

scanf

编辑

与printf函数一样，都被定义在头文件stdio.h里，因此在使用scanf函数时要加上#include <stdio.h>。它是格式输入函数，即按用户指定的格式从键盘上把数据输入到指定的变量之中。

中文名

格式输入

应用学科

计算机软件

外文名

Scan Format

软件语言

C/C++

外语缩写

scanf

属 性

头文件函数

目录

1 函数原型

2 返回值

3 字符说明

- 格式说明符
- 空白符
- 非空白符

4 注意问题

- 问题一
- 问题二
- 问题三
- 问题四

5 发展

1 函数原型

编辑

1 | intscanf(constchar*format,...);

函数scanf()是从标准输入流stdio(标准输入设备，一般是键盘)中读内容的通用子程序，可以说明的格式读入多个字符，并保存在对应地址的变量中。

其调用形式为: scanf("<格式说明字符串>", <变量地址>);变量地址要求有效，并且与格式说明的次序一致。

2 返回值

编辑

scanf()函数返回成功赋值的数据项数，读到文件末尾出错时则返回EOF。

如：

1 | scanf("%d %d",&a,&b);

如果a和b都被成功读入，那么scanf的返回值就是2

如果只有a被成功读入，返回值为1

如果a和b都未被成功读入，返回值为0

如果遇到错误或遇到end of file，返回值为EOF。

且返回值为int型。

例：使用scanf函数输入数据。

1 | #include<stdio.h>
2 | int main(void)
3 | {
4 | int a,b,c;
5 | printf("输入a,b,c\n");
6 | scanf("%d%d%d",&a,&b,&c);

scanf图册

相关术语

纠错

链表

冒泡排序

递归

数组

getchar

ascii

fwrite

快速排序算法

itoa

词条统计

浏览次数: 591117次

编辑次数: 91次 历史版本

最近更新: 2015-02-12

创建者: sunhaofen

百科消息:

安全用耳，保护听力的正确方式

3D恐龙博物馆，邀你一起来体验

【公告】词条打标签功能上线啦！

景宁畲族自治县畲族博物馆上线啦！

```
7  printf("a=%d,b=%d,c=%d\n",a,b,c);
8  fflush(stdin);
9  return 0;
10 }
```

&a,&b,&c中的&是地址运算符，&a指a在内存中的地址。scanf的作用是：按照a，b，c的内存地址将输入的数据存到a，b，c中去。变量a，b，c的地址是在编译连续阶段分配的（存储顺序由编译器决定）。

这里注意：如果scanf中%d是连着写的如"%d%d%d",在输入数据时，数据之间不可以加逗号，只能是空格或tab键或者回车键——“2 3 4”或“2（按tab）3（按tab）4（按tab）”。若是"%d，%d，%d"，则在输入数据时需要加“，”，如“2，3，4”。

3

字符说明

编辑

在 C++ 中，format 用 restrict 修饰。

format 指向的控制串由以下三类[字符](#)组成：

格式说明符

[转换字符](#)(就是%后跟的部分)

a 读[浮点值](#)(仅适用于 C++)

A 读浮点值(仅适用于 C++)

c 读单字符

d 读十进制整数

i 读十进制、[八进制](#)、十六进制整数

e 读[浮点数](#)

E 读浮点数

f 读浮点数

F 读浮点数(仅适用于 C++)

g 读浮点数

G 读浮点数

o 读八进制数

s 读[字符串](#)

x 读十六进制数

X 读十六进制数

p 读[指针值](#)

n 至此已读入值的等价字符数

u 读无符号十进制整数

[] 扫描字符集合

% 读 % 符号([百分号](#))

附加格式说明[字符表](#)[修饰符](#)说明

L/l 长度修饰符 输入"长"数据

h 长度修饰符 输入"短"数据

W 整型常数 指定输入数据所占宽度

* 表示本输入项在读入后不赋值给相应的[变量](#)

