比一个
大（越往后越大）。

基于直接计算各数位数值的算法实现

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

void find_it() {
    for (int i=32; i<100; i++) {
        int n = i * i;
        if (((n/1000) < (n/100 % 10)) &&
            ((n/100 % 10) < (n/10 % 10)) &&
            ((n/10 % 10) < (n % 10)))
            cout << "肇事汽车号码为" << n << endl;
    }
}

int main() {
    find_it();
    return 0;
}
```

基于 `sprintf` 函数的算法实现



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

```
void find_it() {
    for (int i = 32; i < 100; i++) {
        int n = i * i;
        char code[5];
        sprintf(code, "%d", n); // 把整数转成数字字符数组，再进行比较。
        if (code[0] < code[1] && code[1] < code[2] && code[2] < code[3])
            cout << "肇事汽车号码为" << n << endl;
    }
}
```

比较字符与比较
数值是等效的

```
int main() {
    find_it();
    return 0;
}
```

1369



'1' '3' '6' '9' '\0'

【任务6.2】矩阵相乘

矩阵是代数课程中的重要概念，在各类工程学科中应用极广，矩阵的运算是工程问题中的常见运算。

请你编程实现一个程序，能对任意两个 3×3 大小矩阵进行相加和相乘运算。

矩阵的内容用键盘输入，一行一行从左往右（即逐行逐列）输入数据。输出要求排列成 3×3 的矩阵形式。

矩阵的表示方法：二维数组

在C++程序中，使用二维数组来表示矩阵是比较方便的。元素类型为T的

二维数组定义为：

`T 数组名[行数][列数];`

例如，4行3列的整数矩阵m定义为：

```
int m[4][3];
```


矩阵乘法的“算法”原理

$A = \begin{bmatrix} 2 & 3 & -1 \\ 6 & 1 & -2 \end{bmatrix}$ $B = \begin{bmatrix} 4 & -5 \\ -3 & 0 \\ 1 & 2 \end{bmatrix}$

$A \times B = \begin{bmatrix} \square & \square \\ \square & -34 \end{bmatrix}$

$6 \times -5 = -30$
 $1 \times 0 = 0$
 $-2 \times 2 = -4$
 $-30 + 0 - 4 = -34$

$$AB[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0] + A[0][2] * B[2][0];$$

$$AB[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1] + A[0][2] * B[2][1];$$

$$AB[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0] + A[1][2] * B[2][0];$$

$$AB[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1] + A[1][2] * B[2][1];$$

重复感扑面而来

```
cout << "AB:\n" << AB[0][0] << ' ' << AB[0][1] << endl  
      << AB[1][0] << ' ' << AB[1][1] << endl;
```

矩阵乘法的“规律/规则”

$$\begin{aligned} AB[0][0] &= A[0][0] * B[0][0] + \\ &\quad A[0][1] * B[1][0] + \\ &\quad A[0][2] * B[2][0]; \end{aligned}$$

$$\begin{aligned} AB[0][1] &= A[0][0] * B[0][1] + \\ &\quad A[0][1] * B[1][1] + \\ &\quad A[0][2] * B[2][1]; \end{aligned}$$

$$\begin{aligned} AB[1][0] &= A[1][0] * B[0][0] + \\ &\quad A[1][1] * B[1][0] + \\ &\quad A[1][2] * B[2][0]; \end{aligned}$$

$$\begin{aligned} AB[1][1] &= A[1][0] * B[0][1] + \\ &\quad A[1][1] * B[1][1] + \\ &\quad A[1][2] * B[2][1]; \end{aligned}$$



$$\begin{aligned} AB[x][y] &= A[x][0] * B[0][y] + \\ &\quad A[x][1] * B[1][y] + \\ &\quad A[x][2] * B[2][y]; \end{aligned}$$

从0到2变化

从0到2变化

矩阵乘法的“算法”原理

```
int r[3][3];
```

```
for (int i=0; i<3; i++) // 行
```

```
for (int j=0; j<3; j++) // 列
```

使用多重循环!

```
r[i][j] = m[i][0] * n[0][j] +  
          m[i][1] * n[1][j] +  
          m[i][2] * n[2][j];
```

结果矩阵中的第i行第j列

第i行

第j列

行 是变量时,
用数组第0维

列 是变量时,
用数组第1维

矩阵相乘的算法实现

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

int main() {
    cout << "Input m[][]:" << endl;
    int m[3][3];
    for (int i=0; i<3; i++)
        for (int j=0; j<3; j++)
            cin >> m[i][j];

    cout << "Input n[][]:" << endl;
    int n[3][3];
    for (int i=0; i<3; i++)
        for (int j=0; j<3; j++)
            cin >> n[i][j];
```

```
int r[3][3];
for (int i=0; i<3; i++)
    for (int j=0; j<3; j++)
        r[i][j] = m[i][0] * n[0][j] +
                    m[i][1] * n[1][j] +
                    m[i][2] * n[2][j];
```

```
cout << "m[][] * n[][] = " << endl;
for (int i=0; i<3; i++) {
    for (int j=0; j<3; j++)
        cout << r[i][j] << ' ';
    cout << endl;
}

return 0;
```

```
}
```

【任务6.3】姓名排序

电视歌手大奖赛开赛报名时，由于人数较多，一些参赛信息需要及时录入计算机并用计算机进行管理。其中，有一个很重要的工作是：要按选手姓名（汉语拼音）排序后编号，以决定选手比赛的顺序。

请你编程实现对姓名拼音串按英文字典顺序排序的程序。

为测试程序，假定共有10名选手，选手姓名拼音最长不超过20个英文字符，且中间无空格。

姓名组成的数组：字符串数组

10名选手的姓名拼音序列，组成了一个数组：`TYPE namelist[10];` 这里的TYPE应该怎么写呢？注意：这里每个`namelist[i]`元素是一个姓名拼音字母组成的字符串，数组`namelist`的初始化方式是：

```
TYPE namelist[10] = {"zhang3", "li4", "wang5", "zhao6" };
```

数组`namelist`的元素——“字符串”需要用字符数组来存储，因此，我们需要用“字符数组的数组”这种类型来定义`namelist`。使用“字符的二维数组”，可以定义“字符数组的数组”，其语法形式如下：

```
char namelist[10][20];
```

