第13讲：深入理解指针(3)

目录

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：

代码块

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

代码块

```
//测试环境:X86
#include <stdio.h>
int main()
{
    int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
    printf("&arr[0] = %p\n", &arr[0]);
    printf("arr     = %p\n", arr);
    return 0;
}
```

输出结果：

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？

代码块

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

输出的结果是：40，如果arr是数组首元素的地址，那输出应该的应该是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：

代码块

```
//测试环境:X86
#include <stdio.h>
int main()
{
    int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
    printf("&arr[0] = %p\n", &arr[0]);
    printf("arr     = %p\n", arr);
```

```
    printf("&arr    = %p\n", &arr);
    return 0;
}
```

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?

代码块

```
//测试环境:X86
#include <stdio.h>
int main()
{
    int arr[10] = { 1,2,3,4,5,6,7,8,9,10 };
    printf("&arr[0]   = %p\n", &arr[0]);
    printf("&arr[0]+1 = %p\n", &arr[0]+1);
    printf("arr       = %p\n", arr);
    printf("arr+1     = %p\n", arr+1);
    printf("&arr      = %p\n", &arr);
    printf("&arr+1    = %p\n", &arr+1);
    return 0;
}
```

输出结果:

代码块

```
&arr[0]   = 0077F820
&arr[0]+1 = 0077F824
arr       = 0077F820
arr+1     = 0077F824
&arr      = 0077F820
&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。

代码块

```c
#include <stdio.h>
int main()
{
    int arr[10] = {0};
    //输入
    int i = 0;
    int sz = sizeof(arr)/sizeof(arr[0]);
    //输入
    int* p = arr;
    for(i = 0; i < sz; i++)
    {
        scanf("%d", p+i);
        //scanf("%d", arr+i);//也可以这样写
    }
    //输出
    for(i = 0; i < sz; i++)
    {
        printf("%d ", *(p+i));
    }
    return 0;
}
```

这个代码搞明白后，我们再试一下，如果我们再分析一下，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?

代码块

```c
#include <stdio.h>
int main()
{
    int arr[10] = {0};
    //输入
    int i = 0;
    int sz = sizeof(arr)/sizeof(arr[0]);
    //输入
    int* p = arr;
    for(i = 0; i < sz; i++)
    {
        scanf("%d", p+i);
        //scanf("%d", arr+i);//也可以这样写
    }
```

```
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，那我们可以把数组传给一个函数后，函数内部求数组的元素个数吗？

代码块

```
1   #include <stdio.h>
2   //测试环境是x86
3   void test(int arr[])
4   {
5       int sz2 = sizeof(arr)/sizeof(arr[0]);
6       printf("sz2 = %d\n", sz2);
7   }
8
9   int main()
10  {
11      int arr[10] = {1,2,3,4,5,6,7,8,9,10};
12      int sz1 = sizeof(arr)/sizeof(arr[0]);
13      printf("sz1 = %d\n", sz1);
14      test(arr);
15      return 0;
16  }
```

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。

代码块

```
void test(int arr[])//参数写成数组形式，本质上还是指针
{
    printf("%d\n", sizeof(arr));
}

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

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。

代码块

```
//方法1
void bubble_sort(int arr[], int sz)//参数接收数组元素个数
{
    int i = 0;
    for(i = 0; i < sz-1; i++)
    {
        int j = 0;
        for(j = 0; j < sz-i-1; j++)
        {
            if(arr[j] > arr[j+1])
            {
```

```
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
        }
    }
}

int main()
{
    int arr[] = {3,1,7,5,8,9,0,2,4,6};
    int sz = sizeof(arr)/sizeof(arr[0]);
    bubble_sort(arr, sz);
    int i = 0;
    for(i = 0; i < sz; i++)
    {
        printf("%d ", arr[i]);
    }
    return 0;
}


//方法2 - 优化
void bubble_sort(int arr[], int sz)//参数接收数组元素个数
{
    int i = 0;
    for(i = 0; i < sz-1; i++)
    {
        int flag = 1;//假设这一趟已经有序了
        int j = 0;
        for(j = 0; j < sz-i-1; j++)
        {
            if(arr[j] > arr[j+1])
            {
                flag = 0;//发生交换就说明，无序
                int tmp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = tmp;
            }
        }
        if(flag == 1)//这一趟没交换就说明已经有序，后续无序排序了
            break;
    }
}

int main()
{
```

```c
    int arr[] = {3,1,7,5,8,9,0,2,4,6};
    int sz = sizeof(arr)/sizeof(arr[0]);
    bubble_sort(arr, sz);
    int i = 0;
    for(i = 0; i < sz; i++)
    {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