空白符

空白字符会使scanf()函数在读操作中略去输入中的一个或多个空白字符。

非空白符

一个非空白字符会使scanf()函数在读入时剔除掉与这个非空白字符相同的字符。

说明：

(1) %s 表示读字符串，而 %d 表示读整数。格式串的处理顺序为从左到右，格式说明符逐一与变元表中的变元匹配。为了读取长整数，可以将 L / l 放在格式说明符的前面；为了读取短整数，可以将 h 放在格式说明符的前面。这些修饰符可以与 d、i、o、u 和 x 格式代码一起使用。

(2) 默认情况下，a、f、e 和 g 告诉 scanf() 为 float 分配数据。如果将 L / l 放在这些修饰符的前面，则 scanf() 为 double 分配数据。使用 L 就是告诉 scanf()，接收数据的变量是 long double 型变量。

(3) 如果使用的现代编译器程序支持 1995 年增加的宽字符特性，则可以与 c 格式代码一起，用 l 修饰符说明类型 wchar_t 的宽字符指针；也可以与 s 格式代码一起，用 l 修饰符说明宽字符串的指针。l 修饰符也可以用于修饰扫描集，以说明宽字符。

(4) 控制串中的空白符使 scanf() 在输入流中跳过一个或多个空白行。空白符可以是空格(space)、制表符(tab)和新行符(newline)。本质上，控制串中的空白符使 scanf() 在输入流中读，但不保存结果，直到发现非空白字符为止。

(5) 非空白符使 scanf() 在流中读一个匹配的字符并忽略之。例如，"%d,%d" 使 scanf() 先读入一个整数，读入中放弃逗号，然后读另一个整数。如未发现匹配，scanf() 返回。

(6) scanf() 中用于保存读入值的变元必须都是变量指针，即相应变量的地址。

(7) 在输入流中，数据项必须由空格、制表符和新行符分割。逗号和分号等不是分隔符，比如以下代码：

```
1 | scanf("%d%d",&r,&c);
```

将接受输入 10 20，但遇到 10,20 则失败。

(8) 百分号(%)与格式符之间的星号(*)表示读指定类型的数据但不保存。因此，

```
1 | scanf("%d%c%d",&x,&y);
```

对 10/20 的读入操作中，10 放入变量 x，20 放入 y。

(9) 格式命令可以说明最大域宽。在百分号(%)与格式码之间的整数用于限制从对应域读入的最大字符数。例如，希望向 address 读入不多于 20 个字符时，可以书写成如下形式：

```
1 | scanf("%20s",address);
```

如果输入流的内容多于 20 个字符，则下次 scanf() 从此次停止处开始读入。若达到最大域宽前已遇到空白符，则对该域的读立即停止；此时，scanf() 跳到下一个域。

(10) 虽然空格、制表符和新行符都用做域分割符号，但读单字符操作中却按一般字符处理。例如，对输入流 "x y" 调用：

```
1 | scanf("%c%c%c",&a,&b,&c);
```

返回后，x 在变量 a 中，空格在变量 b 中，y 在变量 c 中。

注意，控制串中的其它字符，包括空格、制表符和新行符，都用于从输入流中匹配并放弃字符，被匹配的字符都放弃。例如，给定输入流 "10t20"，调用：

```
1 | scanf("%dt%d",&x,&y);
```

将把 10 和 20 分别放到 x 和 y 中，t 被放弃，因为 t 在控制串中。

(11) ANSI C 标准向 scanf() 增加了一种新特性，称为扫描集(scanset)。扫描集定义一个字符集合，可由 scanf() 读入其中允许的字符并赋给对应字符数组。扫描集由一对方括号中的一串字符定义，左方括号前必须缀以百分号。例如，以下的扫描集使 scanf() 读入字符 A、B 和 C：

```
1 | %[ABC]
```

使用扫描集时，scanf() 连续吃进集合中的字符并放入对应的字符数组，直到发现不在集合中的字符为止(即扫描集仅读匹配的字符)。返回时，数组中放置以 null 结尾、由读入字符组成的字符串。

