



第五章 程序设计复合类型

模块5.1：数组

主讲教师：同济大学计算机科学与技术学院 陈宇飞
同济大学计算机科学与技术学院 龚晓亮



目录

- 一维数组
- 二维数组
- 巧用智能编程提示



目录

- 一维数组

1. 基本概念
2. 一维数组的特点
3. 一维数组的初始化规则
4. 一维数组的基本使用
5. 一维数组的应用-排序算法
6. 基于范围的for循环



1.1 基本概念

- ✓ 数组（array）是一种数据格式，能够存储多个同类型的值
- ✓ 每个值都存储在一个独立的数组元素中，计算机在内存中连续存储数组的各个元素
- ✓ 数组声明应指出：
 - ❖ 存储在每个元素中的值的类型
 - ❖ 数组名
 - ❖ 数组中的元素数量



1.1 基本概念

✓声明数组的通用格式:

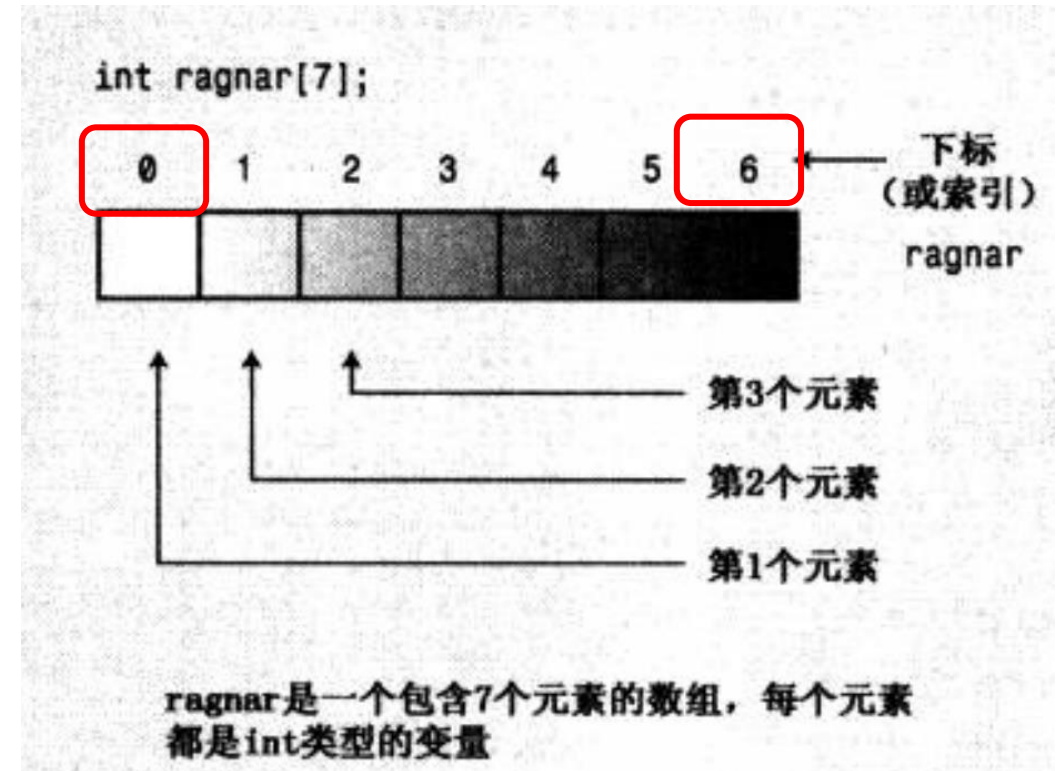
```
typeName arrayName[arraySize]
```

✓arraySize是指定元素数目，它必须是整型常数或const值，也可以是常量表达式如sizeof(int)，即其中所有的值再编译时都是已知的。（后续课程会介绍使用new运算符来避开这种限制）

```
short months[12]; // creates array of 12 short
```

1.2 一维数组的特点

- ✓数组的特点：可以单独访问数组元素
- ✓方式是使用下标或索引来对元素进行编号。C++数组从0开始编号
- ✓使用带索引的方括号表示指定数组元素，months[0]，months[11]



创建数组



1.2 一维数组的特点

- ✓数组声明能够使用一个声明创建大量的变量，然后使用索引来表示和访问各个元素
- ✓**有效下标值**非常重要。编译器不会检查使用的下标是否有效（编译器不会报错）。但运行之后，这种错误赋值可能引发问题，它可能破坏数据或代码，也可能导致程序终止



```
// arrayone.cpp -- small arrays of integers
```

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    using namespace std;
```

```
    int yams[3];    // creates array with three elements
```

```
    yams[0] = 7;    // assign value to first element
```

```
    yams[1] = 8;
```

```
    yams[2] = 6;
```

```
    int yamcosts[3] = { 20, 30, 5 }; // create, initialize array
```

```
    cout << "Total yams = ";
```

```
    cout << yams[0] + yams[1] + yams[2] << endl;
```

```
    cout << "The package with " << yams[1] << " yams costs ";
```

```
    cout << yamcosts[1] << " cents per yam.\n";
```

```
    int total = yams[0] * yamcosts[0] + yams[1] * yamcosts[1];
```

```
    total = total + yams[2] * yamcosts[2];
```

```
    cout << "The total yam expense is " << total << " cents.\n";
```

```
    cout << "\nSize of yams array = " << sizeof yams;
```

```
    cout << " bytes.\n";
```

```
    cout << "Size of one element = " << sizeof yams[0];
```

```
    cout << " bytes.\n";
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 调试 × + ▾

Total yams = 21

The package with 8 yams costs 30 cents per yam.

