

第5讲：数组

目录

1. 数组的概念
2. 一维数组的创建和初始化
3. 一维数组的使用
4. 一维数组在内存中的存储
5. sizeof计算数组元素个数
6. 二维数组的创建
7. 二维数组的初始化
8. 二维数组的使用
9. 二维数组在内存中的存储
10. C99中的变长数组
11. 数组练习

正文开始

1. 数组的概念

数组是一组相同类型元素的集合；从这个概念中我们就可以发现2个有价值的信息：

- 数组中存放的是1个或者多个数据，但是数组元素个数不能为0。
- 数组中存放的多个数据，类型是相同的。

数组分为一维数组和多维数组，多维数组一般比较多见的是二维数组。

2. 一维数组的创建和初始化

2.1 数组创建

一维数组创建的基本语法如下：

```
1 type arr_name[常量值];
```

比特就业课官网链接：<https://www.bitejiuyoke.com>

存放在数组的值被称为**数组的元素**，数组在创建的时候可以指定**数组的大小**和**数组的元素类型**。

- `type` 指定的是数组中存放数据的类型，可以是：`char`、`short`、`int`、`float` 等，也可以自定义的类型
- `arr_name` 指的是数组名的名字，这个名字根据实际情况，起的有意义就行。
- `[]` 中的**常量值**是用来指定数组的大小的，这个数组的大小是根据实际的需求指定就行。

比如：我们现在想存储某个班级的20人的数学成绩，那我们就可以创建一个数组，如下：

```
1 int math[20];
```

当然我们也可以根据需要创建其他类型和大小的数组：

```
1 char ch[8];
2 double score[10];
```

2.2 数组的初始化

有时候，数组在创建的时候，我们需要给定一些初始值，这种就称为初始化的。

那数组如何初始化呢？数组的初始化一般使用大括号，将数据放在大括号中。

```
1 //完全初始化
2 int arr[5] = {1,2,3,4,5};
3
4 //不完全初始化
5 int arr2[6] = {1}; //第一个元素初始化为1，剩余的元素默认初始化为0
6
7 //错误的初始化 - 初始化项太多
8 int arr3[3] = {1, 2, 3, 4};
```

2.3 数组的类型

数组也是有类型的，数组算是一种自定义类型，去掉数组名留下的就是数组的类型。

如下：

```
1 int arr1[10];
2 int arr2[12];
3
4 char ch[5];
```

arr1数组的类型是 `int [10]`

arr2数组的类型是 `int[12]`

ch 数组的类型是 `char [5]`

3. 一维数组的使用

学习了一维数组的**基本语法**，一维数组可以存放数据，存放数据的目的是对数据的操作，那我们如何使用一维数组呢？

3.1 数组下标

C语言规定数组是有下标的，下标是从0开始的，假设数组有n个元素，最后一个元素的下标是n-1，下标就相当于数组元素的编号，如下：

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

数组	1	2	3	4	5	6	7	8	9	10	
下标	0	1	2	3	4	5	6	7	8	9	

数组元素和下标

在C语言中数组的访问提供了一个操作符 `[]`，这个操作符叫：**下标引用操作符**。

有了下标访问操作符，我们就可以轻松的访问到数组的元素了，比如我们访问下标为7的元素，我们就可以使用 `arr[7]`，想要访问下标是3的元素，就可以使用 `arr[3]`，如下代码：

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[10] = {1,2,3,4,5,6,7,8,9,10};
6     printf("%d\n", arr[7]); //8
7     printf("%d\n", arr[3]); //4
```

```
8     return 0;
9 }
```

比特就业课主页：<https://m.cctalk.com/inst/s9yewhfr>

输出结果：

Microsoft Visual Studio 调试控制台

```
8
4
```

3.2 数组元素的打印

接下来，如果想要访问整个数组的内容，那怎么办呢？

只要我们产生数组所有元素的下标就可以了，那我们使用for循环产生0~9的下标，接下来使用下标访问就行了。

如下代码：

```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[10] = {1,2,3,4,5,6,7,8,9,10};
5     int i = 0;
6     for(i=0; i<10; i++)
7     {
8         printf("%d ", arr[i]);
9     }
10    return 0;
11 }
```

输出的结果：

选择 Microsoft Visual Studio 调试控制台

