

Simple Input with Streams

The >> Operator

The >> operator is used to read data from a stream into C++ variables and objects. The global *cin* is the input analog to *cout*. It allows you to read information entered through the keyboard.

The following program reads a word from the user and spits it back out:

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char** argv)
{
    string userInput;
    cin >> userInput;
    cout << "User input was " << userInput << endl;
}
```

By default, input streams are *tokenized*. That is, a single call to the >> operator will return only a single item. If you ran the program above and typed "Hi there!", the output would be "User input was Hi".

Just as the << operator can output data of different types (strings, ints, etc.), the input operator can automatically parse and coerce data into different types. Here's how to read an integer:

```
int age;

cout << "Please enter your age: ";
cin >> age;
```

Obtaining an Entire Line

Input to your program is usually formatted in such a way that reading a single word is appropriate. However, there are many cases where you *don't* want the input tokenized (for example, parts of Assignment 1).

For these situations, C++ input streams have a `getline()` method. Unfortunately, this method uses C-style strings. So instead we'll use the `getline()` function defined in the standard namespace.

To read a line of text from the user, you'd use `getline()` as follows:

```
string fullName;

cout << "What is your full name? ";
std::getline(cin, fullName);
```

The End of Line Bug

You'll often find yourself switching back and forth between reading formatted text with `>>` and lines of text with `getline()`. When that happens, you might encounter some confusing situations with line endings.

For example, suppose your program requests a number from the user and then a sentence. You'll want to use `>>` to read the number directly into an `int` and `getline()` to read the sentence. The following program might be your first attempt:

```
int theNumber;
string theSentence;

cout << "Enter a number: ";
cin >> theNumber;
cout << "Enter a sentence: ";
std::getline(cin, theSentence); // BUG!
```

When you run this program, it will appear that the program never waits for the sentence – there isn't a pause for the user to type anything! What's really happening is that the input stream is still positioned at the end of the line containing the number.

The simplest way to solve this problem is to explicitly end the previous line using a throw-away `getline()` call before reading in the desired line:

```
cin >> theNumber;
string throwaway;
std::getline(cin, throwaway);
cout << "Enter a sentence: ";
std::getline(cin, theSentence);
```