参照`int num[20];`的语法解读，`num`变量的类型是20个整数的数组，因此，`namelist[10]`这个数组变量的每个元素类型是20个字符组成的数组。


```

#include <iostream>
using namespace std;

int main() {
    char namelist[10][20]; // 二维字符数组!
    for (int i=0; i<10; i++) {
        cout << "Input #" << i << " singer's name: ";
        cin >> namelist[i];
    }

    for (int i=0; i<9; i++) // 轮(遍,趟)数=元素数目-1
        for (int j=0; j<9-i; j++) // 比较数=总轮数-当前轮次
            if (strcmp(namelist[j], namelist[j+1]) > 0) {
                
            }

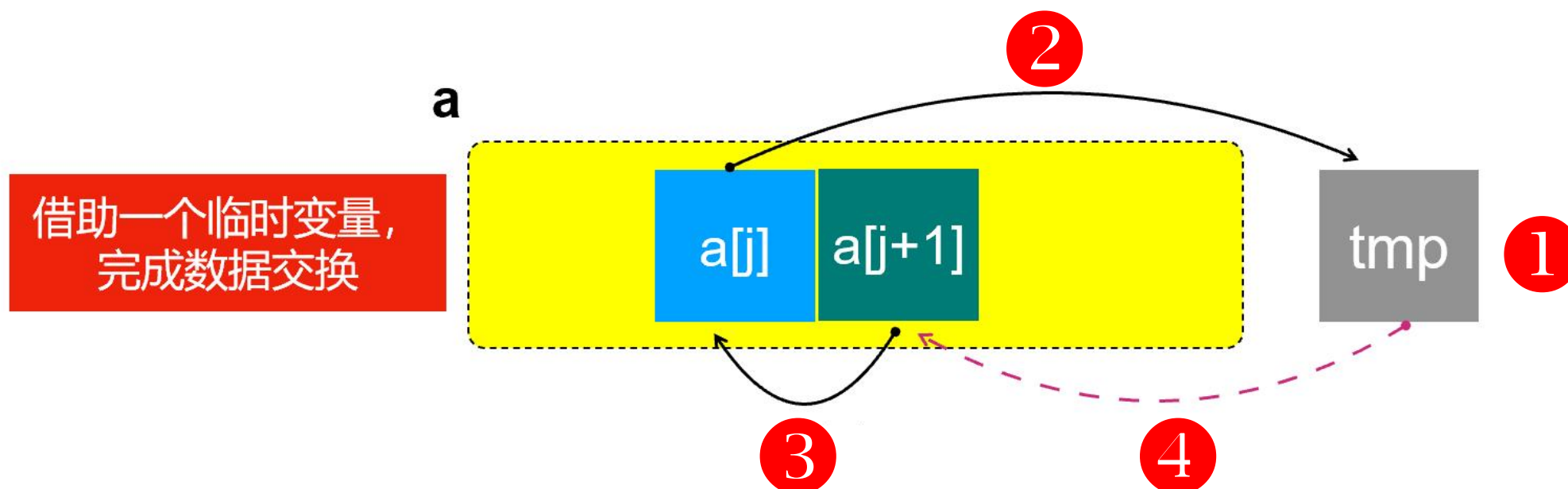
    for (int i=0; i<10; i++) cout << i << ' ' << namelist[i] << endl;
    return 0;
}

```

字符串元素的特殊交换算法

交换两个字符串的内容

```
if (strcmp(namelist[j], namelist[j+1]) > 0)
{
    char tmp[20];
    strcpy(tmp, namelist[j]);
    strcpy(namelist[j], namelist[j+1]);
    strcpy(namelist[j+1], tmp);
}
```



代码重构：使用STL中的string, vector, sort

```
#include <iostream> // cout
#include <string>     // string
#include <vector>     // vector
#include <algorithm> // sort
using namespace std;

int main()
{
    vector<string> namelist(10); // 10个歌手的名字数组（向量）
    for (int i = 0; i < 10; i++)
    {
        cout << "Input #" << i << " singer's name: ";
        cin >> namelist[i];
    }

    sort(namelist.begin(), namelist.end()); // 将向量“头部”和“尾部”传入排序函数
    for (int i = 0; i < 10; i++)
        cout << i << ' ' << namelist[i] << endl;
    return 0;
}
```

STL中的排序算法: sort

包含文件: `#include <algorithm>`

调用方法: `sort(序列头部, 序列尾部);`

```
#include <iostream> // cout
#include <algorithm> // sort
using namespace std;
```

```
int main()
{
```

```
    int array[] = {1, 3, 99, 10, 20, 42, 14, 86, 36};
```

```
    int num = sizeof(array) / sizeof(array[0]); /// ← 计算数组元素数量的小技巧
```

```
    sort(array, array + num);
```

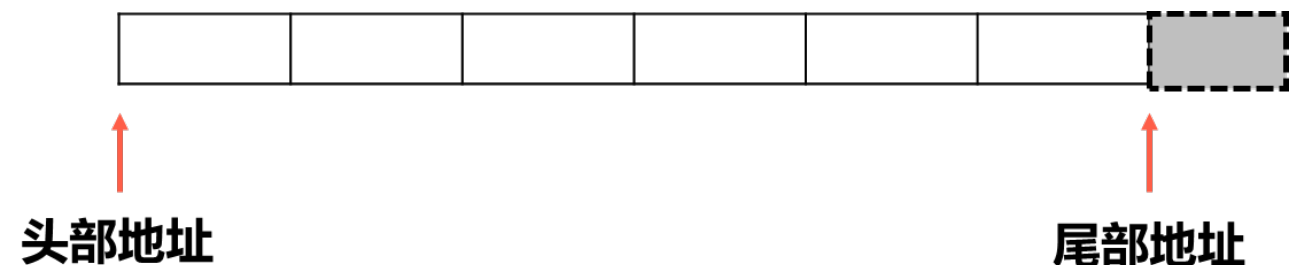
```
    for (int i = 0; i < num; i++)
```

```
        cout << array[i] << ' ';
```

```
    cout << endl;
```

```
    return 0;
```

```
}
```



姓名中有空格怎么办？

【任务6.3】姓名排序

电视歌手大奖赛开赛报名时，由于人数较多，一些参赛信息需要及时录入计算机并用计算机进行管理。其中，有一个很重要的工作是：要按选手姓名（汉语拼音）排序后编号，以决定选手比赛的顺序。

请你编程实现对姓名拼音串按英文字典顺序排序的程序。

为测试程序，假定共有10名选手，选手姓名拼音最长不超过20个英文字符，且**中间无空格**。

Zhang San

Li Si

Wang Wu

...



如何得到包含空格的一行输入？

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      string str1;
8      getline(cin, str1);
9      cout << "INPUT1 = [" << str1 << "]." << endl;
10
11     char str2[100];
12     cin.getline(str2, 99);
13     cout << "INPUT2 = [" << str2 << "]." << endl;
14
15     return 0;
16 }
```

← 要求指定最大长度

终端

```
Tsinghua University
INPUT1 = [Tsinghua University].
Tsinghua University
INPUT2 = [Tsinghua University].
PS D:\FOP.VS.Projects> █
```

【实验探究】一维数组元素的存放位置

```
#include <iostream>
using namespace std;
int main() {
    double array[12]; // 12 elements

    for (int i=0; i<12; i++)
        array[i] = i+1;

    for (int i=0; i<12; i++)
        cout << &(array[i]) << ':'
              << array[i] << endl;

    return 0;
}
```

0x66fdb0	1	低地址
0x66fdb8	2	
0x66fdc0	3	
0x66fdc8	4	
⋮	...	
0x66fdf8	10	
0x66fe00	11	
0x66fe08	12	高地址

【实验探究】二维数组元素的存放位置

```
#include <iostream>
using namespace std;
int main() {
    double array[4][3]; // 4 x 3 matrix

    int num = 1;
    for (int i=0; i<4; i++)           // row
        for (int j=0; j<3; j++)       // col
            array[i][j] = num++;
```

