C/C++中指针与引用的用法

韩 海
(江汉大学 数学与计算机科学学院,湖北 武汉 430056)

摘 要:指针和引用的用法是C/C++语言教学中的难点,在列举指针与引用的基本用法和用作函数参数的基础上,通过对比编译得到的汇编语言代码,说明引用本质上还是指针,并给出在C++中使用引用的建议。

关键词:C++;指针;引用;编译
中图分类号:TP312 **文献标志码**:A **文章编号**:1673-0143(2017)05-0424-05
DOI:10.16389/j.cnki.cn42-1737/n.2017.05.007

Analysis on Usage of Pointer and Reference in C/C++

HAN Hai
(School of Mathematics and Computer Science, Jianghan University, Wuhan 430056, Hubei, China)

Abstract: The usages of pointer and reference in C/C++ are the difficult points in course. On the basis of introduction of the basic usages of pointer and reference, through the comparison of ASM code, the writer illustrates that the reference is pointer in essence, and gives out the suggestion on application of pointer and reference.

Keywords: C++; pointer; reference; compile

指针是C/C++课程重要的章节之一,也是较难理解和把握的章节。同时,指针又因为用途广泛、功能强大、使用灵活被称作是C语言的精华。能够准确地把握指针和熟练地使用指针是驾驭C语言的重要标志。从标准C发展到C++之后,“引用”这个新的概念又会是一个不小的障碍。把C/C++源程序编译成机器语言,再分析机器语言对应的汇编指令可以深入了解C/C++相应语句或者机制的实现原理[1]。从编译结果不难发现,引用本质上还是指针,或者说,引用是一种隐式的指针。

1 指针的用法

指针即内存地址,简称地址。在说明指针及其用法的文字描述中,把“指针”换成“地址”,绝大多数情况下都不改变描述的本意,而且可能是对其含义的更准确理解。

内存是存储数据的“容器”,这种容器的计量单位是字节,地址则是以字节为单位的容器的编号,表现为非负整数。指针即地址,是非负整数、是数据,可以被存放在内存中。用于存放地址的变量称为指针变量。比如:

```
int a=123,*q=&a;
```

该写法规定了 a 是用于存放整数的变量,q 是用于存放地址的变量,或者说 q 是用于存放指针的变量。而且,定义变量 q 时规定了初值,要求把变量 a 的地址存放到 q 中,该操作完成后 q 指向 a,或

收稿日期:2017-05-07
作者简介:韩 海(1968—),男,副教授,硕士,研究方向:图像处理与模式识别。

者说目前有 q 指向 a 的指向关系。

指针的最主要作用是通过指向关系访问指向对象。按照上述变量定义,在后续处理中,对*q的处理就是对 a 的处理,比如:

```
(*q)++;
```

该语句将把 q 所指对象的值加1,即把变量 a 的值加1,该语句执行后 a 中存放的将是124。

这样的用法可以解释指针与指向关系的概念,但无法回答学习者的疑惑:直接写a++不是更简洁明了吗?指针的最重要价值在于用指针作为函数的参数。函数是编写C语言程序的基本单位,类作为C++的基本编程单位则包含了多个函数。C/C++规定在一个函数内定义的局部变量只能在该函数内部访问,而不能被其他函数访问。因此,对于函数调用时基本的参数传递形式,主调函数把数据交给被调函数,被调函数对该数据的处理不会影响主调函数中的原数据。比如:

```
void swap_A(int a,int b)
{ int c;
  c=a,a=b,b=c;}
void main( )
{  int x=3,y=5;
     swap_A(x,y);
     printf("%d , %d",x,y);
}
```

对于函数swap_A,编程者的本意是用于交换两个变量的值。但这个函数仅仅是交换了swap_A内部的形参变量 a 、b 的值,而不是像预期的那样交换主调函数给出的实际参数 x 和 y 的值。而"局部变量谁定义谁使用"的规则又不允许在swap_A函数中直接操作main的局部变量 x 和 y 。解决这一问题的方法就是指针,正确的代码如下:

```
void swap_B(int *a,int *b)
{ int c;
  c=*a,*a=*b,*b=c;
}
void main( )
{  int x=3,y=5;
     swap_B(&x,&y);
     printf("%d , %d",x,y);
}
```

把交换函数的参数设计成指针变量,在发生调用时通过参数传递建立 a 指向 x 、b 指向 y 的指向关系,如图1所示,从而导致在函数swap_B内部时,对*a的操作就是对 x 的操作,对*b的操作就是对 y 的操作。

