

C/C++中指针与数组、指针数组与数组指针

原创 喵的波波鱼 2019-03-14 22:34:42 944 ★ 收藏 3

版权

分类专栏： C/C++重点总结

1.指针与数组

(1) 指针与数组的联系

数组名和指针之间，经常会交替使用这两个变量，

可以把一个指针当成数组来使用，或者是把数组名赋值给指针，通过指针来访问数组成员变量，

代码示例：

```
#include <iostream>

int main()
{
    int array[] = {1,2,3,4,5};
    int *p_array = array;

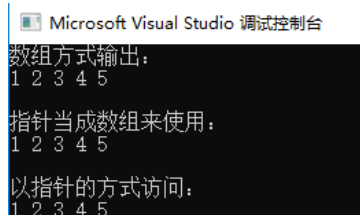
    int len = sizeof(array) / sizeof(array[0]);

    printf("数组方式输出: \n");
    for (int i = 0; i < len; i++)
    {
        printf("%d ", array[i]);
    }

    printf("\n\n指针当成数组来使用: \n");
    for (int i = 0; i < len; i++)
    {
        printf("%d ", p_array[i]);
    }

    printf("\n\n以指针的方式访问: \n");
    int *temp_ptr = p_array;
    for (int i = 0; i < len; i++)
    {
        printf("%d ", *temp_ptr++);
    }
}
```

运行结果：



```
Microsoft Visual Studio 调试控制台
数组方式输出:
1 2 3 4 5
指针当成数组来使用:
1 2 3 4 5
以指针的方式访问:
1 2 3 4 5
```

(2) 指针与数组的区别

《1》数组是固定大小的，数组一经定义，那么数组名就是一个指向数组第一个元素类型的常量指针，也就是说数组名是不允许更改的，而指针可以更改。

示例代码：

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

int main()
```

```

{
    int array[] = {1,2,3,4,5};
    int a = 0;
    int *p_array = array;

    //printf("%p", array++); //编译器报错，不能更改array
    printf("%p", p_array++); //正确，指针可修改
}

```

《2》对数组名取&和对指针取&的意义不同

假设为int数组

对数组名取&：类型变为int[num]*，即指向数组的指针，为**整个数组的首地址**

对指针名取&：类型变为int**，即指向指针的指针，为**指针变量自身的地址**

示例代码：

```

#include <iostream>
using namespace std;

int main()
{
    int array[] = {1,2,3,4,5};
    int a = 0;
    int *p_array = array;

    //重新定义一个用指针定义的数组
    int len = sizeof(array) / sizeof(array[0]);

    //打印每个数组元素的地址
    for (int i = 0; i < len; i++)
    {
        printf("&p_array[%d] =%p\n", i, &array[i]);
    }
    //打印a的地址
    printf("&a =%p\n", &a);

    printf("&p_array =%p\n", &p_array); //打印指针变量p_array的在内存中的地址
    printf("p_array =%p\n", p_array); //打印p_array存储的值

    printf("\n");

    printf("&array =%p\n", &array); //打印的是数组的首地址，&array的类型是int[5]*，即指向数组的指针
    printf("&array+1 =%p\n", &array+1); //因为它本身是指向int[5]*，所以该指针+1，就加了4*5=20个字节

    printf("&array[0] =%p\n", &array[0]); //打印的是数组第一个元素的首地址，&array[0]的类型是int*，即指向int的指针
    printf("&array[0]+1 =%p\n", &array[0]+1); //因为它本身是指向int*，所以该指针+1，就加了4个字节

    printf("array =%p\n", array); //打印的是数组第一个元素的首地址，array的类型是int*，即指向int的指针
    printf("array+1 =%p\n", array+1); //因为它本身是指向int*，所以该指针+1，就加了4个字节

}

```

运行结果：

VS2017下：

```
Microsoft Visual Studio 调试控制
&p_array[0] =00EF FE2C
&p_array[1] =00EF FE30
&p_array[2] =00EF FE34
&p_array[3] =00EF FE38
&p_array[4] =00EF FE3C
&a =00EF FE20
&p_array =00EF FE14
p_array =00EF FE2C

&array =00EF FE2C
&array+1 =00EF FE40
&array[0] =00EF FE2C
&array[0]+1 =00EF FE30
array =00EF FE2C
array+1 =00EF FE30
```

Ubuntu的g++下：

```
boboyu@ubuntu:~/Desktop$ ./test
&p_array[0] =0x7ffd614afbb0
&p_array[1] =0x7ffd614afbb4
&p_array[2] =0x7ffd614afbb8
&p_array[3] =0x7ffd614afbbc
&p_array[4] =0x7ffd614afbc0
&a =0x7ffd614afb9c
&p_array =0x7ffd614afba8
p_array =0x7ffd614afbb0

&array =0x7ffd614afbb0
&array+1 =0x7ffd614afbc4
&array[0] =0x7ffd614afbb0
&array[0]+1 =0x7ffd614afbb4
array =0x7ffd614afbb0
array+1 =0x7ffd614afbb4
```

《3》当对数组名使用sizeof时，得到的是数组元素的个数乘元素类型的字节数，对指针sizeof得到的是指针类型的字节数。

示例代码：

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

int main()
{
    int array[] = {1,2,3,4,5};
    int a = 0;
    int *p_array = array;

    printf("sizeof(array) = %d", sizeof(array));
    printf("\nsizeof(p_array) = %d", sizeof(p_array));
}
```

