



5.2 二维数组的定义及使用

李洁

nijanice@163.com



目录

基本概念

二维数组声明

二维数组初始化

二维数组的基本使用

二维数组与一维数组

二维数组使用典型案例



5.2 一维数组的定义及使用

5.2.1 基本概念

什么是数组？

数组：一种数据格式，存储以组的形式出现的多个同一种数据类型的值。

学生名单(数组)：张三(元素1)，李四(元素2)，...

每月销售额 (数组)：19万元(元素1)，23万元(元素2)，...

学生成绩(数组)：90分(元素1)，40分(元素2)，...



5.2 二维数组的定义和使用

5.2.1 基本概念

一维数组：相当于EXCEL表中的一行或是一列

学生单门成绩：

C/C++	81	88	93
	张三	李四	王五

二维数组：相当于包含多行多列的整张EXCEL表

学生多门成绩：

C/C++	81	87	99
Java	88	83	98
Python	93	59	100
	张三	李四	王五



5.2 二维数组的定义和使用

5.2.1 基本概念

一维数组：相当于EXCEL表中的一行或是一列

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



5.2 二维数组的定义和使用

5.2.1 基本概念

- C/C++支持**多维数组**，其声明的一般形式为
- `typeName arrayName[size1][size2]...[sizeN];`
- 这里仅讨论**二维数组**



5.2 二维数组的定义和使用

5.2.1 基本概念

二维数组的下标(索引)

二维数组的下标包含行下标和列下标两个部分，其中行下标标记元素所在行号，列下标标记元素所在列号。行下标和列下标都从0开始并依次递增。

如 `int s[2][5];`

<code>s[0][0]</code>	<code>s[0][1]</code>	<code>s[0][2]</code>	<code>s[0][3]</code>	<code>s[0][4]</code>
<code>s[1][0]</code>	<code>s[1][1]</code>	<code>s[1][2]</code>	<code>s[1][3]</code>	<code>s[1][4]</code>

注意：

★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；



5.2 二维数组的定义和使用

5.2.1 基本概念

- 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**一维数组的一维数组**，其每行元素对应一个数组（**每列元素不对应一个数组**）。
- 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**，其每列能存储的最大元素个数称为数组的**行长度**。



5.2 二维数组的定义和使用

5.2.1 基本概念

二维数组的存储

二维数组在程序运行时的存储空间中，按一组存放在**连续的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。



5.2 二维数组的定义和使用

5.2.1 基本概念

二维数组的存储

二维数组在程序运行时的存储空间中，**通常以按行存储的方式存储**，即以“先行后列”的规则连续存放。例：

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

在一维的内存中

按行序优先存储

内存地址	存放元素
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]
1020 - 1023	a[1][2]



5.2 二维数组的定义和使用

5.2.1 基本概念

二维数组的存储

二维数组在程序运行时的存储空间中，也可以按列存储的方式存储，即以“先列后行”的规则连续存放。**绝大多数语言都使用行序优先存储数组，仅Fortran是列序优先，例：**

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

在一维的内存中

按列序优先存储

内存地址	存放元素
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]
1020 - 1023	a[1][2]



5.2 二维数组的定义和使用

5.2.1 基本概念

- 二维数组的存储

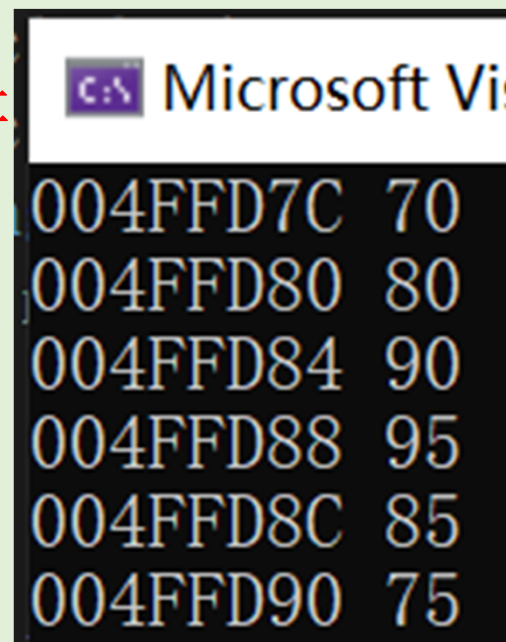
验证编译器默认对二维数组存储使用的行序优先存储

//二维数组中的元素，按？序优先被存储在连续的内存地址中

```
#include <iostream>
using namespace std;
int main()
{
    int grade[2][3] = { 70, 80, 90, 95, 85, 75 };
    int i1, i2;
    for (i1 = 0; i1 < 2; i1++)
        for (i2 = 0; i2 < 3; i2++)
            cout << &(grade[i1][i2]) << ' ' << grade[i1][i2] << endl;
    //使用取元素地址运算符&，获取数组中每个元素的内存地址
    //关于运算符&的内容详见指针部分课程
}
```

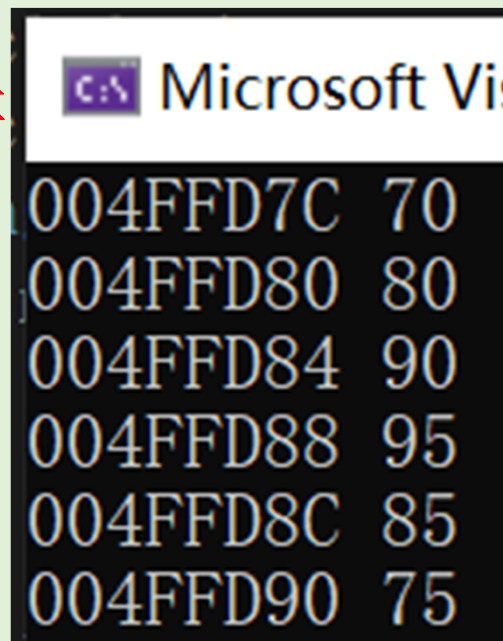
//二维数组中的元素，按？序优先被存储在连续的内存地址

```
#include <iostream>
using namespace std;
int main()
{
    int grade[2][3] = { 70, 80, 90, 95, 85, 75 };
    int i1, i2;
    for (i1 = 0; i1 < 2; i1++)
        for (i2 = 0; i2 < 3; i2++)
            cout << &(grade[i1][i2]) << ' ' << grade[i1][i2] << endl;
    //使用取元素地址运算符&，获取数组中每个元素的内存地址
    //关于运算符&的内容详见指针部分课程
}
```



//二维数组中的元素，按？序优先被存储在连续的内存地址

```
#include <iostream>
using namespace std;
int main()
{
    int grade[2][3] = { 70, 80, 90, 95, 85, 75 };
    int i1, i2;
    for (i1 = 0; i1 < 2; i1++)
        for (i2 = 0; i2 < 3; i2++)
            cout << &(grade[i1][i2]) << ' ' << grade[i1][i2] << endl;
    //使用取元素地址运算符&，获取数组中每个元素的内存地址
    //关于运算符&的内容详见辨析：
}
```



★二维数组存储使用的行序优先存储



5.2 二维数组的定义和使用

5.2.2 二维数组声明

声明/定义格式

```
typeName          arrayName[rowSize1][columnSize2];
```

数组元素类型 数组名 数组长度

在数组声明时，字段rowSize和columnSize应是取**固定值的常数**，且数组的长度在**固定后不可改变**（使用有关程序内存的new运算符可以避开这个限制，详见章节6：指针）。

