[揭开scanf函数工作机制的谜雾](http://blog.csdn.net/chinahuanyang/article/details/5903451)

摘要: scanf函数是C语言常用的输入函数,但在使用过程中我们总是发现它有着这样那样的“陷阱”或“缺陷”,从本质上来说,这是由scanf函数的工作原理和操作系统的内存缓冲机制引起的。将从两个简单程序出发来阐明它们。

关键词: C语言;scanf;陷阱;工作机制;缓冲区1 示例程序引发的疑问

1.1 程序1

试图输入字符串“Hello world!”和“How are you?”分别存入字符数组s1、s2中;再接收一个字符存入字符变量ch中;最后输出s1、s2和内容及ch的ASCII码。

(说明:本文中符号□和↙分别表示键盘上的空格和回车键)

#include

void main( )

{

char ch, s1[20], s2[20];

scanf("%s",s1);

scanf("%s",s2);

scanf(“%c”, ch);

printf("s1=%s,s2=%s,ch=%d",s1,s2,ch);

}

程序运行时,

如果输入:Hello world!↙

结果显示:s1=Hello,s2=world!,ch=10<程序结束>

显然并未达到预期效果,第一:用户没来得及输入s2的内容;第二:s1和s2的内容不对;第三:程序未等待用户输入字符ch的内容,却直接显示值。

1.2 程序2

试图将1和2分别赋值给整型变量a和b。(只给出关键代码)

int a,b;

scanf(“%d,%d”, &a, &b);

printf(“a=%d,b=%d”,a,b);

程序运行时,

输入:1,2↙或者□1,□2↙

显示:a=1,b=2<程序结束>

若将scanf语句稍加修改为scanf(“a=%d,b=%d”, &a, &b),输入:a=1,b=2↙可达到目的,但如果效仿上面第二种输入方案结果却截然不同,

输入:□a=1,□b=2↙或者□a=1,b=2↙

显示:a=随机数,b=随机数<程序结束>

以上两个程序产生的疑问将在后文逐步得到解释。

2 scanf函数的原型

int scanf(格式说明,地址列表);

函数返回值为从标准输入流中接收的数据的个数。

第一个参数是格式说明,它是带双引号的字符串常量。关于格式字符串的说明在很多C语言教材都有详细解释,这里不再赘述。

需要注意的是第二个参数,它应该是内存地址,这就是为什么通常在scanf参数之前会看见&符号的原因,如程序1中的scanf(“%d”, ch) 语句,ch是字符变量,必须在前面加&符号以表示变量的地址。

但并不是所有的地址参数都会有&符号,如程序1中的scanf("%s",s1)语句,因为s1是数组名,而数组名即该数组的内存首地址。(尽管写成scanf("%s",&s1),编译器也不会报错,但这是危险的且不符合功能要求。)

3 空格对于scanf函数的特殊含义

一般情况下,字符串(Space)、回车(Enter)、制表(Tab)键是scanf函数的分隔符,不会被接收进而写入参数指定的空间。

程序1在运行时输入“Hello world!”,正是因为中间加了空格,scanf函数认为它是两个字符串之间的分隔符,从而将”Hello”和“world!”作为独立的数据存入s1和s2中。

但是,当scanf函数以%c格式接收单个字符时,空格、回车、Tab键将不再作为分隔符,它们以ASCII码形式存储到对应参数地址所在内存空间。

4 scanf函数工作机制与内存缓冲区

4.1 scanf函数与内存缓冲区

要解决程序1的第三个疑问,就必须提到内存缓冲区的概念。缓冲区是为了提高存储器访问效率的存在。绝大多数输入输出流是完全缓冲的,数据流先进入内存缓冲区,当它写满时才会刷新(flush)到设备或文件中。“读取”和“写入”操作实际是从缓冲区(buffer)来回复制数据。但使用标准输入或输出时,并不一定会等到缓冲区满才执行刷新操作,缓冲状态由编译器决定。

scanf函数从标准输入流(stdin)读取数据,按照格式说明参数,将数据写入参数地址所在空间。标准输入流默认的是键盘的输入流。但由于操作系统的缓冲区管理机制,从键盘上输入的数据并不是立刻被scanf函数读取,而是暂存于内存缓冲区,只有敲回车键的时候scanf函数才开始工作(尤其要注意,最后敲的回车键('/n')也会送入缓冲区)。也就是说,scanf函数取读的数据是缓冲区里的数据,只有当缓冲区为空的时候程序才会暂停下来,等待用户输入;反之,缓冲区不为空时它就不需要等待用户输入了,转而执行下一句程序指令!

