

第12讲：深入理解指针(2)

目录

1. 数组名的理解
2. 使用指针访问数组
3. 一维数组传参的本质
4. 冒泡排序
5. 二级指针
6. 指针数组
7. 指针数组模拟二维数组

正文开始

1. 数组名的理解

在上一个章节我们在使用指针访问数组的内容时，有这样的代码：

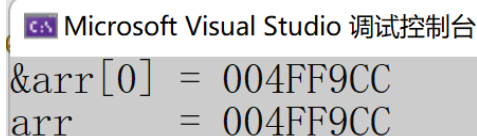
```
1 int arr[10] = {1,2,3,4,5,6,7,8,9,10};
2 int *p = &arr[0];
```

这里我们使用 `&arr[0]` 的方式拿到了数组第一个元素的地址，但是其实数组名本来就是地址，而且是数组首元素的地址，我们来做个测试。

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

输出结果：

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



```
C:\ Microsoft Visual Studio 调试控制台
&arr[0] = 004FF9CC
arr      = 004FF9CC
```

我们发现数组名和数组首元素的地址打印出的结果一模一样，**数组名就是数组首元素(第一个元素)的地址**。

这时候有同学会有疑问？数组名如果是数组首元素的地址，那下面的代码怎么理解呢？

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

输出的结果是：40，如果arr是数组首元素的地址，那输出应该的是4/8才对。

其实数组名就是数组首元素(第一个元素)的地址是对的，但是有两个例外：

- **sizeof(数组名)**，sizeof中单独放数组名，这里的数组名表示整个数组，计算的是整个数组的大小，单位是字节
- **&数组名**，这里的数组名表示整个数组，取出的是**整个数组的地址**（整个数组的地址和数组首元素的地址是有区别的）

除此之外，任何地方使用数组名，数组名都表示首元素的地址。

这时有好奇的同学，再试一下这个代码：

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

三个打印结果一模一样，这时候又纳闷了，那arr和&arr有啥区别呢？

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

输出结果：

```
1 &arr[0]    = 0077F820
2 &arr[0]+1  = 0077F824
3 arr       = 0077F820
4 arr+1     = 0077F824
5 &arr      = 0077F820
6 &arr+1    = 0077F848
```

这里我们发现&arr[0]和&arr[0]+1相差4个字节，arr和arr+1 相差4个字节，是因为&arr[0] 和 arr 都是首元素的地址，+1就是跳过一个元素。

但是&arr 和 &arr+1相差40个字节，这就是因为&arr是数组的地址，+1 操作是跳过整个数组的。

到这里大家应该搞清楚数组名的意义了吧。

数组名是数组首元素的地址，但是有2个例外。

2. 使用指针访问数组

有了前面知识的支持，再结合数组的特点，我们就可以很方便的使用指针访问数组了。

```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[10] = {0};
5     //输入
```

```
6     int i = 0;
7     int sz = sizeof(arr)/sizeof(arr[0]);
8     //输入
9     int* p = arr;
10    for(i=0; i<sz; i++)
11    {
12        scanf("%d", p+i);
13        //scanf("%d", arr+i);//也可以这样写
14    }
15    //输出
16    for(i=0; i<sz; i++)
17    {
18        printf("%d ", *(p+i));
19    }
20    return 0;
21 }
```

这个代码搞明白后,我们再试一下,如果我们在分析一下,数组名arr是数组首元素的地址,可以赋值给p,其实数组名arr和p在这里是等价的。那我们可以使用arr[i]可以访问数组的元素,那p[i]是否也可以访问数组呢?

```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[10] = {0};
5     //输入
6     int i = 0;
7     int sz = sizeof(arr)/sizeof(arr[0]);
8     //输入
9     int* p = arr;
10    for(i=0; i<sz; i++)
11    {
12        scanf("%d", p+i);
13        //scanf("%d", arr+i);//也可以这样写
14    }
15    //输出
16    for(i=0; i<sz; i++)
17    {
18        printf("%d ", p[i]);
19    }
20    return 0;
21 }
```

在第18行的地方,将*(p+i)换成p[i]也是能够正常打印的,所以本质上p[i]是等价于*(p+i)。

同理arr[i] 应该等价于 *(arr+i)，数组元素的访问在编译器处理的时候，也是转换成首元素的地址+偏移量求出元素的地址，然后解引用来访问的。

3. 一维数组传参的本质

数组我们学过了，之前也讲了，数组是可以传递给函数的，这个小节我们讨论一下数组传参的本质。

首先从一个问题开始，我们之前都是在函数外部计算数组的元素个数，那我们可以把函数传给一个函数后，函数内部求数组的元素个数吗？

```
1 #include <stdio.h>
2
3 void test(int arr[])
4 {
5     int sz2 = sizeof(arr)/sizeof(arr[0]);
6     printf("sz2 = %d\n", sz2);
7 }
8 int main()
9 {
10    int arr[10] = {1,2,3,4,5,6,7,8,9,10};
11    int sz1 = sizeof(arr)/sizeof(arr[0]);
12    printf("sz1 = %d\n", sz1);
13    test(arr);
14    return 0;
15 }
```

