# **C语言之清空缓存区**



[宇哲\_安菲尔德](https://blog.csdn.net/qq_26768741" \t "https://blog.csdn.net/qq_26768741/article/details/_blank) 2016-03-19 22:33:11 IMG_257 54361 IMG_258 收藏 99

分类专栏： [C](https://blog.csdn.net/qq_26768741/category_6166214.html" \t "https://blog.csdn.net/qq_26768741/article/details/_blank) [一起学习C/C++](https://blog.csdn.net/qq_26768741/category_9265865.html" \t "https://blog.csdn.net/qq_26768741/article/details/_blank) 文章标签： [c语言](https://www.csdn.net/gather_2c/MtTaEg0sMzc0NzktYmxvZwO0O0OO0O0O.html" \t "https://blog.csdn.net/qq_26768741/article/details/_blank)

版权

在C语言中，我们常常需要去清空缓存区，对于缓存区清空的重要性，接下来我们进行具体的说明。  
首先我们先解释一下缓存区。

## ****1.缓存区****

C语言中的缓冲区又称为缓存，它是内存空间的一部分。  
　　也就是说，在内存空间中预留了一定的存储空间，这些存储空间用来缓冲输入或输出的数据，这部分预留的空间就叫做缓冲区。  
　　　　C语缓冲区分为三种类型：1、全缓冲 2、行缓冲 3、不带缓冲。  
　　缓冲区根据其对应的是输入设备还是输出设备，分为输入缓冲区和输出缓冲区。

例如，在我们平时要在磁盘中读取信息的情况下，先会把数据放到缓存区中，读取完后，再次从磁盘中读取信息。  
缓存区，他的意义就是在高速CPU与低速的设备之间的一个区域，这个区域让CPU工作效率更高。

1) 全缓冲  
当填满标准I/O缓存后才进行实际I/O操作。全缓冲的典型代表是对磁盘文件的读写。

2) 行缓冲  
当在输入和输出中遇到换行符时，执行真正的I/O操作。这时，我们输入的字符先存放在缓冲区，等按下回车键换行时才进行实际的I/O操作。典型代表是标准输入(stdin)和标准输出(stdout)。

3) 不带缓冲  
也就是不进行缓冲，标准出错情况stderr是典型代表，这使得出错信息可以直接尽快地显示出来。

大部分系统默认使用下列类型的缓存：  
标准出错是不带缓存的。  
如果是涉及终端设备的流，则它们是行缓存的，否则是全缓存的。

我们经常用到的输入输出流，在目前的ANSI C 中缓存的特征是：stdin和stdout是行缓存；而stderr是无缓存的。

了解了这些，接下来我们进入正题，接下来我们要清楚为什么要清空缓存区，  
这是因为在你有的时候在输入一个字符后，在输入一个字符，如果你不清空缓冲区，那上一个字符还在你的缓冲区内！这样就造成错误了！

## **2.清空缓存区**

所以我们要对缓存区清空是很重要的，在这里我们给出了下面三种办法来实现：

* 使用fflush（）函数

fflush()函数冲洗流中的信息，该函数通常用于处理磁盘文件。清除读写缓冲区，需要立即把输出缓冲区的数据进行物理写入时。fflush（）函数包含在stdio.h头文件中。  
函数原：int fflush(FILE \*stream)，在这里的stream就是所要清除缓存区的文件。  
函数的返回值：当进行刷新成功返回0，失败返回EOF。没有缓冲区或者只读打开时也返回0值。还有需要注意的是：如果fflush返回EOF,数据可能由于写错误已经丢失。  
用法示例：fflush(stdin)刷新标准输入缓冲区，fflush(stdout)刷新标准输出缓冲区。 printf(“。。。。。。。。。。。”);后面加fflush(stdout)；可提高打印效率  
代码示例：

*#include<stdio.h>#include<stdlib.h>*int main()

{

int num;

char a;

scanf("%d", &num);

a = getchar();

printf("num=%d,a=%c\n", num, a);

system("pause");

return 0;

