Console Programming

Console Programming, Reading

NOTE: The bellow is for your information only. You can use this method if you wish but it is optional.

In console programming, input is gathered from a user who sees a flashing cursor on the screen, types something and presses ENTER or RETURN. This sequence is normally started with a cout "prompt" to tell the user to do something. Here's how this is done in C++, using the conventions you are expected to follow in this course.

How To Input Numbers

There are several ways to input numbers in console programs using "**cin**", pronounced "see in". They all have their advantages and disadvantages, but **in this course we will use a way** that lets programs easily tell the difference between numeric and text input and handle them accordingly. So for example, a user can enter a series of test scores, entering a Q at the end of the sequence to signify the end (instead of some quirky negative number). This shows what to do, and what *not* to do in this course:

  int age; // a C/C++ whole number variable  
  cout << "Enter your age: "; // a prompt  
  **~~cin~~**~~>> age; // no, in this course, do not cin numbers directly~~  
  
  double temperature; // a C/C++ floating point variable  
  cout >> "What's the temperature today? "; // a prompt  
**~~cin~~**~~>> temperature; // no, in this course, do not cin numbers directly~~  
    
  string buf; // a C++ string  
  cout << "Enter your age: "; // a prompt  
  **cin** >> buf; age = atoi(buf.c\_str( )); // in this course, input numbers using "string buffers"  
  
  cout >> "What's the temperature today? "; // a prompt  
  **cin** >> buf; temperature = atof(buf.c\_str( )); // in this course, input numbers using "string buffers"

"**atoi**" ("a to eye") is for whole numbers; "**atof**" ("a to eff") is for floating point numbers. There is a definite difference between the two in C and C++, as will be explained a bit later.

"atoi" and "atof" are **friendly and forgiving**! If a user enters a value that is not fully numeric, like "ten years old" instead of "10" in response to age, these result in zero. They do *not* crash the program. By contrast, if you cin a number directly, and it's not a number, that *will* crash your program!

"atoi" and "atof" are in "**#include <cstdlib>**". Also, the C++ string "buf" is *reusable*. In the sample above, it's used to read age, then reused to read temperature. **Here's what using this method lets you do**:

#include <iostream>  
#include <string>  
using namespace std;  
  
#include <cstdlib> // atoi  
  
int main( )  
{  
  while (true) // a while-true-if-break loop  
  {    
    string buf;  
    cout << "Enter a score, or Q to quit ";  
    **cin** >> buf;  
**if (buf =="Q" || buf == "q") break;**  
    int score =  atoi(buf.c\_str( ));  
    ...  
  }  
}

It's also fine to use C strings for string buffers instead of C++ strings:

#include <iostream>  
using namespace std;  
  
#include <cstdlib> // atoi  
  
int main( )  
{  
  while (true) // a while-true-if-break loop  
  {    
    **char buf[10];**  
    cout << "Enter a score, or Q to quit ";  
    cin >> buf;  
    if (**buf[0] == 'Q'** || **buf[0] == 'q'**) break;  
    int score = **atoi(buf)**;  
    ...  
  }  
}

Another alternative is stoi and stof in the C++ string library, *but only for file input* -- not for console input. They work like atoi and atof in the C library, but allow the parameter to be a C++ string -- no .c\_str( ) required. The reason we cannot use these for console input in this course is that they crash the program if the input string is not a properly-formatted number. atoi and atof simply resolve to zero, or whatever number appears before the first non-digit.

How to Input Multiple Values On A Single Line

If a user types *two* (space-separated) values when you only prompted them for one, cin >> reads the *first* one only. The second one remains in the "keyboard buffer" waiting to be *ignored* or *read*. If you learned cin.ignore in previous course work, that clears the input buffer. Simply by leaving out this ignore statement, a second cin >> would read the second entered value after the space separation and not require a new input line. Like this (taking an opportunity to show stoi):

  int a, b;  
  string buf;  
  cout << "Enter two numbers separated by a space: ";  
  cin >> buf; a = stoi(buf);  
  cin >> buf; b = stoi(buf);

How To Input Text

If text input is expected to consist of **exactly *one* word**, you can use this for C++ strings:

  string favoriteColor; // a C++ string  
  cout << "Favorite color: type red, blue, or green: "; // a "prompt" telling the user what to do next  
  **cin** >> favoriteColor; // wait for the user to type something and press Enter or Return

But if you allow **zero, one, two, or more words**, it's like this:

  string fullName;  
  cout << "Enter your full name (Last, First): ";  
  **getline**(cin, fullName);

How To Allow Both Numeric *And* Text Input In The Same Program

If you use the cin >> and getline constructions in the same program, you *will* have trouble if you don't do it the right way! To avoid trouble, do this --  cin.ignore(1000, 10); after each cin >> statement, like this:

#include <iostream>  
#include <string>  
using namespace std;  
  
#include <cstdlib>  
  
int main( )  
{  
  int age;  
  string buf;  
  cout << "What is your age? ";  
  **cin** >> buf; age = atoi(buf.c\_str( ));  
  cin.ignore(1000, 10); // use after every cin >>  
  
  **string fullName**;  
  cout << "What is your name? ";  
  **getline**(cin, fullName); // do NOT use ignore for this!  
}

1000 And 10?

The first number in the cin.ignore statement shown above is the maximum number of characters to ignore. 1000 is a very large number -- more than one would ever expect to follow typed *useful* input (like a few trailing blanks). 10 is the ASCII code of the character that marks the end of an input line -- a "line feed" or "enter". It's actually written as '\n' and most programmers do write it that way. You may do so in this course, if you like. But its ASCII code is 10, which is just a bit easier to type, so we use that here.

What cin.ignore Does

When a user enters data in a console program, they type an ENTER to send it to the program being run. That ENTER goes along with whatever text is sent. cin >> does not read the ENTER and leaves it in the keyboard buffer. A following getline will be fooled into using what's left in the keyboard buffer instead of waiting for new input. cin.ignore fixes that problem.

But don't get carried away with cin.ignore. Never use it after getline, because getline has it's own built-in ignore -- it does read and discard ENTER. And if you ever want to read multiple cin >> inputs from a single line of input, don't use cin.ignore or else any input after the first item gets erased.