The total yam expense is 410 cents.

Size of yams array = 12 bytes.

Size of one element = 4 bytes.



1.3 一维数组的初始化规则

- ✓只有在定义数组时才能使用初始化，此后就不能使用了，也不能将一个数组赋值给另一个数组

```
int cards[4]={ 3,6,8,10 }; // creates array with four elements, ok
int hand[4];               // creates array with four elements, ok
hand[4]={5,6,7,8};         // assign value to first element, error
hand = cards;              // assign one array to another, error
```

- ✓可以使用下标分别给数组中的元素赋值

```
int yams[3];               // creates array with three elements
yams[0] = 7;               // assign value to first element
yams[1] = 8;
yams[2] = 6;
```



1.3 一维数组的初始化规则

✓如果只对数组的一部分进行初始化，则编译器将把其他元素设置为0

```
float hotelTips[5] = {5.0, 2.5}; // creates, initializes array, ok
```

✓要将数组中所有的元素都初始化为0，只需要显示地将第一个元素初始化0

```
long totals[500] = { 0 }; // 所有元素都设置为0  
long totals[500] = { 1 }; // 第一个元素是1，其他所有元素都设置为0
```



1.3 一维数组的初始化规则

✓如果初始化数组时方括号内（[]）为空，C++编译器将计算元素个数

```
short things[] = {1, 5, 3, 6, 7}; // creates array with five  
                                  elements, ok
```

```
int num_elements = sizeof things / sizeof things[0];  
// determines number of elements in things array, ok
```



1.3 一维数组的初始化规则

- ✓ 数组的声明和初始化使用**一条语句**（推荐）

```
short things[] = {1, 5, 3, 6, 7};
```

- ✓ 数组的声明和初始值赋值使用**分开的两条语句**

```
int grade[5], i;  
for (i = 0; i < 5; i++)  
    grade[i] = 10 - i;
```



1.3 一维数组的初始化规则

✓ 使用大括号对数组进行初始化（初始化列表）的规则

❖ 初始化数组时，可省略等号（=）

```
double earnings[4] = {1.2e4, 1.6e4, 1.1e4, 1.7e4};  
double earnings[4] {1.2e4, 1.6e4, 1.1e4}; // okay
```

❖ 可不在大括号内包含任何东西，将默认将所有元素都设置为0

```
unsigned int counts[10] = { }; // ten elements set to 0  
float balances[100] = { }; // 100 elements set to 0
```



1.3 一维数组的初始化规则

✓ 使用大括号对数组进行初始化（初始化列表）的规则

❖ 列表初始化禁止缩窄转换

```
long plifs[] = {25, 92, 3.0};
```

// 将浮点数转换为整型是缩窄操作，即使浮点数的小数点后面为0

```
char slifs[4] {'h', 'i', 1122011, '\0'};
```

// 1122011超过了char变量的取值范围（这里假设char变量的长度为8位）

```
char tlifs[4] = {'h', 'i', 112, '\0'};
```

// 虽然112是一个int值，但它在char变量的取值范围，allowed

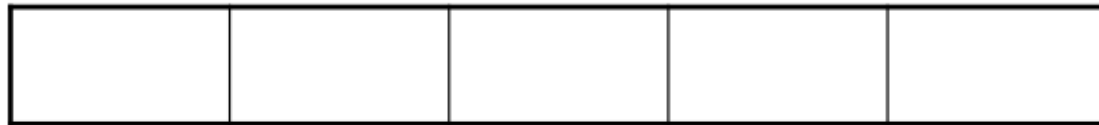


1.4 一维数组的基本使用

✓ 数组的存储

数组在程序运行时，存放在一组**连续的地址单元**内。**数组名**是常量，表示数组在内存中的**首地址**

例：int s[5]; s



s是数组名，也表示数组在内存中的首地址



1.4 一维数组的基本使用

✓ 数组的下标（索引）

数组对于其所含元素进行编号，以便程序对数组汇总的元素进行访问。在C/C++中，数组的元素编号**从0开始**，并依次递增，数组中元素的编号成为下标或索引

例：int s[5];

s	s[0]	s[1]	s[2]	s[3]	s[4]
---	------	------	------	------	------

❖数组的大小为arraySize时，其合理的下标范围是0到arraySize-1



1.4 一维数组的基本使用

- ✓ 数组长度的计算使用运算符 `sizeof`，`sizeof` 在输入数组名作为参数时，会返回整个数组的字节数

```
cout << "\nSize of yams array = " << sizeof yams;  
cout << " bytes.\n";  
cout << "Size of one element = " << sizeof yams[0];  
cout << " bytes.\n";
```

Size of yams array = 12 bytes.
Size of one element = 4 bytes.