```
    for (int i=0; i<4; i++)           // row
        for (int j=0; j<3; j++)       // col
            cout << &(array[i][j]) << ':' << array[i][j] << endl;
```

```
    return 0;
```

```
}
```


【实验探究】二维数组元素的存放位置

`&(array[i][j])`表达式中的`&`是“取地址运算符”，它将得到`array[i][j]`元素在内存中的位置，也称内存地址（简称地址）。`cout`输出地址时，通常以16进制输出，`0x`为16进制数前缀。

`0x66fda0:1`

`0x66fda8:2`

`0x66fdb0:3`

`0x66fdb8:4`

`0x66fdc0:5`

`0x66fdc8:6`

`0x66fdd0:7`

`0x66fdd8:8`

`0x66fde0:9`

`0x66fde8:10`

`0x66fdf0:11`

`0x66fdf8:12`

1	2	3
4	5	6
7	8	9
10	11	12

数学上，矩阵是以二维形式书写的，但在计算机内存中，它是以一维方式“书写”（存储）的。

`0x66fda0`

1

低地址

`0x66fda8`

2

`0x66fdb0`

3

`0x66fdb8`

4

⋮

...

`0x66fde8`

10

`0x66fdf0`

11

`0x66fdf8`

12

高地址

【实验探究】X 维数组元素的存放位置

0x66fda0	1	低地址
0x66fda8	2	
0x66fdb0	3	
0x66fdb8	4	
⋮	...	
0x66fde8	10	高地址
0x66fdf0	11	
0x66fdf8	12	

若某个变量占用的内存如图所示，请问它是一维数组，还是二维数组？



双兔傍地走，安能辨我是雄雌？

二者的存放方式本来就是相同的，所以根本**无法从内存结构来区分它们！**

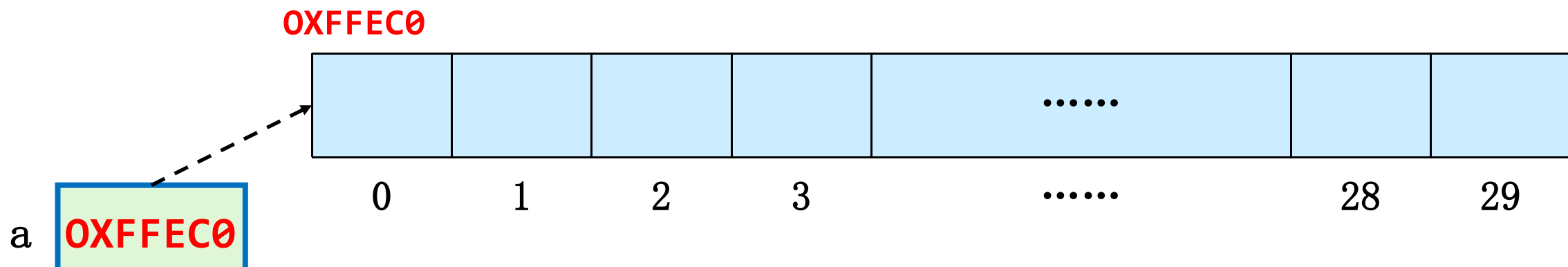
在运行时确定大小的动态数组

若想要在程序执行时再确定“数组”的大小，必须使用“**动态数组**”。这需要通过 **new** 运算符来实现。使用new运算符语法规则如下：

```
TYPE* var = new TYPE[NUM];
```

其中NUM既可以是**变量**，也可以是常量。例如：

```
int* a = new int[30];
```



a是占内存空间的指针变量，图中使用实线框。

安全使用动态数组的注意事项

凡是由 `new` 运算符“生成”的内存空间，在不再需要（使用）它时，应该使用 `delete` 运算符将内存“释放”掉。

对于动态生成的一维数组，参照下面的格式释放分配的内存单元：

```
delete[] a;    // 若之前已有 TYPE* a = new TYPE[SIZE];
```

【实验探究】使用内存释放后的动态数组指针变量

```
#include <iostream>
using namespace std;
void show(int *a, int num)
{
    for (int i = 0; i < num; i++)
        cout << a[i] << ' ';
    cout << endl;
}
int main()
{
    int *a = new int[4];
    for (int i = 0; i < 4; i++)
        a[i] = i * 10 + i % 7;
    show(a, 4);
    cout << "before delete[], a = " << a << endl;
    delete[] a; // 分配给a的内存全部释放了
    cout << "after delete[], a = " << a << endl;
    show(a, 4);
    return 0;
}
```

以一维动态数组为例，说明指针
被delete前后的变化和注意事项

和普通数组的使用方法一致

```
0 11 22 33
before delete[], a = 0x3a1728
after delete[], a = 0x3a1728
3810056 3801284 22 33
```

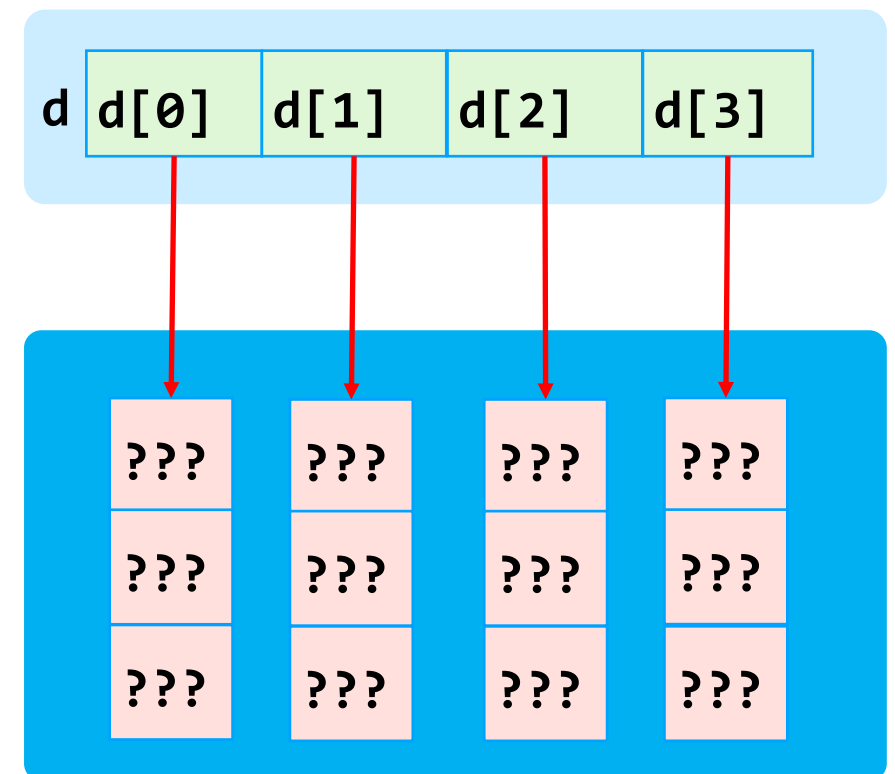
指针数组：元素是指针的静态数组

