

# 一、数组的概念

数组是若干个相同类型的变量在内存中有序存储的集合。

概念理解：

数组用于存储一组数据

数组里面存储的数据类型必须是相同的

数组在内存中会开辟一块连续的空间

`int a[10];` //定义了一个整型的数组a，a是数组的名字，数组中有10个元素，每个元素的类型

都是int类型，而且在内存中连续存储。

这十个元素分别是a[0] a[1] .... a[9]

a[0]~a[9]在内存中连续的顺序存储

## 二、数组的分类

### 2.1 按元素的类型分类

#### 1) 字符数组

即若干个字符变量的集合，数组中的每个元素都是字符型的变量

`char s[10]; s[0],s[1]....s[9];`

#### 2) 短整型的数组

`short int a[10]; a[0],a[9]; a[0]=4;a[9]=8;`

#### 3) 整型的数组

`int a[10]; a[0] a[9]; a[0]=3;a[9]=6;`

#### 4) 长整型的数组

`long int a[5];`

#### 5) 浮点型的数组（单、双）

`float a[6]; a[4]=3.14f;`

`double a[8]; a[7]=3.115926;`

#### 6) 指针数组

`char *a[10]`

`int *a[10];`

#### 7) 结构体数组

`struct stu boy[10];`

### 2.2 按维数分类

## 一维数组

```
int a[30];
```

类似于一排平房

## 二维数组

```
int a[2][30];
```

可以看成一栋楼房 有多层，每层有多个房间，也类似于数学中的矩阵

二维数组可以看成由多个一维数组构成的。

有行，有列，

## 多维数组

```
int a[4][2][10];
```

三维数组是由多个相同的二维数组构成的

```
int a[5][4][2][10];
```

# 三、数组的定义

## 3.1 一维数组的定义

格式：

数据类型 数组名[数组元素个数];

例如：

```
int a[10]; //定义了一个名为a的数组，数组中每一个元素都是int类型，一共有10个元素
```

的

//每一个元素都保存在一个变量中，每一个变量都是有数组名和数组下标组成的

//并且是从0开始的，分别是a[0] a[1] a[2]... a[9]

注意：数组元素的个数在定义的时候也可以不写，但是如果不写，必须初始化（定义的时候赋值）

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5      //定义一个一维数组
6      int a[10];
7      //通过sizeof关键字可以获取数组的大小
8      printf("sizeof(a) = %d %d\n", sizeof(a), 10 * sizeof(int));
9
10     //如果定义数组的同时赋值（初始化），可以不指定数组元素的个数，系统会根据初始化
    元素的个数自动指定数组元素的个数
11     int b[] = {10, 20, 30};
```

```

12     printf("sizeof(b) = %d\n", sizeof(b));
13
14     return 0;
15 }

```

执行结果

```

Starting C:\Users\lzx\Desktop\src\bu
Debug\debug\01_array_define
sizeof(a) = 40 40
sizeof(b) = 12
C:\Users\lzx\Desktop\src\bu

```

## 3.2 二维数组的定义

格式:

数据类型 数组名[行的个数][列的个数];

例如：

```
int a[2][4];
```

解释：

定义一个名为a的二维数组，每一个元素都是int类型

这个二维数组中包含两行四列的元素，一共有8个元素

二维数组也是连续开辟空间，访问元素是行和列都是从

0开始，分别是a[0][0] a[0][1] a[0][2] a[0][3] a[1][0] a[1][1] a[1][2] a[1][3]

注意：二维数组的下标也是可以省略的，但是有条件，在初始化时行数可以省略，但是列数不能省略

```

1 //定义一个二维数组
2 int c[2][4];
3 printf("sizeof(c) = %d %d\n", sizeof(c), 2 * 4 * sizeof(int));
4
5 //二维数组的行数可以省略，但是列数不能省略，在初始化时可以这样操作
6 //系统会根据列数自动指定行数，最终得到的函数所得到元素个数移动是列的整数倍
7 int d[][4] = {1, 2, 3, 4, 5};
8 printf("sizeof(d) = %d\n", sizeof(d));

```

执行结果

```

sizeof(c) = 32 32
sizeof(d) = 32

```

## 四、定义并初始化

## 4.1 一维数组的初始化

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     //以一维数组的初始化
6     //如果不初始化，直接使用会是随机值
7     //int a[4];
8
9     //初始化方式1：全部初始化
10    //int a[4] = {123, 78, 666, 476};
11    //如果是全部 初始化，可以不指定数组元素的个数，系统会自动分配
12    //int a[] = {10, 20, 30, 40};
13
14    //初始化方式2：局部初始化
15    //未初始化的位置的元素自动赋值为0
16    int a[4] = {10, 20};
17
18    printf("%d\n", a[0]);
19    printf("%d\n", a[1]);
20    printf("%d\n", a[2]);
21    printf("%d\n", a[3]);
22
23    return 0;
24 }
```

