程序清单 5.18.2 textin3(2).cpp

// textin3.cpp -- reading chars to end of file

#include <iostream>

#include<cstdio>

int main()

{

using namespace std;

char ch;

int count = 0;

//cin.get(ch); // attempt to read a char

while ( cin.get(ch) ) // test for EOF

{

// ch = cin.get(); //上面已经输入了cin.get(ch)，如果在输入一遍将覆盖上次的值，只有这次的值。

cout << ch; // echo character

++count;

}

cout << endl << count << " characters read\n";

return 0;

}

cin.get(ch) 只是一个单纯的函数执行,对ch的值没有影响(传进去的参数是ch的一个复本)，但是ch = cin.get()是一个赋值操作, 对ch的值有影响。  
 采用cin.get(ch)的时候运行时输入一个^Z程序就结束了。而采用cin>>的时候，输入^Z按回车程序并不结束，可以继续输入,除非紧接着又输入了一个^Z，程序才结束。  
  
原因分析如下：      
   输入缓冲是行缓冲。当从键盘上输入一串字符并按回车后，这些字符会首先被送到输入缓冲区中存储。每当按下回车键后，cin.get()就会检测输入缓冲区中是否有了可读的数据。cin.get()还会对键盘上是否有作为流结束标志的 Ctrl+Z 或者 Ctrl+D 键按下作出检查，其检查的方式有两种：阻塞式以及非阻塞式。         
   阻塞式检查方式指的是只有在回车键按下之后才对此前是否有 Ctrl+Z 组合键按下进行检查，非阻塞式样指的是按下 Ctrl+D之后 立即响应的方式。如果在按 Ctrl+D之前已经从键盘输入了字符，则Ctrl+D的作用就相当于回车，即把这些字符送到输入缓冲区供读取使用，此时Ctrl+D不再起流结束符的作用。如果按  Ctrl+D 之前没有任何键盘输入，则 Ctrl+D 就是流结束的信号。         
   Windows系统中一般采用阻塞式检查   Ctrl+Z、Unix/Linux系统下一般采用非阻塞式的检查   Ctrl+D。楼主是在Windows系统下，因此使用阻塞式的 Ctrl+Z 来标识流的结束。        
   这种阻塞式的方式有一个特点：只有按下回车之后才有可能检测在此之前是否有Ctrl+Z按下。还有一个特点就是：如果输入缓冲区中有可读的数据则不会检测 Ctrl+Z（因为有要读的数据，还不能认为到了流的末尾）。还有一点需要知道：Ctrl+Z产生的不是一个普通的ASCII码值，也就是说它产生的不是 一个字符，所以不会跟其它从键盘上输入的字符一样能够存放在输入缓冲区。明白了这几点之后就可以来解释楼主提出的问题了。         
  从键盘上输入abcd^z加回车之后在Windows系统上是这样处理的：由于回车的作用，前面的 abcd 等字符被送到输入缓冲区 （注意：上面说过了，^z不会产生字符，所以更不会存储到输入缓冲区，缓冲区中没有   ^z 的存在）。这时，cin.get()检测到输入 缓冲区中已经有数据存在（因此不再检查是否有   ^z的输入），于是从缓冲中读取相应的数据。如果都读取完了，则输入缓冲区重新变为 空，cin.get() 等待新的输入。可见，尽管有  ^z按下，但是由于在此之前还有其它输入字符（abcd），所以流也不会结束。         
  因此，输入流结束的条件就是：^z 之前不能有任何字符输入（回车除外），否则   ^z 起不到流结束的作用。

1 地址等相关问题

// fun\_ptr.cpp -- pointers to functions

#include <iostream>

int main()

{

using namespace std;

int p[4] = { 1,2,3,4 };

cout << p << endl; //011EFA1C

cout << p+1 << endl; //011EFA20

cout << &p[1] << endl; //011EFA20

cout << \*(p+1) << endl; //2

cout << p[0] << endl; //1

cout << \*p << endl; //1

cout << \*p+1 << endl; //2

cout << &p << endl; //011EFA1C 整个数组的地址,但和第一个元素首地

址相同

cout << &p[4] << endl; //011EFA2C 进一步验证上面

cout << &p+1 << endl; //011EFA2C 再进一步验证上面

cout << \*&p << endl; //011EFA1C 和第一个元素地址相等

cout << \*&p+1 << endl; //011EFA20 和第二个元素地址相等

//cout << \*\*&P << endl;

return 0;

}