```
int main() {  
    int* d[4];  
    for (int i=0; i<4; i++)  
        d[i] = new int[3];  
}
```

注意区分：

- ◆ `int (*VAR)[SIZE]`；定义了一个指针变量，名为VAR，指向大小为SIZE的整数数组。
- ◆ `int *VAR[SIZE]`；定义了一个数组变量，名为VAR，数组大小为SIZE，数组元素类型为“指向整型变量的指针”

静态分配的空间



动态分配的空间

二维动态数组

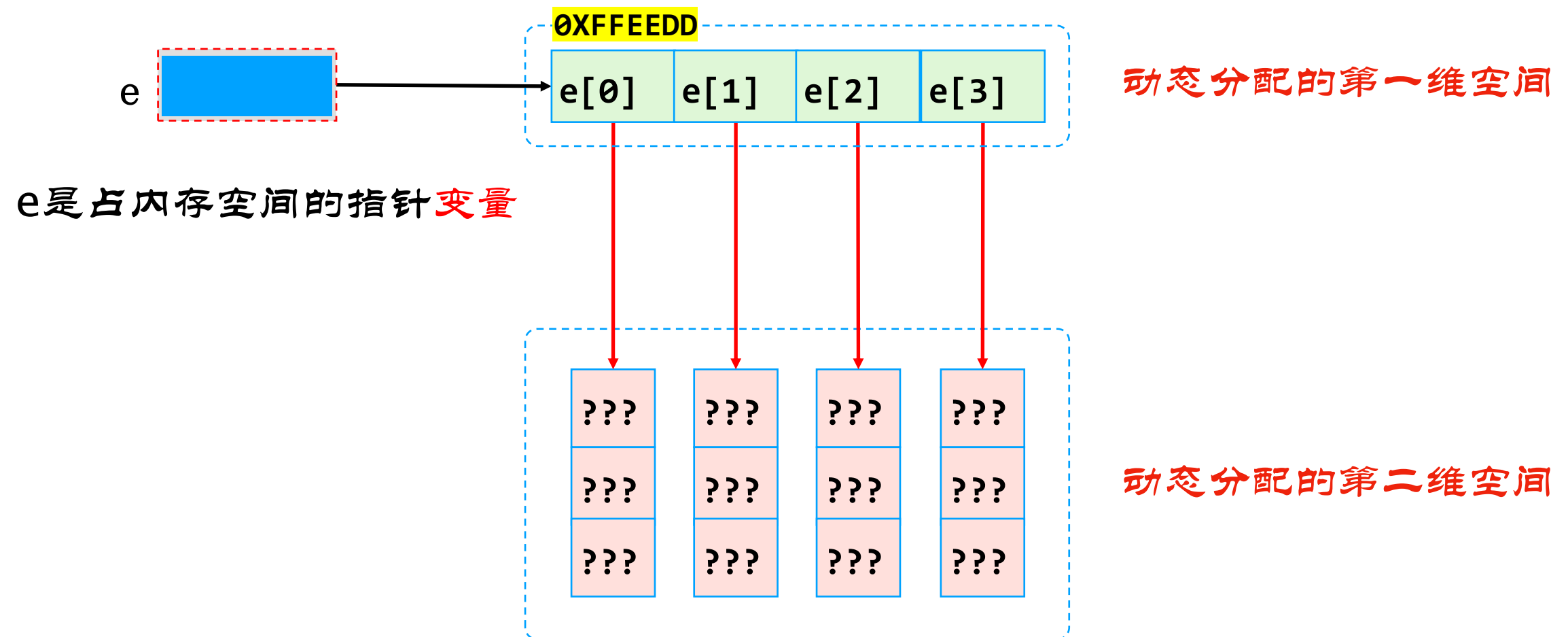
二维数组（设为 $m*n$ ）的动态分配过程，是分两步来完成的：

- ① 分配第一维的空间，即由 m 个指针组成的动态数组。
- ② 分别为 m 个指针分配一维动态数组空间，每个数组都包含 n 个元素。这一步与前面所讲的“一维数组的动态内存分配”是一样的。

示例：

```
int **e; // 注意：必须用两个星号定义“指向指针的指针”
e = new int*[4]; // STEP 1: 创建元素为指针的一维动态数组
for (int i=0; i<4; i++)
    e[i] = new int[3]; // STEP 2: 为每个元素创建一维动态数组
```

二维动态数组



`int **e;` // 注意：必须用两个星号定义“指向指针的指针”

`e = new int*[4];` // STEP 1: 创建元素为指针的一维动态数组

`for (int i=0; i<4; i++)`

`e[i] = new int[3];` // STEP 2: 为每个元素创建一维动态数组

安全释放二维动态数组的内存

二维数组内存的动态释放过程，实际上是分配过程的严格逆过程：

- ① 先通过逐一枚举第一维各元素，将该维中的各元素（即第二维的一维动态数组）指向的内存单元释放掉。
- ② 然后再释放第一维占用的内存空间。

示例：

```
for (int i=0; i<4; i++)  
    delete[] e[i];  
delete[] e;
```

二维动态数组的使用

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

void test(int** a, int len1, int len2)
{
    for (int i=0; i<len1; i++)
    {
        for (int j=0; j<len2; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
}

int main()
{
    int** e;
    e = new int*[4];
    for (int i=0; i<4; i++)
        e[i] = new int[3];
```

```
        for (int i=0; i<4; i++)
            for (int j=0; j<3; j++)
                e[i][j] = 10 * i + j;

    test(e, 4, 3); // 使用二维动态数组（通过函数）

    for (int i=0; i<4; i++) delete[] e[i];
    delete[] e;
    return 0;
}
```

二维动态数组的
定义、创建、赋值、删除

【任务6.4】运行时修改排序的准则

在一般的排序任务中，排序规则（关于大小关系的判定规则）是事先规定好的，先后次序（从大到小，还是从小到大）也是在程序运行前就事先指定好了的。如果希望在程序运行时，再指定是从高到低还是从低到高的排列次序，应该如何做呢？

比如：以整数数组排序为例，程序询问用户排序准则，**从小到大时，输入1；从大到小时，输入2**。程序根据用户的输入，输出指定的排序结果。

请你实现这样的要求。

最平凡的做法（“没有科技含量”）

```
#include <iostream>
using namespace std;
void bubble1(int data[], int num) {
    for (int i=0; i<num-1; i++)
        for (int j=0; j<num-1-i; j++)
            if (data[j] > data[j+1]) {
                int tmp = data[j]; data[j] = data[j+1]; data[j+1] = tmp;
            }
}

void bubble2(int data[], int num) { /// 拷贝粘贴，稍作修改
    for (int i=0; i<num-1; i++)
        for (int j=0; j<num-1-i; j++)
            if (data[j] < data[j+1]) {
                int tmp = data[j]; data[j] = data[j+1]; data[j+1] = tmp;
            }
}
```

最平凡的做法（“没有科技含量”）

```
int main() {  
    int data[] = {8, 4, 7, 9, 3, 5, 1};  
    int ans = 0;  
    cin >> ans;  
    if (ans == 1) bubble1(data, 7);  
    if (ans == 2) bubble2(data, 7);  
  
    for (int i=0; i<7; i++)  
        cout << data[i] << ' ';  
    cout << endl;  
    return 0;  
}
```

分别调用不同的排序函数

有点“科技含量”的方法：函数指针

- 定义函数指针变量的语法：

返回类型 (*指针变量名)(函数形参);

