深入理解C语言指针的奥秘 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/459/2021_2022__E6_B7_B1_E5_85_A5_E7_90_86_E8_c97_459826.htm 指针是一个特殊的变量，它里面存储的数值被解释成为内存里的一个地址。 要搞清一个指针需要搞清指针的四方面的内容：指针的类型，指针所指向的 类型，指针的值或者叫指针所指向的内存区，还有指针本身所占据的内存区。让我们分别说明。 先声明几个指针放着做例子： 例一： (1)int*ptr. (2)char*ptr. (3)int**ptr. (4)int(*ptr)[3]. (5)int*(*ptr)[4]. 如果看不懂后几个例子的话，请参阅我前段时间贴出的文章>。 指针的类型 从语法的角度看，你只要把指针声明语句里的指针名字去掉，剩下的部分就是这个指针的类型。这是指针本身所具有的类型。让我们看看例一中各个指针的类型： (1)int*ptr.//指针的类型是int* (2)char*ptr.//指针的类型是char* (3)int**ptr.//指针的类型是int** (4)int(*ptr)[3].//指针的类型是int(*)[3] (5)int*(*ptr)[4].//指针的类型是int*(*)[4] 怎么样？找出指针的类型的方法是不是很简单？ 指针所指向的类型 当你通过指针来访问指针所指向的内存区时，指针所指向的类型决定了编译器将把那片内存区里的内容当做什么来看待。 从语法上看，你只须把指针声明语句中的指针名字和名字左边的指针声明符*去掉，剩下的就是指针所指向的类型。例如： (1)int*ptr.//指针所指向的类型是int (2)char*ptr.//指针所指向的的类型是char (3)int**ptr.//指针所指向的的类型是int* (4)int(*ptr)[3].//指针所指向的的类型是int()[3] (5)int*(*ptr)[4].//指针所指向的的类型是int*()[4] 在指针的算术运算中，指针所指向的类型有很大的

作用。 指针的类型(即指针本身的类型)和指针所指向的类型是两个概念。当你对C越来越熟悉时，你会发现，把与指针搅和在一起的"类型"这个概念分成"指针的类型"和"指针所指向的类型"两个概念，是精通指针的关键点之一。我看了不少书，发现有些写得差的书中，就把指针的这两个概念搅在一起了，所以看起书来前后矛盾，越看越糊涂。指针的值，或者叫指针所指向的内存区或地址 指针的值是指针本身存储的数值，这个值将被编译器当作一个地址，而不是一个一般的数值。在32位程序里，所有类型的指针的值都是一个32位整数，因为32位程序里内存地址全都是32位长。 指针所指向的内存区就是从指针的值所代表的那个内存地址开始，长度为sizeof(指针所指向的类型)的一片内存区。以后，我们说一个指针的值是XX，就相当于说该指针指向了以XX为首地址的一片内存区域；我们说一个指针指向了某块内存区域，就相当于说该指针的值是这块内存区域的首地址。 指针所指向的内存区和指针所指向的类型是两个完全不同的概念。在例一中，指针所指向的类型已经有了，但由于指针还未初始化，所以它所指向的内存区是不存在的，或者说是无意义的。 以后，每遇到一个指针，都应该问问：这个指针的类型是什么？指针指的类型是什么？该指针指向了哪里？ 指针本身所占据的内存区 指针本身占了多大的内存？你只要用函数sizeof(指针的类型)测一下就知道了。在32位平台里，指针本身占据了4个字节的长度。 指针本身占据的内存这个概念在判断一个指针表达式是否是左值时很有用。 指针的算术运算 指针可以加上或减去一个整数。指针的这种运算的意义和通常的数值的加减运算的意义是不一样的。例如： 例二： 1、chara[20]. 2

、int*ptr=a. ... ... 3、ptr . 在上例中，指针ptr的类型是int*,它指向的类型是int，它被初始化为指向整形变量a。接下来的第3句中，指针ptr被加了1，编译器是这样处理的：它把指针ptr的值加上了sizeof(int)，在32位程序中，是被加上了4。由于地址是用字节做单位的，故ptr所指向的地址由原来的变量a的地址向高地址方向增加了4个字节。 由于char类型的长度是一个字节，所以，原来ptr是指向数组a的第0号单元开始的四个字节，此时指向了数组a中从第4号单元开始的四个字节。 我们可以用一个指针和一个循环来遍历一个数组，看例子： 例三：intarray[20]. int*ptr=array. ... //此处略去为整型数组赋值的代码。 ... for(i=0.i{ (*ptr) . ptr ；}这个例子将整型数组中各个单元的值加1。由于每次循环都将指针ptr加1，所以每次循环都能访问数组的下一个单元。 再看例子： 例四： 1、chara[20]. 2、int*ptr=a. ... ... 3、ptr =5. 在这个例子中，ptr被加上了5，编译器是这样处理的：将指针ptr的值加上5乘sizeof(int)，在32位程序中就是加上了5乘4=20。由于地址的单位是字节，故现在的ptr所指向的地址比起加5后的ptr所指向的地址来说，向高地址方向移动了20个字节。在这个例子中，没加5前的ptr指向数组a的第0号单元开始的四个字节，加5后，ptr已经指向了数组a的合法范围之外了。虽然这种情况在应用上会出问题，但在语法上却是可以的。这也体现出了指针的灵活性。 如果上例中，ptr是被减去5，那么处理过程大同小异，只不过ptr的值是被减去5乘sizeof(int)，新的ptr指向的地址将比原来的ptr所指向的地址向低地址方向移动了20个字节。 总结一下，一个指针ptrold加上一个整数n后，结果是一个新的指针ptrnew，ptrnew的类型和ptrold的类型相同，ptrnew所指向的类型