用字符 ^ 可以说明补集。把 ^ 字符放为扫描集的第一字符时，构成其它字符组成的命令的补集合，指示 scanf() 只接受未说明的其它字符。

对于许多实现来说，用[连字符](#)可以说明一个范围。例如，以下扫描集使 `scanf()` 接受字母 A 到 Z:

```
1 | %[A-Z]
```

重要的是要注意扫描集是区分大小写的。因此，希望扫描大、小写字符时，应该分别说明大、小写字母。

(12) `scanf()` 返回等于成功赋值的域数的值，但由于星号[修饰符](#)而读入未赋值的域不计算在内。遇到文件结束则返回EOF；若出错则返回0。

(13) C99 为 `scanf()` 增加了几个格式[修饰符](#)：`hh`、`ll`、`j`、`z` 和 `t`。`hh` 修饰符可用于 `d`、`i`、`o`、`u`、`x`、`X` 或 `n`。它说明相应的变元是 `signed` 或 `unsigned char` 值，或用于 `n` 时，相应的变元是指向 `long char` 型[变量](#)的指针。`ll` [修饰符](#)也可用于 `d`、`i`、`o`、`u`、`x`、`X` 或 `n`。它说明相应的变元是 `signed` 或者 `unsigned long long int` 值。

`j` 格式[修饰符](#)应用于 `d`、`i`、`o`、`u`、`x`、`X` 或 `n`，说明匹配的变元是类型 `intmax_t` 或 `uintmax_t`。这些类型在 `<stdint.h>`；中声明，并说明最大宽度的整数。

`z` 格式[修饰符](#)应用于 `d`、`i`、`o`、`u`、`x`、`X` 或 `n`，说明匹配的变元是指向 `size_t` 类型对象的[指针](#)。该类型在 `<stddef.h>`；中声明，并说明 `sizeof` 的结构。

`t` 格式[修饰符](#)应用于 `d`、`i`、`o`、`u`、`x`、`X` 或 `n`，说明匹配的变元是指向 `ptrdiff_t` 类型对象的[指针](#)。该类型在 `<stddef.h>`；中声明，并说明两个[指针](#)之间的差别。

4 注意问题

[编辑](#)

(1) 对于字符串数组或字符串指针变量，由于数组名和指针变量名本身就是地址，因此使用`scanf()`函数时，不需要在它们前面加上"&"操作符。

(2) 可以在格式化字符串中的 "%" 各格式化规定符之间加入一个整数，表示任何读操作中的最大位数。

(3) `scanf()` 函数中没有精度控制。

如：`scanf("%5.2f",&a)`；是非法的。不能企图用此语句输入小数为2位的实数。

(4) `scanf`中要求给出[变量](#)地址，如给出变量名则会出错

如 `scanf("%d",a)`；是非法的，应改为`scanf("%d",&a)`；才是合法的。

(5) 在输入多个数值数据时，若格式控制串中没有非[格式字符](#)作输入数据之间的间隔,则可用空格，TAB或回车作间隔。

C编译在碰到空格，TAB，回车或非法数据(如对"`%d`"输入"`12A`"时，`A`即为非法数据)时即认为该数据结束。

(6) 在输入[字符](#)数据(`%c`)时，若格式控制串中无非[格式字符](#)，则认为所有输入的字符均为有效字符。

例如：

```
1 | scanf("%c%c%c",&a,&b,&c);
```

输入为：

```
1 | d e f
```

则把'd'赋予a，' '（空格）赋予b，'e'赋予c。因为%c 只要求读入一个[字符](#)，后面不需要用空格作为两个字符的间隔，因此把' '作为下一个字符送给b。

