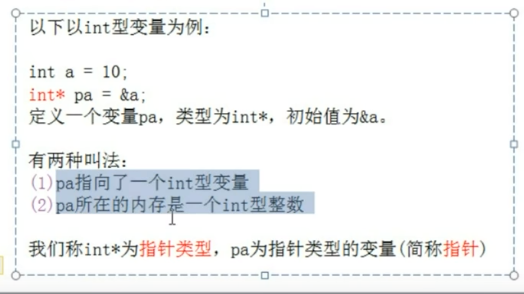
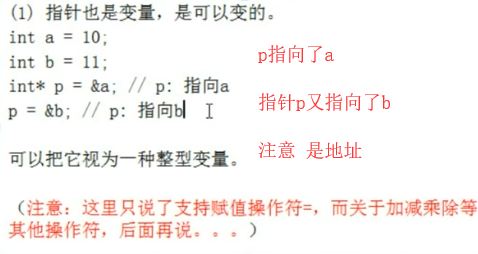
如何表示地址 强调它是一个内存地址



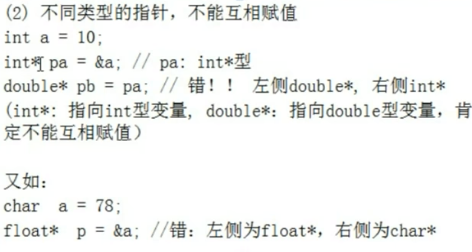


变量pa 地址等同 变量 a地址 其地址里面存放了一个int型的整数

int\* 叫做指针类型 pa为指针变量 简称 指针



其中 不同类型的指针 是不能相互赋值



&a 表示的是 取变量a的地址 变量a是char类型的所以 &a 指针类型为char \*

关于指针 星号\*的位置 是比较自由的

int\* a; //int与\*连写

int \* a; //int与\*中间有空白

int \*a; //\*与变量名连着

这几种写法是一样的 不要去区分

指针的作用

int a =0;

a = 0x11; //是直接使用变量名赋值 a的值为0x11

使用 赋值 右值赋值给左边的变量名

指针 //衍生的赋值方式 按地址来访问，直接修改内存 内存就是 值

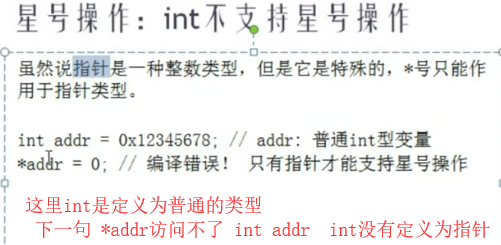
int a =0;

int \* p =&a; //p指向a

\*p = 0x11; //使用指针来访问

\*p +=2; //**不是修改地址 是p所代表的地址的 内存里的值加2**

int\* 是叫指针 \*p是一个动作 取得p处内存的值



重点

**char a =78; cahr \* p =&a; \*p =79;**

p的值为 a的地址 \*p 是往a的内存里赋值 a的值被覆盖为79

指针和数组关系

在软件内存块调试时 查看数组地址不需要 加& 数组名称 就是地址

int arr[4] = {1,2,3,4};

int \* p = arr; // arr是 int 型所以可以赋值地址给指变量p

int\* p = &arr[0]; arr ==&arr[0]; 都是表示首地址

指针加减法

p += 1; //后移一个地址 地址的长度根据定义的类型

p -= 1; //向前 一个地址

arr + 3 == &arr[3] //取下标为3的地址 \* ( arr + 3) /取值

int arr[1,2,3,4];

int \* p =&arr[1];

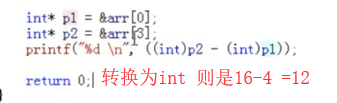
p[0] =0xAA; // p[0] ：自p开始的地址为将 arr[1]设为第0号元素