❖ 数组的长度为：

整个数组的字节数 / 数组中元素的数据类型所占的字节数

```
int num_elements = sizeof things / sizeof things[0];  
// determines number of elements in things array
```



1.4 一维数组的基本使用

- ✓ 对于一维数组中的元素，使用数组下标引用它们，相当于使用一个变量
- ✓ 对于一维数组中的元素，还可以使用指针引用它们（后续课程介绍）
- ✓ 数组的元素做为变量被函数使用，此时参数调用的方式为**按值传递**（后续课程介绍）



```
// arrayone.cpp -- small arrays of integers
#include <iostream>
int main()
{
```

```
    using namespace std;
    int yams[3];    // creates array with three elements
    yams[0] = 7;    // assign value to first element
    yams[1] = 8;
    yams[2] = 6;
```

```
    int yamcosts[3] = { 20, 30, 5 }; // create, initialize array
```

```
    cout << "Total yams = ";
    cout << yams[0] + yams[1] + yams[2] << endl;
    cout << "The package with " << yams[1] << " yams costs ";
    cout << yamcosts[1] << " cents per yam. \n";
    int total = yams[0] * yamcosts[0] + yams[1] * yamcosts[1];
    total = total + yams[2] * yamcosts[2];
    cout << "The total yam expense is " << total << " cents. \n";
```

```
    cout << "\nSize of yams array = " << sizeof yams;
    cout << " bytes. \n";
    cout << "Size of one element = " << sizeof yams[0];
    cout << " bytes. \n";
    return 0;
}
```

Microsoft Visual Studio 调试

```
Total yams = 21
The package with 8 yams costs 30 cents per yam.
The total yam expense is 410 cents.
```

```
Size of yams array = 12 bytes.
Size of one element = 4 bytes.
```

数组的元素作为变量被输出

数组的元素作为变量被表达式使用



1.4 一维数组的基本使用

✓数组遍历：指对数组中连续的几个元素或是全部元素进行某项操作，是对数组的常见操作之一

```
#include <iostream>
using namespace std;
int main()
{
    const int n = 10;
    int arr[n]{};
    for (int i = 0; i < sizeof(arr) / sizeof(int); i++)
    {
        arr[i] = i;
        cout << arr[i] << endl;
    }
    return 0;
}
```

A vertical terminal window showing the output of the C++ program. It displays the numbers 0 through 9, each on a new line, representing the elements of the array. The window has a dark background and a light border.

0
1
2
3
4
5
6
7
8
9



1.4 一维数组的基本使用

✓对于一维数组，在使用**数组名**引用数组时，数组名实际表示数组**存储位置的起始地址**。因此，数组名**不可以用于赋值或是表达式计算**

```
int cards[4]={ 3,6,8,10 }; // creates array with four elements, ok
int hand[4];               // creates array with four elements, ok
hand = cards;              // assign one array to another, error
```



1.4 一维数组的基本使用

- ✓ 数组的复制、拆分和合并：在需要完成数组的“复制”或“拆分”逻辑时，应该将数组的元素依次赋值给另一个数组

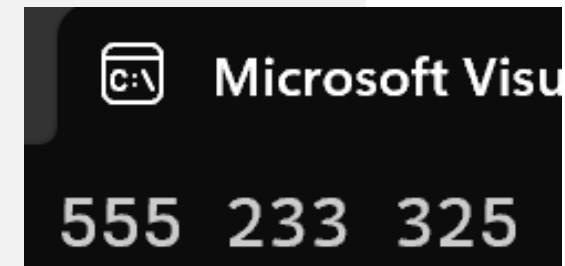
```
#include <iostream>
using namespace std;
int main()
{
    int arr[5] = { 555, 233, 325, 513, 466 };
    int arr2[5] = {   };
    for (int i = 0; i < 5; i++)
    {
        arr2[i] = arr[i]; //数组元素复制
        cout << arr2[i] << endl;
    }
    return 0;
}
```






1.4 一维数组的基本使用

```
#include <iostream>
using namespace std;
int main()
{
    int arr[5] = { 555, 233, 325, 513, 466 };
    int arr_2[3]{};
    for (int i = 0; i < 5; i++)
    {
        if (i < 3 )
            arr_2[i] = arr[i]; //数组拆分
    }
    for(int i = 0; i < 3; i++)
        cout << arr_2[i] << " ";
    return 0;
}
```





```
// not.cpp -- using the not operator
#include <iostream>
using namespace std;
int main()
{
    int grade_1[5] = { 1, 2, 3, 4, 5 };
    int grade_2[5] = { 6, 7, 8, 9, 10 };
    int grade_3[10] = {  };
    int i;
    for (i = 0; i < 10; i++)
    {
        if (i < 5)
            grade_3[i] = grade_1[i]; //数组合并
        else
            grade_3[i] = grade_2[i - 5];
    }
    for (i = 0; i < 10; i++)
    {
        cout << grade_3[i] << endl;
    }
    return 0;
}
```

 Micro
1
2
3
4
5
6
7
8
9
10



1.5 一维数组的应用-排序算法

✓ 冒泡排序

- ❖ 比较相邻的元素，如果第一个比第二个大，就交换他们两个
- ❖ 对每一对相邻元素做同样的工作，执行完毕后找到第一个最大值
- ❖ 重复以上的步骤，每次比较次数-1，直到不需要比较