}

  
在执行这个程序的时候我们会发现，在我们输入第一个字符后敲击‘\n’之后,程序就结束了，这是因为我们没有清空缓存区，getchar（）函数接收的是‘\n’,最后造成了程序的停止。  
接下来我们来使用fflush（）函数。

*#include<stdio.h>#include<stdlib.h>*int main()

{

int num;

char a;

scanf("%d", &num);

fflush(stdin);

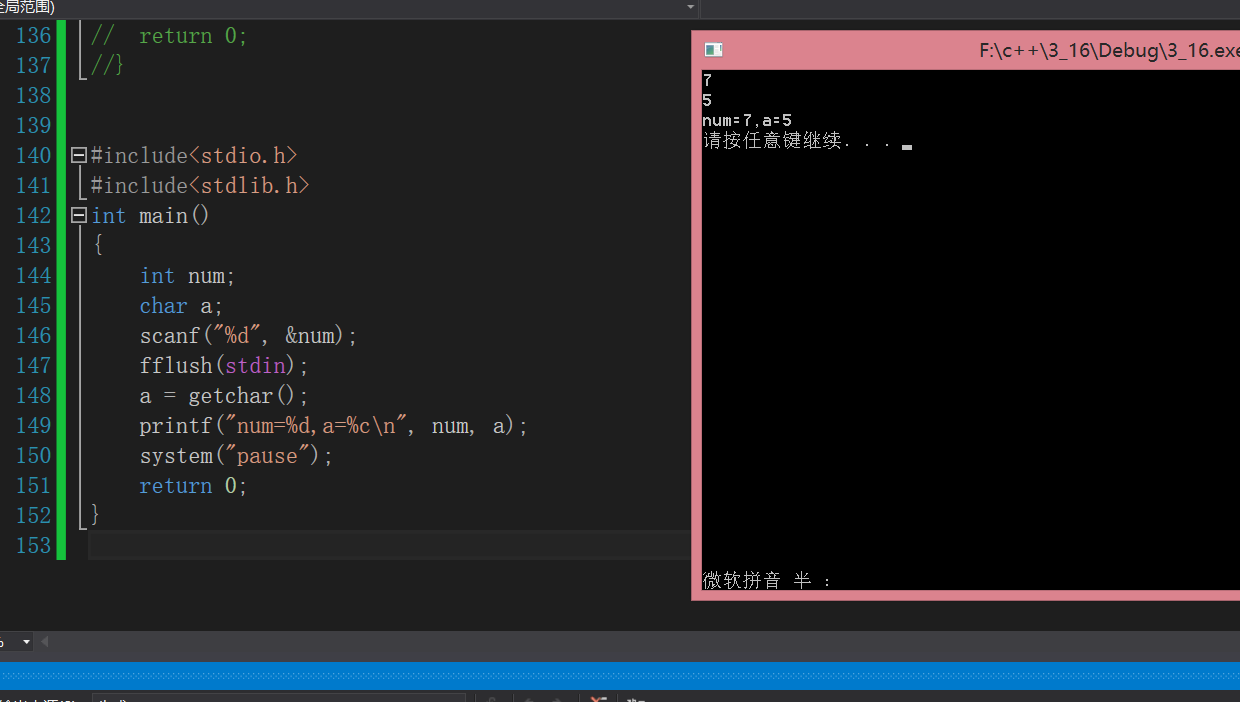
a = getchar();

printf("num=%d,a=%c\n", num, a);

system("pause");

return 0;

}

  
我们在这里可以看到，当使用了fflush（）函数对输入流进行清空缓存区以后，就可以正常让getchar函数接收字符。  
接下来是第二种方法。

使用while((ch = getchar()) != ‘\n’ && ch != EOF);语句

这种方法是最好的方法，可以称为万能清空缓存区御语句。  
代码示例：

#include <stdio.h>#include<stdlib.h>

int main()

