C语言函数声明的陷阱

石鲁生

（苏州大学 计算机科学与技术学院，江苏 苏州 215006）

摘 要：C语言的函数声明是C语言程序设计中非常重要但又十分容易出错的地方，深入分析使用函数声明时各种错误现象，对正确使用C的函数声明很有意义。通过对使用标准C函数声明时所碰到的两个陷阱四种现象的原因的深入分析，找到错误的根源，明确指出了在使用传统和现代函数声明时应注意的问题。

关键词：C语言；函数声明；函数原型；陷阱

中图分类号：TP312C 文献标识码：A 文章编号：1003-7241(2005)11-0019-03

Traps in the C Function Declaration

SHI Lu - sheng

(College of Computer Science and Technology, Soochow University, Soo Chow 215006, China)

Abstract: According to analyzing the reasons of four kinds of phenomena of two traps in C function declaration, the sources of the malfunctions are outlined. It should be paid attention to these problems while using traditional and modern function declarations.

Key words: C language; Function declaration; Function prototype; Traps

1 引言

一个C程序是由一个主函数和若干个其他函数构成的。自C语言的原始定义以来，对函数的描述就越来越复杂。标准C语言引入了新的更好的函数声明方法，即用函数原型指定函数更多的信息。通过函数原型可以将函数的名字和函数类型以及形式参数的个数、类型、顺序通知编译系统，以便在调用函数时，系统可以对照检查。

函数原型的一般形式为：

(1)函数类型 函数名(参数类型1,参数类型2,……)；

(2)函数类型 函数名(参数类型1 参数名1,参数类型2 参数名2,……)；

对于任何被调函数，在调用之前最好都能在主调函数中通过其函数原型进行声明，这样不但可以给编译系统提供足够的被调函数信息，而且对于编程者和程序的阅读者都有很大帮助。但是为了保持向下兼容，标准C中并没有强迫程序员必须使用函数原型声明。因此标准C保留了两个缺口，正是这两个缺口为C的函数声明设下了陷阱。

收稿日期：2005-06-22

2 导致陷阱问题的两个缺口

2.1 缺口一——无需声明的被调函数

2.1.1 最简单的情况

同一源文件中如果被调函数的定义出现在主调函数之前，则主调函数中不需要再进行被调函数的声明，此时系统已经通过被调函数定义得到了其全面而准确的信息，因此可以对函数调用进行全面的检查。

2.1.2 返回值为int的函数

如果不指定函数的返回值，系统会默认函数的返回值类型为int，而当一个返回值为int的函数作为被调函数时，无论它定义在什么位置，都可以被其所在源文件中的其他函数调用。

例1：

```
①main( )
②{ int a=8,b=5,c;
③ c=min(a,b);
④ printf("min is %d",c);
}
⑤min (int m,int n)
⑥{ int t;
```

```
⑦ if(m < n) t = m;
⑧ else t = n;
⑨ return(t);
}
```

对函数 min 来说调用它之前,没有任何关于它的信息存在,但依然能够顺利通过编译、链接,执行的结果也是正确的。

2.2 缺口二 ——使用传统风格函数声明的被调函数

对于例 1 中的 min 函数因为允许使用传统的形参声明方式即:

```
int min(m,n)
int m,n;
{ …
}
```

所以同样存在着传统的函数声明方式即:int min( );

3 产生陷阱问题

3.1 第一个陷阱问题

情景 1:将例 1 中第③行的函数调用改为 min(a)。

调试:更改后的程序仍能够顺利的通过编译和链接,其某一次的执行结果是"min is 8"。当然结果是错误的。

情景 2:将例 1 中的 min 函数定义放到 main 函数之前,同样将原第③行中的函数调用改为 min(a)。

调试:更改后的程序编译时,在函数调用语句 min(a);那一行出错,提示实际参数不够。陷阱问题:情景 1 中的函数调用明明缺少了一个参数,但系统却视而不见,唯一给出的警告就是变量 b 在程序中没有使用。情景 2 中只是将函数 min 的定义提前,编译系统就检查出实际参数不够的错误了! 为什么会这样呢?