- 定义函数指针数组变量的语法：

返回类型 (*指针变量名[数组大小])(函数形参);

- 用函数指针进行函数调用的语法：

指针变量名(函数实参);

有点“科技含量”的方法：函数指针

```
int main() {  
    /// 使用指向函数的指针，函数指针数组（也称为“函数跳转表”）  
    void (*pf[2])(int*, int) = {bubble1, bubble2};  
    /// <<<--- 这可能是C/C++语言中最令人困惑的语法形式了！  
    int data[] = {8, 4, 7, 9, 3, 5, 1};  
    int ans = 0;  
    cin >> ans;  
    pf[ans-1](data, 7);  
  
    for (int i=0; i<7; i++) cout << data[i] << ' ';  
    cout << endl;  
    return 0;  
}
```

- 定义函数指针变量的语法：
返回类型 *指针变量名(函数形参);
- 定义函数指针数组变量的语法：
返回类型 (*指针变量名[数组大小])(函数形参);
- 用函数指针进行函数调用的语法：

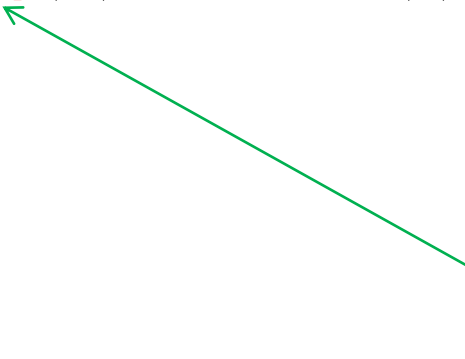
指针变量名（函数实参）;

更有“科技含量”的方法：在函数参数中使用函数指针

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

```
void bubble(int data[], int num, bool (*op)(int, int)) {
    for (int i=0; i<num-1; i++)
        for (int j=0; j<num-1-i; j++)
            if (op(data[j], data[j+1])) {
                int tmp = data[j];
                data[j] = data[j+1];
                data[j+1] = tmp;
            }
}
```

函数指针作为形式
参数时的语法规则



```
bool op1(int a, int b) { return a > b; }
bool op2(int a, int b) { return a < b; }
```


更有“科技含量”的方法：用“函数跳转表”简化代码

```
int main() {  
    bool (*pf[2])(int, int) = {op1, op2}; /// 称pf数组为“函数跳转表”  
}
```

```
int data[] = {8, 4, 7, 9, 3, 5, 1};
```

```
int ans = 0;
```

```
cin >> ans;
```

```
bubble(data, 7, pf[ans-1]);
```

```
for (int i=0; i<7; i++)
```

```
    cout << data[i] << ' ';
```

```
cout << endl;
```

```
return 0;
```

```
}
```

```
if (ans == 1) bubble1(data, 7);  
if (ans == 2) bubble2(data, 7);
```

使用变量，再结合数组、函数指针，使得对算法操作的表达更抽象、更通用，实际上是“**将函数名变量化**”了。

请认真领悟其中的编程思想。

【任务6.5】住宅销售的人员排队系统

根据《XX大学出售新建职工住宅实施办法》的规定，特批人员按下述原则进行排队：

一、离退休特批人员排序原则

- 1、按照票数多少排序；
- 2、票数相同的按职务等级排序；
- 3、职务相同的按任职时间排序；
- 4、任职时间相同的，按出生年月日排序。

二、引进人才特批人员排序原则

- 1、按照票数多少排序；
- 2、票数相同的按职务等级排序；
- 3、职务相同的按来校时间排序；
- 4、来校时间相同按任职时间排序；
- 5、任职时间相同的，按出生年月日排序。

【任务6.5】住宅销售的人员排队系统

姓名	票数	职务等级	来校时间	任职时间	出生时间
Zhang San	48	1	2008	2006	1967
Li Si	50	2	2009	2008	1970
Wang Wu	48	1	2008	2006	1966
Zhao Liu	50	2	2008	2008	1968
Qian Jiu	35	3	2011	2011	1972

请你编写程序，完成上述人员的排队（序）。

要完成这个任务，最方便的办法使用C++语言的自定义类型——“结构体”类型，可以简称为“结构”类型。我们将先介绍关于“结构体”的相关语法知识，再编写程序完成本任务。

用户自定义类型——结构体

定义结构体类型的语法格式：

struct 结构体名

{

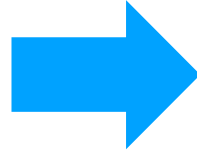
类型名 1 成员名 1;

类型名 2 成员名 2;

. . .

类型名 n 成员名 n;

} ;



```
struct person
```

```
{
```

```
    char xingming[20];
```

```
    int piaoshu;
```

```
    int zhiwu_dengji;
```

```
    int laixiao_shijian;
```

```
    int renzhi_shijian;
```

```
    int chusheng_shijian;
```

```
};
```

struct 是结构体类型的标志，**结构体名**是编程者自己选定的。大括号括起来的语句定义了结构体的**数据成员**，结构体成员和整个结构体类型定义之后有一分号。

结构体中同类型成员的两种定义方法

```
struct person
{
    char xingming[20];
    int piaoshu;
    int zhiwu_dengji;
    int laixiao_shijian;
    int renzhi_shijian;
    int chusheng_shijian;
};
```

```
struct person
{
    char xingming[20];
    int piaoshu,
        zhiwu_dengji,
        laixiao_shijian,
        renzhi_shijian,
        chusheng_shijian;
};
```

可以合并同一类型
的数据成员

结构体变量数据成员的访问方法

- **操作符**：用于对结构体变量my的成员name的引用。

如：

```
person my;
```

```
cin >> my.name ;
```

一般读作：“输出结构体变量 my 的 name 成员”。

- >**操作符**：用于结构体指针对成员的引用，如：

```
person* pmy = &my;
```

```
cin >> pmy->name;
```

结构体变量数据成员的访问方法

```
person p;    // 定义一个名为p的结构体类型变量

cout << "请输入引进人才的信息" << endl;
cout << "姓名: "; cin >> p.xingming;
cout << "票数: "; cin >> p.piaoshu;
cout << "职务等级: ";    cin >> p.zhiwu_dengji;
cout << "来校时间: ";    cin >> p.laixiao_shijian;
cout << "任职时间: ";    cin >> p.renzhi_shijian;
cout << "出生时间: ";    cin >> p.chusheng_shijian;
```

结构体变量数据成员的访问方法

```
cout << "你输入的引进人才的信息是：" << endl;  
cout << p.xingming << ": "  
    << p.piaoshu << ", "  
    << p.zhiwu_dengji << ", "  
    << p.laixiao_shijian << ", "  
    << p.renzhi_shijian << ", "  
    << p.chusheng_shijian << endl;
```


结构体类型变量的定义和初始化

方法一： 在定义结构体类型时，定义和初始化结构体变量。例如：

```
struct person {  
    char name[10];  
    unsigned long birthday;  
    char placeofbirth[20];  
} per = {"Li ming", 19821209, "Beijing"};
```

