

# 指针与函数指针

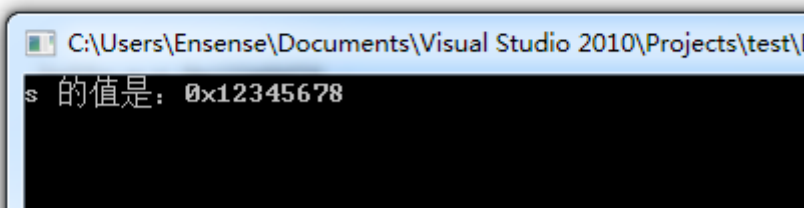
## 1. 指针

指针应该是 C/C++中最容易出错的地方了，一不小心就会被玩坏。这里不介绍指针的基本概念，只聊聊怎样才能不把指针玩坏。

首先记住一个要点：在 C 里 \*是取内容，&是取指针。

看例子：

```
54 |  
55 | int _tmain(int argc, _TCHAR* argv[])  
56 | {  
57 |  
58 |     int nOrg = 0x12345678 ;  
59 |  
60 |     int s = *&nOrg ;  
61 |  
62 |     printf("s 的值是：0x%x ", s);  
63 |  
64 |  
65 |  
66 |  
67 |  
68 |  
69 |  
70 |
```



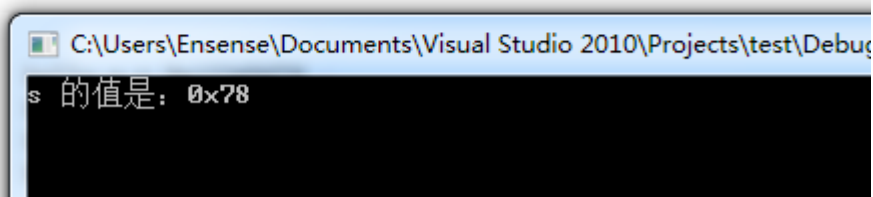
第 60 行，先用&取得变量 nOrg 的地址；再用\*取的地址里的内容。内容当然就是变量的值 0x12345678 。这个应该比较好理解。

要点二：(char \*)、(int \*)等可以显式转换指针的类型。

这个类似数据转换时的显式转换(float)、(double)功能。

看例子：

```
54 |  
55 | int _tmain(int argc, _TCHAR* argv[])  
56 | {  
57 |  
58 |     int nOrg = 0x12345678 ;  
59 |  
60 |     int s = * (char *)&nOrg ;  
61 |  
62 |     printf("s 的值是：0x%x ", s);  
63 |  
64 |  
65 |  
66 |  
67 |  
68 |  
69 |  
70 |
```

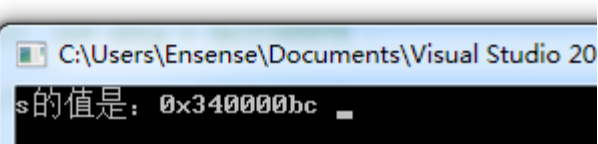


第 60 行，先用&取得变量 nOrg 的地址；再用(char \*)把地址转换成 char 型变量的地址；再用\*取的地址里的内容。内容当然就是变量 nOrg 被转换成 char 型后的值 0x78 。这个结合

之前说的数据类型转换的知识，应该还算好理解。

再来看一个稍微高深一点的例子：

```
38 struct
39 {
40     int a ;
41     unsigned char b ;
42     unsigned char c ;
43     int d ;
44 }hello;
45
46
47
48 int _tmain(int argc, _TCHAR* argv[])
49 {
50
51     hello.a = 0x12345678 ;
52     hello.b = 0x9A ;
53     hello.c = 0xBC ;
54     hello.d = 0xDEF01234 ;
55
56     int s = * (int *)&hello.c ;
57
58     printf("s的值是：0x%x ",s);
59
60
61
62
63
```



第 56 行，先用&取得结构体 hello 内的变量 c 的地址；再用(int \*)把地址转换成 int 型变量的地址；再用\*取的地址里的内容。

结合上次内存对齐的知识，此时的 int 型地址包含的是一个内存没有对齐的 int 数，它的最低字节是原来变量 c 的内存值 0xbc，第二和第三个字节是原结构体数据对齐时的填充字节(填充值为 0x00)，最高字节是 hello.d 的最低字节 0x34。所以最后的结果为：0x340000bc。

当然，由于内存没对齐，再最后一个环节用\*取地址里面的内容时，计算机是先用 char 指针取 0xbc，再用 short 指针取 0x0000，再用 char 指针取 0x34，最后在拼成 0x340000bc 的。效率比较低下。

要点三：数组名称本身可以当作指向数组的指针使用，指向数组首元素地址，但数组里的每个成员名不是指针。

比如 `int a[120] = {0x00}`；这里 `a` 是指针，`a[0]` 就不是指针。

```
47 |  
48 | int _tmain(int argc, _TCHAR* argv[])  
49 | {  
50 |  
51 |     char tt[10] ;  
52 |  
53 |     for(int i=0;i<10;i++) tt[i] = 0x55 ;  
54 |  
55 |     int s = * (int *)tt ;  
56 |  
57 |     printf("s的值是： 0x%x ",s);  
58 |  
59 |  
60 | C:\Users\Ensense\Documents\Visual Studio 2010\Project  
61 | s的值是： 0x55555555  
62 |  
63 |
```