```
1 2 3 4 5 6 7 8 9 10
D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 34832) 已退出，代码为 0。
按任意键关闭此窗口。 . . .
```

3.3 数组的输入

明白了数组的访问，当然我们也根据需求，自己给数组输入想要的数，如下：

```
1 #include <stdio.h>
2 int main()
3 {
```

比特就业课官网链接：<https://www.bitejiuyoke.com>

比特就业课主页: <https://m.cctalk.com/inst/s9yewhfr>

```

4   int arr[10] = {1,2,3,4,5,6,7,8,9,10};
5   int i = 0;
6   for(i=0; i<10; i++)
7   {
8       scanf("%d", &arr[i]);
9   }
10  for(i=0; i<10; i++)
11  {
12      printf("%d ", arr[i]);
13  }
14  return 0;
15 }

```

输入个输出结果：

C:\ Microsoft Visual Studio 调试控制台

```

2 1 4 3 6 5 8 7 0 9
2 1 4 3 6 5 8 7 0 9
D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 55928) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

```

4. 一维数组在内存中的存储

有了前面的知识，我们其实使用数组基本没有什么障碍了，如果我们要深入了解数组，我们最好能了解一下数组在内存中的存储。

依次打印数组元素的地址：

```

1  #include <stdio.h>
2  int main()
3  {
4      int arr[10] = {1,2,3,4,5,6,7,8,9,10};
5      int i = 0;
6      for(i=0; i<10; i++)
7      {
8          printf("&arr[%d] = %p\n ", i, &arr[i]);
9      }
10     return 0;
11 }

```

输出结果我们看看：

```
&arr[0] = 0133F8D0
&arr[1] = 0133F8D4
&arr[2] = 0133F8D8
&arr[3] = 0133F8DC
&arr[4] = 0133F8E0
&arr[5] = 0133F8E4
&arr[6] = 0133F8E8
&arr[7] = 0133F8EC
&arr[8] = 0133F8F0
&arr[9] = 0133F8F4
```

从输出的结果我们分析，数组随着下标的增长，地址是由小到大变化的，并且我们发现每两个相邻的元素之间相差4（因为一个整型是4个字节）。所以我们得出结论：**数组在内存中是连续存放的**。这就为后期我们使用指针访问数组奠定了基础（在讲指针的时候我们在讲，这里暂且记住就行）。

	数组	1	2	3	4	5	6	7	8	9	10
	下标	0	1	2	3	4	5	6	7	8	9

数组元素在内存中是连续存放的

5. sizeof计算数组元素个数

在遍历数组的时候，我们经常想知道数组的元素个数，那C语言中有办法使用程序计算数组元素个数吗？

答案是有的，可以使用**sizeof**。

sizeof 中C语言是一个关键字，是可以计算类型或者变量大小的，其实 **sizeof** 也可以计算数组的大小。

比如：

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[10] = {0};
6     printf("%d\n", sizeof(arr));
7     return 0;
8 }
```

这里输出的结果是40，计算的是数组所占内存空间的总大小，单位是**字节**。

我们又知道数组中所有元素的类型都是相同的，那只要计算出一个元素所占字节的个数，数组的元素个数就能算出来。这里我们选择第一个元素算大小就可以。

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[10] = {0};
6     printf("%d\n", sizeof(arr[0])); //计算一个元素的大小，单位是字节
7     return 0;
8 }
```

接下来就能计算出数组的元素个数：

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[10] = {0};
6     int sz = sizeof(arr)/sizeof(arr[0]);
7     printf("%d\n", sz);
8     return 0;
9 }
```

这里的结果是：10，表示数组有10个元素。

以后在代码中需要数组元素个数的地方就不用固定写死了，使用上面的计算，不管数组怎么变化，计算出的大小也就随着变化了。

6. 二维数组的创建

6.1 二维数组得概念

前面学习的数组被称为一维数组，数组的元素都是内置类型的，如果我们把一维数组做为数组的元素，这时候就是**二维数组**，二维数组作为数组元素的数组被称为**三维数组**，二维数组以上的数组统称为**多维数组**。

6.2 二维数组的创建

```
1 type arr_name[常量值1][常量值2];  
2  
3 例如:  
4 int arr[3][5];  
5 double data[2][8];
```