只有当输入为：`def`（字符间无空格）时，才能把'd'赋予a，'e'赋予b，'f'赋予c。如果在格式控制中加入空格作为间隔，

如

```
1 | scanf("%c %c %c",&a,&b,&c);
```

则输入时各数据之间可加空格。

我们用一些例子来说明一些规则：

```
1 | #include<stdio.h>
2 | int main(void)
3 | {
4 |     char a,b;
5 |     printf("input character a,b\n");
6 |     scanf("%c%c",&a,&b);/*注意两个%c之间没有任何符号*/
7 |     printf("%c%c\n",a,b);
8 |     return 0;
```

```
9 | }
```

由于scanf函数"%c%c"中没有空格，输入M N，结果输出只有M。而输入改为MN时则可输出MN两字符，见下面的输入运行情况：input character a,b

输入：

```
1 |
```

屏幕显示：

```
1 |
1 | #include<stdio.h>
2 | int main(void)
3 | {
4 |     char a,b;
5 |     printf("inputcharactera,b\n");
6 |     scanf("%c %c",&a,&b);/*注意两个%c之间有个空格*/
7 |     printf("\n%c%c\n",a,b);
8 |     return 0;
9 | }
```

本例表示scanf格式控制串"%c %c"之间有空格时，输入的数据之间可以有空区间隔。

(7) 如果格式控制串中有非格式字符则输入时也要输入该非格式字符。

例如：

```
1 | scanf("%d,%d,%d",&a,&b,&c);
```

其中用非格式符“,”作间隔符，故输入时应为：

```
1 | 5,6,7
```

又如：

```
1 | scanf("a=%d,b=%d,c=%d",&a,&b,&c);
```

则输入应为

```
1 | a=5,b=6,c=7
```

如输入的数据与输出的类型不一致时，虽然编译能够通过，但结果将不正确。

```
1 | #include<stdio.h>
2 | int main(void)
3 | {
4 |     int a;
5 |     printf("inputanumber");
6 |     scanf("%d",&a);
7 |     printf("%ld",a);
8 |     return 0;
9 | }
```

由于输入数据类型为整型，而输出语句的格式串中说明为长整型，因此输出结果和输入数据不符。输出并不是输入的值。

如将scanf("%d",&a); 语句改为 scanf("%ld",&a);

输入数据为长整型，输入输出数据才相等。

问题一

如何让scanf()函数正确接受有空格的字符串？如：I love you!

```
1 | #include<stdio.h>
2 | int main(void)
3 | {
4 |     char str[80];
5 |     scanf("%s",str);
6 |     printf("%s",str);
7 |     return 0;
8 | }
```

输入：

```
1 | I love you!
```

上述程序并不能达到预期目的，scanf()扫描到"l"后面的空格就认为对str的赋值结束，并忽略后面的"love you!"。这里要注意是"love you!"还在键盘缓冲区（关于这个问题，网上我所见的说法都是如此，但是，我经过调试发现，其实这时缓冲区字符串首尾指针已经相等了，也就是说缓冲区清空了，scanf()函数应该只是扫描stdin流，这个残存信息是在stdin中）。我们改动一下上面的程序来验证一下：

```
1  #include<stdio.h>
2  #include<windows.h>
3  int main(void)
4  {
5      charstr[80],str1[80],str2[80];
6      scanf("%s",str);/*此处输入:Iloveyou!*/
7      printf("%s\n",str);
8      Sleep(5000);/*这里等待5秒，告诉你程序运行到什么地方*/
9      /*
10     不是sleep(5)
11     1. 函数名是Sleep不是sleep。
12     2. C/C++中，unsignedSleep (unsigned) 应该是毫秒ms.
13     */
14     scanf("%s",str1);/*这两句无需你再输入,是对stdin流再扫描*/
15     scanf("%s",str2);/*这两句无需你再输入,是对stdin流再扫描*/
16     printf("%s\n",str1);
17     printf("%s\n",str2);
18     return 0;
19 }
```

