指针

2018年10月28日　　12:38

一，是什么?

1.指针是C语言中广泛使用的一种数据类型

运用指针编程是C语言最主要的风格之一。利用指针变量可以表示各种数据结构； 能很方便地使用数组和字符串； 并能象汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了C语言的功能。 学习指针是学习C语言中最重要的一环， 能否正确理解和使用指针是我们是否掌握C语言的一个标志。同时， 指针也是C语言中最为困难的一部分，在学习中除了要正确理解基本概念，还必须要多编程，上机调试。只要做到这些，指针也是不难掌握的。

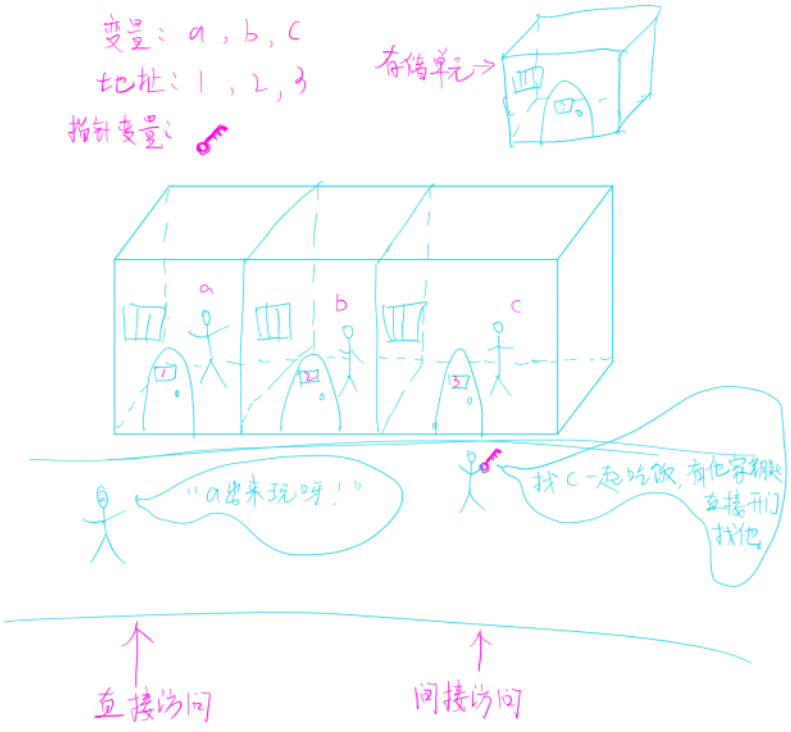
2.指针与地址

我们可以用这个类型声明一个变量并赋值，只不过指针所定义的变量用途比较特殊，是用来保存某个内存地址的。

如果在程序中定义了一个变量，在对程序进行编译时，系统就会给这个变量分配内存单元。编译系统根据程序中定义的变量类型，分配一定长度的空间。内存区的每一个字节有一个编号，这就是“地址”。

由于通过地址能找到所需的变量单元，可以说，地址指向该变量单元。C语言中的地址包括位置信息(内存编号，或称纯地址)和它所指向的数据的类型信息，或者说它是“带类型的地址”。

存储单元的地址和存储单元的内容是两个不同的概念。在程序中一般是通过变量名来引用变量的值。直接按变量名进行的访问，称为“直接访问”方式。还可以采用另一种称为“间接访问”的方式，即将变量的地址存放在另一变量（指针变量）中，然后通过该指针变量来找到对应变量的
地址，从而访问变量。



二，有什么用?

- 可以提高程序的编译效率和执行速度，使程序更加简洁。
- 通过指针被调用函数可以向调用函数处返回除正常的返回值之外的其他数据，从而实现两者间的双向通信。
- 利用指针可以实现动态内存分配。
- 指针还用于表示和实现各种复杂的数据结构，从而为编写出更加高质量的程序奠定基础。
- 利用指针可以直接操纵内存地址，从而可以完成和汇编语言类似的工作。
- 更容易实现函数的编写和调用。

当然，指针也是一把双刃剑，如果对指针不能正确理解和灵活有效的应用，利用指针编写的程序也更容易隐含各式各样的错误，同时程序的可读性也会大打折扣。

三，怎么用?

1.指针的定义

类型名 *指针变量名; 如int *pointer_1, *pointer_2;

左端的int是在定义指针变量时必须指定的“基类型”。指针变量的基类型用来指定此指针变量可以指向的变量的类型。

前面介绍过基本的数据类型(如int,char，float等)，既然有这些类型的变量，就可以有指向这些类型变量的指针，因此，指针变量是基本数据类型派生出来的类型，它不能离开基本类型而独立存在。

在定义指针变量时要注意:

(1) 指针变量前面的“*”表示该变量为指针型变量。指针变量名则不包含“*”。

(2) 在定义指针变量时必须指定基类型。一个变量的指针的含义包括两个方面，一是以存储单元编号表示的纯地址

（如编号为2000的字节），一是它指向的存储单元的数据类型（如int,char,float等）。
(3) 如何表示指针类型。指向整型数据的指针类型表示为“int *”，读作“指向int的指针”或简称“int指针”。
(4) 指针变量中只能存放地址（指针），不要将一个整数赋给一个指针变量。