和ptrold所指向的类型也相同。ptrnew的值将比ptrold的值增加了n乘sizeof(ptrold所指向的类型)个字节。就是说，ptrnew所指向的内存区将比ptrold所指向的内存区向高地址方向移动了n乘sizeof(ptrold所指向的类型)个字节。 一个指针ptrold减去一个整数n后，结果是一个新的指针ptrnew，ptrnew的类型和ptrold的类型相同，ptrnew所指向的类型和ptrold所指向的类型也相同。ptrnew的值将比ptrold的值减少了n乘sizeof(ptrold所指向的类型)个字节，就是说，ptrnew所指向的内存区将比ptrold所指向的内存区向低地址方向移动了n乘sizeof(ptrold所指向的类型)个字节。 运算符amp.是取地址运算符，*是...书上叫做"间接运算符"。 amp.a.//amp.p.//amp.b.//*ptr是个指针，amp.b来给*ptr赋值就是毫无问题的了。 **ptr=34.//*ptr的结果是ptr所指向的东西，在这里是一个指针，对这个指针再做一次*运算，结果就是一个int类型的变量。 指针表达式 一个表达式的最后结果如果是一个指针，那么这个表达式就叫指针表式。 下面是一些指针表达式的例子： 例六： inta,b. intarray[10]. int*pa. pa=amp.a是一个指针表达式。 int**ptr=amp.pa也是一个指针表达式。 *ptr=amp.b都是指针表达式。 pa=array. pa .//这也是指针表达式。例七： char*arr[20]. char**parr=arr.//如果把arr看作指针的话，arr也是指针表达式 char*str. str=*parr.//*parr是指针表达式 str=*(parr 1).//*(parr 1)是指针表达式 str=*(parr 2).//*(parr 2)是指针表达式 由于指针表达式的结果是一个指针，所以指针表达式也具有指针所具有的四个要素：指针的类型，指针所指向的类型，指针指向的内存区，指针自身占据的内存。 好了，当一个指针表达式的结果指针已经明确地具有了指针自身占据的内存的话，这个

指针表达式就是一个左值，否则就不是一个左值。 在例七中，&amp.a不是一个左值，因为它还没有占据明确的内存。*ptr是一个左值，因为*ptr这个指针已经占据了内存，其实*ptr就是指针pa，既然pa已经在内存中有了自己的位置，那么*ptr当然也有了自己的位置。 数组和指针的关系 如果对声明数组的语句不太明白的话，请参阅我前段时间贴出的文章>。 数组的数组名其实可以看作一个指针。看下例： 例八： intarray[10]={0,1,2,3,4,5,6,7,8,9},value. ... ... value=array[0].//也可写成：value=*array. value=array[3].//也可写成：value=*(array 3). value=array[4].//也可写成：value=*(array 4).上例中，一般而言数组名array代表数组本身，类型是int[10]，但如果把array看做指针的话，它指向数组的第0个单元，类型是int*，所指向的类型是数组单元的类型即int。因此*array等于0就一点也不奇怪了。同理，array 3是一个指向数组第3个单元的指针，所以*(array 3)等于3。其它依此类推。 例九： char*str[3]={ "Hello,thisisasample!", "Hi,goodmorning.", "Helloworld" }. chars[80] ； strcpy(s,str[0]).//也可写成strcpy(s,*str). strcpy(s,str[1]).//也可写成strcpy(s,*(str 1)). strcpy(s,str[2]).//也可写成strcpy(s,*(str 2)).上例中，str是一个三单元的数组，该数组的每个单元都是一个指针，这些指针各指向一个字符串。把指针数组名str当作一个指针的话，它指向数组的第0号单元，它的类型是char**，它指向的类型是char*。 *str也是一个指针，它的类型是char*，它所指向的类型是char，它指向的地址是字符串"Hello,thisisasample!"的第一个字符的地址，即H的地址。 str 1也是一个指针，它指向数组的第1号单元，它的类型是char**，它指向的类型是char*。 *(str 1)也是一个指针，

它的类型是char*，它所指向的类型是char，它指向"Hi,goodmorning."的第一个字符H，等等。 下面总结一下数组的数组名的问题。声明了一个数组TYPEarray[n]，则数组名称array就有了两重含义：第一，它代表整个数组，它的类型是TYPE[n]；第二，它是一个指针，该指针的类型是TYPE*，该指针指向的类型是TYPE，也就是数组单元的类型，该指针指向的内存区就是数组第0号单元，该指针自己占有单独的内存区，注意它和数组第0号单元占据的内存区是不同的。该指针的值是不能修改的，即类似array 的表达式是错误的。 在不同的表达式中数组名array可以扮演不同的角色。 在表达式sizeof(array)中，数组名array代表数组本身，故这时sizeof函数测出的是整个数组的大小。 在表达式*array中，array扮演的是指针，因此这个表达式的结果就是数组第0号单元的值。sizeof(*array)测出的是数组单元的大小。 表达式array n（其中n=0，1，2，....。）中，array扮演的是指针，故array n的结果是一个指针，它的类型是TYPE*，它指向的类型是TYPE，它指向数组第n号单元。故sizeof(array n)测出的是指针类型的大小。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com