输入：

```
1 I love you!
```

输出：

```
1 I
2 love
3 you!
```

好了，原因知道了，所以结论是：残留的信息 love you是存在于stdin流中，而不是在键盘缓冲区中。那么scanf()函数能不能完成这个任务？回答是：能！别忘了scanf()函数还有一个 %[] 格式控制符（如果对%[]不了解的请查看本文的上篇），请看下面的程序：

```
1  #include<stdio.h>
2  int main(void)
3  {
4      charstr[50];
5      scanf("%[^\\n]",str);/*scanf("%s",string);不能接收空格符*/
6      printf("%s\\n",str);
7      return 0;
8  }
```

问题二

键盘缓冲区残余信息问题

```
1  #include<stdio.h>
2  int main(void)
3  {
4      inta;
5      charc;
6      while(c!='\\n')
7      {
8          scanf("%d",&a);
9          scanf("%c",&c);
10         printf("a=%dc=%c\\n",a,c);/*printf("c=%d\\n",c);*/
11     }
12     return 0;
13 }
```

scanf("%c", &c);这句不能正常接收字符,什么原因呢？我们用printf("c = %d\\n", c);将C用int表示出来，启用printf("c = %d\\n", c);这一句，看看scanf()函数赋给C到底是什么，结果是c=10,ASCII值为10是什么？换行即\\n。对了，我们每击打一下"Enter"键，向键盘缓冲区发去一个"回车"(\\r),一个"换行"(\\n),在这里\\r被scanf()函数处理掉了（姑且这么认为吧^_^），而\\n被scanf()函数"错误"地赋给了c.解决办法：可以在两个scanf()函数之后加getch(), getchar(),但是要视具体scanf()语句加那个，这里就不分析了，读者自己去摸索吧。

```
1  #include<stdio.h>
2  int main(void)
3  {
4      inta;
5      charc;
6      while(c!='\\n')
7      {
```

```
8 scanf("%d",&a);
9 fflush(stdin);
10 scanf("%c",&c);
11 fflush(stdin);
12 printf("a=%dc=%c\n",a,c);
13 }
14 return 0;
15 }
```

这里再给一个用“空格符”来处理缓冲区残余信息的示例：

版本1：运行出错的程序

```
1 #include<stdio.h>
2 int main(void)
3 {
4     inti;
5     charj;
6     for(i=0;i<10;++i)
7         scanf("%c",&j);/*这里%前没有空格*/
8         printf("%c",j);/*在输入十个字符之后*/
9     return 0;
10 }
```

版本2：使用了空格控制符后

```
1 #include<stdio.h>
2 int main(void)
3 {
4     inti;
5     charj;
6     for(i=0;i<10;++i)
7         scanf(" %c",&j);/*注意这里%前有个空格*/
8         printf("%c",j);/*在输入十个字符之后，验证打印出来的字符是否是自己输入的最后一个字符（即输
9     return 0;
10 }
```

接着，我们运行看看，首先，运行第一个版本（错误的程序）

我们输入：

0 1 2 3 4 5 6 7 8 9

结果是一个空字符

再运行第二个版本（正确的程序）

同样输入：

0 1 2 3 4 5 6 7 8 9

这一次就显示字符9，故此程序正确。

那么为什么第二个程序就正确呢，原因何在，在%前面加一个空格就这么有用，答案是肯定的，就是%前面的空格在起作用，读者看看此文章的前面部分，在scanf的使用过程中应注意的问题中已经指出：“scanf()的格式控制串可以使用空白字符或其它非空白字符，使用空白字符会使scanf()函数在读操作中略去输入中的零个或多个空白字符。”

所以在%前面加上了空格（空格属于空白字符，此外还有像制表符等也属于空白字符），在输入过程中，将略去输入中的一个或多个空白字符，所以我们输入的0 1 2 3 4 5 6 7 8 9这些字符中的空白字符就被略去了，字符9也就正确的打印出来了，这样子解释，相信大家都看明白勒吧！