3.2 第二个陷阱问题

情景 3:例 2

```
①main( )
②{ float min( );
③ float a = 5.5,b = 3.8,c;
④ c = min(a,b);
⑤ printf("min is %f",c);
}
⑥float min (float m , float n)
⑦{ float t;
⑧ if(m < n) t = m;
⑨ else t = n;
return(t);
}
```

调试:例 2 可以顺利通过编译和链接,但某次的执行结果却是"min is 0.000000"。结果是错误的。

情景 4:例 3

```
文件 L3_1.c   文件 L3_2.c
# include "L3_2.c"    float exam(a)
float exam(float);    float a;
main( )   ①{
{ float f = 3.58f,t;   ② return a;
  t = exam(f);   }
  printf("t = %f \ n",t);
}
```

调试:例 3 可以顺利通过编译和链接,但某次的执行结果是"t = 168.959991"。结果是错误的。

陷阱问题:例 2 和例 3 的 main 函数中都有被调函数的声明,看似没有问题,但是结果却不对,这是为什么呢?

4 分析陷阱问题

4.1 陷阱问题一的分析

C 中对函数调用语句"fun(p1,p2,…,pn);"通常是产生这样一串汇编的代码:

```
①push pn
②… / * 将参数 p1,
③push p2 p2, …,pn 压栈 * /
④push p1
⑤call fun //调用函数的代码
⑥add esp,xxx //通过调整栈指针,恢复栈到调用前的状态,xxx 代表某个值
```

由于情景 1 中没有在主函数中声明被调函数 min,故系统在判断"min(a);"是一条函数调用语句后(C 中只有函数调用是这种形式),编译器就开始进行参数压栈操作,然后"call min"来执行函数的代码,其他事情根本不管,也没有必要管。实际上函数 min 是在 m(其值由实际参数 a 传递而来,为 8)和 n(其值为随机数,可以通过在 min 函数中加入输出语句"printf("n = %d",n);"来证明)中寻求最小值。但情景 2 就不同了,因为在函数调用语句"min(a);"之前已经有了函数 min 的定义,系统会根据定义的信息对函数的调用情况进行检查,当然可以查出其中的不当之处。

结论:C 在没有函数声明和定义的情况下根本不对函数的参数进行检查,不管给什么样的参数,仅仅进行压栈的操作,然后就开始调用函数,容易出错。类似情景 1 中的错误可能任何人都看得出来,但是如果在程序的规模和复杂度大大增加的情况下就不好说了! 因此虽然函数类型为 int 的函数可以不加声明就直接调用,但是当被调函数的调用先于定义出现时,最好还是在主调函数中加上被调函数的声明[2]。 (下转第 80 页)

过点亮不同的方向灯,告诉货车司机如何控制货车停到正确位置,通过方向灯的闪烁,告诉司机货车离准确位置的距离。

3.5 系统软件可实现的功能

a.定位一个车道上单个 20ft,40ft,45ft 的货柜;

b.定位一个车道上两个 20ft 的货柜(指在同一货柜车上的两个 20ft 货柜);

c.分辨同一货柜车上的两个 20ft 货柜,并定位距货柜车驾驶室较近的 20ft 货柜;

d.定位一个车道上两个 20ft 的货柜时,将计算两个货柜总长度及货柜间距离;

3.6 测量精度

扫描器每旋转扫描 0.25 度作一次位置点采集,也就是说采集点与采集点之间的距离就是系统的测量误差值。由于 0.25° 角度足够小,根据三角函数公式可以推出以下计算公式:

误差距离 = tg(0.25°) × 地面的高度

再考虑到外界位置因素的干扰,当 LMI221 高度在 14000mm ~ 24000mm 之间时,误差距离小于 100mm,这样的精度是能满足现场要求的。

4 成功应用实例

香港 HIT 码头:

HIT 码头在 14 号岸吊安装了该套系统,系统共使用了两套激光扫描器。

深圳 YICT 码头:

YICT 码头在 21 号岸吊安装了该套系统,系统共使用了 6 个激光扫描器。

下表所列出的结果是在 YICT 实际测量的数据,可以看出测量的准确率为 100%,同时效率也提高了约 8 - 10%。系统安装至今已快一年,运行状况稳定可靠,深受用户的好评并在逐步推广中。

测量次数	车 道	货车类型	货车数量	准确率	对位效率(次/小时)	
					人工	自动
第一次	第四道	货柜长 40ft,高 8.6ft	32	100%	35	39
第二次	第四道	货柜长 45ft,高 9.6ft	26	100%	35	41
第三次	第四道	货柜长 20ft,高 8.6ft,双柜	70	100%	35	37

5 参考文献:

[1] 西门子公司. SIMATIC S7 - 200 中文系统手册[Z]. 北京:西门子公司,2000

[2] 谭浩强 编著. C 程序设计(第二版)[M]. 北京:清华大学出版社,1999

作者简介:刘刚(1976 -),男,西安交通大学能源与动力工程学院在读硕士研究生,主要从事火电厂自动控制和工业自动化的研究。

(上接第 20 页)

4.2 陷阱问题二的分析

我们先在情景 3 例 2 的第⑦和第⑧行之间加入输出语句“printf("m = %d , n = %d",m , n);”,调试后发现输出的结果是“m = 0.000000,n = 2.343750”。在情景 4 例 3 中文件 L3 _ 2.c 的第①和第②行之间插入输出语句“printf("a = %f \ n",a);”调试发现输出的结果是“a = 168.959991”。通过输出我们发现 m、n 和 a 的值都很奇怪,并不是我们想象的从实际参数 a、b 和 f 接收过来的 5.5,3.8 和 3.58。

先看这样一个事实,在早期的 C 中为了处理上的方便,规定传递给函数的实际参数如果其值为浮点数类型,一律先扩展成 double 类型再入栈[3]。但是当有了函数原型的声明以后,在进行函数调用时首先会先根据函数原型对调用的参数进行分析,不会再按老的规则进行数据的调整。

情景 3 和情景 4 中的被调函数既有定义又有声明,为什么结果还会出错呢? 仔细分析发现情景 3 例 2 中被调函数的声明是传统形式(不是函数原型),而被调函数的定义却是现代形式;同样情景 4 例 3 中被调函数是通过函数原型声明的,是现代形式,而被调函数的定义却又是传统形式。所以情景 3 中函数调用时系统会首先将实际参数 a 和 b 转换成 double 型压栈,而 min 函数却从栈中按 float 类型取 a 和 b 的值分别给 m 和 n;情景 4 中由于有了函数原型的声明,函数调用时系统直接将 f 按 float 型压栈,而 exam 函数却认为栈里的 f 是被转换成 double 型压栈的,故虽然 a 是 float,但仍然从栈中取出一个 double 类型的“f”,因此就乱套了,输出 m、n 和 a 得到一些奇怪的值也就可以解释了。

结论:函数定义有传统和现代两种形式,函数的声明也有传统和现代两种形式,在使用的过程中必须配对使用,如果将传统声明和现代定义或将传统定义和现代声明结合在一起使用就会带来意想不到的错误。

5 总结

为了保持兼容性和可移植性,标准 C 的函数声明保留了两个缺口,这些缺口给我们提供了相当的方便,但同时也埋下了陷阱。为了保持与非标准实现兼容,应避免使用函数原型声明;在不涉及兼容性问题的情况下,应严格按照函数原型的方式进行声明,万万不可将传统和现代的函数声明方式混用。

6 参考文献:

[1] 姚新颜. C/C + + 深层探索[M]. 北京:人民邮电出版社,2002,55 - 72

[2] (美)SAMUEL P. HARBISON Ⅲ, GUY L. STEELE JR. 著,邱仲潘等译. C 语言参考手册[M]. 北京:机械工业出版社,2004,208 - 212

[3] 谭浩强. C 语言程序设计(第二版)[M]. 北京:清华大学出版社,2002,151 - 155

作者简介:石鲁生(1978 -),男,江苏宿迁人,在职研究生,助教,主要研究方向为动态模糊理论在计算机科学中的应用。