第 55 行，直接把数组名 `tt` 作为指针，用 `(int *)` 修改指针类型后，在取内容，得到 `0x55555555` 。

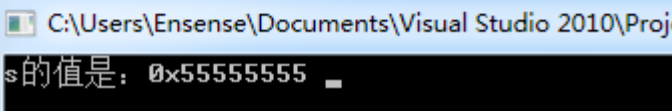
其效果与下面是一样的

```
46 |  
47 |  
48 | int _tmain(int argc, _TCHAR* argv[])  
49 | {  
50 |  
51 |     char tt[10] ;  
52 |  
53 |     for(int i=0;i<10;i++) tt[i] = 0x55 ;  
54 |  
55 |     int s = * (int *)&tt[0] ;  
56 |  
57 |     printf("s的值是： 0x%x ",s);  
58 |  
59 |  
60 | C:\Users\Ensense\Documents\Visual Studio 2010\Project  
61 | s的值是： 0x55555555  
62 |  
63 |
```

这个是先对数组首个元素 `tt[0]` 取地址，用 `(int *)` 修改指针类型后，在取内容，得到 `0x55555555` 。

其实数组名的作用还不知与此，对数组名取地址得到的还是一样的结果，感兴趣的可以自己去看 [google](#)，内容太多就不说了。

```
47 |  
48 | int _tmain(int argc, _TCHAR* argv[])  
49 | {  
50 |  
51 |     char tt[10] ;  
52 |  
53 |     for(int i=0;i<10;i++) tt[i] = 0x55 ;  
54 |  
55 |     int s = * (int *)&tt ;  
56 |  
57 |     printf("s的值是: 0x%x ",s);  
58 |  
59 |  
60 | }  
61 |  
62 |
```



## 2. 函数指针

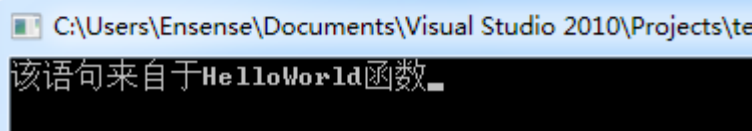
就像前面说的数组名就是指向数组首元素的常量指针，对于一个函数而言，函数名也是指向函数第一条指令的常量指针。

我们可以用一个指针来保存这个地址，而这个指针就是函数指针，该指针可以看作是它指向函数的别名，所以我们可以用该指针来调用这个函数。

看例子：

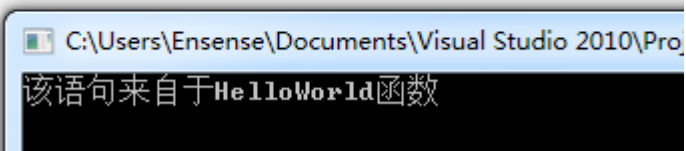
一般的函数这样写：

```
1 |  
2 | void HelloWorld(void)  
3 | {  
4 |     printf("该语句来自于HelloWorld函数");  
5 | }  
6 |  
7 | int _tmain(int argc, _TCHAR* argv[])  
8 | {  
9 |  
10 |     HelloWorld();  
11 |  
12 | }  
13 |
```



使用函数指针的程序是这样的：

```
51 |  
52 | void HelloWorld(void)  
53 | {  
54 |     printf("该语句来自于HelloWorld函数");  
55 | }  
56 |  
57 | int _tmain(int argc, _TCHAR* argv[])  
58 | {  
59 |     void (*p)(void) ;  
60 |     p = HelloWorld ;  
61 |  
62 |     p() ;  
63 |  
64 |  
65 |  
66 |  
67 |  
68 |
```



第 59 行定义了一个函数指针 `p`，该指针的返回值是 `void`，参数也是 `void`。

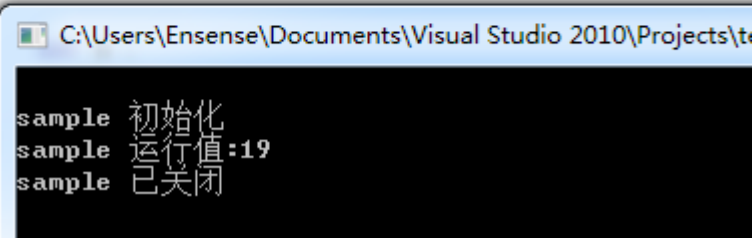
第 60 行把指针指向 `HelloWorld` 函数的地址。

第 62 行可以直接使用指针 `p` 调用函数 `HelloWorld` 了。

这个玩意的应用太灵活，可以自行脑补。来个有意思的。

```
37
38 bool sampleInit(void);
39 int sampleHandle( char val);
40 bool sampleClose(void);
41
42 #typedef struct
43 {
44     int a ;
45     int b ;
46     bool (*Init)(void);
47     int (*Handle)(char val);
48     bool (*Close)(void);
49 #}MODULE;
50
51 MODULE sample = {
52     52,
53     11,
54     sampleInit,
55     sampleHandle,
56     sampleClose ,
57 };
58
59 #bool sampleInit(void)
60 {
61     printf("\r\nsample 初始化");
62     return true ;
63 }
64
65 #int sampleHandle( char val)
66 {
67     int tmp ;
68     tmp = val+ sample.a + sample.b ;
69     printf("\r\nsample 运行值:%d",tmp);
70     return tmp ;
71 }
72
73 #bool sampleClose(void)
74 {
75     printf("\r\nsample 已关闭");
76     return true ;
77 }
78
```

```
80
81
82 int _tmain(int argc, _TCHAR* argv[])
83 {
84     sample.a = 3 ;
85     sample.b = 4 ;
86     sample.Init();
87     sample.Handle(12);
88     sample.Close();
89
90
91
92
93
94 }
95
96
97
```



是不是觉得 sample 有点 class 的意思了？