输出的结果：

Microsoft Visual Studio 调试控制台

```
sz1 = 10
sz2 = 1
```

我们发现在函数内部是没有正确获得数组的元素个数。

这就要学习数组传参的本质了，上个小节我们学习了：数组名是数组首元素的地址；那么在数组传参的时候，传递的是数组名，也就是说**本质上数组传参本质上传递的是数组首元素的地址**。

所以函数形参的部分理论上应该使用指针变量来接收首元素的地址。那么在函数内部我们写

`sizeof(arr)` 计算的是一个地址的大小（单位字节）而不是数组的大小（单位字节）。正是因为函数的参数部分是本质是指针，所以在函数内部是没办法求的数组元素个数的。

```
1 void test(int arr[])//参数写成数组形式，本质上还是指针
2 {
```

```
3     printf("%d\n", sizeof(arr));
4 }
5
6 void test(int* arr) //参数写成指针形式
7 {
8     printf("%d\n", sizeof(arr)); //计算一个指针变量的大小
9 }
10 int main()
11 {
12     int arr[10] = {1,2,3,4,5,6,7,8,9,10};
13     test(arr);
14     return 0;
15 }
```

总结：一维数组传参，形参的部分可以写成数组的形式，也可以写成指针的形式。

4. 冒泡排序

冒泡排序的核心思想就是：两两相邻的元素进行比较。

```
1 //方法1
2 void bubble_sort(int arr[], int sz) //参数接收数组元素个数
3 {
4     int i = 0;
5     for(i=0; i<sz-1; i++)
6     {
7         int j = 0;
8         for(j=0; j<sz-i-1; j++)
9         {
10             if(arr[j] > arr[j+1])
11             {
12                 int tmp = arr[j];
13                 arr[j] = arr[j+1];
14                 arr[j+1] = tmp;
15             }
16         }
17     }
18 }
19
20 int main()
21 {
22     int arr[] = {3,1,7,5,8,9,0,2,4,6};
23     int sz = sizeof(arr)/sizeof(arr[0]);
24     bubble_sort(arr, sz);
```

```
25     for(i=0; i<sz; i++)
26     {
27         printf("%d ", arr[i]);
28     }
29     return 0;
30 }
31
32
33 //方法2 - 优化
34 void bubble_sort(int arr[], int sz)//参数接收数组元素个数
35 {
36     int i = 0;
37     for(i=0; i<sz-1; i++)
38     {
39         int flag = 1;//假设这一趟已经有序了
40         int j = 0;
41         for(j=0; j<sz-i-1; j++)
42         {
43             if(arr[j] > arr[j+1])
44             {
45                 flag = 0;//发生交换就说明, 无序
46                 int tmp = arr[j];
47                 arr[j] = arr[j+1];
48                 arr[j+1] = tmp;
49             }
50         }
51         if(flag == 1)//这一趟没交换就说明已经有序, 后续无序排序了
52             break;
53     }
54 }
55
56 int main()
57 {
58     int arr[] = {3,1,7,5,8,9,0,2,4,6};
59     int sz = sizeof(arr)/sizeof(arr[0]);
60     bubble_sort(arr, sz);
61     for(i=0; i<sz; i++)
62     {
63         printf("%d ", arr[i]);
64     }
65     return 0;
66 }
```

5. 二级指针

指针变量也是变量, 是变量就有地址, 那指针变量的地址存放在哪里?

```
#include <stdio.h>
```

```
int main()
```

```
{
```

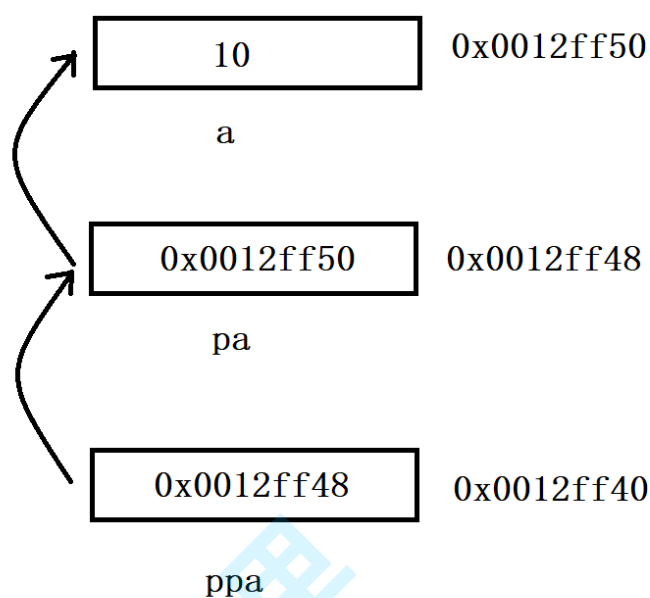
```
    int a = 10;
```

```
    int * pa = &a;
```

```
    int ** ppa = &pa;
```

```
    return 0;
```

```
}
```