- 3表示数组有3行
- 5表示每一行有5个元素
- int 表示数组的每个元素是整型类型
- arr 是数组名，可以根据自己的需要指定名字

7. 二维数组的初始化

那二维数组如何初始化呢？像一维数组一样，也是使用大括号初始化的。

```
1 int arr1[3][5] = {1,2};
2 int arr2[3][5] = {0};
```


	0	1	2	3	4		0	1	2	3	4
0	1	2	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	2	0	0	0	0	0
arr1数组							arr2数组				

7.2 完全初始化

```
1 int arr3[3][5] = {1,2,3,4,5, 2,3,4,5,6, 3,4,5,6,7};
```

	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7
arr3数组					

7.3 按照行初始化

```
1 int arr4[3][5] = {{1,2},{3,4},{5,6}};
```

	0	1	2	3	4
0	1	2	0	0	0
1	3	4	0	0	0
2	5	6	0	0	0
arr4数组					

7.4 初始化时省略行, 但是不能省略列

```
1 int arr5[][5] = {1,2,3};
```

```
2 int arr6[][5] = {1,2,3,4,5,6,7};
3 int arr7[][5] = {{1,2}, {3,4}, {5,6}};
```

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

arr5数组

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	0	0	0
2	0	0	0	0	0

arr6数组

	0	1	2	3	4
0	1	2	0	0	0
1	3	4	0	0	0
2	5	6	0	0	0

arr7数组

8. 二维数组的使用

8.1 二维数组的下标

当我们掌握了二维数组的创建和初始化，那我们怎么使用二维数组呢？

其实二维数组访问也是使用下标的形式的，二维数组是有行和列的，只要锁定了行和列就能唯一锁定数组中的一个元素。

C语言规定，二维数组的行是从0开始的，列也是从0开始的，如下所示：

```
1 int arr[3][5] = {1,2,3,4,5, 2,3,4,5,6, 3,4,5,6,7};
```

	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	4	5	6	7

arr数组

图中最右侧绿色的数字表示行号，第一行蓝色的数字表示列号，都是从0开始的，比如，我们说：第2行，第4列，快速就能定位出7。

```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[3][5] = {1,2,3,4,5, 2,3,4,5,6, 3,4,5,6,7};
5     printf("%d\n", arr[2][4]);
6     return 0;
```

输出的结果如下：

Microsoft Visual Studio 调试控制台

7

D:\code\2022\test\test_6_3\Debug\test_6_3.exe (进程 61388) 已退出，代码为 0。
按任意键关闭此窗口. . .

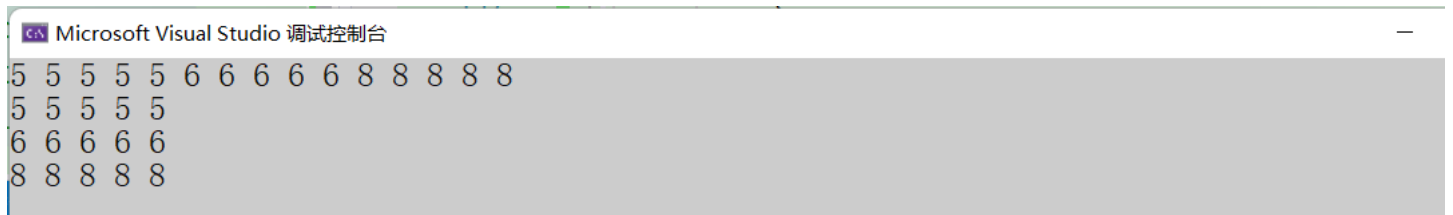
8.2 二维数组的输入和输出

访问二维数组的单个元素我们知道了，那如何访问整个二维数组呢？

其实我们只要能够按照一定的规律产生所有的行和列的数字就行；以上一段代码中的arr数组为例，行的选择范围是0~2，列的取值范围是0~4，所以我们可以借助循环实现生成所有的下标。

```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[3][5] = {1,2,3,4,5, 2,3,4,5,6, 3,4,5,6,7};
5     int i = 0; //遍历行
6     //输入
7     for(i=0; i<3; i++) //产生行号
8     {
9         int j = 0;
10        for(j=0; j<5; j++) //产生列号
11        {
12            scanf("%d", &arr[i][j]); //输入数据
13        }
14    }
15    //输出
16    for(i=0; i<3; i++) //产生行号
17    {
18        int j = 0;
19        for(j=0; j<5; j++) //产生列号
20        {
21            printf("%d ", arr[i][j]); //输出数据
22        }
23        printf("\n");
24    }
25    return 0;
26 }
```

