指针 pointer

目录

& 取地址符

scanf("%d",&i);//&i表示变量i的地址，

相当于将数据存入以变量i的地址为地址的变量中，也就是变量i。

打印变量的地址

- %p：以16进制输出，不够8位，左边补零。
- Sizeof（&i）或sizeof（p）

在32位机器上运行的结果，显示出8位16进制格式的地址，也就是32位的地址值。

如果是在64位机器上运行，将会输出16位16进制格式的地址，即64位地址值。

```
00000000066FE1C
```

```
int  main( ) {
    int i;
    printf("0x%x",&i);
}
```

```
0x66fe1c
```

```
int  main( ) {
    int i;
    printf("0x%x\n",&i);
    printf("%p",&i);
}
```

```
0x66fe1c
000000000066FE1C
```

在32位的机器上，地址是32个0或者1组成二进制序列，那地址就得用4个字节的空间来存储，所以一个指针变量的大小就应该是4个字节。

那如果在64位机器上，如果有64个地址线，那一个指针变量的大小是8个字节，才能存放一个地址。

打印数组的地址

```
int  main( ) {
     int a[10];
     printf("%p\n",a);
     printf("%p\n",&a);
     printf("%p\n",&a[0]);
     printf("%p\n",&a[1]);

return 0;
}
```

```
000000000066FDF0
000000000066FDF0
000000000066FDF0
000000000066FDF4
```

能否找到一个存放地址的变量？

我们需要找到一个存放地址的变量

如何表达存放地址的变量?

6.1 指针的引出

一. 地址与指针

1. 地址与取地址运算

C程序中的变量在内存中占有一个可标识的存储区， 每一个存储区是由若干个字节组成, 每一个字节都有 自己的地址, 而一个存储区的 地址是指该存储区中第一个字节的地址

C语言允许在程序中使用变量的地址

(通过地址运算符&可得到)

如: float x ; 变量 x 的地址 ---- &x

int a[10] ; 数组变量 a 的地址 ---- 数组名 a

1 指针是什么？

我们口头上说的指针其实指的是**指针变量**。指针变量就是一个存放地址的变量。

指针（pointer）和 int，char 类似，是一种独立的数据类型。区别在于，当我们说“指针”时，其实是说一系列的数据类型（泛指），就像我们说“数值型数据类型”，也是泛指（包括 int, float, double 等等）。当我们具体说“int 型指针”时，我们是说 int*（而不是 float* 或者 double*），就像是指出了数值类型中的某一个具体类型（例如 int）。