Dev++环境可以使用变量定义数组的长度，Visual Studio不可以（同一维数组定义）。



5.2 二维数组的定义和使用

5.2.2 二维数组声明

例：

`int studentNumber[4][20];` //大小为80的4行20列的int型二维数组，存储4个班级，每个班级20个学生学号

`double grade[5][50];` //大小为250的5行50列double型二维数组，存储50个学生5门课的成绩

`char character[2][26];` //大小为52的2行26列char型二维数组，分别存储大写和小写的26个英文字符

数组的长度rowSize和columnSize的值为**大于等于1的整数，是取值固定的常数，且数组的长度在固定后不可改变**（使用有关程序内存的new运算符可以避开这个限制，详见章节6：指针）。



5.2 二维数组的定义和使用

5.2.2 二维数组声明

- 在数组的长度字段rowSize和columnSize的值为**大于等于1的整数**，其取值上限取决于编程环境的内存限制
- 全局变量数组使用**静态存储区分配**存储空间，局部变量数组在程序的**运行堆栈上分配**存储空间，因而全局变量数组可用的内存空间大小明显大于局部变量数组（有关存储空间的内容详见之后的操作系统课程）



5.2 二维数组的定义和使用

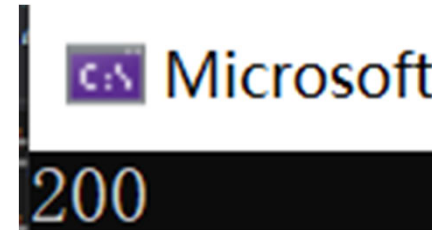
5.2.2 二维数组声明

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

```
int grade[5][10] = {};
```

```
sizeof(grade);
```

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



5.2 二维数组的定义和使用

5.2.2 二维数组声明

二维数组长度的计算使用运算符`sizeof`，运算符`sizeof`使用数组中的一个元素`grade[1][2]`做为参数时，会返回数组中一个元素的内存大小

```
int grade[5][10] = {};  
  
sizeof(grade[1][2]);
```





5.2 二维数组的定义和使用

5.2.2 二维数组声明

二维数组长度的计算使用运算符`sizeof`，运算符`sizeof`使用`grade[1]`做为参数时，返回的值是什么？

```
int grade[5][10] = {};  
  
sizeof(grade[1]);
```

The logo for C:\ Microsoft, featuring a purple square with a white 'C' and the word 'Microsoft' in blue.

40

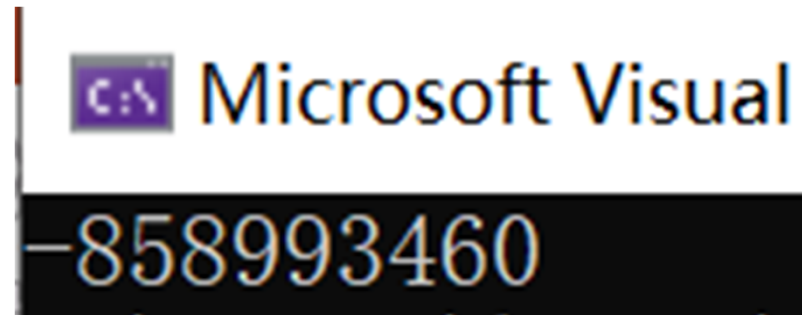


5.2 二维数组的定义和使用

5.2.3 二维数组初始化

一个数组在被声明后，通常需要先进行初始化的过程来**赋予初值**，以防止因数组中存储着含有我们不知道的值的元素而**导致的BUG**

```
int main()
{
    int grade[5][10];
    cout << grade[2][3];
    return 0; }
```



会出现什么问题？

- 此时数组在声明后没有进行初始化，数组对应的存储空间存放着程序员不知道的**未知值**，正确的做法是进行下页所示的数组初始化后，再使用数据元素



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

初始化方法（元素全部初始化）：

- `int grade[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};`
- `int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`
- `int grade[][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

以上三个语句初始化的二维数组结果都是：



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

初始化方法（元素全部初始化）：

- `int grade[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};`
//初始化使用双层括号时，内括号的个数不超过行数，内括号内数据个数不超过列数
- `int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`
- `int grade[][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`
// 编译器计算出行长度为3，自动设置行长度

以上三个语句初始化的二维数组结果都是：

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



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

初始化方法（元素部分初始化）：

- `int grade [3][4] = {};`
- `int grade [3][4] = {0};`
- `int grade[3][4] = {{1}, {5}, {9}};`
- `int grade[3][4] = {{1, 2}, {5, 6, 7}, {9}};`



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

初始化方法（元素部分初始化）：

- `int grade [3][4] = {};` //所有元素初始化为默认值0

- `int grade [3][4] = {0};` //所有元素初始化为默认值0

- `int grade[3][4] = {{1}, {5}, {9}};` //初始化结果是：
//对每行第一个元素赋值，剩下元素为默认值0

1	0	0	0
5	0	0	0
9	0	0	0

- `int grade[3][4] = {{1, 2}, {5, 6, 7}, {9}};` //初始化结果是：
//第一行元素为{1, 2, 0, 0}
//第二行元素为{5, 6, 7, 0}
//第三行元素为{9, 0, 0, 0}

1	2	0	0
5	6	7	0
9	10	0	0



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

错误示例:

```
int grade[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};  
int grade1[3][4];  
grade1 = grade;
```

```
int grade [3][4] = {1, 2, 3, 4, 5, 6, 7, 8.8, 9, 10, 11, '1' };
```

```
int grade1[4] = {1, 2, 3, 4};  
int grade2[4] = {5, 6, 7, 8};  
int grade3[4] = {9, 10, 11, 12};  
int grade[3][4] = {grade1, grade2, grade3};
```



5.2 数组的基本使用

5.2.3 二维数组初始化

错误示例:

```
int grade[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};  
int grade1[3][4];  
grade1 = grade; // 不能通过地址赋值
```

```
int grade [3][4] = {1, 2, 3, 4, 5, 6, 7, 8.8, 9, 10, 11, '1' };
```

//数组初始化禁止使用缩窄转换

```
int grade1[4] = {1, 2, 3, 4};  
int grade2[4] = {5, 6, 7, 8};  
int grade3[4] = {9, 10, 11, 12};  
int grade[3][4] = {grade1, grade2, grade3}; // 不能使用一维数组名初始化二维数组
```



5.2 数组的基本使用

5.2.3 二维数组初始化

错误示例:

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

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



5.2 数组的基本使用

5.2.3 二维数组初始化

错误示例:

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

//二维数组根据内括号内的元素初始化每一行元素，此时用于初始化第一行元素的整数超过了行的长度

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

//二维数组的初始化不能省略列数(多维数组的初始化不能省略第二维及更高的维数),因为在程序中二维数组按行序优先存储,编译器使用列数值计算二维数组的行数以及每行的元素个数,并分配相应的存储空间



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

★二维数组的声明和初始化过程，可以分开成两条语句，也可以合在一起写成一条语句。

★二维数组的初始化语句**并不是必须要写**的程序语句，但使用未初始化的数组可能会导致程序出现错误，合理的初始化数组是**编程的好习惯**之一。



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

★二维数组的声明和初始化使用**一条**语句(推荐):

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

★二维数组的声明和**初始值赋值**使用**分开的两条**语句:

```
int grade[3][4], i, j;  
    for (i = 0; i < 3; i++)  
        for (j = 0; j < 4; j++)  
            grade[i][j] = i * 4 + j + 1;
```




5.2 二维数组的定义和使用

5.2.4 二维数组使用

如何单独访问二维数组元素？

二维数组的**下标(索引)**

二维数组的下标包含**行下标和列下标**两个部分，其中行下标标记元素所在行号，列下标标记元素所在列号。行下标和列下标都从0开始并依次递增。

如 `int s[2][5];`

<code>s[0][0]</code>	<code>s[0][1]</code>	<code>s[0][2]</code>	<code>s[0][3]</code>	<code>s[0][4]</code>
<code>s[1][0]</code>	<code>s[1][1]</code>	<code>s[1][2]</code>	<code>s[1][3]</code>	<code>s[1][4]</code>

