

复合类型：数组和字符串

COMP250205：计算机程序设计

李昊

hao.li@xjtu.edu.cn

西安交通大学计算机学院

数组

使用大量相同类型的变量

对于一个水果店，记录今年**每个月**的营业额

```
float jan, feb, mar, apr, may, jun, july, aug, sep, oct, nov, dec;
```

对于一个水果店，记录今年**每天**的营业额

```
float jan1st, jan2nd, jan3rd, jan4th, ..., dec31th;
```

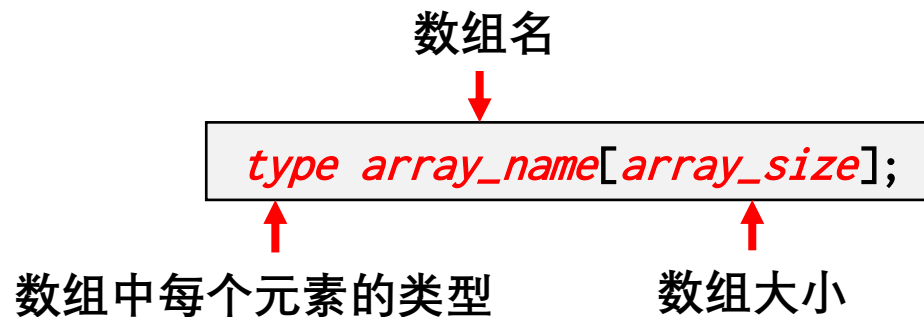
对于一个水果店，记录**过去三年每天**的营业额



使用数组记录相同类型的变量

数组是复合类型：从基本/自定义类型拓展而来
不存在单独的数组类型，但存在

整型数组、浮点型数组、字符数组、xxx数组



对于一个水果店，记录今年**每天**的营业额

```
float per_day_income[365];
```

数组的初始化

数组的初始化方式

通过花括号显式初始化 → `int array[4] = {0, 1, 2, 3};`

不允许复制初始化 → `int array2[4] = array;`

`int array3[4];`

不允许赋值 → `array3 = array;`

数组的长度必须在声明时确定/可推断

可推断 → `int array[] = {0, 1, 2, 3};`

只初始化前4个 → `int array[5] = {0, 1, 2, 3};`

越界 → `int array[2] = {0, 1, 2, 3};`

访问数组元素

可以单独访问数组中的任一个特定位置的元素

下标： 标记数组中元素的位置，**从0开始**

代表了数组的第0个元素



```
float per_day_income[365];  
cin >> per_day_income[0];  
cin >> per_day_income[365];
```



不存在这样一个元素，**数组越界**

数组越界是最常见的运行时错误，需要特别小心

使用数组 - 1

零售店有三种商品，每种商品价格不一样，售出的数量也不一样，求总收入是多少

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

int main()
{
    int amount[3] = {12, 7, 8};
    float price[3] = {24.1, 5.9, 13.9};
    float income = amount[0] * price[0] + amount[1] * price[1] +
                    amount[2] * price[2];
    cout << income << endl;
    return 0;
}
```

如果是30种商品呢？



使用数组 - 2


数组可以用下标访问，与循环天然搭配

```
int main()
{
    int amount[3] = {12, 7, 8};
    float price[3] = {24.1, 5.9, 13.9};
    float income = 0.0;

    for (int i = 0; i < 3; ++i) {
        income += amount[i] * price[i];
    }

    cout << income << endl;
    return 0;
}
```

使用循环中的
变量作为下标



使用数组 - QUIZ

定义一个长度为10的整型数组，每个元素的值就是它的下标，将这个数组打印出来

```
int main()
{
    int array[10];
    for (int i = 0; i < 10; ++i) {
        array[i] = i;
    }
    for (int i = 0; i < 10; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;
    return 0;
}
```

多维数组 - 1

使用数组存储多维数据

一个 10×10 的矩阵，每个元素 (x, y) 的值为 $x * y$

2维数组