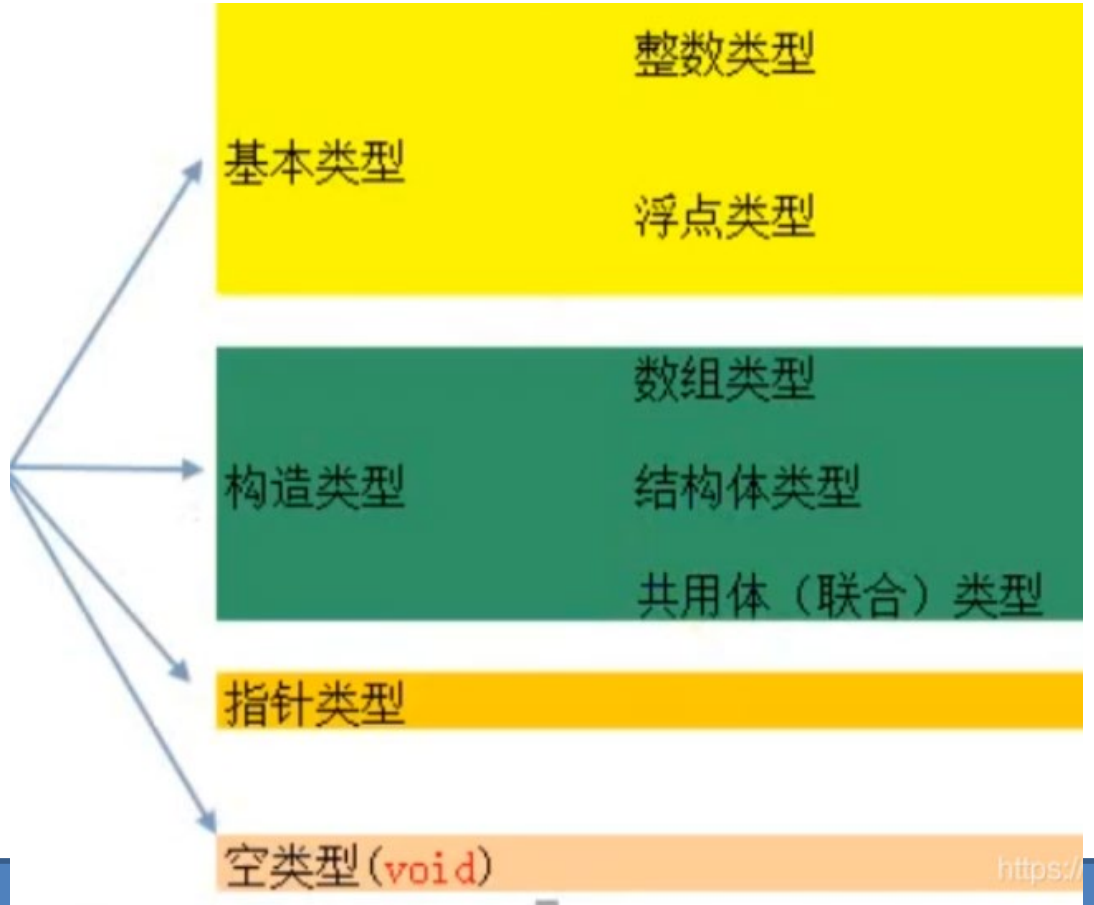
因此：指针是数据类型，而指针变量是用来保存这些地址的变量。

指针的定义：指针是数据类型。

指针变量：是存放内存单元编号的变量（存放地址的变量）

一种数据类型

在C语言中，指针类型就是数据类型，是给编译器看的，也就是说，指针类型与数组、int、char这种类型是平级的，是同一类的。



（1）可以提高程序的编译效率和执行速度，使程序更加简洁。

（2）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。

（3）利用指针可以实现动态内存分配。

（4）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。

（5）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。

（6）更容易实现函数的编写和调用

2 指针的定义

任何变量都可以带＊，加上＊以后变成新的类型，统称“指针类型”。

```
int *p1;
char *p2;
```

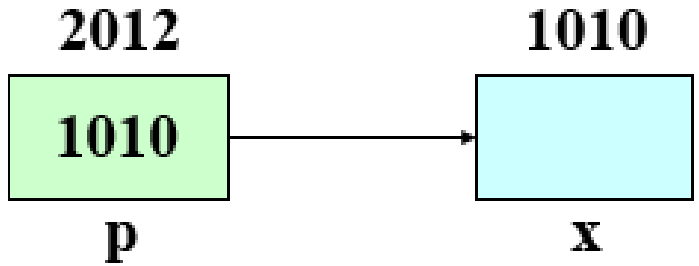
(1) 变量的访问方式

① **直接访问**：通过变量名或地址访问变量的存储区

例：scanf (“%d” , &x) ;

x = sqrt(x) ;

printf (“%d” , x) ;



② **间接访问**：将一个变量的地址存放在另一个变量中.

如将变量 x 的地址存放在 变量p 中, 访问x 时先找到p, 再由p 中存放的地址找到 x

```
ch 的地址:000000000066FE1F
num的地址:000000000066FE18
```

指针的值实质是内存单元（即字节）的编号，所以指针 单独从数值上看，也是整数，他们一般用16进制表示。

```
int main(void)
{

    char ch = 'a';
    int  num = 97;
    printf("ch 的地址:%p\n",&ch);
    printf("num的地址:%p\n",&num);
    return 0;

}
```

二、 指针变量的声明

1. 格式： 数据类型 *指针变量名；

```
int *p1 ;
char *p2 ;
```

```
int i;
int *p=&i;
int *p,q;
```

Int* p int * p或int *p ： * 的位置靠前靠后靠中间都可以。

2. 说明：

(1) 在变量定义时,* 号表示该变量是指针变量 （注意: 指针是p1，p2，不是*p1，*p2）

(2) 定义指针变量后, 系统为其分配存储空间, 用以存放其他变量的地址, 但在对指针变量赋值前, 它并没有 确定的值, 也不指向一个确定的变量

(3) 使指针变量指向一个确定的变量必须进行赋值

```
int x, *p ;
x = 5 ;
p = &x ;
```

```
int a=10;
    int width=sizeof(a);
    int *p ;
    printf("width=%d\n",width);
    printf("pointer=%p\n",p);
    p=&a;
    printf("pointer=%p\n",p);
    *p=20;
    printf("a=%d",a);
```