注意：

★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；



5.2 二维数组的定义和使用

5.2.4 二维数组使用

★对于二维数组中的元素，使用**数组下标**引用它们，相当于使用一个**变量**。数组的第一个元素下标为 $[0, 0]$ ，**最后一个元素下标为 $[\text{行长度}-1, \text{列长度}-1]$** 。

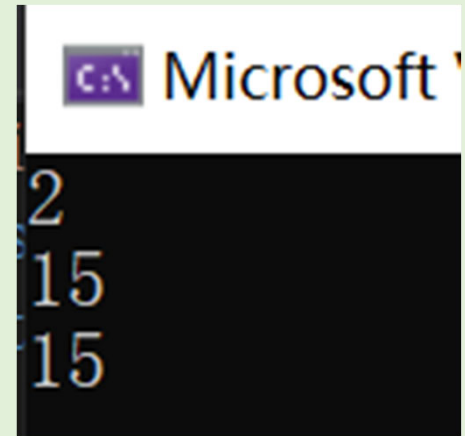
如 `int a[3][4]`，则使用`a[3][0]`和`a[1][4]`都是**越界**的。

★对于二维数组中的元素，也可以使用指针引用它们，具体内容见章节六。

★**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**，即对于形式参数的修改**不会改变**真实参数的值。

```
#include <iostream>
using namespace std;
int add(int a, int b)
{
    int c; c = a + b;
    return c;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout << grade[0][1] << "\n" ;
    int count;
    cout << grade[1][0] + grade[2][1] << "\n";
    cout << add(grade[1][0], grade[2][1]) << "\n";
    return 0;
}
```

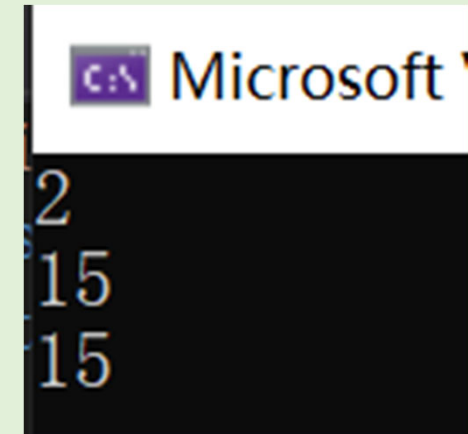
```
#include <iostream>
using namespace std;
int add(int a, int b)
{
    int c; c = a + b;
    return c;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout << grade[0][1] << "\n" ;//数组的元素做为变量被输出
    int count;
    cout << grade[1][0] + grade[2][1] << "\n";//数组的元素做变量被表达式使用
    cout << add(grade[1][0], grade[2][1]) << "\n";//数组的元素做变量被函数使用
    return 0;
}
```



```

#include <iostream>
using namespace std;
int add(int a, int b)
{
    int c; c = a + b;
    return c;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout << grade[0][1] << "\n" ; //数组的元素做为变量被输出
    int count;
    cout << grade[1][0] + grade[2][0];
    cout << add(grade[1][0], grade[2][0]);
    return 0;
}

```



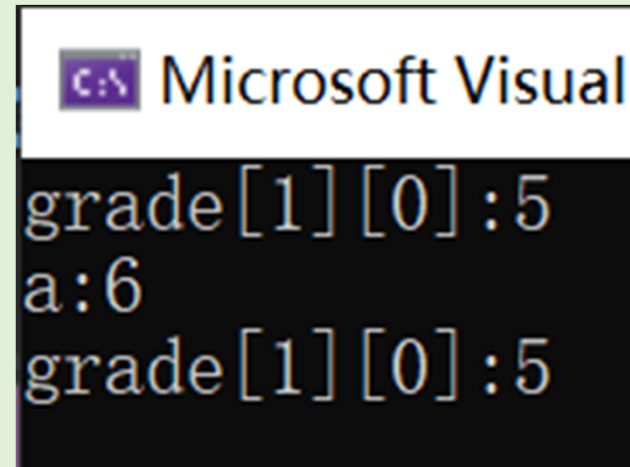
辨析：

★对于二维数组中的元素，使用数组下标引用它们，相当于使用一个变量。

用

```
#include <iostream>
using namespace std;
void add_one(int a) //形式参数
{
    a = a + 1;
    cout << "a:" << a << "\n"; //形式参数
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout << "grade[1][0]:" << grade[1][0] << "\n"; //实际参数
    add_one(grade[1][0]);
    cout << "grade[1][0]:" << grade[1][0] << "\n"; //实际参数
    return 0;
}
```

```
#include <iostream>
using namespace std;
void add_one(int a)  //形式参数
{
    a = a + 1;
    cout << "a:" << a << "\n";  //形式参数
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout << "grade[1][0]:" << grade[1][0] << "\n";  //实际参数
    add_one(grade[1][0]);
    cout << "grade[1][0]:" << grade[1][0] << "\n";  //实际参数
    return 0;
}
```

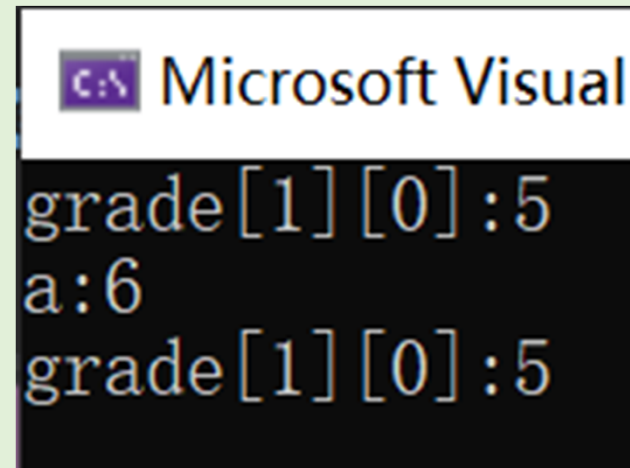


C:\ Microsoft Visual
grade[1][0]:5
a:6
grade[1][0]:5

```

#include <iostream>
using namespace std;
void add_one(int a)  //形式参数
{
    a = a + 1;
    cout << "a:" << a << "\n";  //形式参数
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4,
    cout << "grade[1][0]:" << grade[1][0];
    add_one(grade[1][0]);
    cout << "grade[1][0]:" << grade[1][0];
    return 0;
}

```



```

C:\ Microsoft Visual
grade[1][0]:5
a:6
grade[1][0]:5

```

辨析:

★对于二维数组中的元素，使用**数组下标**引用它们，相当于使用一个**变量**。

★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**，对于形式参数的修改**不会改变**真实参数的值。

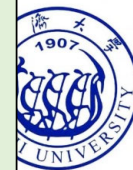


5.2 数组的基本使用

5.2.4 二维数组使用

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，是对于数组的常见操作之一。

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int grade[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++)
        {
            grade[i][j] = grade[i][j] - 1; //遍历操作数组元素
            cout << "grade[" << i << "]" << "[" << j << "]: ";
            cout << setw(2) << grade[i][j] << " "; //遍历输出数组元素
        }
        cout << endl;
    }
    return 0;
}
```





```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int grade[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++)
```

Microsoft Visual Studio 调试控制台

```
grade[0][0]: 0  grade[0][1]: 1  grade[0][2]: 2  grade[0][3]: 3
grade[1][0]: 4  grade[1][1]: 5  grade[1][2]: 6  grade[1][3]: 7
grade[2][0]: 8  grade[2][1]: 9  grade[2][2]: 10 grade[2][3]: 11
```