对于程序1,输入的Hello□world!↙暂存于缓冲区,第一个scanf函数开始工作,识别到“hello”之后的空格后认为字符串结束,将其存入字符数组s1,结束工作;第二个scanf函数开始工作,继续从缓冲区中读入数据(此时缓冲区内的数据为□world!↙),忽略无效字符空格,将world!存入第二个字符数组;第三个scanf函数开始工作,它希望接下来的数据是一个字符型,正好此时缓冲区内还有个回车字符'/n',当然不必等待用户输入而直接读取它存入对应地址参数所指空间ch中,所以程序会显示输出字符'/n'对应的ASCII码10。

4.2 scanf函数接收处理数据的过程

Scanf函数将缓冲区的数据与其格式说明参数字符串逐一进行匹配,遇“非法字符”或者格式参数字符串达到末尾就结束工作(这里的“非法字符”指的是与格式参数中指定类型不匹配的字符)。匹配流程如图1所示。

以程序2中的scanf(“a=%d,b=%d”, &a, &b)语句为例,格式说明参数为字符串“a=%d,b=%d”,其中“%d”为格式控制符,“a=”和“,b=”分别为第一、二个待接收数据的前驱字符(很多书称做格式说明参数中的“其它字符”);“ &a”和“ &b”为地址参数。

注意:在进行第二步数据类型匹配的时候(即匹配%d的时候),允许在对应数据之前输入0个或多个空格、回车、Tab键,scanf会自动忽略它们。

因此,输入:a=1,b=2↙或a=□1,b=□2↙效果是一样的,都会让a值为1,b值为2。

scanf函数对上面输入数据的匹配过程如下:

1)将前驱字符“a=”与缓冲区的数据进行匹配,完全相同;

2)进行数据类型“%d”的匹配,如果这时缓冲区内有空格之类分隔符,忽略它们,检测到整型数据1,类型合法(根据1后面的非整型数据逗号判断出整型数据输入结束),将1写入变量a的空间;

3)继续进行下一轮匹配,先对照第二个数据的前驱字符“,b=”,完全相等;

4)按格式说明参数匹配“%d”,忽略空格,接收数据2,存入对应空间;

5)检测到格式说明参数字符串已达到末尾,结束工作。

但是,如果输入:□a=1,□b=2↙则在scanf函数测试到第一个输入字符空格的时候就认为它不匹配格式参数中要求的“a=”,立刻停止工作。因此变量a、b并没有被写入数据,它们的值是随机数(a、b是局部变量)。

再如,输入:a=1,□b=2↙则a值为1,b值为一个随机数。因为当检测到“,□b=”与格式说明参数中的前驱字符“,b=”不匹配时,scanf函数结束工作,用户输入的整数2并未被识别进而存入变量b中。

说明:若格式说明参数中有空格,将自动被忽略。如语句scanf(“□%d□%d”, &a, &b)等价于scanf(“%d%d”, &a, &b),可以认为没有“前驱字符”,匹配流程跳过第一步。

5 scanf引起的死循环分析

在进行菜单显示时，我们常常采用这种形式的程序框架。

while（1）

{

//显示菜单

……

//获取用户选择的菜单功能

scanf("%d",a);

//后续处理

……

}

问题：当输入一个字符时就会死循环，如果在scanf后面加一个getchar（）；就好了，但是我不明白为什么。

研究过scanf函数是怎样接收整型数字的吗？它要跳过所有非数字的字符专接收数字。那么输入了字符它就一直留在输入缓冲区；只要缓冲区有字符，scanf就要去读，一读不是数字，它就跳过，试图等到输入缓冲区没有字符了才等待你输入……如此反复。具体来说，当用户第一次输入了一个非数字的字符，如输入了’x’，这个字符会驻留在缓冲区中，scanf函数开始执行，检查缓冲区，没有接收到需要的数值量，会返回0，然后本次循环的scanf函数就执行完了，但缓冲区中的字符’x’没有清除；当下次循环执行到scanf函数时，由于缓冲区中仍有字符’x’，不是空的，此时scanf函数会直接从缓冲区尝试获取一个数值量，当然找不到，仍会返回0；注意由于scanf函数执行时缓冲区非空，此时scanf函数不会暂停程序等待用户的输入，这样由于’x’字符一直驻留在缓冲区中，后续每次scanf函数执行时缓冲区都不为空，此时就就会“死循环”了。

解决办法1是可以用 fflush(stdin)语句清除缓冲区多余的字符。

解决办法2是用getchar（）函数，当scanf函数失败，缓冲区的字符会被getchar（）函数从缓冲区里面读出来，这时缓冲区里面又空了，所以程序提醒用户可以再次输入数据。其实不管哪一种情况都是死循环的，只是表现不一样而已，第二种情况只是你还能输入东西罢了。