1个内存单元是1个字节

2个指针运算符

*：取指针所指向地址的变量的内容

&：取变量的地址

*：取值运算符（间接访问运算符）

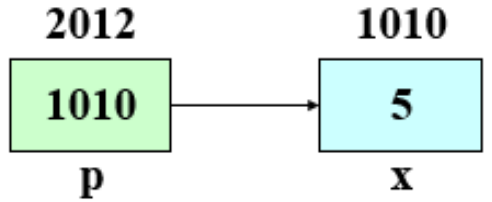
&：取地址运算符

三、 指针变量的引用

(1) p与*p不同,p是指针变量,p的值是p所指向的变量的地址

*p 是p 所指向的变量 ,*p的值是p所指向的变量的值

*p 的值为 5(*p 表示 x) , 而p 的值为 1010



*p

```
int  main( ) {
    int i=6;
    int *p=&i;
    printf("p=%p\n",p);
    printf("*p=%d\n",*p);
    printf("i=%d\n",i);

    return 0;
}
```

*区别

(2) 引用指针变量时的*与 定义指针变量时的*不同

定义变量时的*只是表示其后的变量是指针变量

```
int  a, *p ;
p = &a ;
printf (“%d\n”, *p) ;
*p = 12 ;
printf (“%d\n”, *p) ;
```

2. & 与*

p = &a ;

&*p → &(*p) → &a

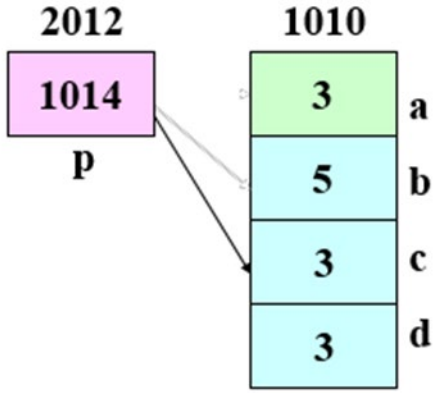
*&a → *(&a) → *p → a

3. *与 ++ , – –

int a = 3 , b = 5 , c , d , *p ;

(1) p = &a ;

p 的值为a 的地址, *p 的值为2



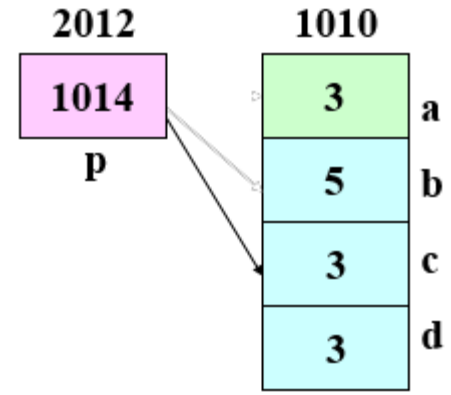
(*p)++ ;(等价于 a++ ;)

p 的值不变,*p 的值为 3

(2) c = *p++ ;

c = *(p++) ; → { c = *p ; p++ ; }

执行后 c 的值为 3 ,*p 的值为 5



(3) d = *++p ;

d = *(++p) ; → { ++p ; d = *p ; }

执行后 d 的值为 3 ,*p 的值为 3

就近原则：*（P++）与*（p--）的运算

一般来讲自增、自减运算符在后，先取地址运算，再自增自减运算，但自增自减，离谁近就是对谁，例如*（p++）

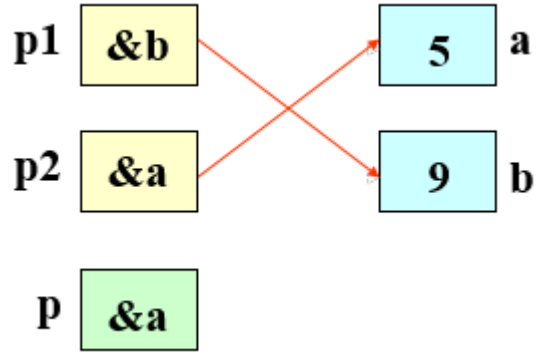
先取*p所指向的值后再做++运算，这里++运算不是对*p的 值++而是对p的值++,其中P++是加一个基类型所占的字节数。

同理*（++p）与*(--p)是先自增自减运算后指针运算。

```c
#include  <stdio.h>

int  main( ) {
     int *p1, *p2, *p, a, b ;
     scanf("%d%d", &a, &b);
     p1 = &a ;
     p2 = &b ;
     if (a<b) {
          p = p1 ;
          p1 = p2 ;
          p2 = p ;
     }
     printf("a=%d, b=%d \n", a, b);
     printf("max=%d, min=%d \n",*p1,*p2) ;

return 0;
}
```



输出结果:

a=5 , b=9

max=9 , min=5

6 指针的大小

p1=8, p2=8

指针在32位机器下是4个字节，在64位机器下是8个字节。注：（指针的大小与类型无关）

```
int a;
char b;
int *p1=&a;
char *p2=&b;
printf("p1=%d,p2=%d",sizeof(p1),sizeof(p2));
return 0;
```

疑问？

既然指针的大小跟类型无关，占的字节都是4byte（32位）或8byte（64位）

那么为何还要区分类型呢?

