cin<<, cin.get,cin.getline等函数深入分析  
  
                  问题描述  
  
很多初学者都认为cin函数是一个很简单的函数，其实不然！cin函数有很多需要了解的知识（比如：cin的返回值是什么，cin提供了哪些成员函数且分 别是什么作用，如cin.clear(), cin.ignore(), cin.fail(), cin.good()等等），如果没有很好的掌握，在使用的时候很可能会出问题却不知其原因！而且很多人也确确实实遇到过不少问题，以下是几个简单的例 子：  
  
程序1：  
  
#include <iostream>  
  
using namespace std;  
  
int main()  
  
{  
  
      int m, n;   
  
      cin>>m;  
  
    cin>>n;  
  
      return 0;  
  
}  
  
测试情况：  
  
如果用户每次都输入两个合法的数，程序不会出问题！  
  
但是如果用户第一次输入时给一个非法的输入，比如说输入一个字符'a'，你会发现程序不  
  
会再执行第二条输入语句。似乎有点奇怪！！  
  
（当输入与变量类型不同时，cin不会执行下去。）  
  
程序2：  
  
#include <iostream>  
  
using namespace std;  
  
int main()  
  
{  
  
      char str[8];   
  
      cin.getline(str, 5);   
  
      cout<<str<<endl;   
  
      cin.getline(str, 5);   
  
      cout<<str<<endl;   
  
      return 0;  
  
}  
  
程序的功能很简单，就是输入一个字符串再输出，再次输入一个字符串输出。程序执行情况：  
  
测试一：  
  
abcd (回车)  
  
abcd (输出)  
  
efgh (回车)  
  
efgh (输出)  
  
当用户第一次输入的字符串字符数小于4时，程序执行正常！  
  
  
  
测试二：  
  
abcdefgh (回车)  
  
abcd (输出)  
  
    (输出-换行)  
  
当用户第一次输入的字符数字符数大于4时，第一个字符串接受输入的前四个字符，而第二次的输入操作没有执行，第二个字符串输出为空。似乎也很奇怪！！！  
  
其实在很多时候都会遇到诸如此类的问题，如果不熟悉程序输入的原理和cin等一些函数的原理就不知道怎么解决！我在这里做一个简单的介绍，也许介绍得不是很准确和全面，或者存在一些误解，请大家包涵！  
  
  
  
输入操作的原理与前一节中提到的scanf函数一样，程序的输入都建有一个缓冲区，即输入缓冲区。一次输入过程是这样的，当一次键盘输入结束时会将输入的 数据存入输入缓冲区，而cin函数直接从输入缓冲区中取数据。正因为cin函数是直接从缓冲区取数据的，所以有时候当缓冲区中有残留数据时，cin函数会 直接取得这些残留数据而不会请求键盘输入，这就是例子中为什么会出现输入语句失效的原因！  
  
  
  
cin的一些输入函数和操作符cin is a extern istream object。提供了很多可用的成员函数和重载的操作符，如：cin<<, cin.get(), cin.getline()等。下面我们来了解一下这几个函数：  
  
  
  
一. cin>>  
  
该操作符是根据后面变量的类型读取数据。  
  
输入结束条件  ：遇到Enter、Space、Tab键。(这个很重要！)  
  
对结束符的处理 ：丢弃缓冲区中使得输入结束的结束符(Enter、Space、Tab)  
  
  
  
读字符的情况：  
  
程序3：  
  
#include <iostream>  
  
using namespace std;  
  
int main()  
  
{  
  
      char c1, c2;  
  
      cin>>c1;  
  
      cin>>c2;  
  
      cout<<c1<<" "<<c2<<endl;  
  
      return 0;    
  
}  
  
测试一输入：  
  
a  
  
b  
  
输出:  
  
a b  
  
  
  
测试二输入：  
  
a b  
  
输出：  
  
a b  
  
  
  
  
  