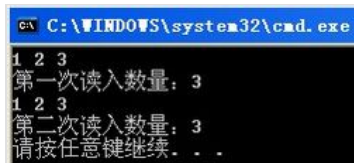
问题三

输入类型与格式化字符串不匹配导致stdin流的阻塞。

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int a=0,b=0,c=0,ret=0;
5     ret=scanf("%d%d%d",&a,&b,&c);
6     printf("第一次读入数量: %d\n",ret);
7     ret=scanf("%d%d%d",&a,&b,&c);
8     printf("第二次读入数量: %d\n",ret);
9     return 0;
10 }
```

我们定义了a, b, c三个变量来接受输入的内容, 定义了变量ret来接收scanf函数的返回值。

正确输入的话:



```
C:\WINDOWS\system32\cmd.exe
1 2 3
第一次读入数量: 3
1 2 3
第二次读入数量: 3
请按任意键继续. . .
```

但是当输入内容与格式换字符串不匹配时, 结果会令人大跌眼镜(仔细分析会对scanf函数和stdin流有更深入的了解):



```
C:\WINDOWS\system32\cmd.exe
1 b 2
第一次读入数量: 1
第二次读入数量: 0
请按任意键继续. . .
```

执行到第一个scanf时, 当输入字符'b'的时候与ret=scanf("%d%d%d",&a,&b,&c);中的格式化字符串不匹配, stdin流被阻塞, scanf函数不在读取后面的部分, 直接将1返回, 表示只将stdin流中的1读入到了变量a中。

执行到第二个scanf时, 字符'b'还是与格式化字符串不匹配, stdin流仍然被阻塞, 所以没有提示输入, scanf函数将0返回。

将代码作如下修改, 可以有力的证明上述结论。

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int a=0,b=0,c=0,ret=0;
5     ret=scanf("%d%d%d",&a,&b,&c);
6     printf("第一次读入数量: %d\n",ret);
7     ret=scanf("%c%d%d",&a,&b,&c);
8     printf("第二次读入数量: %d\n",ret);
9     return 0;
10 }
```

当把第二个scanf函数内的格式化字符串改为"%c%d%d"时, 运行结果如下:



```
C:\WINDOWS\system32\cmd.exe
1 b 2
第一次读入数量: 1
6
第二次读入数量: 3
请按任意键继续. . .
```

执行到第一个scanf函数时, 由于输入'b'的原因scanf函数直接返回1, stdin流阻塞。

执行到第二个scanf函数时, 字符'd'与格式化字符串"%c%d%d"中的%c匹配, stdin流终于疏通, 在输入6, 则将变量a, b, c分别赋值为98('b'的ASCII码)、2、6, scanf函数返回3。

有上述问题可知, 当使用scanf函数时, 如果遇到一些匪夷所思的问题, 在scanf函数后正确使用fflush(stdin);, 清空输入缓冲区, 可以解决很多问题。以本题为例:

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int a=0,b=0,c=0,ret=0;
5     ret=scanf("%d%d%d",&a,&b,&c);
6     fflush(stdin);
7     printf("第一次读入数量: %d\n",ret);
8     ret=scanf("%d%d%d",&a,&b,&c);
9     fflush(stdin);
10    printf("第二次读入数量: %d\n",ret);
11    return 0;
12 }
```

运行结果:



```
C:\WINDOWS\system32\cmd.exe
1 b 2
第一次读入数量: 1
1 3 6
第二次读入数量: 3
请按任意键继续. . .
```


问题解决。

问题四

如何处理scanf()函数误输入造成程序死锁或出错

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int a,b,c;
5     scanf("%d,%d",&a,&b);
6     c=a+b; /*计算a+b*/
7     printf("%d+%d=%d",a,b,c);
8     return 0;
9 }
```