```
    return 0;
}
```



```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int grade[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++)
        {
            grade[i][j] = grade[i][j] - 1; //遍历操作数组元素
            cout << "grade[" << i << "]" << "[" << j << "]: ";
            cout << setw(2) << grade[i][j] << " "; //遍历输出数组元素
        }
        cout << endl;
    }
    return 0;
}
```

辨析:

★对于二维数组的遍历，需要两层循环。



5.2 二维数组的定义和使用

5.2.4 二维数组使用

用二维数组，同时统计3门课的高于平均分的人数

- 当同时统计多门课的数据时，使用一维数组需要定义多个一维数组并分开统计每个一维数组的数据，此时可使用一个二维数组同时统计多门课的数据



```
int main()//用一维数组实现求一门课高于平均分人数:
{
    int k = 0, i;
    double s[100], ave, sum = 0;
    for (i = 0;i < 100;i++)
    {
        cin >> s[i];    // 数组元素的输入
        sum = sum + s[i]; // 数组元素的求和
    }
    ave = sum / 100;
    for (i = 0;i < 100;i++)
        if (s[i] > ave)
            k++;
    cout << k;
}
```



```
int main()//用二维数组实现三门课高于平均分人数:
{
    int k[3]= {0}, i, j;
    double s[3][100], ave[3] = {0}, sum[3] = {0};
    for (j = 0; j < 3; j++)
        for (i = 0; i < 100; i++)
        {
            cin >> s[j][i];    // 数组元素的输入
            sum [j]= sum[j] + s[j][i]; // 数组元素的求和
        }
    ave[j]= sum[j] / 100;
    for (j = 0; j < 3; j++)
        for (i = 0; i < 100; i++)
            if (s[j][j]> ave[j])
                k[j]++;
    cout << k[0] << "\n"<< k[1] << "\n"<< k[2] << "\n";
    return 0;
}
```



5.2 二维数组的定义和使用

5.2.4 二维数组使用

用二维数组，同时输出3门课的最高分

- 当同时统计多门课的数据时，使用一维数组需要定义多个一维数组并分开统计每个一维数组的数据，此时可使用一个二维数组同时统计多门课的数据



```
int main() //用一维数组实现输出一门课最高分:
{
    int k = 0, i;
    double s[100], ave, sum = 0;
    for (i = 0; i < 100; i++)
    {
        cin >> s[i];    // 数组元素的输入
    }
    int max = s[0], imax = 0;    //假设第一个元素值最大
    for (i = 1; i < 100; i++)
        if (s[i] > max)
        {
            max = s[i];    //求最大元素
            imax = i;    //求最大元素下标
        }
    cout << max << "\n" << imax; // 输出最高分及下标
}
```

```
int main()//用二维数组实现输出三门课最高分:
{
    int k[3]= {0}, i, j;
    double s[3][100], ave[3] ={0}, sum[3] ={0};
    for (j = 0;j < 3;j++)
        for (i = 0;i < 100;i++)
            cin >> s[j][i];    // 数组元素的输入
    int max[3]={0}, imax[3]={0};    //假设第一个元素值最大
    for (j = 0;j < 3;j++)
        for (i = 0;i < 100;i++)
            if (s[j][i]> max[j])
            {
                max [j]= s[j][i];    //求最大元素
                imax[j] = i;        //求最大元素下标
            }
    cout << max[0] << " " << imax[0] << "\n" << max[1]
        << " " << imax[1] << "\n" << max[2] << " " << imax[2] << "\n";
    return 0; }
```



5.2 二维数组的定义和使用

5.2.4 二维数组使用

二维数组在遍历时应注意：

- 对于二维数组，当它的长度较大时，遍历数组会使程序的效率变低，此时可以使用更优的数据结构进行优化，如稀疏矩阵的优化（稀疏矩阵：矩阵中大部分元素为没有意义的0）。
- 在二维数组遍历以查找数组中的元素时，基于行优先或是基于列优先的顺序遍历数组，程序的效率会有差别。此时可以通过更优化的算法来提高效率。



5.2 二维数组的定义和使用

5.2.4 二维数组使用

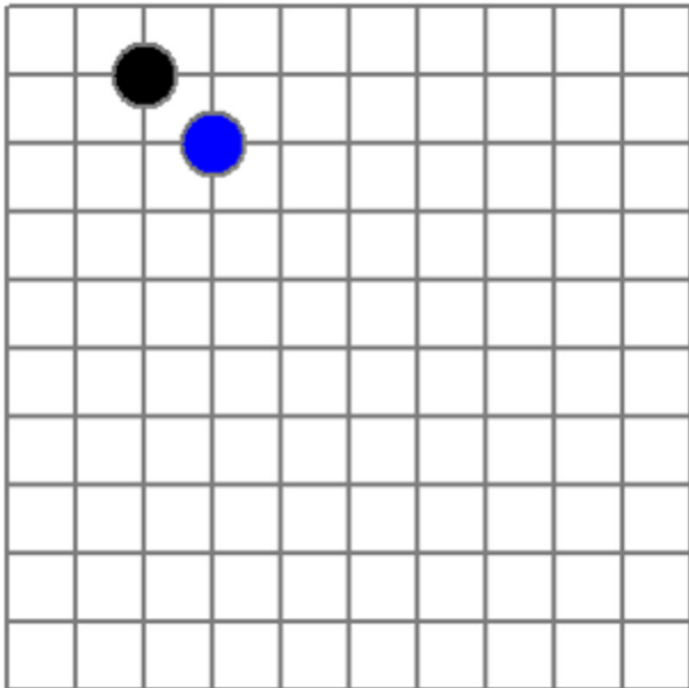
- 对于二维数组，当它的长度较大时，遍历数组会使程序的效率变低，此时可以使用更优的数据结构进行优化，如稀疏矩阵的优化。



5.2 二维数组的定义和使用

5.2.4 二维数组使用

使用二维数组记录棋盘



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

数组中存在过多没有意义的0

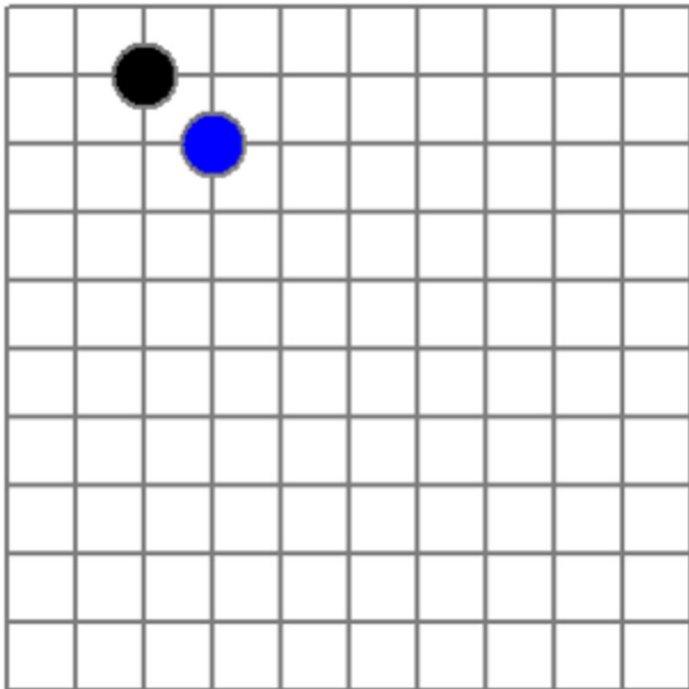


5.2 二维数组的定义和使用

5.2.4 二维数组使用

使用二维数组记录棋盘

(具体内容详见之后的数据结构课程)



0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	0	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

节省了很大的存储空间

行号	列号	值
1	2	1
2	3	2



5.2 二维数组的定义和使用

5.2.4 二维数组使用

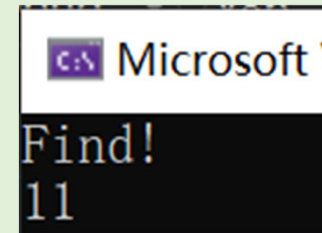
- 在二维数组遍历以查找数组中的元素时，基于行优先或是基于列优先的顺序遍历数组，程序的效率会有差别。

例: 对于一个含有20个有序的整数的数组 $a[2][10]$, 查找元素 $a[1][0]$ 所对应的整数在数组中的位置时，分别用行有优先和列优先实现。