读字符串的情况：  
  
程序4：  
  
#include <iostream>  
  
using namespace std;  
  
int main()  
  
{  
  
      char str1[10], str2[10];  
  
      cin>>str1;  
  
      cin>>str2;  
  
      cout<<str1<<endl;  
  
      cout<<str2<<endl;  
  
      return 0;    
  
}  
  
测试一输入：  
  
abcd  
  
efgh  
  
输出：  
  
abcd  
  
efgh  
  
【分析】输入遇到回车符结束，很正常。  
  
  
  
测试二输入：  
  
abcd efgh  
  
输出：  
  
abcd  
  
efgh  
  
【分析】第一次读取字符串时遇到空格则停止了，将abcd读入str1，并舍弃了空格，将后面的字符串给了第二个字符串。这证明了cin读入数据遇到空格结束；并且丢弃空格符；缓冲区有残留数据室，读入操作直接从缓冲区中取数据。  
  
  
  
  
  
二．cin.get()  
  
该函数有三种格式：无参，一参数，二参数  
  
即cin.get(),  cin.get(char ch),  cin.get(array\_name, Arsize)  
  
  
  
读取字符的情况：  
  
输入结束条件：Enter键  
  
对结束符处理：不丢弃缓冲区中的Enter  
  
cin.get() 与 cin.get(char ch)用于读取字符，他们的使用是相似的，  
  
即：ch=cin.get() 与 cin.get(ch)是等价的。  
  
程序5：  
  
#include <iostream>  
  
using namespace std;  
  
int main()  
  
{  
  
      char c1, c2;  
  
      cin.get(c1);  
  
      cin.get(c2);  
  
      cout<<c1<<" "<<c2<<endl;  // 打印两个字符  
  
      cout<<(int)c1<<" "<<(int)c2<<endl;  // 打印这两个字符的ASCII值  
  
      return 0;    
  
}  
  
测试一输入：  
  
a  
  
输出:  
  
a  
  
  
  
97 10  
  
【分析】会发现只执行了一次从键盘输入，显然第一个字符变量取的'a', 第二个变量取的是Enter(ASCII值为10)，这是因为该函数不丢弃上次输入结束时的Enter字符，所以第一次输入结束时缓冲区中残留的是上次输入结束时的Enter字符！  
  
  
  
测试二输入：  
  
a b  
  
输出：  
  
a    
  
97 32  
  
【分析】显然第一个字符变量取的'a',  第二个变量取的是Space(ASCII值为32)。原因同上，没有丢弃Space字符。  
  
  
  
读取字符串的情况：  
  
cin.get(array\_name, Arsize)是用来读取字符串的，可以接受空格字符，遇到Enter结束输入，按照长度(Arsize)读取字符, 会丢弃最后的Enter字符。  
  
程序6：  
  
#include <iostream>  
  
using namespace std;  
  
int main ()  
  
{  
  
char a[20];  
  
cin.get(a, 10);  
  
cout<<a<<endl;  
  
return 0;  
  
}  
  
测试一输入：  
  
abc def  
  
输出：  
  
abc def  
  
【分析】说明该函数输入字符串时可以接受空格。  
  
  
  
测试二输入：  
  
1234567890  
  
输出：  
  
123456789  
  
【分析】输入超长，则按需要的长度取数据。  
  
  
  
程序7：  
  
#include <iostream>  
  
using namespace std;  
  
int main ()  
  
{  
  
char ch, a[20];  
  
cin.get(a, 5);  
  
cin>>ch;  
  
cout<<a<<endl;  
  
cout<<(int)ch<<endl;  
  
return 0;  
  
}  
  
测试一输入：  
  
12345  
  
输出：  
  
1234  
  
53  
  
【分析】第一次输入超长，字符串按长度取了"1234"，而'5'仍残留在缓冲区中，所以第二次输入字符没有从键盘读入，而是直接取了'5'，所以打印的ASCII值是53('5'的ASCII值)。  
  
