# 指针的简单运用

## 1.了解指针

指针，是C语言中的一个重要概念及其特点，也是掌握C语言比较困难的部分。指针也就是内存地址，指针变量是用来存放内存地址的变量，不同类型的指针变量所占用的存储单元长度是相同的，而存放数据的变量因数据的类型不同，所占用的存储空间长度也不同。有了指针以后，不仅可以对数据本身，也可以对存储数据的变量地址进行操作。

## 2.指针与指针变量的区别

指针：

指针就是地址，地址就是指针。

指针变量：

指针变量是变量。定义一个指针变量，是在内存中开辟一个空间，该空间里面存放地址。

如何使用：

指针更多强调的是内容（对应右值），指针变量更多强调的是空间（对应左值）。判断一个指针和一个指针变量要通过判断它是左值还是右值

左值与右值:

空间 对应 左值

例：（对a来说）

a = 20; //当把值赋给a时，是给a的空间写入20。

内容 对应 右值

例：（对a来说）

b = a; //把a的内容赋给b。

## 3.指针与指针类型

（1）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]

（2）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]

（3）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。

（5）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（后面会解释）是否是左值时很有用。

## 4.指针的安全问题

（1）看下面的例子：

int main()   
{  
 char *s* = 'a';  
 int \**ptr*;  
 *ptr* = (int \*) &*s*;  
 \**ptr* = 1298;  
}

指针ptr 是一个int \*类型的指针，它指向的类型是int。它指向的地址就是s 的首地址。在32 位程序中，s 占一个字节，int 类型占四个字节。最后一条语句不但改变了s 所占的一个字节，还把和s 相临的高地址方向的三个字节也改变了。这三个字节是干什么的？只有编译程序知道，而写程序的人是不太可能知道的。也许这三个字节里存储了非常重要的数据，也许这三个字节里正好是程序的一条代码，而由于你对指针的马虎应用，这三个字节的值被改变了！这会造成崩溃性的错误。

（2）野指针

野指针就是指针指向的位置是不可知的（随机的、不正确的、没有明确限制的）

野指针成因

A.指针未初始化

#include <stdio.h>  
int main()   
{  
 int \**p*;*//局部变量指针未初始化，默认为随机值* \**p* = 20;  
 return 0;  
}

B.指针访问越界

#include <stdio.h>  
int main()  
{  
 int *arr*[10] = {0};  
 int \**p* = *arr*;  
 int *i* = 0;  
 for(*i*=0; *i*<=11; *i*++)  
 {  
 *//当指针指向的范围超出数组arr的范围时，p就是野指针* \*(*p*++) = *i*;  
 }  
 return 0;  
}

C.指针指向的空间释放

如何规避野指针的方法：

a.指针初始化

b.小心指针越界

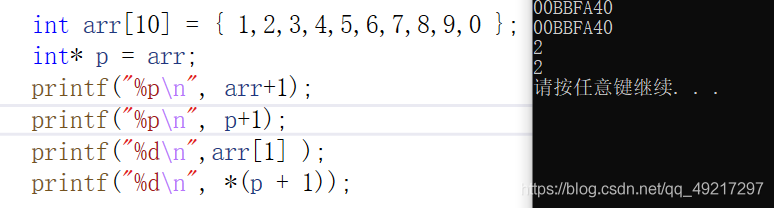
c.指针指向空间释放即使置NULL

d.指针使用之前检查有效性

## 5.指针与数组

### #include<stdio.h> int main() { int *arr*[10] = { 1,2,3,4,5,6,7,8,9,0 }; int\* *p* = *arr*; printf("%p\n", *arr*+1); printf("%p\n", *p*+1); printf("%d\n",*arr*[1] ); printf("%d\n", \*(*p* + 1)); return 0; }