{

char c1 = 0;

char c2 = 0;

scanf("%c", &c1);

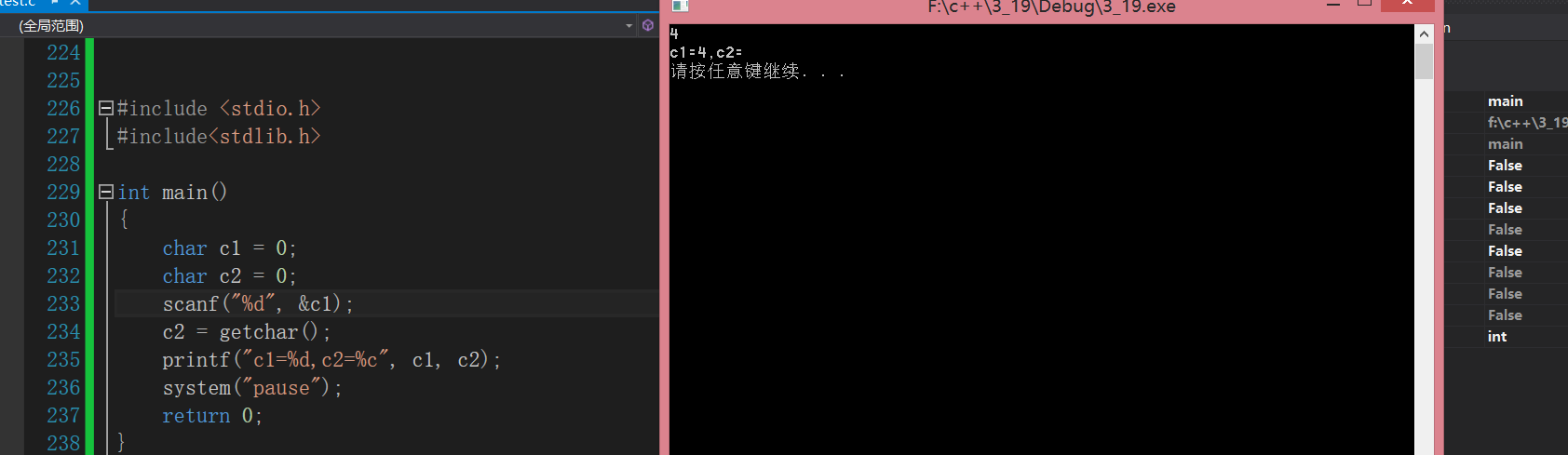
c2 = getchar();

printf("c1=%d,c2=%c", c1, c2);

system("pause");

return 0;

}



在这里我们依然没有清空缓存区，得到的效果是这样的，getchar（）函数依然接收了’\n’,造成程序停止。  
接下来，我们使用while((ch = getchar()) != ‘\n’ && ch != EOF);语句

#include <stdio.h>#include<stdlib.h>

int main()

{

char c1 = 0;

char c2 = 0;

int ch;

scanf("%d", &c1);

while ((ch = getchar()) != EOF && ch != '\n')

{

;