输入和输出的结果：



```
Microsoft Visual Studio 调试控制台
5 5 5 5 5 6 6 6 6 6 8 8 8 8 8
5 5 5 5 5
6 6 6 6 6
8 8 8 8 8
```

9. 二维数组在内存中的存储

像一维数组一样，我们如果想研究二维数组在内存中的存储方式，我们也是可以打印出数组所有元素的地址的。代码如下：

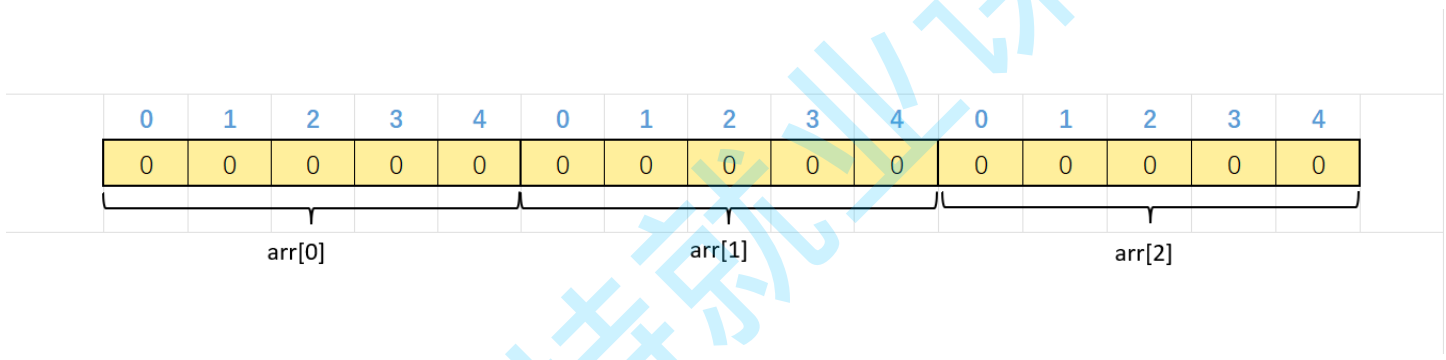
```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[3][5] = { 0 };
5     int i = 0;
6     int j = 0;
7     for (i = 0; i < 3; i++)
8     {
9         for (j = 0; j < 5; j++)
10        {
11            printf("&arr[%d][%d] = %p\n", i, j, &arr[i][j]);
12        }
13    }
14    return 0;
15 }
```

输出的结果：

```
&arr[0][0] = 00CFF860
&arr[0][1] = 00CFF864
&arr[0][2] = 00CFF868
&arr[0][3] = 00CFF86C
&arr[0][4] = 00CFF870
&arr[1][0] = 00CFF874
&arr[1][1] = 00CFF878
&arr[1][2] = 00CFF87C
&arr[1][3] = 00CFF880
&arr[1][4] = 00CFF884
&arr[2][0] = 00CFF888
&arr[2][1] = 00CFF88C
&arr[2][2] = 00CFF890
&arr[2][3] = 00CFF894
&arr[2][4] = 00CFF898
```

从输出的结果来看，每一行内部的每个元素都是相邻的，地址之间相差4个字节，跨行位置处的两个元素（如：arr[0][4]和arr[1][0]）之间也是差4个字节，所以**二维数组中的每个元素都是连续存放的**。