如上程序，如果正确输入a,b的值，那么没什么问题，但是，你不能保证使用者每一次都能正确输入，一旦输入了错误的类型，你的程序不是死锁，就是得到一个错误的结果,呵呵，这可能所有人都遇到过的问题吧？解决方法：scanf()函数执行成功时的返回值是成功读取的变量数,也就是说，你这个scanf()函数有几个变量，如果scanf()函数全部正常读取，它就返回几。但这里还要注意另一个问题，如果输入了非法数据，键盘缓冲区就可能还有个残余信息问题。正确的例程：

```
1 #include<stdio.h>
2 int main(void)
3 {
4     int a,b,c;
5     while(scanf("%d%d",&a,&b)!=2)
6         fflush(stdin);
```

scanf

cwc836578502

```
7     return 0;
10 }
```

补充

fflush(stdin)这个方法在GCC下不可用。（在VC6.0下可以）

以下是 C99 对 fflush 函数的定义：

int fflush(FILE *stream);

分享

如果stream指向输出流或者更新流（update stream），并且这个更新

执行的操作不是输入，那么fflush函数将把任何未被写入的数据写入st

指向的文件（如标准输出文件stdout）。否则，fflush函数的行为是不

C和C++的标准里从来没有定义过 fflush(stdin)。

fflush（NULL）清空所有输出流和上面提到的更新流。如果发生写错误，fflush

函数会给那些流打上错误标记，并且返回EOF，否则返

由此可知，如果 stream 指向输入流（如 stdin），那么 不确定的。故而使用

fflush(stdin) 是不正确的，至少是移植性不好的。

可采用如下方法：

方法一：

```
1 /*此函数可以和scanf函数一起使用，但使用%c输入时要注意，即此函数只能用于缓冲区非空的情况*/
2 #include<stdio.h>
3 void flush()
4 {
5     char c;
6     while((c=getchar())!='\n'&&c!=EOF);
7 }
8 int main(void)
9 {
10     int a,b,c; /*计算a+b*/
11     while(scanf("%d%d",&a,&b)!=2)
12         flush();
13     c=a+b;
14     printf("%d+%d=%d",a,b,c);
15     return 0;
16 }
```

- 1 函数原型
- 2 返回值
- 3 字符说明
 - 3.1 格式说明符
 - 3.2 空白符
 - 3.3 非空白符
- 4 注意事项
 - 4.1 问题一
 - 4.2 问题二
 - 4.3 问题三
 - 4.4 问题四
- 5 发展

方法二：

使用getchar()代替fflush(stdin)

程序示例：

```
1  #include<stdio.h>
2  int main(void)
3  {
4      inti,c;
5      while(1)
6      {
7          printf("Pleaseinputaninteger:");
8          scanf("%d",&i);
9          if(feof(stdin)||ferror(stdin))
10         {
11             //如果用户输入文件结束标志（或文件已被读完），或者发生读写错误，则退出循环
12             //dosomething
13             break;
14         }
15         //没有发生错误，清空输入流。通过while循环把输入流中的余留数据“吃”掉
16         while((c=getchar())!='\n'&&c!=EOF);
17         //可直接将这句代码当成fflush(stdin)的替代，直接运行可清除输入缓存流
18         //使用scanf("%*[^\\n]");也可以清空输入流，不过会残留\\n字符。
19         printf("%d\\n",i);
20     }
21     return 0;
22 }
```

5 发展

[编辑](#)

使用scanf函数进行输入，必须指定输入的数据的类型和格式，不仅繁琐复杂，而且很容易出错.C++保留scanf只是为了和C兼容，以便过去用C语言写的程序可以在C++的环境下运行。C++的编程人员都愿意使用cin进行输入，很少使用scanf。

词条图册

更多图册▶

```
#include<stdio.h>
int main()
{
    int a=0;
    scanf("%d",&a);
    printf("%d",a);
}
```

词条图片 (5张)

1/1

词条标签：[计算机科学](#)

新手上路

- 成长任务
- 编辑入门
- 编辑规则
- 百科术语

我有疑问

- 常见问题
- 我要提问
- 参加讨论
- 意见反馈

投诉建议

- 举报不良信息
- 未通过词条申诉
- 投诉侵权信息
- 封禁查询与解封