```
int main()
{
    int a[2][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
    int i, j, count = 0;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 10; j++)    //行优先查找
        {
            count++;                //统计比较操作次数
            if (a[i][j] == 11)
            {
                cout << "Find!" << endl;
                cout << count;
                break;
            }
        }
    }
    return 0; }
```



```
int main()
{
    int a[2][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
    int i, j, count = 0;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 10; j++)    //行优先查找
        {
            count++;                //统计比较操作次数
            if (a[i][j] == 11)
            {
                cout << "Find!" << endl;
                cout << count;
                break;
            }
        }
    }
    return 0; }
```

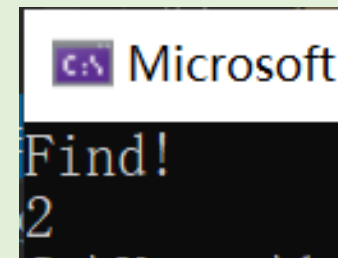


```
int main()
{
    int a[2][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
    int i, j, count = 0;
    for (i = 0; i < 10; i++)
    {
        for (j = 0; j < 2; j++)    //列优先查找
        {
            count++;              //统计比较操作次数
            if (a[j][i] == 11)
            {
                cout << "Find!" << endl;
                cout << count;
                break;
            }
        }
    }
    return 0; }
```

```

int main()
{
    int a[2][10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
    int i, j, count = 0;
    for (i = 0; i < 10; i++)
    {
        for (j = 0; j < 2; j++)    //列优先查找
        {
            count++;                //统计比较操作次数
            if (a[j][i] == 11)
            {
                cout << "Find!" << endl;
                cout << count;
                break;
            }
        }
    }
    return 0;
}

```



辨析:

★使用行优先查找需要进行11次比较操作，而使用列优先查找只需要2次比较操作。



5.2 二维数组的定义和使用

5.2.4 二维数组使用

在使用**数组名**引用数组时，数组名实际表示的是数组在程序的存储空间中**存储位置的起始地址**。对于数组名的引用，**不可以直接赋值或是在表达式中进行计算**。

数组的复制，拆分和合并：在需要完成数组的“复制”或是“拆分”的逻辑时，**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复制给另一个数组



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int grade1[3][4] = {};
    grade1 = grade;
    grade = grade + 1;
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int grade1[3][4] = {};
    grade1 = grade;      //数组名不可用于赋值
    grade = grade + 1;   //数组名不可用于表达式计算
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int grade1[3][4] = {};
    grade1 = grade;
    grade = grade + 1;
    return 0;
}
```

错误列表

整个解决方案



错误 4



警告 0



消息 0



代码

说明

abc

E0137

表达式必须是可修改的左值

abc

E0137

表达式必须是可修改的左值



C3863

不可指定数组类型“int [3][4]”



C3863

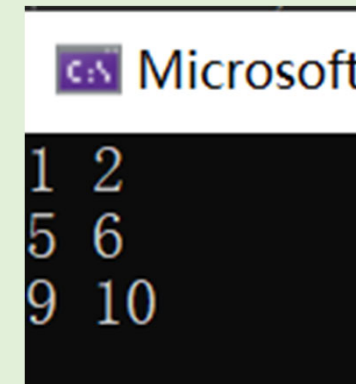
不可指定数组类型“int [3][4]”

```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int grade1[3][4] = {};
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 4; j++)
        {
            grade1[i][j] = grade[i][j];
            grade[i][j] = grade[i][j]+1;
        }
    return 0;
}
```





```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int part_grade[3][2] = {};
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 2; j++)
        {
            part_grade[i][j] = grade[i][j]; //数组拆分
            cout << part_grade[i][j] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```



```
int main()
{
    int grade[3][4] = { }; int i, j;
    int part_grade1[3][2] = {1, 2, 5, 6, 9, 10};
    int part_grade2[3][2] = {3, 4, 7, 8, 11, 12};
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if(j < 2)
                grade[i][j] = part_grade1[i][j];
            else
                grade[i][j] = part_grade2[i][j-2];
            cout << grade[i][j] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```





```
int main()
{
    int grade[3][4] = { }; int i, j;
    int part_grade1[3][2] = {1, 2, 5, 6, 9, 10};
    int part_grade2[3][2] = {3, 4, 7, 8, 11, 12};
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if(j < 2)
                grade[i][j] = part_grade1[i][j]; //数组合并
            else
                grade[i][j] = part_grade2[i][j-2]; //数组合并
            cout << grade[i][j] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

```
C:\N Microsoft
1 2 3 4
5 6 7 8
9 10 11 12
```



5.2 二维数组的定义和使用

5.2.4 二维数组使用

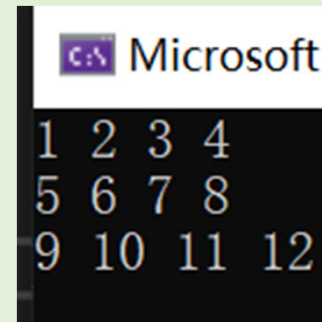
数组名作为函数参数：在使用**数组名**引用数组时，数组名实际表示的是数组在程序的存储空间中**存储位置的起始地址**。对于数组名的引用，常用于函数的参数传递。

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





```
#include <iostream>
using namespace std;
int print_grade(int grade[3][4])
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j];
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade(grade); //函数调用
    return 0;
}
```





5.2 二维数组的定义和使用

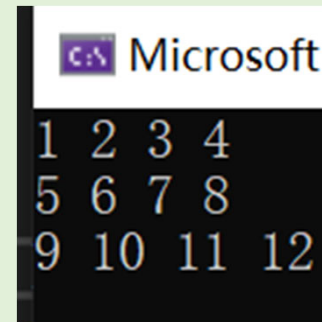
5.2.4 二维数组使用

数组名作为函数参数：在使用**二维数组名**引用数组时，函数的形式参数可以指定二维数组所有维数的大小，也**可以省略**第一维的维数大小，**但不能省略**其第二维或者是更高维度的维数。

因为在程序中二维数组按**行序优先存储**，编译器**使用列数值计算**二维数组的行数以及每行的元素个数，并**分配相应的存储空间**。



```
#include <iostream>
using namespace std;
int print_grade1(int grade[][4]) //形式参数可以省略第一维的维数
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j];
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade1(grade); //函数调用
    return 0;
}
```




```
#include <iostream>
using namespace std;
int print_grade1(int grade[][])
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j];
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade1(grade); //函数调用
    return 0;
}
```

```
#include <iostream>
using namespace std;
int print_grade1(int grade[][]) //形式参数不可以省略第二维或更高维的维数?
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j];
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade1(grade); //函数调用
    return 0;
}
```

```
#include <iostream>
using namespace std;
int print_gradel(int grade[3][])
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j];
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_gradel(grade); //函数调用
    return 0;
}
```