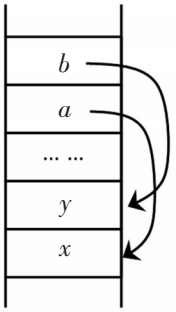

图1　调用swap_B时的指向关系

Fig. 1　Point-to relationship as calling function swap_B

不难发现,这样的处理与“局部变量不得在函数外部访问”的规则是冲突的。这是对规则的说明不清造成的,该规则完整的说法是“局部变量不得在函数外部通过变量名直接访问,但可以通过指针间接访问”。

2 引用的用法

引用是C++中引入的新概念。通常,对这个概念的解释是一段让人难以把握的描述,核心意思是“引用是变量(对象)的别名”,各种C++教材以及百度百科、维基百科无不如此[2-3]。为此,通常会用简单的例子对引用做初步的解释:

```
int a=123,&q=a;
q++;
```

在定义了变量 a 及其引用 q 之后,a 是整型变量,q 是 a 的别名,想要对 a 进行操作既可以用变量名 a 也可以用引用 q,对 q 的操作就是对 a 的操作,q ++的效果是使得变量 a 的值加1,变成124。

但是,这样的例子并不能体现引入引用这个新概念的价值,而且会让人感觉多此一举。与指针类似,引用的价值同样表现在函数调用与参数传递。以引用作为函数的参数,一方面可以简化函数代码,另一方面可以让主调函数中的调用命令更简洁[4]。比如对于前面编写的交换函数,用引用来处理写作:

```
void swap_C(int &a,int &b)
{ int c;
 c=a,a=b,b=c;
}
void main( )
{  int x=3,y=5;
      swap_C(x,y);
      printf("%d , %d",x,y);
}
```

调用swap_C时首先进行参数传递,确立 a 是 x 的引用、b 是 y 的引用,在进入函数swap_C内部后,对 a 的操作就是对 x 的操作,对 b 的操作就是对 y 的操作。

这样的用法会导致另一个困惑:规则不是要求在声明引用的时候必须对其初始化吗?这里在函数首部声明了两个引用参数,但是却是在调用的时候才初始化。关于引用还有一项规定,声明一个引用是某个变量的别名之后不得更改。但是,在上面的例子当中如果再添加一条调用命令“swap_C(y,x);”,又会出现一个矛盾:在前一次调用时 a 是 x 的引用、后一次调用时变成了 a 是 y 的引用。解释这些矛盾的根本方法是深入到引用的实现机制当中,查看源代码编译成机器语言的结果就可以揭开引用的本质。

3 引用是隐式指针

把函数swap_B、swap_C放在同一个源程序文件中,添加一个简单的主函数main分别对两个函数进行调用,在VC++ 6.0环境下进行编译并生成asm文件,截取两个函数对应的汇编语言代码见图2。

两个函数对应的汇编语言指令竟然完全相同。继续观察后续代码发现,对两个函数进行调用的如下两条C语言语句对应的汇编语言指令也相同。

```
swap_B(&x,&y);
swap_C(x,y);
```

换言之,以引用的形式编写的函数swap_C与以指针的形式编写的函数swap_B,无论是从函数内部实现还是从参数传递的方法来看,都没有本质差别。

```
?swap_B@@YAXPAH0@Z PROC NEAR
; 2    : {  int c;
        push    ebp
        mov     ebp, esp
        sub     esp, 68
        push    ebx
        push    esi
        push    edi
        lea     edi, DWORD PTR [ebp-68]
        mov     ecx, 17
        mov     eax, -858993460
        rep stosd
; 3    :    c=*a,*a=*b,*b=c; }
        mov     eax, DWORD PTR _a$[ebp]
        mov     ecx, DWORD PTR [eax]
        mov     DWORD PTR _c$[ebp], ecx
        mov     edx, DWORD PTR _a$[ebp]
        mov     eax, DWORD PTR _b$[ebp]
        mov     ecx, DWORD PTR [eax]
        mov     DWORD PTR [edx], ecx
        mov     edx, DWORD PTR _b$[ebp]
        mov     eax, DWORD PTR _c$[ebp]
        mov     DWORD PTR [edx], eax
        pop     edi
        pop     esi
        pop     ebx
        mov     esp, ebp
        pop     ebp
        ret     0
?swap_B@@YAXPAH0@Z ENDP
```

```
?swap_C@@YAXAAH0@Z PROC NEAR
; 5    : {  int c;
        push    ebp
        mov     ebp, esp
        sub     esp, 68
        push    ebx
        push    esi
        push    edi
        lea     edi, DWORD PTR [ebp-68]
        mov     ecx, 17
        mov     eax, -858993460
        rep stosd
; 6    :    c=a,a=b,b=c; }
        mov     eax, DWORD PTR _a$[ebp]
        mov     ecx, DWORD PTR [eax]
        mov     DWORD PTR _c$[ebp], ecx
        mov     edx, DWORD PTR _a$[ebp]
        mov     eax, DWORD PTR _b$[ebp]
        mov     ecx, DWORD PTR [eax]
        mov     DWORD PTR [edx], ecx
        mov     edx, DWORD PTR _b$[ebp]
        mov     eax, DWORD PTR _c$[ebp]
        mov     DWORD PTR [edx], eax
        pop     edi
        pop     esi
        pop     ebx
        mov     esp, ebp
        pop     ebp
        ret     0
?swap_C@@YAXAAH0@Z ENDP
```