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？

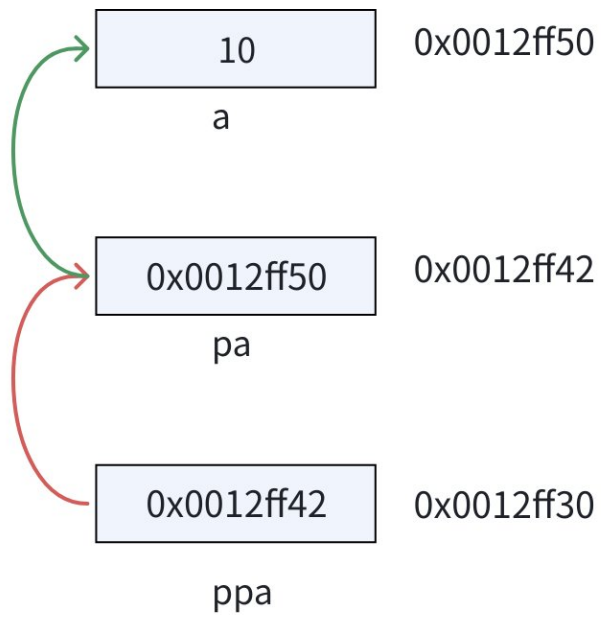
这就是 `二级指针` 。

代码块

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int *pa = &a;
    int** ppa = &pa;

    return 0;
}
```



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` .

代码块

```c
int b = 20;
*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` .

代码块

```
**ppa = 30;
//等价于*pa = 30;
//等价于a = 30;
```

6. 指针数组

指针数组是指针还是数组?

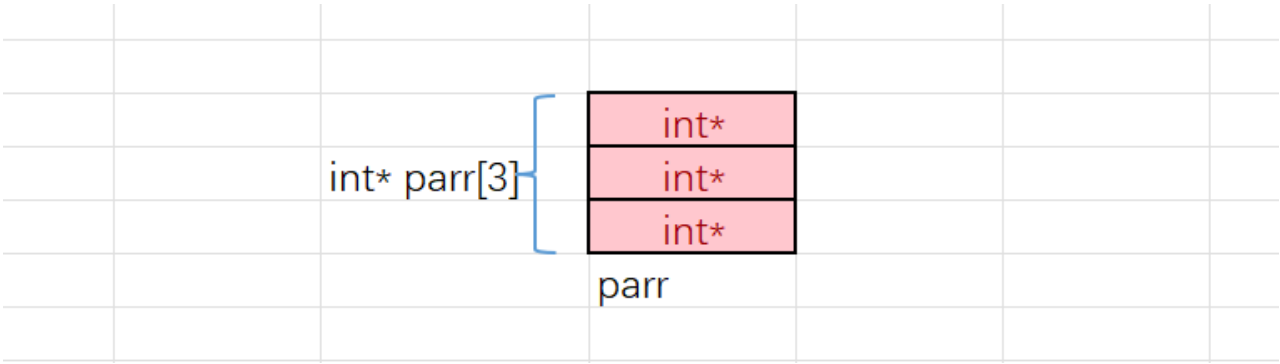
我们类比一下，整型数组，是存放整型的数组，字符数组是存放字符的数组。

那指针数组呢？是存放指针的数组。

整型数组和字符数组

指针数组的每个元素都是用来存放地址（指针）的。

如下图：



`parr` 是一个数组，数组有3个元素，每个元素的类型是 `int*` ；

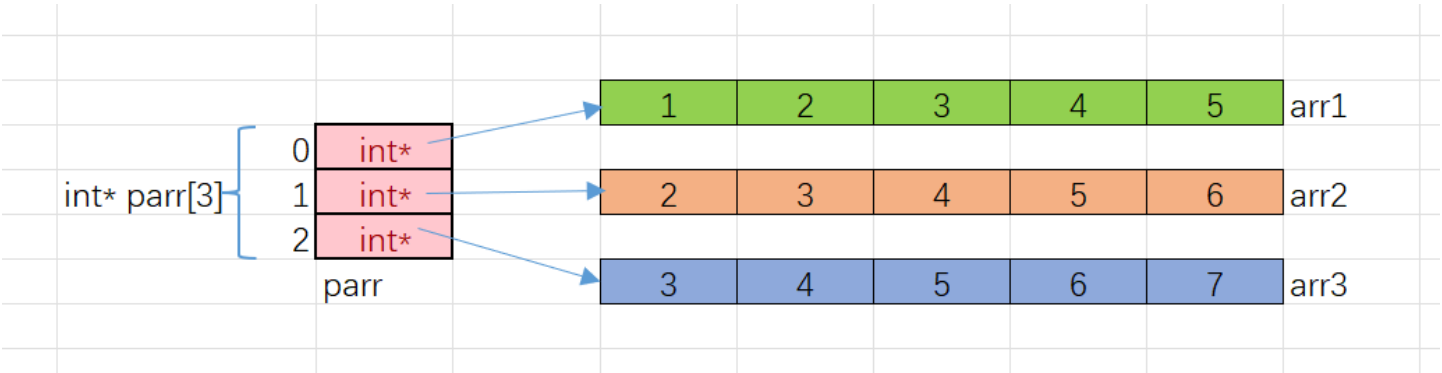
指针数组的每个元素是地址，又可以指向一块区域。

7. 指针数组模拟二维数组

代码块

```
#include <stdio.h>
int main()
```

```c
{
    int arr1[] = {1,2,3,4,5};
    int arr2[] = {2,3,4,5,6};
    int arr3[] = {3,4,5,6,7};
    //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
    int* parr[3] = {arr1, arr2, arr3};
    int i = 0;
    int j = 0;
    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 5; j++)
        {
            printf("%d ", parr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



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