方法二： 定义结构体类型后，再进行结构体变量的定义（和初始化）。例如：

```
struct person {  
    char name[10];  
    unsigned long birthday;  
    char piaceofbirth[20];  
};  
  
person per = {"Li ming", 19821209, "Beijing"};
```

结构体数组的定义与初始化

```
struct student {  
    char name[20];  
    char sex;  
    long int birthday;  
    float height;  
    float weight;  
};
```

结构体数组的元素初始化方法，与普通结构体变量的初始化相同：等号后面是用花括号括起来的若干个初始值（即结构体变量值，该值是用花括号括起来的）组成的序列，数值之间用逗号隔开。

```
student Room[4] = {  
    { "Li li", 'M', 19840318, 1.82, 65.0 },  
    { "Mi mi", 'M', 19830918, 1.75, 58.0 },  
    { "He lei", 'M', 19841209, 1.83, 67.1 },  
    { "Ge li", 'M', 19840101, 1.70, 59.0 } };
```

结构体变量的
初始化值

括结构体变
量初始化值
的花括号

括数组元素初始
化值的花括号

结构体与函数：结构体可作为函数参数和函数返回值

```
#include <iostream> // cout
#include <cstring> // strcpy
using namespace std;

struct package
{
    char msg[4];
    int v1, v2;
};

package transfer(package data)
{
    cout << "&data= " << &data << ", value = "
        << data.msg << ' ' << data.v1 << ' ' << data.v2 << endl;
    package tmp;
    strcpy(tmp.msg, data.msg);
    tmp.v1 = data.v1 + 1;
    tmp.v2 = data.v2 + 2;
    cout << "&tmp = " << &tmp << ", value = "
        << tmp.msg << ' ' << tmp.v1 << ' ' << tmp.v2 << endl;
    return tmp;
}
```

```
int main()
{
    package pkg = {"ABC"}, res;
    cout << "&pkg = " << &pkg << ", value = "
        << pkg.msg << ' ' << pkg.v1 << ' ' << pkg.v2 << endl;

    res = transfer(pkg);
    cout << "&res = " << &res << ", value = "
        << res.msg << ' ' << res.v1 << ' ' << res.v2 << endl;
    return 0;
}
```

程序运行输出结果：

终端

```
&pkg = 0x62fe14, value = ABC 0 0
&data= 0x62fde0, value = ABC 0 0
&tmp = 0x62fdf0, value = ABC 1 2
&res = 0x62fe08, value = ABC 1 2
```

结构体与函数：结构体可作为函数参数和函数返回值

```
package transfer(package data)
{
    cout << "&data= " << &data << ", value = "
        << data.msg << ' ' << data.v1 << ' ' << data.v2 << endl;

    res = transfer(pkg);
    cout << "&res = " << &res << ", value = "
        << res.msg << ' ' << res.v1 << ' ' << res.v2 << endl;
}
```

终端

程序运行输出结果

```
&pkg = 0x62fe14, value = ABC 0 0
&data= 0x62fde0, value = ABC 0 0
&tmp = 0x62fdf0, value = ABC 1 2
&res = 0x62fe08, value = ABC 1 2
```

- ◆ 结构体变量作为函数参数时，传入形参中的只是实参的内容，形参与实参存储在内存的不同单元中，它们互相不影响。
- ◆ 多个变量的值可以通过让函数返回结构体变量而传递给调用者，即可以视为函数可以一次性返回多个变量的值 —— 只要将它们“打包”到一个结构体变量中即可。

如何按规则输出人员排序结果？

姓名	票数	职务等级	来校时间	任职时间	出生时间
Zhang San	48	1	2008	2006	1967
Li Si	50	2	2009	2008	1970
Wang Wu	48	1	2008	2006	1966
Zhao Liu	50	2	2008	2008	1968
Qian Jiu	35	3	2011	2011	1972

可以参照整数数组的排序示例来设计程序算法。

算法实现

```
#include <iostream>
using namespace std;
struct person {
    char xingming[20];
    int piaoshu;
    int zhiwu_dengji;
    int laixiao_shijian;
    int renzhi_shijian;
    int chusheng_shijian;
};
```

```
////// ..... NEXT PAGE
```

```
int main() {
    person array[5] = { {"zhangsang", 48, 1, 2008, 2006, 1967},
                        {"lisi", 50, 2, 2009, 2008, 1970},
                        {"wangwu", 48, 1, 2008, 2006, 1966},
                        {"zhaoliu", 50, 2, 2008, 2008, 1968},
                        {"qianjiu", 35, 3, 2011, 2011, 1972} };

    bubble(array, 5);
    for (int i=0; i<5; i++) output(array[i]);
    return 0;
}
```

排版整齐，有利于防止输入错误，也方便查找BUG



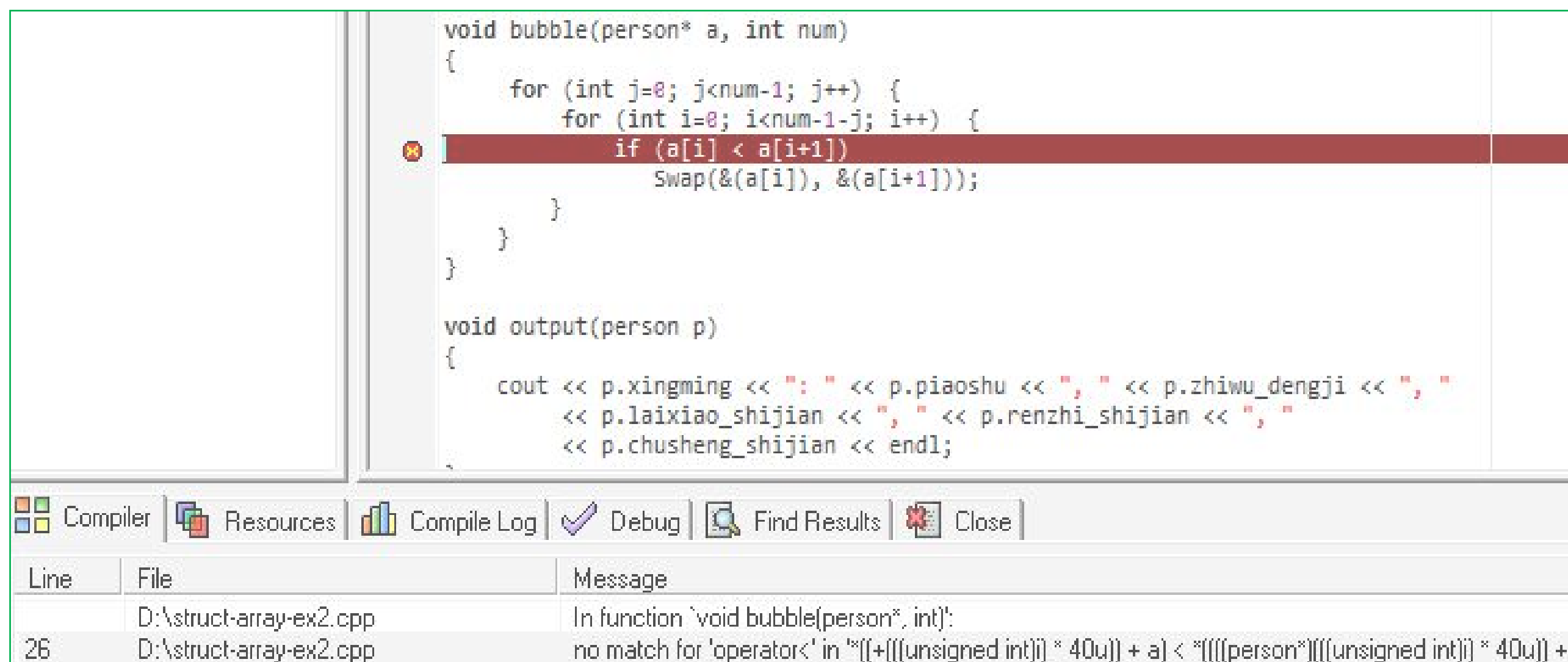
```
void output(person p) {  
    cout << p.xingming << ": " << p.piaoshu << ", "  
        << p.zhiwu_dengji << ", " << p.laixiao_shijian << ", "  
        << p.renzhi_shijian << ", " << p.chusheng_shijian << endl;  
}
```

```
void Swap(person* first, person* second) {  
    person p;  
    p = *first;  
    *first = *second;  
    *second = p;  
}
```