```
int matrix[10][10];
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
        matrix[i][j] = i * j;
    }
}
```

双重循环，每一重循环对应一维

```
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}
```

内层循环换行，可以按行对齐

多维数组 - 2

多维数组实际上并不是一个独立的数据类型

2维数组：数组的数组

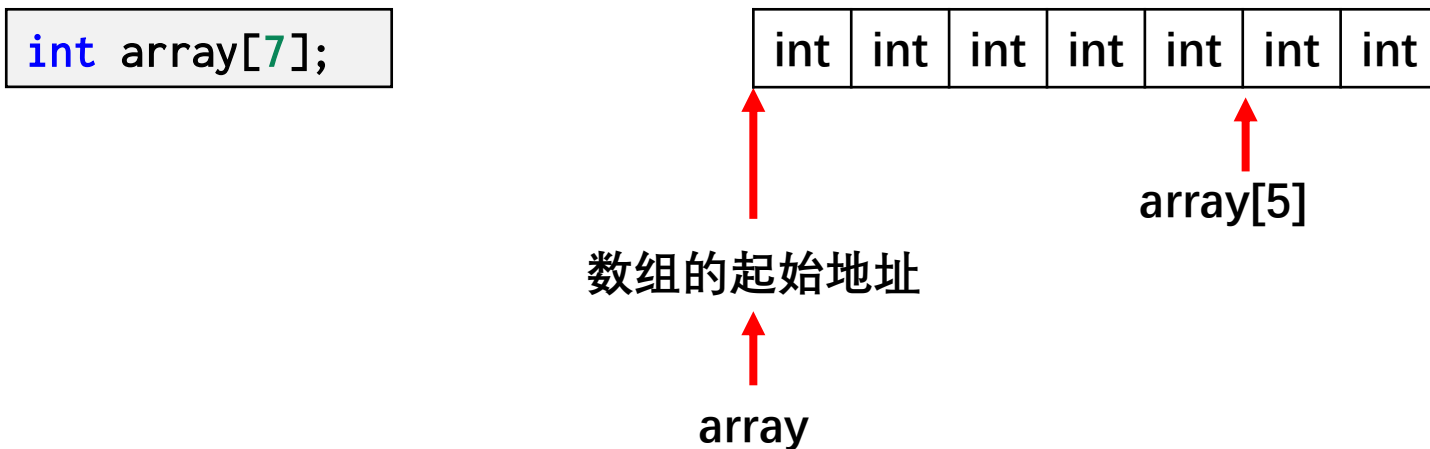
3维数组：数组的数组的数组