2.复杂类型的说明
int p; //这是一个普通的整型变量

int *p; //首先从P 处开始,先与*结合,所以说明P 是一个指针,然后再与int 结合,说明指针所指向的内容的类型为int 型.所以P是一个返回整型数据的指针

int p[3]; //首先从P 处开始,先与[]结合,说明P 是一个数组,然后与int 结合,说明数组里的元素是整型的,所以P 是一个由整型数据组成的数组

int *p[3]; //首先从P 处开始,先与[]结合,因为其优先级比*高,所以P 是一个数组,然后再与*结合,说明数组里的元素是指针类型,然后再与int 结合,说明指针所指向的内容的类型是整型的,所以P 是一个由返回整型数据的指针所组成的数组

int (*p)[3]; //首先从P 处开始,先与*结合,说明P 是一个指针然后再与[]结合(与"()"这步可以忽略,只是为了改变优先级),说明指针所指向的内容是一个数组,然后再与int 结合,说明数组里的元素是整型的.所以P 是一个指向由整型数据组成的数组的指针

int **p; //首先从P 开始,先与*结合,说是P 是一个指针,然后再与*结合,说明指针所指向的元素是指针,然后再与int 结合,说明该指针所指向的元素是整型数据.由于二级指针以及更高级的指针极少用在复杂的类型中,所以后面更复杂的类型我们就不考虑多级指针了,最多只考虑一级指针.

int p(int); //从P 处起,先与()结合,说明P 是一个函数,然后进入()里分析,说明该函数有一个整型变量的参数,然后再与外面的int 结合,说明函数的返回值是一个整型数据

Int (*p)(int); //从P 处开始,先与指针结合,说明P 是一个指针,然后与()结合,说明指针指向的是一个函数,然后再与()里的int 结合,说明函数有一个int 型的参数,再与最外层的int 结合,说明函数的返回类型是整型,所以P 是一个指向有一个整型参数且返回类型为整型的函数的指针

int *(*p(int))[3]; //可以先跳过,不看这个类型,过于复杂从P 开始,先与()结合,说明P 是一个函数,然后进入()里面,与int 结合,说明函数有一个整型变量参数,然后再与外面的*结合,说明函数返回的是一个指针,,然后到最外面一层,先与[]结合,说明返回的指针指向的是一个数组,然后再与*结合,说明数组里的元素是指针,然后再与int 结合,说明指针指向的内容是整型数据.所以P 是一个参数为一个整数据且返回一个指向由整型指针变量组成的数组的指针变量的函数.

说到这里也就差不多了,我们的任务也就这么多,理解了这几个类型,其它的类型对我们来说也是小菜了,不过我们一般不会用太复杂的类型,那样会大大减小程序的可读性,请慎用,这上面的几种类型已经足够我们用了.

3.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？

4.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。

5.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si zeof(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？（重点注意）

6.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。

7.指针的算术运算

一个指针ptrold 加(减)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(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 所指向的内存区向高(低)地址方向移动了n 乘

sizeof(ptrold 所指向的类型)个字节。指针和指针进行加减：两个指针不能进行加法运算，这是非法操作，因为进行加法后，得到的结果指向一个不知所向的地方，而且毫无意义。两个指针可以进行减法操作，但必须类型相同，一般用在数组方面，不多说了。

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，以单元为单位。例如：

```
char a[20];
int *ptr=(int *)a; //强制类型转换并不会改变a 的类型
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型变量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，因为在32 位程序中，int 占4 个字节。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。

8.运算符&和*

这里&是取地址运算符，*是间接运算符。

&a 的运算结果是一个指针，指针的类型是a 的类型加个*，指针所指向的类型是a 的类型，指针所指向的地址嘛，那就是a 的地址。

*p 的运算结果就五花八门了。总之*p 的结果是p 所指向的东西，这个东西有这些特点：它的类型是p 指向的类型，它所占用的地址是p所指向的地址。

例：

int a=12; int b; int *p; int **ptr;

p=&a; //&a 的结果是一个指针，类型是int*，指向的类型是 int，指向的地址是a 的地址。

*p=24; //*p 的结果，在这里它的类型是int，它所占用的地址是p 所指向的地址，显然，*p 就是变量a。

ptr=&p; //&p 的结果是个指针，该指针的类型是p 的类型加个*， 在这里是int **。该指针所指向的类型是p 的类型，这里是int*。该指针所指向的地址就是指针p 自己的地址。

*ptr=&b; //*ptr 是个指针，&b 的结果也是个指针，且这两个指针的类型和所指向的类型是一样的，所以用&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，结果是一个int 类型的变量。

9.数组和指针的关系

数组的数组名其实可以看作一个指针。看下例：

例：

```
int array[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。

10.NULL指针

指针变量在使用前或者使用完，好的习惯是赋值为NULL ，NULL 是编号为0的字节的地址。指向NULL表示不指向任何变量。NULL就像剑鞘，野指针就像暗箭，如果你不像被暗箭所伤，就让他归鞘。