函数Swap的形式参数为
何要用指针类型？

```
void bubble(person* a, int num) {  
    for (int j=0; j<num-1; j++) {  
        for (int i=0; i<num-1-j; i++) {  
            if (a[i] < a[i+1])  
                Swap(&a[i], &a[i+1]);  
        }  
    }  
}
```

编译错误：结构体变量无法直接比较大小



```
void bubble(person* a, int num)
{
    for (int j=0; j<num-1; j++) {
        for (int i=0; i<num-1-j; i++) {
            if (a[i] < a[i+1])
                Swap(&a[i], &a[i+1]));
        }
    }
}

void output(person p)
{
    cout << p.xingming << ": " << p.piaoshu << ", " << p.zhiwu_dengji << ", "
    << p.laixiao_shijian << ", " << p.renzhi_shijian << ", "
    << p.chusheng_shijian << endl;
}
```

Compiler | Resources | Compile Log | Debug | Find Results | Close

Line	File	Message
26	D:\struct-array-ex2.cpp	In function 'void bubble(person*, int)': no match for 'operator<' in '[[+(((unsigned int)j) * 40u)) + a] < *[[[(person*)]((unsigned int)j) * 40u]] + ...'


```
void bubble(person* a, int num) {  
    for (int j=0; j<num-1; j++) {  
        for (int i=0; i<num-1-j; i++) {  
            if (person_cmp(a[i], a[i+1]))  
                Swap(&(a[i]), &(a[i+1]));  
        }  
    }  
}
```

算法实现

```
bool person_cmp(person a, person b) {  
    if (a.piaoshu < b.piaoshu) return true;  
    if (a.piaoshu > b.piaoshu) return false;  
  
    if (a.zhiwu_dengji < b.zhiwu_dengji) return true;  
    if (a.zhiwu_dengji > b.zhiwu_dengji) return false;  
  
    if (a.laixiao_shijian > b.laixiao_shijian) return true;  
    if (a.laixiao_shijian < b.laixiao_shijian) return false;  
  
    if (a.renzhi_shijian > b.renzhi_shijian) return true;  
    if (a.renzhi_shijian < b.renzhi_shijian) return false;  
  
    if (a.chusheng_shijian > b.chusheng_shijian) return true;  
    if (a.chusheng_shijian < b.chusheng_shijian) return false;  
  
    return true;  
}
```

代码重构

使用STL中的排序算法: `sort(head, tail, func);`

```
#include <iostream> // cout
```

```
#include <algorithm> // sort
```

```
using namespace std;
```

```
struct person { ... };
```

```
void output(person p) { ... }
```

```
bool person_cmp(person a, person b) { ... }
```

```
int main()
```

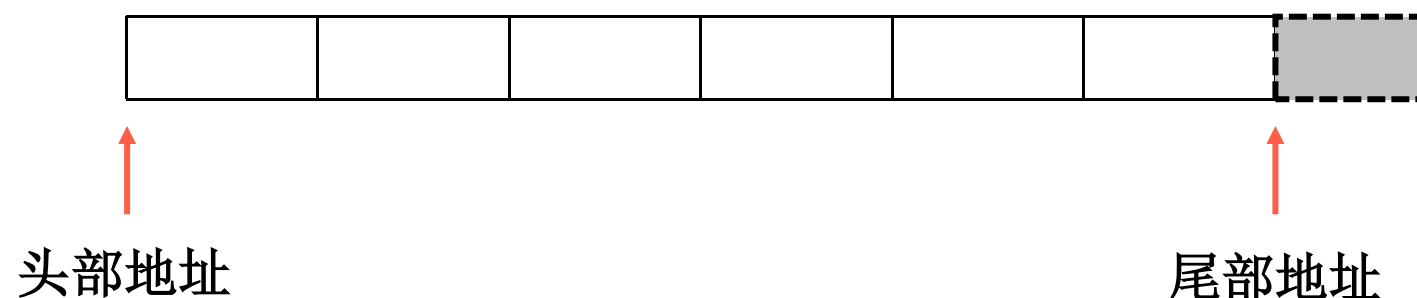
```
{  
    person array[5] = {{"zhangsan", 48, 1, 2008, 2006, 1967},  
                        {"lisi", 50, 2, 2009, 2008, 1970},  
                        {"wangwu", 48, 1, 2008, 2006, 1966},  
                        {"zhaoliu", 50, 2, 2008, 2008, 1968},  
                        {"qianjiu", 35, 3, 2011, 2011, 1972}};
```

```
    sort(array, array + 5, person_cmp);
```

```
    for (int i = 0; i < 5; i++) output(array[i]);
```

```
    return 0;
```

```
}
```



