一、回顾

1.字符指针

详细内容移步：指针进阶之字符指针

①字符

```
char ch = 'w'; //字符变量ch里面存放一个w
char* p = &ch; //定义一个字符指针p，将ch的地址赋值给p，p是字符指针
```

②字符串

```
const char* p2 = "abcdef";
```

这里是将整个字符串存进p2里面了吗？不是的。p2是一个指针变量，4个字节，根本存不下“abcdef”七个字节。

当我们把一个字符串赋给p2时，其实是把这个字符串首元素a的地址交给了p2。能找到a，就能找到整个字符串。

“abcdef”是常量字符串，内容不允许被修改。所以最合理应加上“const”。

const去修饰指针变量，const放在`*`左边，修饰的是`*p2`，也就是p2指向的内容不能被修改。

2.指针数组和数组指针

详细内容移步：指针进阶之指针数组和数组指针

（1）指针数组

> 本质上是数组，用来存放指针。

比如：

```
int* arr[10];
```

arr首先与[]结合，是**数组**，有10个元素。

除去数组名（arr）和元素个数（10），剩下的是数组的元素类型，即：int*，每个元素是int*类型，即指针类型。

又比如：

```
char* ch[5];
```

ch数组有5个元素，每个元素是char*类型-->每个元素是字符指针类型。

（2）数组指针

> 本质上是指针，用来存放数组。

以前学过：

```
int* p3 --> p3为整型指针 （p3指向的元素是整型的） --> 指向整形的指针
char* p4 --> p4为字符指针（p4指向的元素是字符类型的）--> 指向字符的指针
```

那么数组指针 --> 指向数组的指针

比如现在有一个数组arr2，将arr2的地址取出来：

```
int arr2[5];//数组
&arr2;//取出数组的地址
```

再将数组arr2的地址（&arr2）存起来：

```
int(*pa)[5] = &arr2;
```

pa先和*结合，是一个指针；指向的是一个数组，数组里面有5个元素，每个元素是int类型。

pa就是一个数组指针。

pa是什么类型？去掉pa，剩下的就是它的类型“int (*)[5]”，即指向数组的指针类型。

二、数组参数

在写代码的时候难免要把【数组】或者【指针】传给函数，那函数的参数该如何设计呢？

1.一维数组传参

（1）整型数组

现在有一个存放整型的一维数组arr：

```
int main() {
    int arr[10] = { 0 };//arr是一维数组，有10个元素，每个元素为整型
    test(arr);  //将arr传参，传给test函数
    return 0;
}
```

要设计函数test，有以下几种方式，看一下可不可行：

①用整型**数组**接收

传递过来的是arr，是一个整型数组，那么就可以拿一个整型数组来接收。

```
void test(int arr[]) {
    //这种写法OK
}
```

这个整型数组，写上10个元素，也是可以的。

如下：

```
void test(int arr[10]) {
    //这种写法ok
}
```

②用**指针**接收

数组名arr是首元素地址，那么就可以拿一个指针来接收。

数组arr首元素是int类型，所以指针是`int*`类型。

可以用整型指针来接收。如下：

```
void test(int* arr) {
    //传过来的是arr，数组名是首元素地址，原来的每个元素都是Int类型的，这种写法OK
}
```

🏠总结

> 当一维数组进行传参的时候，**参数部分**可以写成`数组`，数组大小可以省略也可以不省略（写错也无所谓）；也可以写成`指针`。

(2) 指针数组

现在有一个存放指针的整型数组arr2：

```
int main() {
    int* arr2[20] = { 0 };//arr2是指针数组，数组有20个元素，每个元素是int*类型
    test(arr2);
    return 0;
}
```

要设计函数test2，有以下几种方式，看一下可不可行：

①用指针**数组**接收

```
void test2(int* arr[20]) {
    //传来的是一个数组，20个元素，每个元素是int*，这种写法OK，想省略20也没问题
}
```

arr2是一个整型指针数组，有20个元素，每个元素是int*类型。

既然传上去的是数组，就可以用数组（20个元素，每个元素是int*类型）来接收。

②用**指针**接收

```
void test2(int** arr) {
    //传来的arr是数组名，即首元素地址，每个元素是int*类型，那参数部分可以写成指针。
}
```

数组名arr2也是首元素地址，那么参数部分可以用指针来接收。

❓那这个指针如何写呢？

arr2数组的每个元素是`int*类型`，`int*`是一级指针。

arr2数组名表示首元素地址，首元素是一个`int*类型`，那么arr2表示的是一级指针的地址。

一级指针的地址传递给函数，当然要用二级指针来接收啦。

所以可以写成：`int** arr`。

(3) 总结

参数部分可以写成数组，数组大小可以省略，写错也没事（尽量别写错）。

也可以写成指针的形式，写成指针的时候，应该找一个合理的指针类型。

比如arr传递的是一个整型数组的数组名，那么函数的参数部分就要拿整型指针来接收。数组名表示首元素地址，即整型的地址，当然要放到整型指针里面啦。

arr2是指针数组，数组名是一级指针的地址，函数的参数就要用二级指针来接收。

2.二维数组传参

现在有一个二维数组：

```
int main() {
    int arr[3][5] = { 0 };
    test(arr);//二维数组传参
    return 0;
}
```