  
  
测试二输入：  
  
1234  
  
a  
  
输出：  
  
1234  
  
97  
  
【分析】第二次输入有效，说明该函数把第一次输入后的Enter丢弃了！  
  
  
  
  
  
三．cin.getline()  
  
cin.getline() 与 cin.get(array\_name, Arsize)的读取方式差不多，以Enter结束，可以接受空格字符。按照长度(Arsize)读取字符, 会丢弃最后的Enter字符。  
  
但是这两个函数是有区别的：  
  
cin.get(array\_name, Arsize)当输入的字符串超长时，不会引起cin函数的错误，后面的cin操作会继续执行，只是直接从缓冲区中取数据。但是 cin.getline()当输入超长时，会引起cin函数的错误，后面的cin操作将不再执行。（具体原因将在下一部分"cin的错误处理"中详细介 绍）  
  
程序8：  
  
#include <iostream>  
  
using namespace std;  
  
int main ()  
  
{  
  
char ch, a[20];  
  
cin.getline(a, 5);  
  
cin>>ch;  
  
cout<<a<<endl;  
  
cout<<(int)ch<<endl;  
  
return 0;  
  
}  
  
测试输入：  
  
12345  
  
输出：  
  
1234  
  
-52  
  
【分析】与cin.get(array\_name, Arsize)的例程比较会发现，这里的ch并没有读取缓冲区中的5，而是返回了-52，这里其实cin>>ch语句没有执行，是因为cin出错了！下一部分将详细介绍。  
  
问题描述一：（分析scanf()和getchar()读取字符）  |  
----------------------------------------------------  
    scanf(), getchar()等都是标准输入函数，一般人都会觉得这几个函数非常简单，没什么特殊的。但是有时候却就是因为使用这些函数除了问题，却找不出其中的原因。下面先看一个很简单的程序：  
程序1：  
    #include <stdio.h>  
    int main()  
    {  
char ch1, ch2;  
scanf("%c", &ch1);   
scanf("%c", &ch2);  
printf("%d  %d\n", ch1, ch2);  
return 0;  
    }  
    或者是：  
    #include <stdio.h>  
    int main()  
    {  
char ch1, ch2;  
ch1 = getchar();  
ch2 = getchar();  
printf("%d  %d\n", ch1, ch2);  
return 0;  
    }  
    程序的本意很简单，就是从键盘读入两个字符，然后打印出这两个字符的ASCII码值。可是执行程序后会发现除了问题：当从键盘输入一个字符后，就打印出了 结果，根本就没有输入第二个字符程序就结束了。例如用户输入字符'a', 打印结果是97，10(这是回车的ASCII码)。这是为什么呢？  
【分析】：  
    首先我们呢看一下输入操作的原理， 程序的输入都建有一个缓冲区，即输入缓冲区。一次输入过程是这样的，当一次键盘输入结束时会将输入的数据存入输入缓冲区，而cin函数直接从输入缓冲区中 取数据。正因为cin函数是直接从缓冲区取数据的，所以有时候当缓冲区中有残留数据时，cin函数会直接取得这些残留数据而不会请求键盘输入，这就是例子 中为什么会出现输入语句失效的原因！  
    其实这里的10恰好是回车符！这是因为scanf()和getchar()函数是从输入流缓冲区中读取值的，而并非从键盘(也就是终端)缓冲区读取。而读 取时遇到回车(\n)而结束的，这个\n会一起读入输入流缓冲区的，所以第一次接受输入时取走字符后会留下字符\n，这样第二次的读入函数直接从缓冲区中 把\n取走了，显然读取成功了，所以不会再从终端读取！这就是为什么这个程序只执行了一次输入操作就结束的原因！  
  