如 int *p;

Char *p;

Flaot *p

Double *p

```
int *pi;
char *pc;
float *pf;
double *pd;
int * a[1];
printf("pi=%d,pc=%d,pf=%d,pd=%d\n",sizeof(pi),sizeof(pc),sizeof(pf),sizeof(pd));
printf("pa=%d",sizeof(a));
return 0;
```

7指针类型的区别

相同点:

char型指针和int型指针，指针变量本身都是占4个（32位）或8个（64位）字节的内存空间，可以通过sizeof(char*)或者sizeof(int*)来得到占用的字节空间数，存放的都是一个32位或64位的地址值。

不同点：

char*指针类型和*int*指针类型在做算术运算的时候，地址值的变化是不一样的。

```
char *pointer_c;
```
假设ps存放的地址值是0x1000 pointer_c++;
pointer_c自增加1，则pointer_c存放的地址值就变为了0x1001，因为char类型是占一个字节
也就是说，pointer_c指向了下一个char型的内存地址。
```
 int *pointer_i;
```
如果是int型指针，假设初始也是0x1000 pointer_i++;
pointer_i自增加1，则pointer_i存放的地址值就变为了0x1004，因为int类型是占4个字节，
也就是说，pointer_i指向了下一个int型的内存地址。

```
p1=000000000066FE0C, p2=000000000066FE0B
p1=000000000066FE10, p2=000000000066FE0C
```

P1++:移动了四个字节

P2++：移动了一个字节

```
int main(void)
{
    int a=3;
    char b=4;
    int *p1=&a;
    char *p2=&b;
    printf("p1=%p,p2=%p\n",p1,p2);
    p1++;
    p2++;
    printf("p1=%p,p2=%p",p1,p2);
    return 0;
}
```

对于指向数组的指针而言：

P1++:移动了四个字节

P2++：移动了一个字节

```
int arr1[6]={0};
char arr2[]="Long live the motherland";
int *p1=arr1;
char *p2=arr2;
printf("p1=%p\n",p1);
printf("p1+1=%p\n",p1+1);
printf("p2=%p\n",p2);
printf("p2+1=%p\n",p2+1);
```

```
p1=000000000066FDF0
p1+1=000000000066FDF4
p2=000000000066FDD0
p2+1=000000000066FDD1
```

为什么需要指针?

指针解决了一些编程中基本的问题

(1) .快速的传递数据，减少内存的使用
(2) .可以使函数返回一个以上的值
(3) .可以直接访问物理硬件
(4) .可以方便的处理字符串
(5) .是理解面向对象语言中"引用"功能的基础
(6) .可以表示一些复杂的数据结构

应用场景1

交换两个变量的值

```
void swap(int *pa, int *pb)
{
    int t = *pa;
    *pa = *pb;
    *pb = t;
}
```

应用场景2

- 函数返回多个值，某些值就只能通过指针返回
  - 传入的参数实际上是需要保存带回的结果的变量

```c
void minmax(int a[], int len, int *max, int *min);

int main(void)
{
    int a[] = {1,2,3,4,5,6,7,8,9,12,13,14,16,17,21,23,55,};
    int min,max;
    minmax(a, sizeof(a)/sizeof(a[0]), &min, &max);
    printf("min=%d,max=%d\n", min, max);

    return 0;
}

void minmax(int a[], int len, int *min, int *max)
{
    int i;
    *min = *max=a[0];
    for ( i=1; i<len; i++ ) {
        if ( a[i] < *min ) {
            *min = a[i];
        }
        if ( a[i] > *max ) {
            *max = a[i];
        }
    }
}
```

应用场景2.1

- 函数返回运算的状态，结果通过指针返回
- 常用的套路是让函数返回特殊的不属于有效范围内的值来表示出错：
  - -1或0（在文件操作会看到大量的例子）
- 但是当任何数值都是有效的可能结果时，就得分开返回了

C++和java python等语言会用异常来解决这样的问题！

```c
#include <stdio.h>

/**
    @return 如果除法成功，返回1；否则返回0
*/
int divide(int a, int b, int *result);

int main(void)
{
    int a=5;
    int b=2;
    int c;
    if ( divide(a,b,&c) ) {
        printf("%d/%d=%d\n", a, b, c);
    }
    return 0;
}

int divide(int a, int b, int *result)
{
    int ret = 1;
    if ( b == 0 ) ret = 0;
    else {
        *result = a/b;
    }
    return ret;
}
```

使用指针常见的错误

- 定义了指针变量，还没有指向任何变量，就开始使用指针

```
int *p;
int k;
k = 12;
*p = 12;
```