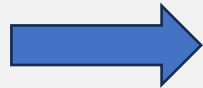


```
#include<iostream>
using namespace std;
int main() {
    int arr[10] = { };
    cout << "请输入10个整数: " << endl;
    for (int i = 0; i < 10; i++) //数组元素逐一输入
    {
        cin >> arr[i];
    }
    for (int i = 0; i < 10 - 1; i++)
    {
        for (int j = 0; j < 10 - 1 - i; j++) //两两相邻的元素比较并交换
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < 10; i++) //数组元素逐一输出
    {
        cout << arr[i] << endl;
    }
    return 0;
}
```

```
请输入10个整数:
3 7 29 23 12 82 1 72 8 5
1
3
5
7
8
12
23
29
72
82
```



```
for (int i = 0; i < 10 - 1; i++)  
{  
    for (int j = 0; j < 10 - 1 - i; j++)  
    {  
        if (arr[j] > arr[j + 1])  
        {  
            int temp = arr[j];  
            arr[j] = arr[j + 1];  
            arr[j + 1] = temp;  
        }  
    }  
}
```



```
#define N 10  
for (int i = 0; i < N - 1; i++)  
    for (int j = 0; j < N - 1 - i; j++)  
        if (arr[j] > arr[j + 1])  
        {  
            int temp = arr[j];  
            arr[j] = arr[j + 1];  
            arr[j + 1] = temp;  
        }
```

```

#define N 10
for (int i = 0; i < N - 1; i++)
    for (int j = 0; j < N - 1 - i; j++)
        if (arr[j] > arr[j + 1])
        {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }

```

设arr为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环i=0，内循环j=0~8

内循环	0	1	2	3	4	5	6	7	8	
0	3	-								3
1	7	-	-							7
2	29		-	23						23
3	23			29	12					12
4	12				29	-				29
5	82					-	1			1
6	1						82	72		72
7	72							82	8	8
8	8								82	5
9	5									82

第1次外循环后，最大数82在最后

```

#define N 10
for (int i = 0; i < N - 1; i++)
    for (int j = 0; j < N - 1 - i; j++)
        if (arr[j] > arr[j + 1])
        {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }

```

设arr为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第2次，外循环i=1，内循环j=0~7

内循环	0	1	2	3	4	5	6	7	8
0	3								
1	7								
2	23								
3	12								
4	29								
5	1								
6	72								
7	8								
8	5								
9	82								

第1次外循环后，最大数82在最后，
第2次外循环就不必再比较该数了



1.6 基于范围的for循环

- ✓ 简化了一种常见的循环任务：对数组的每个元素执行相同的操作

```
double prices[5] = {4.99, 10.99, 6.87, 7.99, 8.49};  
for (double x : prices)  
    cout << x << std::endl; // display all prices
```

- ✓ 要修改数组的元素，需要使用不同的循环变量语法：

```
double prices[5] = {4.99, 10.99, 6.87, 7.99, 8.49};  
for (double &x : prices) //符号&表明x是一个引用变量（后续课程具体介绍）  
    x = x * 1.06;        // add 6% sales tax
```



1.6 基于范围的for循环

- ✓ 还可以结合使用基于范围的for循环和初始化列表:

```
for (int x : {3, 5, 2, 8, 6} )  
    cout << x << std::endl;  
cout << ' \n' ;
```



目录

- 二维数组

1. 二维数组的基本特点
2. 二维数组的初始化规则
3. 二维数组的基本使用
4. 二维数组与一维数组
5. 二维数组的应用



2.1 二维数组的基本特点

- ✓ 一维数组：相当于excel表中的一行或一列

```
typeName    arrayName[arraySize]
```

数组元素类型 数组名 数组长度

- ✓ 二维数组：相当于包含多行多列的整张excel表

```
typeName    arrayName[rowSize][columnSize]
```

数组元素类型 数组名 数组长度

```
int maxtemps[4][5];
```

```
// creates 2D array with 4 rows and 5 columns, ok
```



2.1 二维数组的基本特点

- ✓ C/C++支持多维数组，其声明的通用格式

```
typeName    arrayName[Size1][Size2]... [SizeN]
```

数组元素类型	数组名	数组长度
--------	-----	------



2.1 二维数组的基本特点

✓ 二维数组的下标（索引）

二维数组的下标包括行下标和列下标两个部分，行下标和列下标都从0开始并依次递增

```
int maxtemps[4][5];
```

`int maxtemps[4][5];`

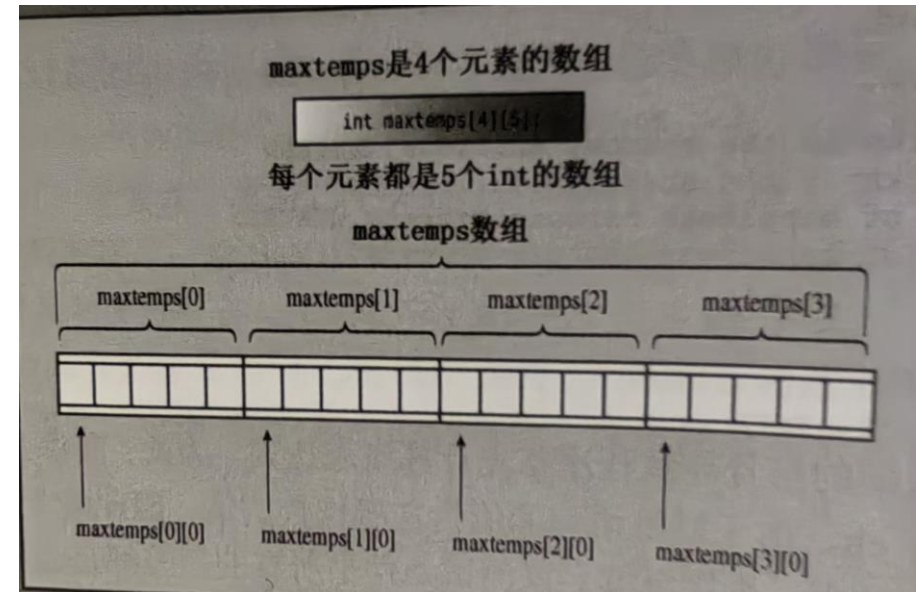
maxtemps数组被看成表格:

		0	1	2	3	4
maxtemps[0]	0	maxtemps[0][0]	maxtemps[0][1]	maxtemps[0][2]	maxtemps[0][3]	maxtemps[0][4]
maxtemps[1]	1	maxtemps[1][0]	maxtemps[1][1]	maxtemps[1][2]	maxtemps[1][3]	maxtemps[1][4]
maxtemps[2]	2	maxtemps[2][0]	maxtemps[2][1]	maxtemps[2][2]	maxtemps[2][3]	maxtemps[2][4]
maxtemps[3]	3	maxtemps[3][0]	maxtemps[3][1]	maxtemps[3][2]	maxtemps[3][3]	maxtemps[3][4]

使用下标访问数组

2.1 二维数组的基本特点

- ✓ 一个二维数组可以含有若干个元素，
这些元素数据类型必须相同
- ✓ 二维数组本质上可以看作是以数组作为数组元素的数组，即由一维数组组成的一维数组，其每行元素对应一个数组（每列元素不对应一个数组）



由数组组成的数组



2.1 二维数组的基本特点

- ✓ 一个二维数组能存储的最大元素个数称为数组的**长度**，其每行能存储的最大元素个数称为数组的**列长度**，其每列存储的最大元素个数称为数组的**行长度**
- ✓ 二维数组合理的行、列下标范围为0到（行长度-1），0到（列长度-1）



2.1 二维数组的基本特点

- ✓ 二维数组按一组存放在连续地址单元的一维数组的形式存放，其存储方式有按行序存储和按列序存储两种

	内存地址	存放元素
int a[2][3] 在一维的内存中按行序优先存储	1000-1003	a[0][0]
	1004-1007	a[0][1]
	1008-1011	a[0][2]
	1012-1015	a[1][0]
	1016-1019	a[1][1]
	10120-1023	a[1][1]



2.1 二维数组的基本特点

✓ 绝大多数语言都是使用行优先存储数组，仅Fortran是列序优先

	内存地址	存放元素
int a[2][3] 在一维的内存中 <u>按列</u> <u>序优先</u> 存储	1000-1003	a[0][0]
	1004-1007	a[1][0]
	1008-1011	a[0][1]
	1012-1015	a[1][1]
	1016-1019	a[0][2]
	10120-1023	a[1][2]



2.1 二维数组的基本特点

- ✓ 在数组声明时，字段rowSize和columnSize应取**固定值的常数**，且数组的长度在**固定后不可改变**（使用程序内存的new运算符可以避开这个限制，后续课程介绍）

```
typeName    arrayName[rowSize][columnSize]
```

数组元素类型 数组名 数组长度

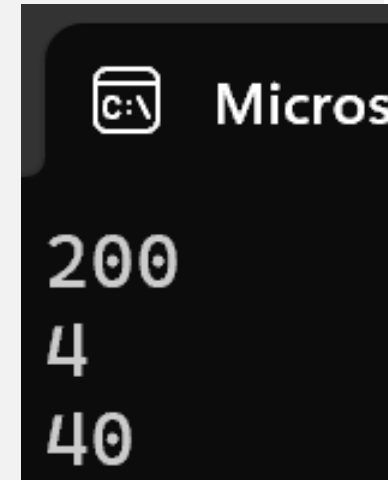


2.1 二维数组的基本特点

- ✓ 二维数组长度的计算使用运算符sizeof，运算符sizeof在输入数组名做为参数时，会返回整个二维数组的字节数
- ✓ 二维数组的长度即为：
$$\text{整个二维数组的字节数} / \text{数组中元素的数据类型所占的字节数}$$



```
#include<iostream>
using namespace std;
int main()
{
    int grade[5][10] = {};
    //(1)find the size of the array
    cout << sizeof(grade) << endl;
    //(2)find the size of one element in the array
    cout << sizeof(grade[1][2]) << endl;
    //(3)find the size of one row in the array
    cout << sizeof(grade[0]) << endl;
    return 0;
}
// Explanation:
//The size of the array grade is 200 bytes (5 * 10 * 4).
//The size of one element in the array grade is 4 bytes.
//The size of one row in the array grade is 40 bytes (10 * 4).
```





2.2 二维数组的初始化规则

- ✓ 对于二维数组来说，由于每个元素本身就是一个数组，因此可以使用于上述代码类似的格式来初始化每个元素

数据类型 数组名[整常量表达式][整常量表达式]={ 初始化数据 };

(1) 分行进行初始化

在 {} 内部再用 {} 把各行分开。初始化的数据个数不能超过数组元素的个数

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



```
int a[2][3];  
a[0][0]=1; a[0][1]=2; a[0][2]=3; a[1][0]=4; a[1][1]=5; a[1][2]=6;
```