要设计函数test，有以下几种方式，看一下可不可行：

（1）用数组接收

```
void test(int arr[3][5]) {
    //可以
    //与下面传上来的数组保持一致，是最中规中矩的写法
}
```

数组传参，那么函数的参数部分可以写为数组。

同样，用数组来接收，可以省略行的部分，如下：

```
void test(int arr[][5]) {
    //可以
    //二维数组传参，函数形参的设计只能省略第一个[]的数字。
}
```

但是下面的不可以：

```
void test(int arr[3][]) {
    //不行    //行可以省略，列不能省略
}
```

因为对一个二维数组，可以不知道有多少行，但是必须知道有多少列（一行有多少个元素）。

数组行可以省略，列不能省略！

（2）用指针接收

传上去的数组名是首元素地址，那么就可以用指针来接收。

<1> 第一种写法

arr是二维数组数组名，那么这样写行不行呢？

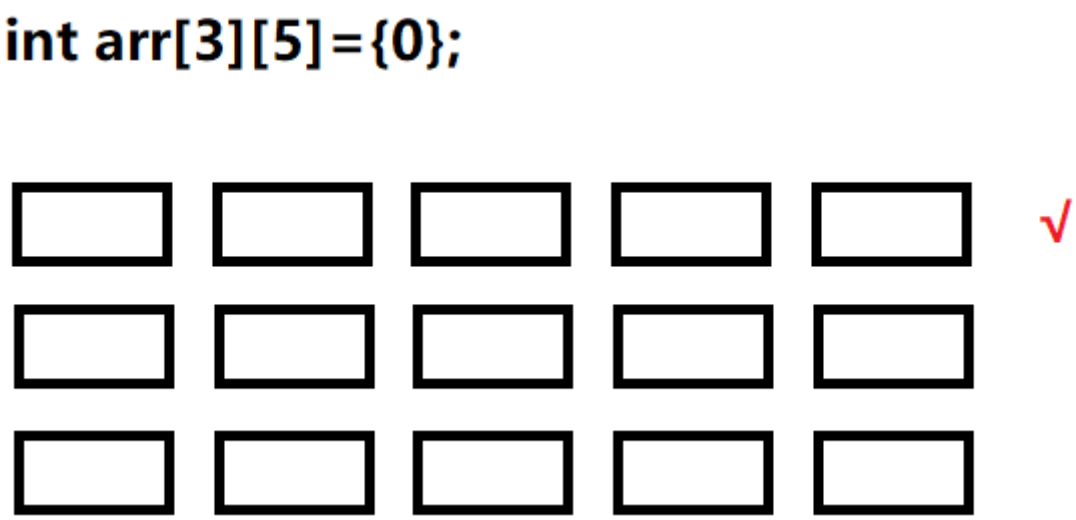
```
void test(int* arr) {
    //整型指针，不行
}
```

报警告：`int*`与`int[3][5]`间接级别不同。

传上来的是一个二维数组arr，二维数组的数组名表示的是首元素地址，首元素是第一行（可以看成一个一维数组）。

那么**二维数组数组名表示的是第一个一维数组的地址。**

如图：



传上来的是一个一维数组的地址，不能存放到整型指针里面去。整型指针是用来存放整型变量的地址的。

所以上面写法不行！

<2> 第二种写法

那究竟该如何书写呢？

这样行不行？

```
void test(int** arr) {
    //二级指针，不行
    //参数部分写的是二级指针，存放的是一级指针变量的地址
    //刚才传来的是一个一维数组的地址！不能放到二级指针里面去。
}
```

上面说了，数组名arr是首元素的地址，是第一行的地址，第一行是一个一维数组。

那么arr就是第一行的地址，即一维数组的地址。

一维数组的地址怎么能放到二级指针里面去？

二级指针是用来存放一级指针变量的地址的。

> `*数组名`是第一行的地址，`**数组名`是首元素。
>
> `*形参`得到一个`int*`指针（已经不是一行的信息了），`**形参`直接跑到指针指向的不知道哪里去了。

所以这种写法也不行！

<3> 第三种写法

这种写法也不正确：

```
void test(int* arr[5]) {
    //指针数组，不行
    //arr是数组，5个元素，每个元素是int*
}
```