如下图所示：



二维数组的每一行在内存中连续存放

了解清楚二维数组在内存中的布局，有利于我们后期使用指针来访问数组的学习。

10. C99中的变长数组

在C99标准之前，C语言在创建数组的时候，数组大小的指定只能使用常量、常量表达式，或者如果我们初始化数据的话，可以省略数组大小。

如：

```
1 int arr1[10];
2 int arr2[3+5];
3 int arr3[] = {1,2,3};
```

这样的语法限制，让我们创建数组就不够灵活，有时候数组大了浪费空间，有时候数组又小了不够用的。

C99中给一个**变长数组（variable-length array，简称 VLA）**的新特性，允许我们可以使用变量指定数组大小。

请看下面的代码：

```
1 int n = a+b;
2 int arr[n];
```

上面示例中，数组 `arr` 就是变长数组，因为它的长度取决于变量 `n` 的值，编译器没法事先确定，只有运行时才能知道 `n` 是多少。

变长数组的根本特征，就是数组长度只有运行时才能确定，**所以变长数组不能初始化**。它的好处是程序员不必在开发时，随意为数组指定一个估计的长度，程序可以在运行时为数组分配精确的长度。有一个比较迷惑的点，变长数组的意思是数组的大小是可以使用变量来指定的，在程序运行的时候，根据变量的大小来指定数组的元素个数，而不是说数组的大小是可变的。数组的大小一旦确定就不能再变化了。

遗憾的是在VS2022上，虽然支持大部分C99的语法，没有支持C99中的变长数组，没法测试；下面是我在gcc编译器上测试，可以看一下。

```
1 #include <stdio.h>
2 int main()
3 {
4     int n = 0;
5     scanf("%d", &n); //根据输入数值确定数组的大小
6     int arr[n];
7     int i = 0;
8     for (i = 0; i < n; i++)
9     {
10         scanf("%d", &arr[i]);
11     }
12     for (i = 0; i < n; i++)
13     {
14         printf("%d ", arr[i]);
15     }
16     return 0;
17 }
```

第一次测试，我给n中输入5，然后输入5个数字在数组中，并正常输出

第二次测试，我给n中输入10，然后输入10个数字在数组中，并正常输出

调试控制台 终端

```
PS D:\code\vscode\test_12_13_1> gcc .\test.c -o test
PS D:\code\vscode\test_12_13_1> .\test.exe
5
1 2 3 4 5
1 2 3 4 5
PS D:\code\vscode\test_12_13_1> .\test.exe
10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
PS D:\code\vscode\test_12_13_1>
PS D:\code\vscode\test_12_13_1>
PS D:\code\vscode\test_12_13_1> █
```

11. 数组练习

练习1：多个字符从两端移动，向中间汇聚

编写代码，演示多个字符从两端移动，向中间汇聚

```
1 #include <stdio.h>
2 int main()
3 {
4     char arr1[] = "welcome to bit...";
5     char arr2[] = "#####";
6     int left = 0;
7     int right = strlen(arr1)-1;
8     printf("%s\n", arr2);
9     while(left<=right)
10    {
11        Sleep(1000);
12        arr2[left] = arr1[left];
13        arr2[right] = arr1[right];
14        left++;
15        right--;
16        printf("%s\n", arr2);
17    }
18    return 0;
19 }
20
```

练习2：二分查找

在一个升序的数组中查找指定的数字n，很容易想到的方法就是遍历数组，但是这种方法效率比较低。

比如我买了一双鞋，你好奇问我多少钱，我说不超过300元。你还是好奇，你想知道到底多少，我就让你猜，你会怎么猜？你会1，2，3，4...这样猜吗？显然很慢；一般你都会猜中间数字，比如：150，然后看大了还是小了，这就是二分查找，也叫折半查找。

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int arr[] = {1,2,3,4,5,6,7,8,9,10};
6     int left = 0;
7     int right = sizeof(arr)/sizeof(arr[0])-1;
8     int key = 7; //要找的数字
9     int mid = 0; //记录中间元素的下标
10    int find = 0;
11    while(left<=right)
12    {
13        mid = (left+right)/2;
14        if(arr[mid]>key)
15        {
16            right = mid-1;
17        }
18        else if(arr[mid] < key)
19        {
20            left = mid+1;
21        }
22        else
23        {
24            find = 1;
25            break;
26        }
27    }
28    if(1 == find )
29        printf("找到了,下标是%d\n", mid);
30    else
31        printf("找不到\n");
32 }
```

求中间元素的下标，使用 `mid = (left+right)/2`，如果left和right比较大的时候可能存在问题，可以使用下面的方式：

```
1 mid = left+(right-left)/2;
```


完

比特就业课