2.2 二维数组的初始化规则

(2) 不分行进行初始化

把 {} 中的数据依据赋给a数组各元素（按行赋值）

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

(3) 为部分数组元素初始化，其余数组元素的初值均为0

```
int a[2][3] = { {1, 2}, {4} }; // 1 2 0 4 0 0
```



2.2 二维数组的初始化规则

(4) 可以省略第一维度的定义，**但不能省略第二维的定义**。系统根据初始化的数据个数和第2维的长度来确定第一维的长度

```
int a[][3] = { 1, 4, 5, 6 }; // 1 4 5 6 0 0
```

```
cpp-demo.cpp  x
cpp-demo      (全局范围)

1      #include <iostream>
2      using namespace std;
3      int main()
4      {
5          int a[3][ ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
6          return 0;
7      }
```

The screenshot shows a C++ IDE with a file named 'cpp-demo.cpp'. The code is as follows:

```
1      #include <iostream>
2      using namespace std;
3      int main()
4      {
5          int a[3][ ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
6          return 0;
7      }
```

A red arrow points to the second dimension of the array 'a' in line 5, which is an empty bracket '[]'. A green vertical bar is positioned to the left of the opening curly brace of the 'main' function body.



2.3 二维数组的基本使用

- ✓ 必须先定义，后使用，数组名的命名规则同变量名
- ✓ 数组的大小在声明时确定，不能动态定义，在内存中存放时**先行后列**，占用一块连续的空间
- ✓ 若数组定义中整型常量表达式为 m, n ，则表示有 $m*n$ 个元素，数组长度应为 $m*n$ ，也称数组有 m 行 n 列



2.3 二维数组的基本使用

- ✓ 每个数组元素占用一个数据类型所占用的字节数，数组元素的表示方法为数组名[行下标][列下标]，行下标的范围从[0..m-1]，列下标的范围从[0..n-1]，下标表示为整型常量/变量表达式

```
int a[10][20];  
a[0][7] = 15;    // ok  
float c; long d;  c + d * a[5][3];    // ok  
int k = 3;  a[k * 4 - 3][k / 2 + 4] = 18;    // ok  
10 + a[0][9] + a[1][3] - a[2][2]; // ok
```



2.3 二维数组的基本使用

- ✓ 数组的使用只能逐个引用其中元素，**不能整体使用**，引用时数组下标的表示为整型常量/变量表达式

```
int a[3][4], i, j;  
cout << a; //error  
cout << a[0]; //error  
cout << a[0][0] << a[1][2] << a[2][3]; //ok  
for (i = 0; i < 3; i++)  
    for (j = 0; j < 4; j++)  
        cout << a[i][j]; //ok
```

错误是指无法得到预期结果，编译本身没错，得到的是另外值(地址)



2.3 二维数组的基本使用

```
#include<iostream>
using namespace std;
int main()
{
    int a[3][4]{1,2,3,4}, i, j;
    cout << a << endl; //error
    cout << a[0] << endl; //error
    cout << a[0][0] << a[1][2] << a[2][3]<<endl; //ok
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << a[i][j] << endl; //ok
    return 0;
}
```

Microsoft Visual Studio

```
0000002CF0FCFA18
0000002CF0FCFA18
100
```

```
1
2
3
4
0
0
0
0
0
0
0
0
```



2.3 二维数组的基本使用

✓ 引用时若数组下标超出范围编译器不会报错，因此编程者要控制

注意：编译都不会报错，
是执行时错误

=> 数组下标越界即使执行时不错，也是严重错误

```
#include<iostream>
using namespace std;
int main()
{
    int a[3][4]{1, 2, 3, 4};
    cout<< 3 * 5 + a[10][5] << endl;
    cout<< 3 * 5 + a[5][20] << endl;
    a[5 * 3][21] = 16;
    return 0;
}
```

Microsoft
32781
14

```
#include<iostream>
using namespace std;
int main()
{
    int a[3][4]{1, 2, 3, 4};
    cout<< 3 * 5 + a[10][5] << endl; // cc(引用操作)
```

C6385: 正在从 "a" 读取无效数据。

C6385: 正在从 "a[10]" 读取无效数据。

C6201: 索引"10"超出了"0"至"2"的有效范围(对于可能在堆栈中分配的缓冲区"a")。



2.3 二维数组的基本使用

✓ 对于二维数组的遍历，需要两层循环

```
#include<iostream>
#include<iomanip>      //for setw()
using namespace std;
int main()
{
    int a[3][4]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            cout << "a[" << i << "][" << j << "]=";    // "a[i][j]="
            cout << setw(2) << a[i][j]; //setw() is used to set the width of the output
        }
        cout << endl;
    }
    return 0;
}
```



2.3 二维数组的基本使用

- ✓ 在使用**数组名**引用数组时，数组名实际表示该数组**存储位置的起始地址**。数组名**不可以直接赋值或是在表达式中进行计算**

```
int a_1[3][4] = {1, 2, 3};  
int a_2[3][4] = { };  
a_2 = a_1;           // ×数组名不可用于赋值  
a_1 = a_1+1;         // ×数组名不可用于表达式计算
```