```
#include <iostream>
using namespace std;
int print_grade1(int grade[3][ ]) //形式参数省略第二维或更高维的维数?
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j];
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade1(grade); //函数调用
    return 0;
}
```



5.2 二维数组的定义和使用

5.2.3 二维数组初始化

错误示例:

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

//二维数组根据内括号内的元素初始化每一行元素，此时用于初始化第一行元素的整数超过了行的长度

```
int grade [3][] = {1, 2, 3, 4, 5, 6, 7, 8.8, 9, 10, 11, 12};
```

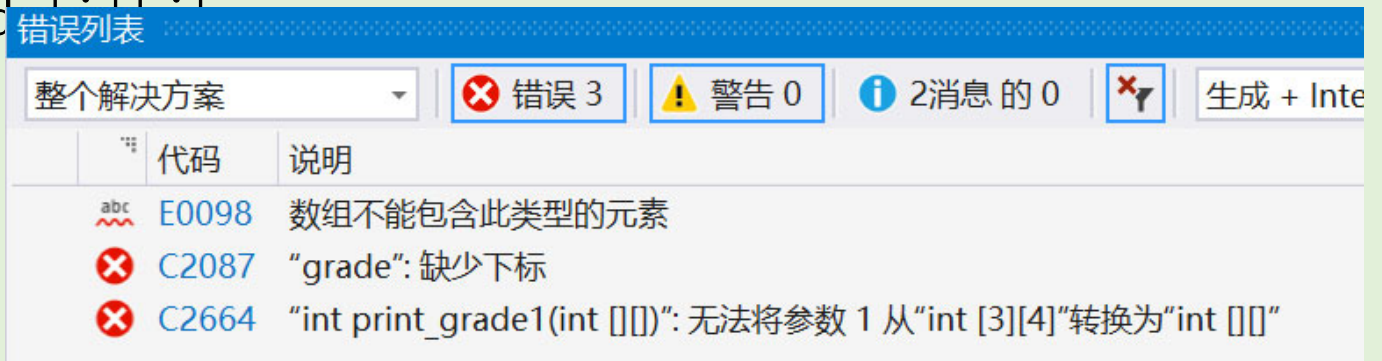
//二维数组的初始化不能省略列数(多维数组的初始化不能省略第二维及更高的维数),因为在程序中二维数组按行序优先存储,编译器使用列数值计算二维数组的行数以及每行的元素个数,并分配相应的存储空间

```

#include <iostream>
using namespace std;
int print_grade1(int grade[3][4]) //形式参数不可省略第二维或更高维的维数
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j] << " ";
    return 0;
}

int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade1(grade); //函数调用
    return 0;
}

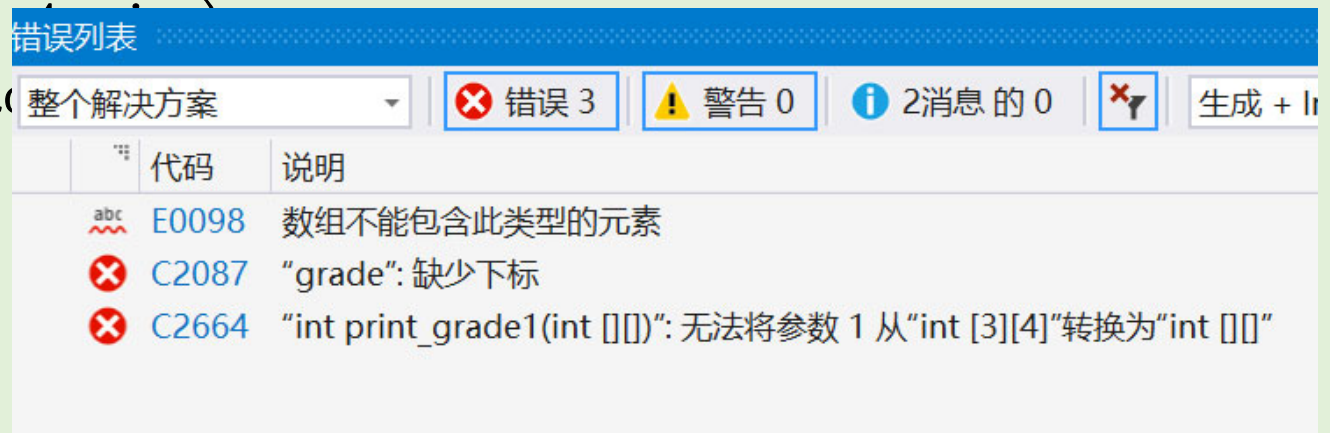
```



```

#include <iostream>
using namespace std;
int print_grade1(int grade[][]) //形式参数不可省略第二维或更高维的维数
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cout << grade[i][j] << " ";
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade1(grade); //函数调用
    return 0;
}

```





5.2 二维数组的定义和使用

5.2.4 二维数组使用

数组名作为函数参数：在使用**二维数组名**引用数组时，此时参数调用的方式为**按地址传递**，即对于形式参数的修改**会改变**真实参数的值



```
#include <iostream>
using namespace std;
void array_add_one1(int grade[3][4]) //形式参数
{
    int i, j;
    for (j = 0; j < 3; j++)
    {
        for (i = 0; i < 4; i++)
        {
            grade[j][i]++;
            cout << grade[j][i] << " " ;//形式参数
        }
        cout << endl;
    }
    cout << endl;
} //程序未结束，因页面大小问题写在后页
```



```
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }, i, j;
    for (j = 0; j < 3; j++)
    {
        for (i = 0; i < 4; i++)
            cout << grade[j][i] << " "; //实际参数
        cout << endl;
    }
    cout << endl;
    array_add_one1(grade);
    for (j = 0; j < 3; j++)
    {
        for (i = 0; i < 4; i++)
            cout << grade[j][i] << " "; //实际参数
        cout << endl;
    }
    return 0; }
```

```
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }, i, j;
    for (j = 0; j < 3; j++)
    {
        for (i = 0; i < 4; i++)
            cout << grade[j][i] << " "; //实际参数
        cout << endl;
    }
    cout << endl;
    array_add_one1(grade);
    for (j = 0; j < 3; j++)
    {
        for (i = 0; i < 4; i++)
            cout << grade[j][i] << " "; //实际参数
        cout << endl;
    }
    return 0; }
```

C:\ Microsoft Visual

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

```
2 3 4 5
6 7 8 9
10 11 12 13
```

```
2 3 4 5
6 7 8 9
10 11 12 13
```



5.2 二维数组的定义和使用

5.2.5 二维数组与一维数组

1. 二维数组可以看作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，但不可以使用二维数组的列下标来引用二维数组的一整列元素。
2. 使用1中方法，可以将二维数组中的某行元素构成的一维数组提取出。
3. 使用二维数组行下标来引用二维数组的一整行元素时，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)。



5.2 二维数组的定义和使用

5.2.5 二维数组与一维数组

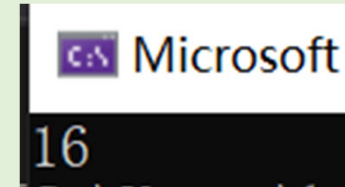
对于二维数组中的一行元素，使用**数组行下标**引用它们，相当于使用一个**一维数组的数组名**



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout<< sizeof(grade[1]); //使用二维数组行下标
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    cout<< sizeof(grade[1]); //使用二维数组行下标
    return 0;
}
```



辨析:

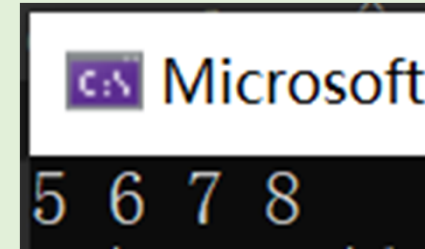
★二维数组grade[3][4]可以看作是由三个一维数组grade[0], grade[1], grade[2]构成, 其中这三个一维数组的所占的字节数都为16。



```
#include <iostream>
using namespace std;
int print_grade(int grade[4])
{
    int i;
    for (i = 0; i < 4; i++)
        cout << grade[i] << " ";
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade(grade[1]); //使用二维数组行下标
    return 0;
}
```



```
#include <iostream>
using namespace std;
int print_grade(int grade[4])
{
    int i;
    for (i = 0; i < 4; i++)
        cout << grade[i] << " ";
    return 0;
}
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    print_grade(grade[1]); //使用二维数组行下标
    return 0;
}
```



辨析:

★二维数组grade[3][4]可以看作是由三个一维数组grade[0], grade[1], grade[2]构成。grade[1]代表该二维数组的第二行元素所构成的一维数组名。



5.2 二维数组的定义和使用

5.2.5 二维数组与一维数组

对于二维数组中的一行元素，使用**数组行下标**引用它们，相当于使用一个一维数组数组名，如用于函数的参数传递，此时参数调用的方式为**按地址传递**，即对于形式参数的修改**会改变**真实参数的值



```
void array_add_one(int grade[4]) //形式参数
{
    int i;
    for (i = 0; i < 4; i++)
    {
        grade[i]++;
        cout << grade[i] << " "; //形式参数
    }
    cout << endl << endl; }

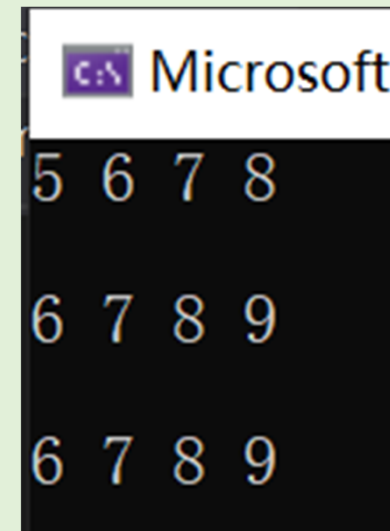
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }, i;
    for (i = 0; i < 4; i++)
        cout << grade[1][i] << " "; //实际参数
    cout << endl << endl;
    array_add_one(grade[1]);
    for (i = 0; i < 4; i++)
        cout << grade[1][i] << " "; //实际参数
    cout << endl << endl;
    return 0; }
```

```
void array_add_one(int grade[4]) //形式参数
```

```
{    int i;
    for (i = 0; i < 4; i++)
    {
        grade[i]++;
        cout << grade[i] << " "; //形式参数
    }
    cout << endl << endl; }
```

```
int example17()
```

```
{    int grade[3][4] = { 1, 2, 3, 4,
    for (i = 0; i < 4; i++)
        cout << grade[1][i] << " ";
    cout << endl << endl;
    array_add_one(grade[1]);
    for (i = 0; i < 4; i++)
        cout << grade[1][i] << " "; //实际参数
    cout << endl << endl;
    return 0; }
```



辨析:

★一维数组数组名`grade[1]` 用于函数的参数传递, 此时参数调用的方式为按地址传递, 即对于形式参数的修改会改变真实参数的值。



5.2 二维数组的定义和使用

5.2.5 二维数组与一维数组

对于二维数组中的一行元素，使用**数组行下标**引用它们，相当于使用一个一维数组数组名，**不可以用于赋值或是在表达式中进行计算。**



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int grade1[4] = {};
    grade1 = grade[1];
    grade[1] = grade[1] + 1;
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
    int grade1[4] = {};
    grade1 = grade[1];    //二维数组行下标不可用于赋值
    grade[1] = grade[1] + 1; //二维数组行下标不可用于表达式计算
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main()
{
    int grade[3][4]
    int grade1[4] =
    grade1 = grade[
    grade[1] = grad
    return 0;
}
```

错误列表

整个解决方案



错误 4



警告 0



消息 0

	代码	说明
abc	E0137	表达式必须是可修改的左值
abc	E0137	表达式必须是可修改的左值
	C3863	不可指定数组类型“int [4]”
	C3863	不可指定数组类型“int [4]”



5.2 二维数组的定义和使用

5.2.6 二维数组使用典型案例


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加减、转置和乘除计算。

```
void matrix_add(int a[2][3], int b[2][3])
{
    int i, j, c[2][3];
    for (i = 0; i < 2; i++)           //A+B矩阵，每个对应元素相加
        for (j = 0; j < 3; j++)
            c[i][j] = a[i][j] + b[i][j];
    cout << "矩阵A+B: " << endl;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            cout << c[i][j] << " ";
        cout << "\n";
    }
}
```





```
int main()
{
    int a[2][3], b[2][3], i, j;
    cout << "矩阵A: " << endl;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            cin >> a[i][j];
    cout << "矩阵B: " << endl;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            cin >> b[i][j];
    matrix_add(a, b, 2, 3); //调用矩阵相加函数
    return 0;
}
```

 Microsoft

矩阵A:

1 2 3

4 5 6

矩阵B:

7 8 9

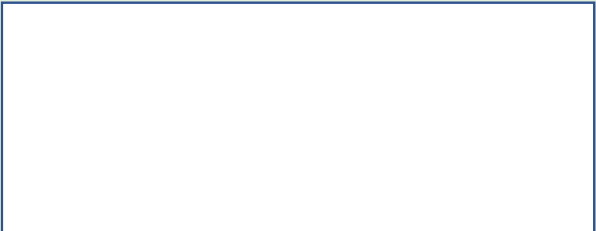
10 11 12

矩阵A+B:

8 10 12

14 16 18



```
void matrix_transform(int a[2][3])
{
    int i, j, temp;
    for (i = 0; i < 3; i++)
        for (j = 0; j < i; j++) //矩阵转置
        {
            
        }
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            cout << a[i][j] << " ";
        cout << "\n";
    }
}
```



```
void matrix_transform(int a[2][3])
{
    int i, j, temp;
    for (i = 0; i < 3; i++)
        for (j = 0; j < i; j++) //矩阵转置
        {
            temp = a[i][j];
            a[i][j] = a[j][i];
            a[j][i] = temp;
        }
    cout << "转置后矩阵: " << endl;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            cout << a[i][j] << " ";
        cout << "\n";
    }
}
```



```
int main()
{
    int a[3][3], temp, i, j;
    cout << "原矩阵: " << endl;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            cin >> a[i][j];
    matrix_transform(a); //调用矩阵转置函数
    return 0;
}
```

C:\ Microsoft V

原矩阵:

1 2 3

4 5 6

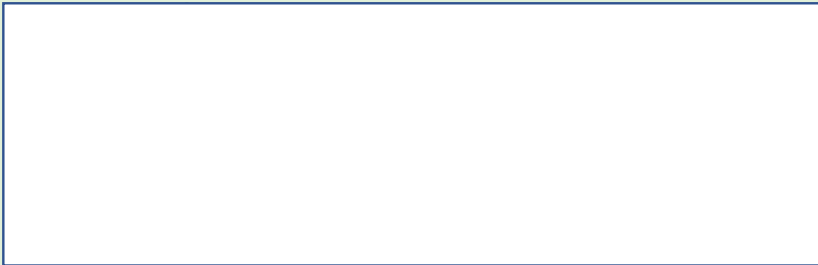
7 8 9

转置后矩阵:

1 4 7

2 5 8

3 6 9