----------------------------------------------------  
|  问题描述二：（分析scanf()和gets()读取字符串）  |  
----------------------------------------------------  
首先我们看一下scanf()读取字符串的问题：  
程序2：  
    #include <stdio.h>  
    int main()  
    {  
char str1[20], str2[20];  
scanf("%s",str1);   
printf("%s\n",str1);      
scanf("%s",str2);    
printf("%s\n",str2);    
return 0;  
    }  
    程序的功能是读入一个字符串输出，在读入一个字符串输出。可我们会发现输入的字符串中不能出现空格，例如：  
测试一输入：  
Hello world!  
输出：  
Hello  
world!  
【分析】到此程序执行完毕，不会执行第二次的读取操作！这个问题的原因跟问题一类似，第一次输入Hello world!后，字符串Hello world!都会被读到输入缓冲区中，而scanf()函数取数据是遇到回车、空格、TAB就会停止，也就是第一个scanf()会取出"Hello"， 而"world!"还在缓冲区中，这样第二个scanf会直接取出这些数据，而不会等待从终端输入。  
  
测试二：  
Hello   
Hello[输出]  
world  
world[输出]  
【分析】程序执行了两次从键盘读入字符串，说明第一次输入结束时的回车符被丢弃！即：scanf()读取字符串会舍弃最后的回车符！  
  
  
我们再看一下gets()读取字符串的情况：  
用scanf来读取一个字符串时，字符串中是不可以出现空格的，一旦出现空格，后面的数据就会舍弃残留在缓冲区中。其实有另外一个函数是可以接受空格的，那就是gets()，下面我们看一下这个函数的应用，我们把程序2改动一下：  
程序3：  
#include <stdio.h>  
int main()  
{  
char str1[20], str2[20];  
gets(str1);   
printf("%s\n",str1);      
gets(str2);    
printf("%s\n",str2);    
return 0;  
}  
测试：  
Hello world! [输入]  
Hello world! [输出]  
12345 [输入]  
12345 [输出]  
【分析】显然与上一个程序的执行情况不同，这次程序执行了两次从键盘的读入，而且第一个字符串取了Hello world! 接受了空格符，而没有像上一个程序那样分成了两个字符串！所以如果要读入一个带空格符的字符串时因该用gets(), 而不宜用scanf()!  
  
  
--------------------------------------------------------  
| 问题描述三：（getchar()暂停程序，查看程序执行结果）|  
--------------------------------------------------------  
    不知道大家有没有遇到过这样的问题，有的编译器程序执行完后的结果界面不会停下而是一闪就没了，以至于看不到执行结果。所以很多人在程序最后加上 getchar()语句，目的是想让程序执行完后停下来，等待从终端接收一个字符再结束程序。可是发现有时候这样根本没用，程序照样跳出去了。这是为什么 呢？  
【分析】原因跟上面例子讲的一样，是因为输入缓冲区中还有数据，所以getchar()会成果读到数据，所以就跳出了！  
  
  
  
------------------  
|    【总结】    |  
------------------  
第一：要注意不同的函数是否接受空格符、是否舍弃最后的回车符的问题!  
读取字符时：  
scanf()以Space、Enter、Tab结束一次输入，不会舍弃最后的回车符（即回车符会残留在缓冲区中）；  
getchar()以Enter结束输入，也不会舍弃最后的回车符；  
读取字符串时：  
scanf()以Space、Enter、Tab结束一次输入  
gets()以Enter结束输入（空格不结束），接受空格，会舍弃最后的回车符！  
  
第二：为了避免出现上述问题，必须要清空缓冲区的残留数据，可以用以下的方法解决：  
方法1：C语言里提供了函数清空缓冲区，只要在读数据之前先清空缓冲区就没问题了！  
      这个函数是fflush(stdin)。  
方法2：自己取出缓冲区里的残留数据。  
（说实话这个语句我也没看懂，呵呵！为什么格式控制是这样的！希望高手指点一下！）  
      scanf("%[^\n]",string);