指针数组里面存的是指针，而不能存放地址。

这种写法也不行！

<4> 第四种写法

这种写法才是正确的：

```
void test(int(*arr)[5]) {
    //数组指针，可以    //传过去的是第一行的地址，第一行是一维数组（5个整型）。
    //arr先和*结合，是指针，指向的数组（5个元素，每个元素是int类型）。
}
```

刚才说了，arr就是数组第一行的地址，即一维数组的地址。

第一行，是由5个整型构成的一维数组。

首先确定下来，是用指针接收，那么就先用小括号括起来，即：`(*arr)`。

然后再写这个指针类型，这个指针接收的是一个一维数组的地址，数组里面有5个int类型的元素。

那么指针类型就可以用`int [5]`表示。

即：`int (*arr)[5]`，这个指针有能力指向二维数组的首元素（第一行）。

三、指针参数

1.一级指针传参

(1) 写法

直接来看一段代码：

```
int main() {
   int arr[10] = { 1,2,3,4,5,6,7,8,9 }; //数组arr有10个元素，每个元素是int类型
   int* p = arr;    //数组名arr是首元素地址，我们把它放到指针里面去，p是指针
   int sz = sizeof(arr) / sizeof(arr[0]);   //sz是元素个数
   //一级指针p，传给函数，那么函数应该用指针来接收
   print(p, sz);
   return 0;
}
```

一级指针p，传给函数，那么函数应该用指针（`int* p`）来接收。

那么函数的参数部分就很好写：

```
 void print(int* p, int sz) {//p为指针，sz为个数
    int i = 0;
    for (i = 0; i < sz; i++) {
        printf("%d\n", *(p + i));
    }
}
```

📂 **思考**：当一个函数的参数部分为一级指针的时候，函数能接收什么参数？

(2) 案例一

参数部分是一级指针，那传上去的可能是什么呢？

```
void test1(int* p){

}
// test1函数能接收什么参数？
```

①既然参数部分是`int*`**指针类型**，那么一定可以接收地址。

那么是可以接收**地址**的，如下：

```
 int a = 10;
 test1(&a);//传地址过去
```

②同样，也可以直接传递一个**指针**过去，如下：

```
 int* p1 = &a;
 test1(p1);//p1本身就是指针
```

(3) 案例二

字符指针也是类似。

比如：

```
 void test2(char* pa){

 }
// test2函数能接收什么参数?
```

①直接传一个**字符的地址**

```
char ch = 'w';
test2(&ch);//传字符变量的地址
```

②还可以直接传**字符指针的地址**

```
 char* pc = &pca;
 test2(pc);//pc本身就是指针
```

(4) 总结

总结

> 如果函数的参数部分，写的是一级指针。那么可以传一个**变量的地址**上去，也可以传一个**存放地址的一级指针变量。**

2.二级指针传参

(1) 写法

直接看一段代码：

```
int main() {
    int n = 10; //整型变量n
    int* p = &n;    //p为一级指针，将n的地址存放到p里面
    int** pp = &p;  //pp为二级指针，将p指针的地址存放进pp里面
    test(pp);   //传一个二级指针变量上去
    test(&p);
    return 0;
}
```

`test(&p);`传了一个**一级指针p的地址**上去，需要拿二级指针（`int** ptr`）来接收。

`test(pp);`传了一个**二级指针变量（pp）**上去，自然要拿一个二级指针（`int** ptr`）来接收。

如下：

```
void test(int** ptr) {
     printf("num=%d\n", **ptr);
}
```

> 二级指针进行传参的时候，参数部分可以直接设计为二级指针。

📂 **思考：**当一个函数的参数部分为二级指针的时候，函数能接收什么参数？

（2）案例

上面说明了，参数部分是二级指针（`int** p`）的时候。

```
void test(int** p) {

}
```

传上去的可能是：

①一级指针变量的地址

```
int* ptr;
test(&ptr); //一级指针变量的地址传过去
```

②二级指针变量本身

```
int** pp = &ptr;
test(pp);   //二级指针传过去
```

③一级指针数组

除了上面两种，还有其他可能的。

既然函数参数部分是二级指针，无非就是接收一级指针变量的地址或者二级指针本身。

如果现在有一个一级指针数组arr，那么此时的数组名arr是首元素地址。

首元素是`int*`类型，那么`int*`元素的地址，需要用二级指针变量来接收。如下：

```
int* arr[10];   //一级指针数组arr，10个元素，每个元素类型是int*
test(arr);  //数组名是首元素地址，即int的地址
```

所以，传一个存放一级指针的数组的数组名（指针数组），也是可以的。

欢迎关注，一位喜欢慢慢生活的博主。

自述文件

雨翼轻尘

110	3.3w	2.1w
原创内容	作者排名	粉丝数量

CSDN