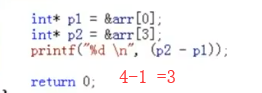
int a =10;

int \* p =&a;

p[0]=11; //长度为1的数组 a也看做是数组的首地址

遍历 for( int \*p =arr; p <arr+4; p++) //arr是数组

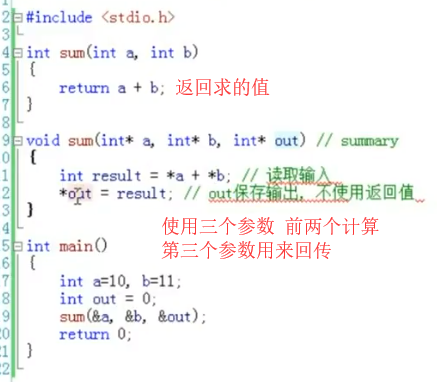


数组原型

指针作为函数的参数

参数 可以分为 输入参数 和 输出参数

主要是可以修改上层函数的值因为传递的是地址



int avg(int\* p, int len) //如果是数组传递 第一个参数地址 第二个参数长度

{

int sum = 0;

for(int i=0; i<len;i++)

{

sum+=p[i];

}

return sum/len;

}

int main()

{

int arr [] = {1,2,,3,4}; // arr 是 int \*

int ret ;

ret = avg (arr,4); //范围从arr[0] 到arr[3]

return 0;

}

int avg (int \* p, int len) == int avg(int p[], int len) ; //p[] 是不用写 只是表示是一个数组地址

传指针 作用

1. 返回多个值

比如一个函数 （参数1，参数2，参数3，参数4）

//参数需要返回则可以把参数1、2输入参数 把3、4输出参数使用指针直接回传修改不用return

1. 效率高

//普通传递参数

int n =0;

test1(n);

void test1 (int a) // int a =n 是拷贝的方式 如果参数是大数据 则拷贝占用的字节过多 造成浪费空间 运行也慢

//指针传递参数

int n =0;

test1(&n); // 传地址，把n的地址传递给函数 直接

void test1 (int \* p ) //效率高 传递的速度快不浪费空间

const 指针

只读 模式 不修改 只是读取

当函数的参数中使用到 const 可以看成是只作输入参数

int \* const p p不能修改

int \*p[5]; 是指针数组， 有 5 个成员，每个成员都是一个指针，共有5 个指针  
int (\*p)[5]； 小括号优先，所以 是 1个指针，用来指向 有5个元素的数组。  
--------  
一般来说  
int (\*p)[n]; 是数组指针（也称行指针）  
()优先级高，说明p是一个指针，指向一个整型的一维数组，这个一维数组的长度是n，也可以说是p的步长。也就是说执行p+1时，p要跨过n个整型数据的长度。  
如要将二维数组赋给一指针，应这样赋值：  
int a[3][4];  
int (\*p)[4]; //该语句是定义一个数组指针，指向含4个元素的一维数组。  
p=a; //将该二维数组的首地址赋给p，也就是a[0]或&a[0][0]  
p++; //该语句执行过后，也就是p=p+1;p跨过行a[0][]指向了行a[1][]  
所以数组指针也称指向一维数组的指针，亦称行指针。  
----  
int \*p[n]; 指针数组  
[]优先级高，先与p结合成为一个数组，再由int\*说明这是一个整型指针数组，它有n个指针类型的数组元素。这里执行p+1是错误的，这样赋值也是错误的：p=a；因为p是个不可知的表示，只存在p[0]、p[1]、p[2]...p[n-1],而且它们分别是指针变量可以用来存放变量地址。但可以这样 \*p=a; 这里\*p表示指针数组第一个元素的值，a的首地址的值。  
如要将二维数组赋给一指针数组:  
int \*p[3];  
int a[3][4];  
for(i=0;i<3;i++)  
p[i]=a[i];  
这里int \*p[3] 表示一个一维数组内存放着三个指针变量，分别是p[0]、p[1]、p[2]  
所以要分别赋值。