2.3 二维数组的基本使用

- ✓ 数组的复制、拆分和合并：需要在完成数组的“复制”或“拆分”逻辑时，不能直接把一个数组赋值给另一个数组，而是应该将数组的元素依次赋值给另一个数组

```
int a_1[3][4] = {1, 2, 3};  
int a_2[3][4] = { };  
for (int i = 0; i < 3; i++)  
    for (int j = 0; j < 4; j++)  
    {  
        a_2[i][j] = a_1[i][j];    // ✓  
        a_2[i][j] = a_1[i][j]+1;  // ✓  
    }
```



2.4 二维数组与一维数组

- ✓ 二维数组可以看做是由一维数组构成的数组，因而**可以通过使用二维数组的行下标来引用二维数组的一整行元素**，相当于使用该行元素对应的一维数组的数组名（即起始存储地址或指针）

```
int a_1[3][4] = {1, 2, 3};  
cout << sizeof(a_1[1]); // ✓ 使用二维数组行下标引用
```

- ✓ 但**不可以**使用数组行下标用于赋值或是表达式计算

```
int a_1[3][4] = {1, 2, 3};  
int a_2[4] = {};  
a_2 = a_1[1]; // × 不可以使用二维数组行下标进行赋值  
a_1[1] = a_1[1]+1; // × 不可以使用二维数组行下标进行表达式计算
```



2.5 二维数组的应用

- ✓ 二维数组与矩阵：二维数组的结构与矩阵非常相似，因而在程序中经常使用二维数组表示矩阵，并通过二维数组进行矩阵的转置和算术运算等

// 矩阵运算

part1



```
#include <iostream>
using namespace std;
void matrix_add(int a[3][3], int b[3][3], int c[3][3])
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    cout << "The sum of the two matrices is:" << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << c[i][j] << " ";
        }
        cout << endl;
    }
}
```



```
void matrix_transpose(int a[3][3], int b[3][3])  
{
```

part2



```
    cout << "The original matrix is:" << endl;  
    for (int i = 0; i < 3; i++)  
    {  
        for (int j = 0; j < 3; j++)  
        {  
            cout << a[i][j] << " ";  
        }  
        cout << endl;  
    }  
    cout << "The transpose of the matrix is:" << endl;  
    for (int i = 0; i < 3; i++)  
    {  
        for (int j = 0; j < 3; j++)  
        {  
            b[i][j] = a[j][i];  
            cout << b[i][j] << " ";  
        }  
        cout << endl;  
    }  
}
```

```
int main()
{
```

part3



```
    int a[3][3] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
    int b[3][3] = { {90, 80, 70}, {60, 50, 40}, {30, 20, 10} };
    int c[3][3];
    matrix_add(a, b, c);
    matrix_transpose(a, c);

    return 0;
}
```

```
Microsoft Visual Studio 调试器 × + ▾

The sum of the two matrices is:
91 82 73
64 55 46
37 28 19
The original matrix is:
1 2 3
4 5 6
7 8 9
The transpose of the matrix is:
1 4 7
2 5 8
3 6 9
```



2.5 二维数组的应用

- ✓ 编写一个程序，来计算平均每天捕获鱼的重量
 - 假设每天最多捕获5条鱼
 - 如果数组被填满或输入了非数字输入，循环将结束

```
// cinfish.cpp -- non-numeric input terminates loop
```

```
#include <iostream>
```

```
const int Max = 5;
```

```
int main()
```

```
{
```

```
    using namespace std;
```

```
    // get data
```

```
    double fish[Max];
```

```
    cout << "Please enter the weights of your fish.\n";
```

```
    cout << "You may enter up to " << Max
```

```
        << " fish <q to terminate>.\n";
```

```
    cout << "fish #1: ";
```

```
    int i = 0;
```

```
    while (i < Max && cin >> fish[i]) {
```

```
        if (++i < Max)
```

```
            cout << "fish #" << i + 1 << ": ";
```

```
    }
```

```
    // calculate average
```

```
    double total = 0.0;
```

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

```
        total += fish[j];
```

```
    // report results
```

```
    if (i == 0)
```

```
        cout << "No fish\n";
```

```
    else
```

```
        cout << total / i << " = average
```

```
        weight of " << i << " fish\n";
```

```
    cout << "Done.\n";
```

```
    return 0;
```

```
}
```

Microsoft Visual Studio 调试

```
Please enter the weights of your fish.
You may enter up to 5 fish <q to terminate>.
fish #1: 30
fish #2: 35
fish #3: 25
fish #4: 40
fish #5: 1
26.2 = average weight of 5 fish
Done.
```



2.5 二维数组的应用

- ✓ 程序要求用户提供5个高尔夫得分，以计算平均成绩
 - 如果用户输入非数字输入，程序将拒绝，并要求用户继续输入数字
 - 程序发现用户输入了错误内容时，应采取的步骤
 - 重置cin以接受新的输入
 - 删除错误输入
 - 提示用户再输入

```
// cingolf.cpp -- non-numeric input skipped
#include <iostream>
const int Max = 5;
int main()
{
    using namespace std;
    // get data
    int golf[Max];
    cout << "Please enter your golf scores.\n";
    cout << "You must enter " << Max << " rounds.\n";
    int i;
    for (i = 0; i < Max; i++)
    {
        cout << "round #" << i+1 << ": ";
        while (!(cin >> golf[i])) {
            cin.clear(); // reset input
            while (cin.get() != '\n')
                continue; // get rid of bad input
            cout << "Please enter a number: ";
        }
    }
}
```