}

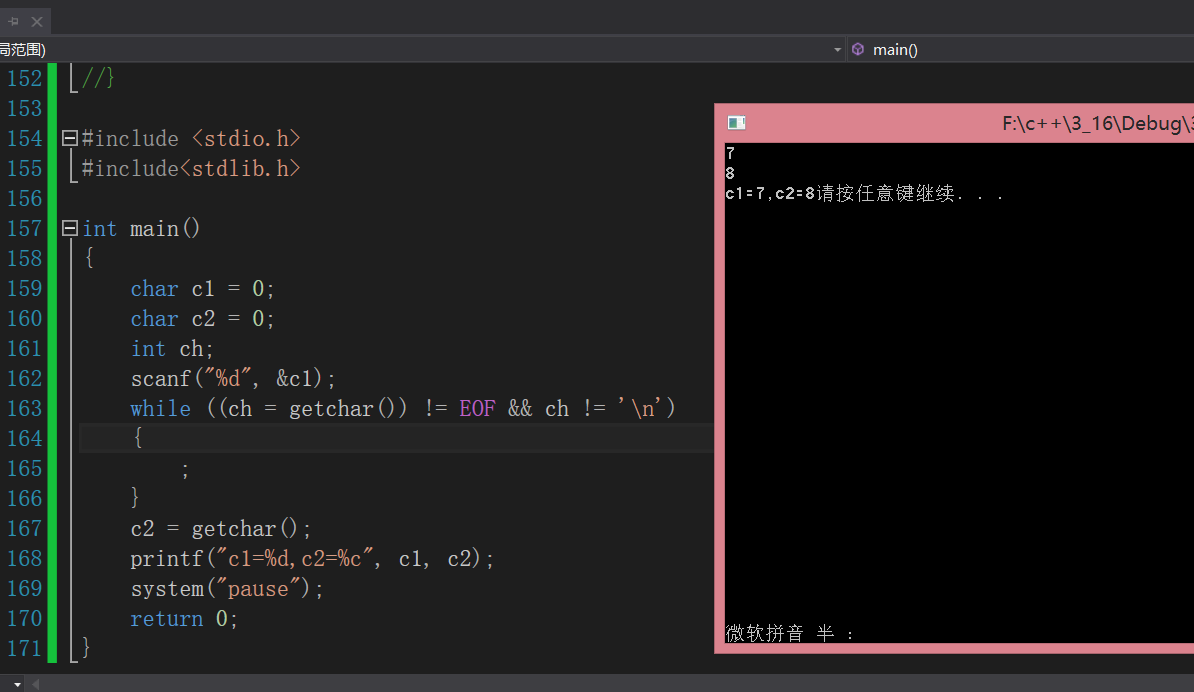
c2 = getchar();

printf("c1=%d,c2=%c", c1, c2);

system("pause");

return 0;

}



加入了while ((ch = getchar()) != EOF && ch != ‘\n’)；语句，完成了清空缓存区  
事实上有时我们会将这个语句封装在一个函数中，这样会让程序的可移植性更强。

#define CLEAR\_BUF \

{\

int ch; \while ((ch = getchar()) != EOF && ch != '\n')\

{\

; \

}\

}

#include <stdio.h>#include<stdlib.h>

int main()

{

char c1 = 0;

char c2 = 0;

scanf("%d", &c1);

