指针数组与数组指针详解

1.什么是指针数组和数组指针？

指针数组：指针数组可以说成是”指针的数组”，首先这个变量是一个数组，其次，”指针”修饰这个数组，意思是说这个数组的所有元素都是指针类型，在32位系统中，指针占四个字节。

数组指针：数组指针可以说成是”数组的指针”，首先这个变量是一个指针，其次，”数组”修饰这个指针，意思是说这个指针存放着一个数组的首地址，或者说这个指针指向一个数组的首地址。

根据上面的解释，可以了解到指针数组和数组指针的区别，因为二者根本就是种类型的变量。

2.指针数组和数组指针到底是什么？

2.1指针数组

首先先定义一个指针数组，既然是数组，名字就叫arr

char \*arr[4] = {"hello", "world", "shannxi", "xian"};

//arr就是我定义的一个指针数组，它有四个元素，每个元素是一个char \*类型的指针，这些指针存放着其对应字符串的首地址。

（当一个变量出现左右都出现一个运算符时，没有记住运算符优先级的人就会纠结arr变量到底跟哪一个运算符先结合。如果是自己定义一个指针数组，搞不清楚运算符的优先级，那么就加上小括号()，比如定义一个指针数组，可以写成char \*(arr[4])，不过在定义之前一定要清楚自己定义的变量，如果目的是一个数组，那就把arr[4]括起来，如果是一个指针，就把\*arr括起来。如果是看到一段这样的代码，可以从他的初始化来分别它是数组还是指针，很明显，我这定义的是一个数组，如果是指针，会用NULL来初始化。）

这个指针数组有多大呢？答案是16个字节，因为它是一个指针数组。(这是废话，正话下面说)

每当出现这些问题时，脑子里一定要第一时间反应出内存映像图

内存映像象图 内容 权限

栈区 函数中的普通变量 可读可写

堆区 动态申请的内存 可读可写

静态变量区 static修饰的变量 可读可写

数据区 用于初始化变量的常量 只读

代码区 代码指令 只读

这里最左侧一列是一个很简陋但能说明意思的内存图，一般情况下，从栈区到代码区，是从高地址到低地址。栈向下增长，堆向上增长。

arr[4]是一个在主函数定义的数组。把它对应到对应到内存中，arr是一个在栈区,有四个元素的数组，而每一个数组又是一个指针，所以说它的四个元素各占四个字节，所以变量arr的大小是16个字节。

那么就有人问了？初始化arr的{“hello”, “world”, “shannxi”, “xian”};的是什么鬼？

这四个不是什么鬼，他们也存在在内存中，只是跟arr这个变量不在同一段空间，它们被分配在只读数据区，数组arr[4]的四个指针元素，分别存放着这四个字符串的首地址，想象一下，从栈区有四只无形的手指向数据区的空间。arr+1会跳过四个字节，。也就是一个指针的大小

这就相当与定义char \*p1 = “hello”，char \*p1 = “world”，char \*p3 = “shannxi”， char \*p4 = “xian”，这是四个指针，每个指针存放一个字符串首地址，然后用arr[4]这个数组分别存放这四个指针，就形成了指针数组。

2.2数组指针

首先来定义一个数组指针，既然是指针，名字就叫pa

char (\*pa)[4];

如果指针数组和数组指针这俩个变量名称一样就会是这样：char \*pa[4]和char (\*pa)[4]，原来指针数组和数组指针的形成的根本原因就是运算符的优先级问题，所以定义变量是一定要注意这个问题，否则定义变量会有根本性差别！

pa是一个指针指向一个char [4]的数组，每个数组元素是一个char类型的变量，所以我们不妨可以写成：char[4] (\*pa);这样就可以直观的看出pa的指向的类型，不过在编辑器中不要这么写，因为编译器根本不认识，这样写只是帮助我们理解。

既然pa是一个指针，存放一个数组的地址，那么在我们定义一个数组时，数组名称就是这个数组的首地址，那么这二者有什么区别和联系呢？

char a[4];，

a是一个长度为4的字符数组，a是这个数组的首元素首地址。既然a是地址，pa是指向数组的指针，那么能将a赋值给pa吗？答案是不行的！因为a是数组首元素首地址，pa存放的却是数组首地址，a是char 类型，a+1，a的值会实实在在的加1，而pa是char[4]类型的，pa+1，pa则会加4，虽然数组的首地址和首元素首地址的值相同，但是两者操作不同，所以类型不匹配不能直接赋值，但是可以这样：pa = &a，pa相当与二维数组的行指针，现在它指向a[4]的地址。

3.指针数组和数组指针的使用

3.1指针数组在参数传递时的使用

指针数组常用在主函数传参，在写主函数时，参数有两个，一个确定参数个数，一个这是指针数组用来接收每个参数（字符串）的地址

int main(int argc, char \*argv[])

此时可以想象内存映像图，主函数的栈区有一个叫argv的数组，这个数组的元素是你输入的参数的地址，指向着只读数据区。

如果是向子函数传参，这和传递一个普通数组的思想一样，不能传递整个数组过去，如果数组很大，这样内存利用率很低，所以应该传递数组的首地址，用一个指针接收这个地址。因此，指针数组对应着二级指针

void fun(char \*\*pp);//子函数中的形参

fun(char \*p[]);//主函数中的实参

3.2指针数组的排序

指针数组的排序非常有趣，因为这个数组中存放的是指针，通过比较指针指向的空间的大小，排序这些空间的地址。函数实现如下：

void sort(char \*\*pa, int n)//冒泡排序

{

int i, j;

char \*tmp = NULL;

for(i = 0; i < n-1; i++){

for(j = 0; j < n-1-i; j++){

if((strcmp(\*pa+j), \*(pa+j+1)) > 0){

tmp = \*(pa + j);

\*(pa + j) = \*(pa + j + 1);

\*(pa + j + 1) = tmp;

}

}

}

}

在函数中定义指针数组，并且打印结果如下：

char \*pa[4] = {"abc", "xyz", "opq", "xyz"};

abc

ijk

opq

xyz

数组指针传参时的使用

数组指针既然是一个指针，那么就是用来接收地址，在传参时就接收数组的地址，所以数组指针对应的是二维数组。

void fun(int (\*P)[4]);//子函数中的形参，指针数组

a[3][4] = {0};//主函数中定义的二维数组

fun(a);//主函数调用子函数的实参，是二维数组的首元素首地址