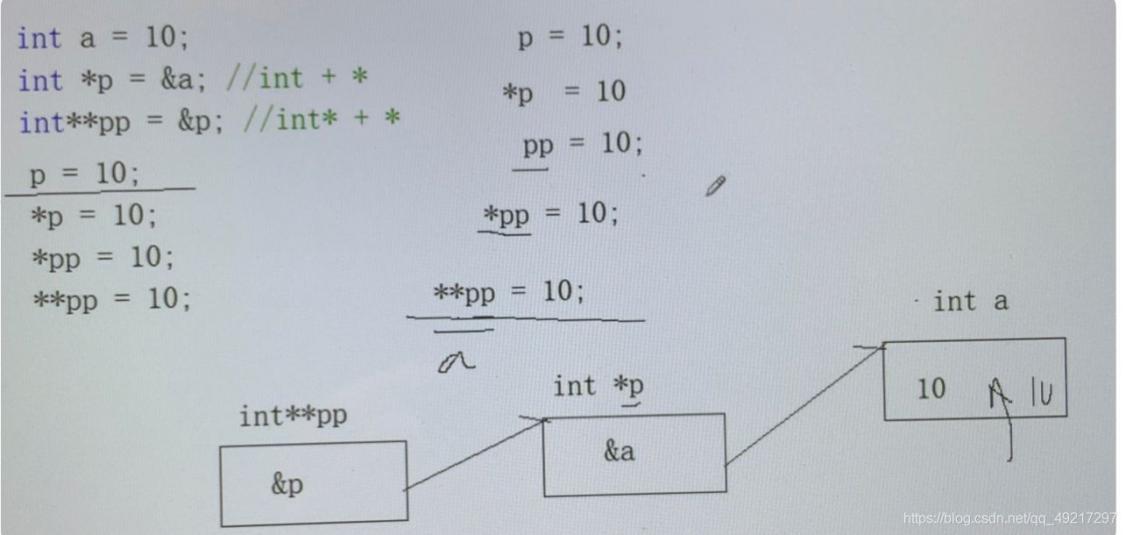
打印结果:



所以可以用指针保存数组首元素地址，使用指针进行访问数组元素。

## 6.二级指针

指针变量也是变量，是变量就有地址，那指针变量的地址存放在哪里？ 就是二级指针。



对于二级指针的运算有：

\*pp通过对pp中的地址进行解引用，这样找到的是 p ， \*pp其实访问的就是p .

int a = 20;

\*pp = &a;//等价于 p = &a;

\*\*pp 先通过 \*pp 找到 p ,然后对 p 进行解引用操作：\*p ，那找到的是 a .

\*\*pp = 10;

//等价于\*p = 10;

//等价于a = 10;

结论：

二级指针及以上的指针自加1是加上4字节（32位系统）或8字节（64位系统）

一级指针自加1就是加上所指向类型的大小

## 7.指针数组和数组指针

数组指针：是指针，指向数组。例：int (\*arr)[10]

指针数组：是数组，数组内容存放的是指针。例：int \*arr[10]

然后，需要明确一个优先级顺序：()>[]>\*

所以：

(\*p)[n]：根据优先级，先看括号内，则p是一个指针，这个指针指向一个一维数组，数组长度为n，这是“数组的指针”，即数组指针；

\*p[n]：根据优先级，先看[]，则p是一个数组，再结合\*，这个数组的元素是指针类型，共n个元素，这是“指针的数组”，即指针数组。

## 8.指针和结构类型的关系

struct MyStruct  
{  
 int a;  
 int b;  
 int c;  
};  
struct MyStruct ss = {20, 30, 40};

struct MyStruct \*ptr=&ss;

可以声明一个指向结构类型对象的指针。

 //声明了结构对象ss，并把ss 的成员初始化为20，30 和40。

 //声明了一个指向结构对象ss 的指针。它的类型是

    //MyStruct \*,它指向的类型是MyStruct。

请问怎样通过指针ptr 来访问ss 的三个成员变量？

答案：

ptr->a; //指向运算符，或者可以这们(\*ptr).a,建议使用前者

ptr->b;

ptr->c;