```
int matrix[3][4] = {  
    {0, 1, 2, 3},  
    {4, 5, 6, 7},  
    {8, 9, 10, 11}  
};
```

每一个元素都是一个数组

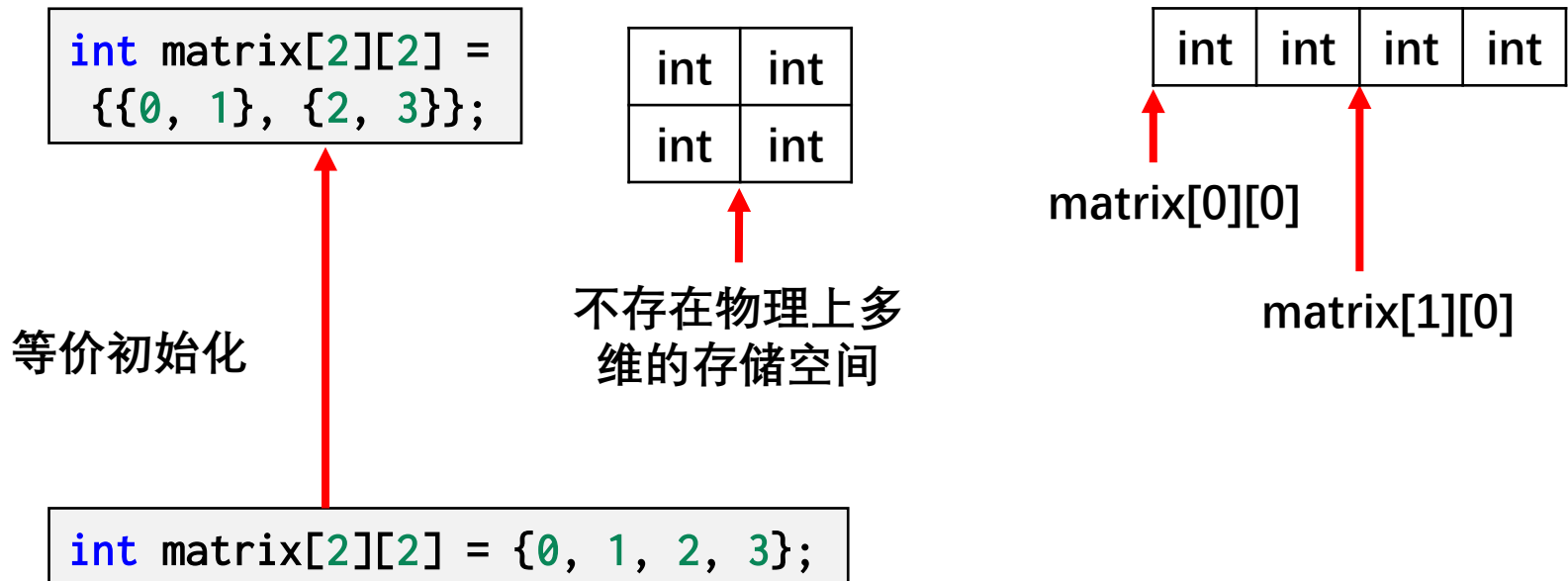
深入数组的内存模型 - 1

所有数据，都以二进制方式，公平均一的存储



深入数组的内存模型 - 2

所有数据，都以二进制方式，公平均一的存储



(C风格) 字符串

字符数组和字符串 - 1

字符数组：每一个数组元素都是一个字符

```
char str[] = {'h', 'e', 'l', 'l', 'o'};
```

str的起始地址 →

'h'	'e'	'l'	'l'	'o'
-----	-----	-----	-----	-----

字符串：一串以'\0'结尾的字符

```
cout << "hello" << endl;
```

↑
双引号括起来的一串字符

hello的实际存储方式 →

'h'	'e'	'l'	'l'	'o'	'\0'
-----	-----	-----	-----	-----	------

字符数组和字符串 - 2

使用字符数组来存储字符串

字符数组视角 → `char str[] = {'h', 'e', 'l', 'l', 'o', '\0'};`

字符串视角
类型转换 → `char str[] = "hello";`

错误! 需要6个
字节存储 → `char str[5] = "hello";`

错误! 不能使
用字符变量存
储字符串 → `char str = "h";`

字符串的输出 - 1

对于非字符数组类型，cout将其作为地址看待

```
int array[] = {1, 2, 3};  
cout << array << endl;
```



数组名被当作数组起始地址

对于字符数组类型，cout将其作为字符串看待

```
char str1[] = {'h', 'e', 'l', 'l', 'o'};  
char str2[] = "hello";  
cout << str1 << str2 << endl;
```



尝试打印到'\0'为止

字符串的输出 - 2

```
#include <cstring> ← 包含了对字符串进行处理的常用函数
#include <iostream>

int main() {
    using namespace std;
    const int SIZE = 15;
    char name1[SIZE];
    char name2[SIZE] = "C++owboy"; ← 较短的字符串，后面用'\0'补齐
    cout << "Hi! I'm " << name2 << "! What's your name?" << endl;
    cin >> name1; 计算字符串的长度，到'\0'为止（不包括'\0'）
    cout << "Well, " << name1 << ", your name has ";
    cout << strlen(name1) << " letters and is stored" << endl;
    cout << "in an array of " << sizeof(name1) << " bytes." << endl;
    cout << "Your initial is " << name1[0] << "." << endl;
    name2[3] = '\0';
    cout << "The first 3 letters of my name: " << name2 << endl;
    return 0;
}
```

字符串的输入 - 1

使用cin作为字符串输入的缺陷