CLEAR\_BUF

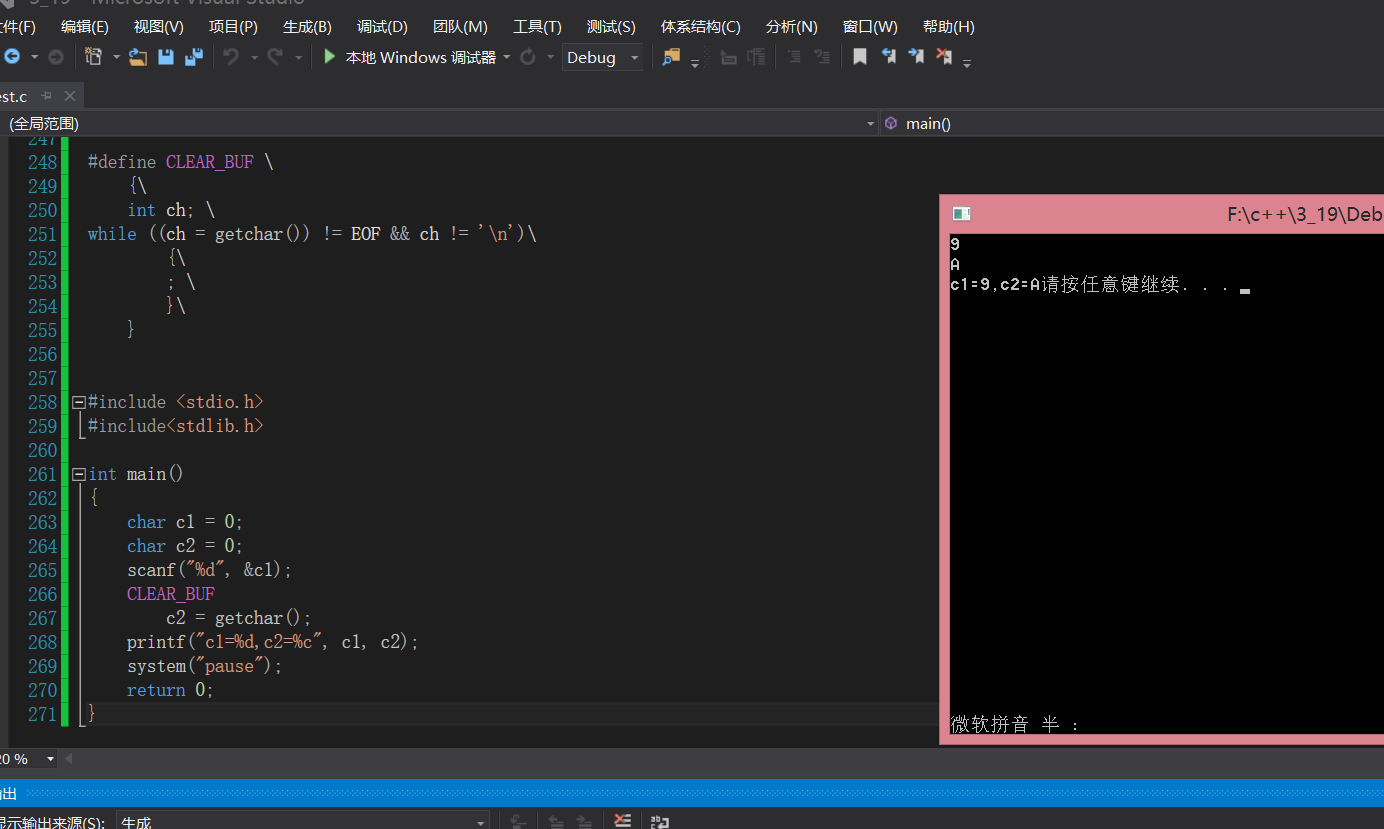
c2 = getchar();

printf("c1=%d,c2=%c", c1, c2);

system("pause");

return 0;

}

  
依然可以达到我们要的效果。

* setbuf（）函数关闭缓存区

接下来我们要说另外一种清空缓存区的方法，与其说是清空缓存区，不如说他是关闭缓存区。  
setbuf（）函数  
是linux中的C函数，主要用于打开和关闭缓冲机制。包含在头文件stdio.h中。  
setbuf函数具有打开和关闭缓冲机制。为了带缓冲进行I/O，参数buf必须指向一个长度为BUFSIZ的缓冲区。通常在此之后该流就是全缓冲的，但是如果该流与一个终端设备相关，那么某些系统也可以将其设置为行缓冲。为了关闭缓冲，可以将buf参数设置为NULL。  
函数原型：void setbuf(FILE \*stream,char \*buf);一个参数是文件流，一个参数是buf指向的缓冲区长度，这个长度就是在stdio.h中定义的宏BUFSIZ所决定的。当定义buf为空时，setbuf函数将使的文件I/O不带缓冲。

#include <stdio.h>#include<stdlib.h>char outbuf[50];int main(void)

{

*/\* 将outbuf与stdout输出流相连接 \*/*

setbuf(stdout, outbuf);

*/\* 向stdout中放入一些字符串 \*/*

puts("This is a test of buffered output.");

puts("This output will go into outbuf");

puts("and won't appear until the buffer");

puts("fills up or we flush the stream.\n");

*/\* 以下是outbuf中的内容 \*/*

puts(outbuf);

*/\*刷新流\*/*

fflush(stdout);

system("pause");

return 0;

}