图2　函数swap_B和swap_C的汇编语言代码

Fig. 2　Assembly language code of functions of swap_B and swap_C

与指针不同,引用不能称作是数据类型。从计算机和程序设计语言的角度来看,数据类型是对数据存储形式的规定,包括规定用几个字节存储该类型的数据,以及每个二进制位的含义。比如int、double等固有类型对此都有明确规定,用sizeof(类型)可以得到存储这种类型的数据需要的字节数。自定义类型以及类(class)则是把固有类型进行组合、封装,用sizeof(类)可以得到存储一个对象的字节数。但是,引用并不具备这些特性,不存在“引用型变量”的概念。比如:

```
double a , *b , &c=a ;
```

这是定义变量的基本用法,它规定了 a 是一个double型变量、b 是一个指针型变量,c 是 a 的引用。但是,这并不表示“ c 是引用型变量”,引用实际上是一种隐式的指针。引用的重要价值就表现在作为函数的形式参数,比如前述的swap_C,不妨把swap_C看作是swap_B的变形,不理解swap_B的用法也就很难掌握swap_C。

4　引用导致函数用法的拓展

在C++中引入引用这个概念,在用于函数的形式参数时可以简化编程。无论是从编写函数代码的角度看,还是从调用命令的角度看,swap_C都比swap_B简单、好用。可以说作为函数参数的引用是指针的简化版,是隐藏了星号的指针。

引用作为函数参数时,不仅仅可以作为函数的入口参数(即形式参数),也可以作为函数的出口参数(即函数返回值),并且对函数的用法有所拓展。通常情况下,函数返回值是一个数据,对函数值的进一步处理可以有计算、输出等。引用作为函数值的用法导致函数返回的是一个变量,从而出现了新用法——把函数调用放在赋值号的左边,已经有文献对这种用法的相关规则进行了多方面的探讨[5]。比如,有如下的两个求最大值的函数:

```
int max_A(int x[ ],int n)
{ int i,k=0;
  for(i=1;i<n;i++)
 if(x[k]<x[i])
        k=i;
    return x[k];}
```

```
int & max_B(int x[ ],int n)
{ int i,k=0;
  for(i=1;i<n;i++)
 if(x[k]<x[i])
        k=i;
    return x[k];}
```

针对函数max_A,函数调用放在赋值号的左边是错误的,而max_B就可以:

```
max_A(x,10)=0;   //错误
max_B(x,10)=0;   //正确
```

赋值号的用途是把一个数据存放到变量中。前一行是错误的赋值,因为max_A函数的返回值是一个整数而不是变量;max_B返回的是一个变量,因而可以把0存入该变量。

5 对相关教学的建议

指针因为功能强大、使用灵活而受到很多C语言程序员的喜爱。在C语言的教学中,应该从指针是内存字节编号这个基本概念开始,先用填充了具体数据和地址的内存图讲解指针与指向关系,让学员掌握指针的本质。在学员已具备初步概念后再像图1那样用箭头表示指向关系。既要让学员知道内存的实际情况,又要让学员重点关注指向关系导致的间接访问这种用法。

引用的概念会对学习者造成一些困扰——引用表现为类型却不是类型。引用通常用作函数的形式参数,以达到"对形参的处理就是对实参的处理"的效果。引用偶尔也用于函数的返回值,从而该函数的调用得到一个变量,导致函数调用之后的用法有所拓展。因此,除了作为函数的形式参数之外,不建议在其他情况下使用引用,而且,只有掌握了指针才能准确地使用引用,讲解引用的最根本方法就是把它转换成指针对应的用法。

参考文献(References)

[1] 火善栋, 杨旭东. 用汇编语言解析C/C++函数调用中值传递、指针传递和引用传递的内在机制[J]. 计算机时代, 2012(9):49-50.

[2] 陈维兴,林小茶. C++面向对象程序设计[M]. 北京:中国铁道出版社,2004:34-38.

[3] 韩祥波, 张艳华, 黄晶晶, 等.《C++面向对象程序设计》中引用类型的教学实践[J]. 科技视界, 2016(27):205.

[4] 钟治初. 返回指针与引用的函数[J]. 廊坊师范学院学报(自然科学版),2009,9(2):15-17.

[5] 段新娥,周锁成. 浅析C++中的引用与指针[J]. 福建电脑,2001,26(1):164.

(责任编辑:曾 婷)