Microsoft Visual Studio 调试

```
Please enter your golf scores.
You must enter 5 rounds.
round #1: 88
round #2: 87
round #3: must i
Please enter a number: 103
round #4: 94
round #5: 86
91.6 = average score 5 rounds
```

```
// calculate average
double total = 0.0;
for (i = 0; i < Max; i++)
    total += golf[i];
// report results
cout << total / Max << " = average
score " << Max << " rounds\n";
return 0;
}
```



目录

- 巧用智能编程提示



3.1 巧用智能编程提示

✓ 二维数组的长度即为：

整个二维数组的字节数 / 数组中元素的数据类型所占的字节数

```
#include<iostream>
using namespace std;
int main()
{
    int grade[5][10] = {};
    cout << sizeof(grade) << endl; // 200 //find the size of the array
    cout << sizeof(grade[1][2]) << endl; // 4 //find the size of one element in the array
    cout << sizeof(grade[0]) << endl; // 40 //find the size of one row in the array
    return 0;
}

// Output:
// 200
// 4
// 40
// Explanation: The size of the array grade is 200 bytes (5 * 10 * 4). The size of one element in the array grade is 4 bytes.
// The size of one row in the array grade is 40 bytes (10 * 4).
```

利用智能编程环境的提示，
帮助自己理解程序



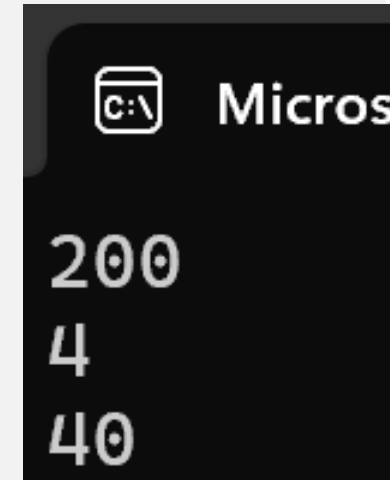
```
#include<iostream>
using namespace std;
int main()
{
    int grade[5][10] = {};
    //(1)find the size of the array
    cout << sizeof(grade) << endl; // 200
    //(2)find the size of one element in the array
    cout << sizeof(grade[1][2]) << endl; // 4
    //(3)find the size of one row in the array
    cout << sizeof(grade[0]) << endl; // 40
    return 0;
}
```

```
// Explanation:
```

```
//The size of the array grade is 200 bytes (5 * 10 * 4).
```

```
//The size of one element in the array grade is 4 bytes.
```

```
//The size of one row in the array grade is 40 bytes (10 * 4).
```



常用的提示词需要积累和摸索，
智能编译环境会越来越懂你

```
// ifelse.cpp -- using the if else statement
#include <iostream>
int main()
{
    char ch;
    std::cout << "Type, and I shall repeat.\n";
    std::cin.get(ch);
    while (ch != '.')
    {
        if (ch == '\n')
            std::cout << ch;    // done if newline
        else
            std::cout << ++ch;  // done otherwise
        std::cin.get(ch);
    }
    std::cout << "\nPlease excuse the slight confusion.\n";
    return 0;
}
```

需求：
假设要通过对字母进行加密编码来
修改输入的文本（换行符不变）

智能编程辅助预测运行结果，
但要注意可能存在错误

```
// Output:
// Type, and I shall repeat.
// abcdefghijklmnopqrstuvwxyz
// bcdefghijklmnopqrstuvwxyz
// cdefghijklmnopqrstuvwxyz.
// defghijklmnopqrstuvwxyz.
// Please excuse the slight confusion.
```

```
// Output:
// Type, and I shall repeat.
// abcefgghijklmnopqrstuvwxyz
// bcdfhjkmnqrvwxy
// Please excuse the slight confusion.
```



巧用智能编程辅助预测运行结果，但要注意可能存在错误，要通过实际运行确认

```
// Output:  
// Type, and I shall repeat.  
// abcdefghijklmnopqrstuvwxyz  
// bcdefghijklmnopqrstuvwxyz  
// cdefghijklmnopqrstuvwxyz.  
// defghijklmnopqrstuvwxyz.  
// Please excuse the slight confusion.
```

```
// Output:  
// Type, and I shall repeat.  
// abcefgghijklmnopqrstuvwxyz  
// bcdfhjkmnqrvwxy  
// Please excuse the slight confusion.
```

Microsoft Visual Studio 调试器

```
Type, and I shall repeat.  
abcdefghijklmnopqrstuvwxyz  
bcdefghijklmnopqrstuvwxyz{  
abcdefghijklmnopqrstuvwxyz.  
bcdefghijklmnopqrstuvwxyz{  
Please excuse the slight confusion.
```

D:\c++\Project5\x64\Debug\I

```
Type, and I shall repeat.  
abcefgghijklmnopqrstuvwxyz  
bcdfhjkmnqrvwxyz{
```



总结

• 一维数组

1. 基本概念
2. 一维数组的特点
3. 一维数组的初始化规则
4. 一维数组的基本使用
5. 一维数组的应用-排序算法
6. 基于范围的for循环

• 二维数组

1. 二维数组的特点
2. 二维数组的初始化规则
3. 二维数组的基本使用
4. 二维数组与一维数组
5. 二维数组的应用

• 巧用智能编程提示