运行结果：

```
Microsoft Visual Studio 调试
sizeof(array) = 20
sizeof(p_array) = 4
```

总结:可以发现，对于数组名在不同场景下其实有两种语义：

- 1.直接使用数组名，数组名代表的是指向数组第一个元素的指针，所以在+1之后也只是加了一个元素类型的字节
- 2.对数组名取&或取sizeof，数组名代表的是整个数组，而取地址之后就变成了指向数组的指针，所以在+1之后是加了一整个数组的字节

2.指针数组与数组指针

(1) 数组指针

定义 int (*p)[n];

()优先级高，首先说明p是一个指针，指向一个整型的一维数组，这个一维数组的长度是n，也可以说是p的步长。也就是说执行p+1时，p要跨过n个int数据的长度。

示例代码：

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

void test_array_pointer()
{
    printf("数组指针\n");
    int(*p)[3]; //定义一个数组指针，指向含有三个元素的一维数组
    int a[2][3]; //定义一个二维数组

    printf("sizeof(p) = %d\n", sizeof(p)); //打印的是指针的大小，为4个字节
    p = a;
    printf("a = %p\n", a); //输出第一个元素的地址
    printf("a+1 = %p\n", a+1); //第一个元素的类型是int[3],所以+1加的是4*3=12个字节
    printf("p = %p\n", p); //p是一个指向int[3]的指针，p=a的时候，a是指向第一个元素的指针，
    //即为指向int[3]的指针，即p指向数组a的第一个元素
    printf("p+1 = %p\n", p+1); //p指向的类型为int[3],+1即跳过int[3]所代表的字节数，即4*3=12个字节
}

int main()
{
    test_array_pointer();
}
```

运行结果：



```
Microsoft Visual Studio 调试控制台
数组指针
sizeof(p) = 4
a = 00D4FD64
a+1 = 00D4FD70
p = 00D4FD64
p+1 = 00D4FD70
```

（2）指针数组

定义 `int *p[n];`

[]优先级高，先与p结合成为一个数组，再由int*说明这是一个整型指针数组，它有n个指针类型的数组元素。这里执行p+1时，则p指向下一个数组元素，这样赋值是错误的：p=a；因为p是指向p数组的第一个元素的常量指针（类型为int** const），不可以进行赋值，p[0]、p[1]、p[2]...p[n-1]（类型为int*），它们分别为指针变量可以被赋值存放变量地址。但可以这样 *p=a;

这里*p表示指针数组第一个元素的值，a为数组a的第一个元素的地址，即&a[0]

示例代码：

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

void test_pointer_array()
{
    printf("指针数组\n");
    int *p[2]; //定义一个指针数组，有两个元素
    int a[2][3] = { {11,12,13}, {21,22,23} }; //定义一个二维数组，定义了2行3列

    printf("sizeof(p) = %d\n", sizeof(p)); //打印的是数组的大小，为8个字节
    for (int i = 0; i < 2; i++)
    {
        p[i] = a[i]; //a[i]本身是一个int[3]*，值是每一行的首地址
    }
    printf("a = %p\n", a); //&a[0]
    printf("a+1 = %p\n", a + 1); //&a[1]
    printf("p = %p\n", p); //指向数组p的第一个元素的地址，p的类型为int**
    printf("p+1 = %p\n", p + 1); //指向数组p的第一个元素的地址，给p+1相当于加p所指向元素的类型的大小，
    //即int*，即4个字节

    printf("*p = %p\n", *p); // 等价于p[0] 指向a的第一行起始位置
}
```

```

printf("(p + 1) = %p\n", *(p + 1)); // 等价于p[1] 指向a的第二行的起始位置

for (int i = 0; i < 3; i++)
{
    printf("%d ", *(p + i)); // *p即p[0]是int*,即指针, 给指针+1,表示加它所指向的类型的大小,
                                //这里即加int的大小,即4个字节,每一次迭代都4个字节然后解引用即取到第

}

printf("\n");
for (int i = 0; i < 3; i++)
{
    printf("%d ", (*(p + 1) + i));

}

printf("\n\n");
for (int i = 0; i < 3; i++)
{
    printf("%d ", *(p[0] + i));
}
printf("\n");
for (int i = 0; i < 3; i++)
{
    printf("%d ", *(p[1] + i));
}

}

int main()
{
    test_pointer_array();
}

```

运行结果：

Microsoft Visual Studio 调试控制台

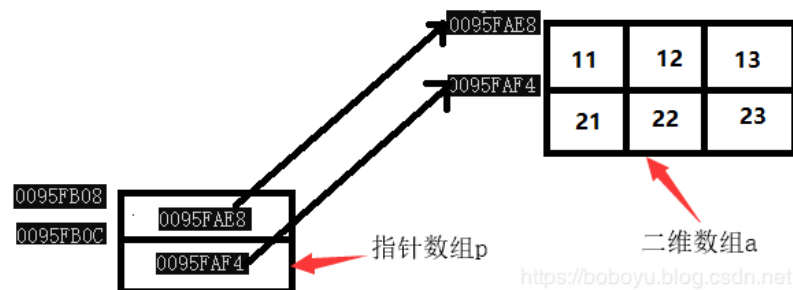
```

指针数组
sizeof(p) = 8
a = 0095FAE8
a+1 = 0095FAF4
p = 0095FB08
p+1 = 0095FB0C
*p      = 0095FAE8
*(p + 1) = 0095FAF4
11 12 13
21 22 23

11 12 13
21 22 23

```

上述代码图解：



优先级：() > [] > *