## 4.2 二维数组的初始化

按行初始化：

a、全部初始化

```
int a[2][2]={{1,2},{4,5}};
a[0][0] = 1; a[0][1] = 2; a[1][0] = 4,a[1][1]=5;
```

b、部分初始化

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

逐个初始化：

全部初始化：

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

部分初始化：

```
int a[2][3]={3,5,6,8};
```

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5      //二维数组的初始化
6      //int a[2][3];
7
8      //初始化方式1：按行初始化
9      //全部初始化
10     //int a[2][3] = {{10, 20, 30}, {666, 777, 888}};
11     //局部初始化
12     //没有赋值的位置的元素自动为0
13     //int a[2][3] = {{10, 20}, {666}};
14
15     //初始化方式2：逐个初始化
16     //全部初始化
17     //int a[2][3] = {1, 2, 3, 4, 5, 6};
18     //局部初始化
19     //没有赋值的位置的元素自动为0
20     int a[2][3] = {1, 2, 3};
21
22     printf("%d\n", a[0][0]);
23     printf("%d\n", a[0][1]);
24     printf("%d\n", a[0][2]);
25     printf("%d\n", a[1][0]);
26     printf("%d\n", a[1][1]);
27     printf("%d\n", a[1][2]);
28
29     return 0;
30 }
```

## 五、数组元素的引用方法

### 一维数组元素的引用方法

数组名[下标]；//下标代表数组元素在数组中的位置，注意从0开始

```
int a[10];
```

a[2];

## 二维数组元素的引用方法

数组名[行下标][列下标];

int a[3][4];

a[1][2]

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(int argc, char *argv[])
6  {
7      //一维数组的引用以及一维数组的遍历
8      int a[6] = {111, 222, 333, 444, 555, 666};
9
10     a[3] = 10000;
11
12     //一维数组的遍历
13     int i;
14     for(i = 0; i < sizeof(a) / sizeof(int); i++)
15     {
16         printf("a[%d] = %d\n", i, a[i]);
17     }
18
19     printf("*****\n");
20
21     //二维数组的引用以及二维数组的遍历
22     int b[3][4] = {1, 2, 3, 4,
23     5, 6, 7, 8,
24     9, 10, 11, 12};
25
26     b[2][0] = 666;
27
28     //二维数组的遍历
29     int m, n;
30     //外层循环控制行数
31     for(m = 0; m < 3; m++)
32     {
33         //内层循环控制列数
34         for(n = 0; n < 4; n++)
```

```

35  {
36  printf("%-4d", b[m][n]);
37  }
38  printf("\n");
39  }
40
41  return 0;
42  }

```

执行结果

Starting C:\Users\lzx\Desktop\src\build-6  
 \debug\04\_quote.exe...

a[0] = 111  
 a[1] = 222  
 a[2] = 333  
 a[3] = 10000  
 a[4] = 555  
 a[5] = 666

\*\*\*\*\*

1 2 3 4  
 5 6 7 8  
 666 10 11 12

C:\Users\lzx\Desktop\src\build-6

## 六、字符数组的定义和初始化问题

```
char c1[] = { 'c' , ' ' , 'p' , 'r' , 'o' , 'g' };
```

```
char c2[] = "c prog" ;
```

```
char a[][5] = {
    { 'B' , 'A' , 'S' , 'I' , 'C' },
    { 'd' , 'B' , 'A' , 'S' , 'E' }
};
```

```
char a[][6] = { "hello" , "world" };
```

### 字符数组的引用

1.用字符串方式赋值比用字符逐个赋值要多占1

个字节,用于存放字符串结束标志 '\0' ;

2.上面的数组c2在内存中的实际存放情况为：

'c'	' '	'p'	'r'	'o'	'g'	'\0'
-----	-----	-----	-----	-----	-----	------


注：'\0'是由C编译系统自动加上的

3.由于采用了'\0'标志，字符数组的输入输出将变得简单方便.

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     //定义一个字符数组，通过scanf函数输入字符串并输出结果
6     //通过赋值""这样的方式可以清除字符数组中的垃圾字符，让每一个元素都是\0
7     char ch[32] = "";
8
9     //数组名就是当前数组的首地址，所以scanf的第二个参数直接传数组名即可
10    scanf("%s", ch);
11
12    printf("ch = %s\n", ch);
13
14    return 0;
15 }
```

执行结果

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe



```
nihao
ch = nihao
```

注意：在QT中如果要使用scanf函数这样的从终端输入函数，需要通过命令终端输入输出  
点击左边工具栏中的“项目”，点击“build&run”下方的“run”，然后再右侧点击，  
再“run in terminal”前打对勾