```
#include <iostream>
int main() {
    using namespace std;
    const int SIZE = 20;
    char name[SIZE];
    char dessert[SIZE];

    cout << "Enter your name:\n";
    cin >> name;
    cout << "Enter your favorite dessert:\n";
    cin >> dessert;
    cout << "I have some delicious " << dessert << " for you, ";
    cout << name << endl;
    return 0;
}
```

字符串的输入 - 2

标准输入中的字符被缓冲，cin从中将普通字符付给变量

H	a	o	'	L	i	\n
---	---	---	---	---	---	----



```
char ch;
while (cin >> ch) {}
```



在缓冲中不停的看下一个字符，并且跳过（消耗）空格和回车

str1, '\0'替换结尾 str2, '\0'替换结尾

str1, '\0'替换结尾			str2, '\0'替换结尾			
H	a	o	'	L	i	\n



```
char str1[20];      char str2[20];
cin >> str1;      cin >> str2;
```



一次性读入缓冲中的输入，直到空格、回车为止 继续从当前缓冲中读取数据，直到空格、回车为止

字符串的输入 - 3

使用cin.getline()来进行整行的字符串输入

```
#include <iostream>
int main() {
    using namespace std;
    const int SIZE = 20;
    char name[SIZE];
    char dessert[SIZE];

    cout << "Enter your name:\n";
    cin.getline(name, SIZE);
    cout << "Enter your favorite dessert:\n";
    cin.getline(dessert, SIZE);
    cout << "I have some delicious " << dessert << " for you, ";
    cout << name << endl;
    return 0;
}
```

查看教材4.2.3节，了解更多使用cin读取整行的方法和细微差别

字符串的操作函数

cstring头文件中定义了一系列操作字符串的函数

函数原型	意义
strlen(p)	返回p的长度，空字符不计算
strcmp(p1, p2)	比较p1和p2，如果相等，返回0；如果p1>p2，返回正数；否则返回负数
strcat(p1, p2)	将p2附加到p1后面，返回p1
strcpy(p1, p2)	将p2拷贝给p1，返回p1

字符串操作函数的陷阱

所有字符串操作函数均基于两个假设

1. 操作对象是字符串，即以'\0'结尾
2. 操作对象有足够的空间进行合法的操作

```
char str1[10] = "hello, ";  
char str2[10] = "world!";  
char str3[] = "hello, world!";  
char str4[5] = {'h', 'e', 'l', 'l', 'o'};
```

危险! str1的空间不足



```
strcat(str1, str2);  
strcpy(str1, str3);
```

危险! str4的长度不明



```
strcat(str1, str4);
```

字符串操作函数的陷阱 - QUIZ

输入一串字符，然后将其每个字符变成后继字符

```
#include <iostream>
#include <cstring>
int main() {
    using namespace std;
    const int SIZE = 10;
    char str[SIZE], str2[2*SIZE];
    cin.getline(str, SIZE);
    strcpy(str2, str);
    for (int i = 0; i < SIZE; ++i) {
        str[i] = str[i] + 1;
    }
    strcat(str2, str);
    cout << str2 << endl;
    return 0;
}
```

会将'\0'修改掉 ← 使用strlen()

str长度未知，str2可能空间不够

内容总结

数组：存储大量相同类型的对象

长度在声明时确定

使用循环来处理数组，注意越界

多维数组实际是数组的数组

(C风格) 字符串：使用字符数组存储，'\0'结尾

字符串的输入和输出与普通数组不同

字符串操作函数对操作对象有严格假设