```
void matrix_multiplication(int a[2][3], int b[3][4])
{
    int c[2][4], i, j, k, s;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 4; j++) //矩阵乘法, 2*3矩阵乘以3*4矩阵得到2*4矩阵
            {

            }
    cout << "矩阵A*B: " << endl;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 4; j++) cout << c[i][j] << " ";
        cout << "\n";
    }
    cout << "\n";}
```

```
void matrix_multiplication(int a[2][3], int b[3][4])
{
    int c[2][4], i, j, k, s;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 4; j++) //矩阵乘法, 2*3矩阵乘以3*4矩阵得到2*4矩阵
            {
                s = 0;
                for (k = 0; k < 3; k++)
                    s += a[i][k] * b[k][j];
                c[i][j] = s;
            }
    cout << "矩阵A*B: " << endl;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 4; j++) cout << c[i][j] << " ";
        cout << "\n";
    }
    cout << "\n"; }
```




```
int main()
{
    int a[2][3], b[3][4], i, j;
    cout << "矩阵A: " << endl;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            cin >> a[i][j];
    cout << "矩阵B: " << endl;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            cin >> b[i][j];
    matrix_multiplication(a, b); //调用矩阵乘法函数
    return 0;
}
```



Microsoft Visual Studio

矩阵A:

3 5 7

4 6 8

矩阵B:

1 4 7 10

2 5 8 11

3 6 9 12

矩阵A*B:

34 79 124 169

40 94 148 202



5.2 二维数组的定义和使用

5.2.6 二维数组使用典型案例

稀疏矩阵的与二维矩阵的转化

```
int main()    //稀疏矩阵转存
{int grade[5][5] = {1,0,0,0,0, 3,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,7,9,0,0};
  int matrix[5][3] = {}, i, j, count = 1;
  matrix[0][0] = 5;    //记录矩阵行数
  matrix[0][1] = 5;    //记录矩阵列数
  for (i = 0;i < 5;i++)
  {    for (j = 0;j < 5;j++)
      {
          if (grade[i][j] != 0)
          {
              matrix[0][2] ++; //矩阵非0元素个数加1
              matrix[count][0] = i;    //记录该非0元素位置和数值
              matrix[count][1] = j;
              matrix[count][2] = grade[i][j];
              count++;
          }
      }
  }
} //程序未结束，因页面大小问题写在后页
```

```

cout << "稀疏矩阵: " << endl;
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 5; j++)
    {
        cout << grade[i][j] << ' ';
    }
    cout << endl;
}
cout << endl << "转化矩阵: " << endl;
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 3; j++)
    {
        cout << matrix[i][j] << ' ';
    }
    cout << endl;
}
return 0; }

```

稀疏矩阵:

```

1 0 0 0 0
3 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 7 9 0 0

```

转化矩阵:

```

5 5 4
0 0 1
1 0 3
4 1 7
4 2 9

```

int main() //稀疏矩阵还原

{

int grade[5][5] = {}, i, j, count = 1;

int matrix[5][3] = {{5, 5, 4}, {0, 0, 1}, {1, 0, 3}, {4, 1, 7}, {4, 2, 9}};

int row, col;

for (i = 1; i <= matrix[0][2]; i++) //循环整个记录数组


{



元素的坐标

//将非0元素写入稀疏矩阵

} //程序未结束，因页面大小问题写在后页



```
int main()    //稀疏矩阵还原
{
    int grade[5][5] = {}, i, j, count = 1;
    int matrix[5][3] = {{5, 5, 4}, {0, 0, 1}, {1, 0, 3}, {4, 1, 7}, {4, 2, 9}};
    int row, col;
    for (i = 1; i <= matrix[0][2]; i++)    //循环整个记录数组
    {
        row = matrix[i][0];    //提取非0元素的坐标
        col = matrix[i][1];
        grade[row][col] = matrix[i][2];    //将非0元素写入稀疏矩阵
    }    //程序未结束，因页面大小问题写在后页
```

```

cout << "转化矩阵: " << endl;
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 3; j++)
    {
        cout << matrix[i][j] << ' ';
    }
    cout << endl;
}
cout << endl << "稀疏矩阵: " << endl;
for (i = 0; i < 5; i++)
{
    for (j = 0; j < 5; j++)
    {
        cout << grade[i][j] << ' ';
    }
    cout << endl;
}
return 0; }

```

转化矩阵:

```

5 5 4
0 0 1
1 0 3
4 1 7
4 2 9

```

稀疏矩阵:

```

1 0 0 0 0
3 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 7 9 0 0

```



5.2 二维数组的定义和使用

5.2.6 二维数组使用典型案例

查找递增的二维数组中，是否含有值为X的元素



5.2 二维数组的定义和使用

5.2.6 二维数组使用典型案例

查找递增的二维数组中，是否含有值为X的元素

可用二分法查找来提高效率，思路如下：

1. 首先，从数组的中间元素开始搜索，如果该元素正好是目标元素，则搜索过程结束，否则执行下一步。
2. 如果目标元素大于/小于中间元素，则在数组大于/小于中间元素的那一半区域查找，然后重复步骤（1）的操作。
3. 如果某一步数组为空，则表示找不到目标元素。

可以证明，二分法查找的时间复杂度 $O(\log n)$



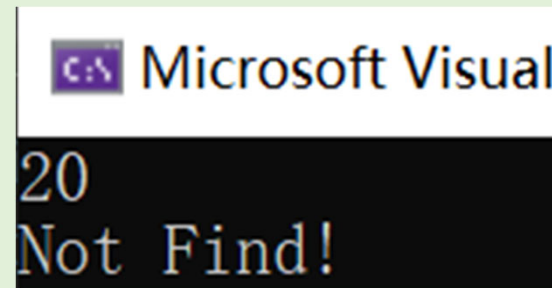
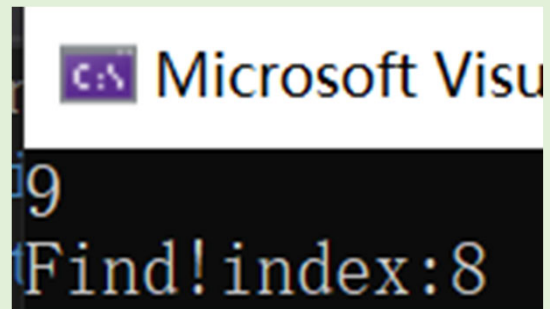
```
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int i, array_length, target, mark = 0, head = 0, tail;
    cin >> target;
    array_length = sizeof(grade) / 4;
    tail = array_length - 1;
    while (head <= tail)
    {
```

```
    } //程序未结束，因页面大小问题写在后页
```



```
int main()
{
    int grade[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int i, array_length, target, mark = 0, head = 0, tail;
    cin >> target;
    array_length = sizeof(grade) / 4;
    tail = array_length - 1;
    while (head <= tail)
    {
        i = (head + tail) / 2;
        if(grade[i / 4][i % 4] == target)
        {
            cout << "Find!" << "index:" << i;
            mark = 1;
            break;
        }
        if (grade[i / 4][i % 4] < target)
        {
            head = i + 1;
        }
    } //程序未结束，因页面大小问题写在后页
```

```
    if (grade[i / 4][i % 4] > target)
    {
        tail = i - 1;
    }
}
if (mark == 0)
    cout << "Not Find!";
return 0;
}
```





目录

基本概念	二维数组意义/二维数组下标/二维数组存储
二维数组声明	二维数组声明格式
二维数组初始化	二维数组初始化方法
二维数组使用	下标引用/遍历/复制、合并、拆分/函数参数
二维数组与一维数组	二维数组与一维数组关系
二维数组使用典型案例	矩阵运算/矩阵查找/稀疏矩阵转换