C++ Reading a Line of Text

Because there are times when you do not want to skip whitespace before inputting a character, there is a function to input the next character in the stream regardless of what it is. The function is named **get** and is applied as shown.

```
cin.get(character);
```

The next character in the input stream is returned in **char** variable **character**. If the previous input was a numeric value, **character** contains whatever character ended the inputting of the value.

There are also times when you want to skip the rest of the values on a line and go to the beginning of the next line. A function named **ignore** defined in file **<iostream>** allows you to do this. It has two parameters. The first is an **int** expression and the second is a character. This function skips the number of characters specified in the first parameter or all the characters up to and including the character specified in the second parameter, whichever comes first. For example,

```
cin.ignore(80, '\n');
```

skips 80 characters or skips to the beginning of the next line depending on whether a newline character is encountered before 80 characters are skipped (read and discarded).

As another example, consider:

```
cin.ignore(4,'g');
cin.get(c);
cout << c << endl;
```

If the input to this program is "agdfg" then the input is ignored up to and including the 'g' so the next character read is 'd'. The letter "d" is then output.

If the input to this program is "abcdef" then the input is ignored for the first four characters, so the next character read is 'e'. The letter "e" is then output.

A more common usage is `cin.ignore()` with nothing in parentheses. This ignores the next input character.

We said that initial whitespace is skipped when reading data into a variable of type **string** and that the next whitespace character encountered stops the reading. How, then, can we get blanks or other whitespace characters into a string? We can use the function **getline**, which takes two parameters. The first is the name of the input stream, and the second is the name of the **string** variable into which the string is to be read. For example,

```
getline(cin, newString);
```

begins immediately reading and collecting characters into **newString** and continues until a newline character is encountered. The newline character is read but not stored in **newString**.

To illustrate the difference, consider the following snippet of code:

```
string s1;
cout << "Enter your name " << endl;
cin >> s1;
cout << s1;
```

If we run this program and enter "Bill Gates" then as output we will have:

Enter your name
Bill Gates
Bill

That is, string s1 only contains "Bill" because the space between Bill and Gates is whitespace and is used to denote the end of the input string. If we want to include everything up to the newline then we would use getline:

```
string s1;
cout << "Enter your name " << endl;
getline(cin, s1);
cout << s1;
```

If we run this program and enter "Bill Gates" then as output we will have:

Enter your name
Bill Gates
Bill Gates

We now get the entire string up to the newline character.

Mixing getline with cin

It is worthwhile to point out problems that can arise if we have code that mixes both cin and getline. **getline removes the newline from the input stream while cin does not**. This does not cause any problems if our program uses all cin's, because cin will ignore leading whitespace. But since getline does not ignore the leading whitespace, it can lead to undesirable results, namely empty strings. Consider the following example:

```
string s;
char c;
getline(cin, s);
cout << s << endl;
cin >> c;
cout << c << endl;
getline(cin, s);
cout << s << endl;
```

If our input to such a program is:

Bill Gates
Z
Foobar

Then we would expect the output to be

Bill Gates
Z
Foobar

Instead, our output is:

Bill Gates
Z
(blank, an empty string)

What is the reason for the empty string? In the line "cin >> c" we input the letter 'Z' into variable c. However, the newline when we hit the carriage return is left in the input stream. If we use another cin, this newline is considered whitespace and is ignored. However, if we use getline, this is not ignored. Instead we read a newline into the second getline, and exit. A blank string has been read into s in the second getline.

This is illustrated in the following code:

```
string s;
char c;
getline(cin, s);
cout << s << endl;
cin >> c;
cout << c << endl;
cin.get(c);
cout << int(c) << endl;
getline(cin, s);
cout << s << endl;
```

If our input is the same, the output is now:

Bill Gates
Z
10
Foobar

Here we explicitly read the newline with an extra cin.get() function call, and printed it's ASCII value out (which is 10).

In general, if we want to immediately throw away the newline that follows using cin to not mess things up when we later use getline, then use cin.ignore() immediately after a "cin >>" to discard the next value.

```
string s;
char c;
cin >> c;
cin.ignore();                    // Ignore NEWLINE
getline(cin, s);
// Use c and s here as intended
```