二级指针画图

对于二级指针的运算有：

- `*ppa` 通过对 `ppa` 中的地址进行解引用，这样找到的是 `pa`，`*ppa` 其实访问的就是 `pa`。

```
1 int b = 20;  
2 *ppa = &b; // 等价于 pa = &b;
```

- `**ppa` 先通过 `*ppa` 找到 `pa`，然后对 `pa` 进行解引用操作：`*pa`，那找到的是 `a`。

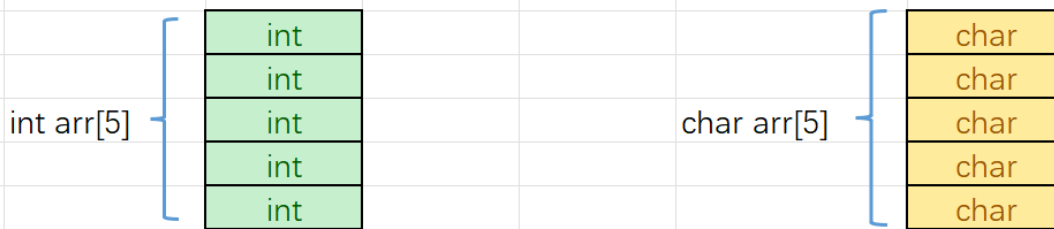
```
1 **ppa = 30;  
2 // 等价于 *pa = 30;  
3 // 等价于 a = 30;
```

6. 指针数组

指针数组是指针还是数组？

我们类比一下，整型数组，是存放整型的数组，字符数组是存放字符的数组。

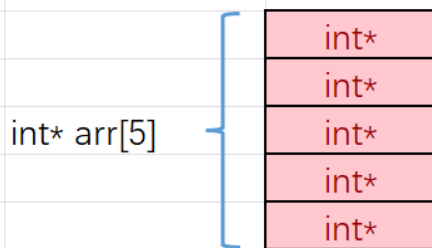
那指针数组呢？是存放指针的数组。



整型数组和字符数组

数组指针的每个元素都是用来存放地址（指针）的。

如下图：



数组指针的每个元素是地址，又可以指向一块区域。

7. 指针数组模拟二维数组

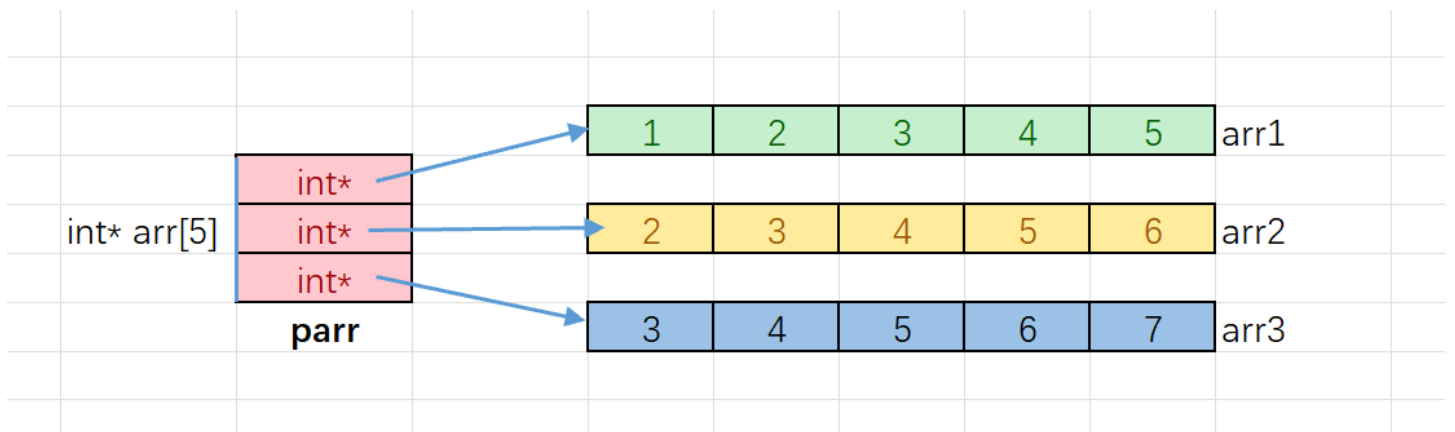
```

1  #include <stdio.h>
2  int main()
3  {
4      int arr1[] = {1,2,3,4,5};
5      int arr2[] = {2,3,4,5,6};
6      int arr3[] = {3,4,5,6,7};
7      //数组名是数组首元素的地址，类型是int*的，就可以存放在parr数组中
8      int* parr[3] = {arr1, arr2, arr3};
9      int i = 0;
10     int j = 0;
11     for(i=0; i<3; i++)
12     {
13         for(j=0; j<5; j++)
14         {
15             printf("%d ", parr[i][j]);
16         }
17         printf("\n");
18     }

```

```
19     return 0;  
20 }
```

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



parr数组的画图演示

parr[i]是访问parr数组的元素，parr[i]找到的数组元素指向了整型一维数组，parr[i][j]就是整型一维数组中的元素。

上述的代码模拟出二维数组的效果，实际上并非完全是二维数组，因为每一行并非是连续的。

完