参数1: 数组“头部”的地址

参数2: 数组“尾部”的地址

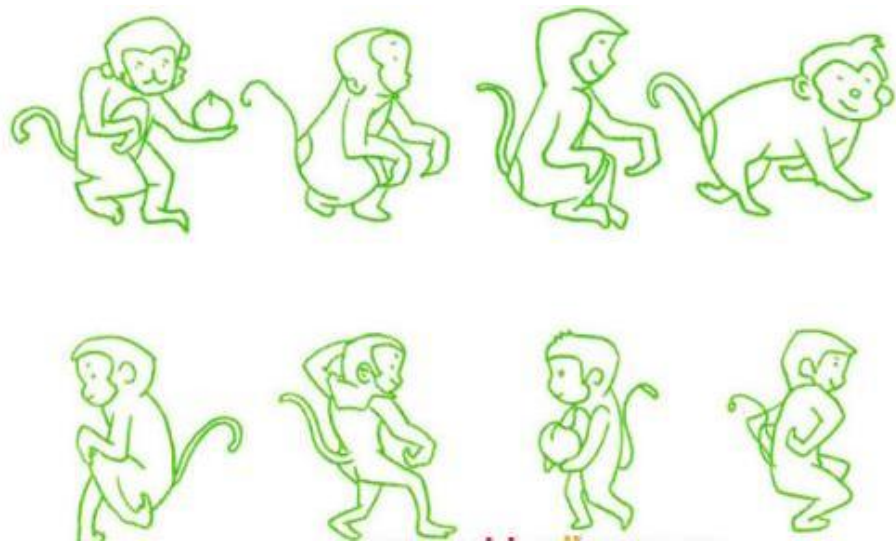
参数3: 比较元素大小的函数

【任务6.6】猴子选大王（环形数组）

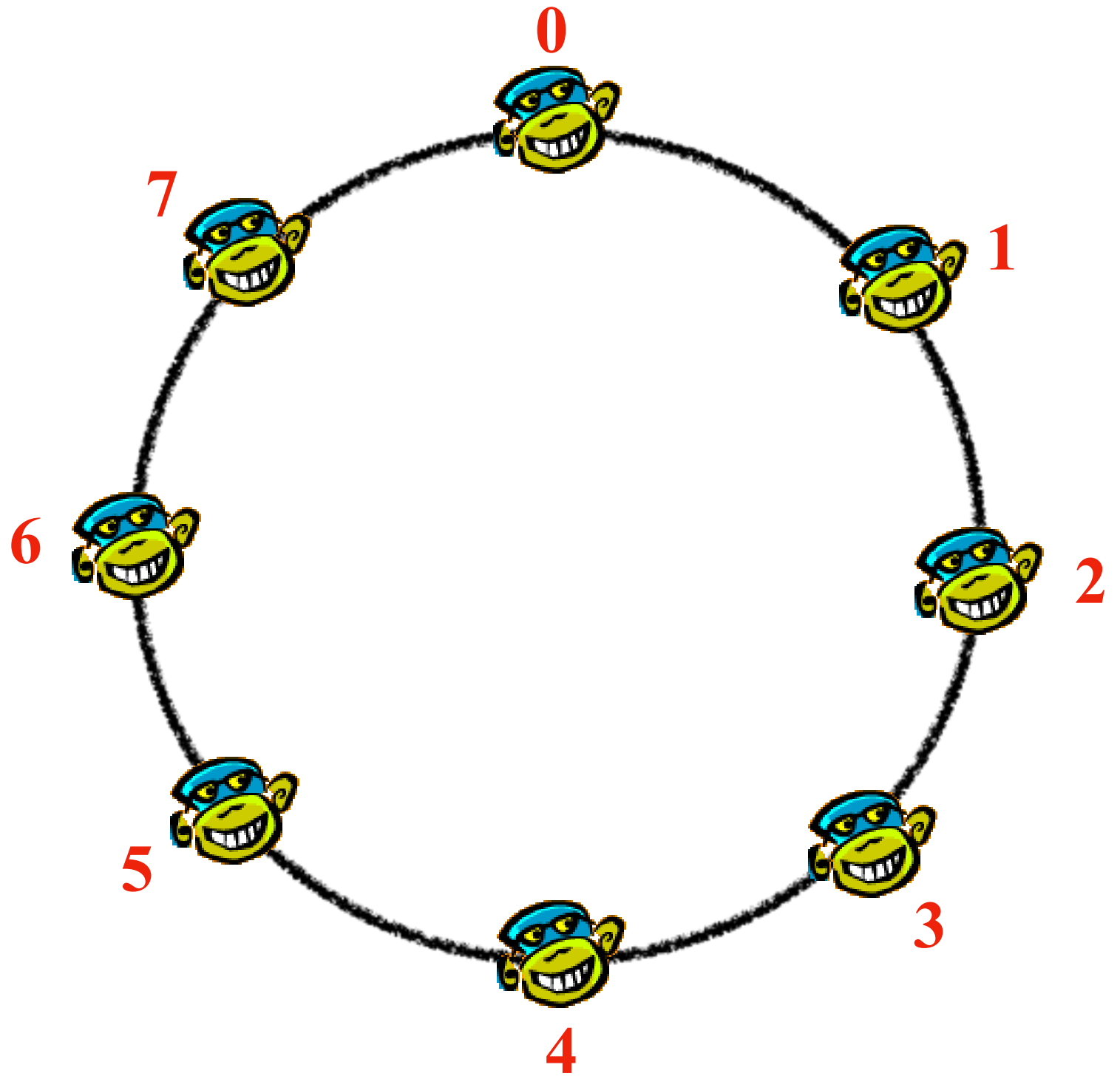
n 只猴子围成一圈，顺时针方向从 0 到 $n-1$ 编号。之后从 0 号开始沿顺时针方向让猴子按 1, 2, \dots , m 依次报数。凡报到 m 的猴子，都让其出圈，取消其候选资格。不停地按顺时针方向逐一让报出 m 的猴子出圈，最后剩下那只就是猴王。

请你编写程序模拟这个过程。要求按取消候选资格的次序，依次输出被淘汰猴子的编号，以及最后剩下的猴王编号。

【解题技巧】观察实例（以 $n=8$, $m=3$ 为例）



猴王



猴子被淘汰的顺序

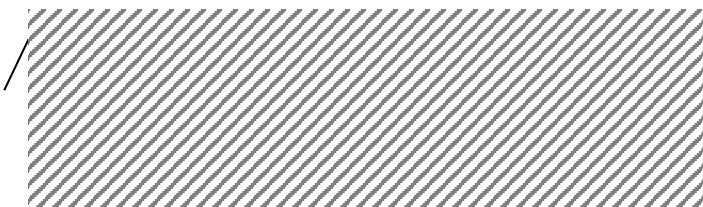
2 5 0 4 1 7 3

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

int main()
{
    int n, m;
    cout << "请输入猴子总数: ";
    cin >> n;
    cout << "请输入报数间隔: ";
    cin >> m;

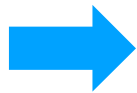
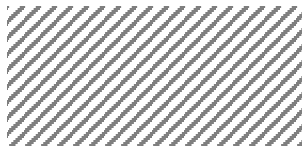
    // 根据猴子总数创建相应长度的向量, 各元素值均为0
    vector<int> ring(n);
```

```
int idx = 0;

for (int i = 0; i < n; i++)
{
    
    ring[idx] = 1;          // 标记报数为m的猴子
    cout << idx << ' '; // 输出报数为m的猴子位置（从0编号）
}

cout << "---> 猴王位置为 " << idx << endl;

return 0;
}
```



```
int cnt = 0;
while (1)
{
    cnt++;
    // 1. 寻找能够报数cnt的位置
    
    // 2. 若报数为m，则停止报数
    
    // 3. 当前位置已安全报数，转下一位置
    
}
```


本讲知识点小结

- ✓ 字符串的基本操作
- ✓ 用STL的string存储和处理字符串
- ✓ 用STL的sort算法对数组和向量进行排序
- ✓ 二维数组（定义、初始化）
- ✓ 矩阵乘法
- ✓ 字符串数组排序
- ✓ 动态数组（一维、二维）、指针数组
- ✓ 函数指针、函数跳转表
- ✓ 结构、结构数组、结构与函数结合使用
- ✓ 结构数组排序
- ✓ 环形数组

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

int main()
{
    string msg = "END. See you later!";
    cout << << endl;
    return 0;
}
```