How to read strings and chars from the console

When you use the extraction operator (>>) to read data from the cin object, it uses *whitespace* (spaces, tabs, or new lines) to separate the data in the input stream into one or more *tokens*. Then, it reads one token per extraction expression. So, if a user enters "Grace M. Hopper",

```
cin >> name;
```

only stores "Grace" in the name variable.

You could fix this by declaring variables for the first name, middle initial, and last name and then chaining the extraction expressions like this:

```
cin >> first_name >> middle_initial >> last_name;
```

However, what if you want to give your users the flexibility to enter only a first name or a first and last name? One way to do that is to use the getline() function that's presented in figure 2-16 to read the entire line of user input.

Unlike an extraction expression, the getline() function only uses line breaks to separate the data in the stream. As a result, it reads all data up to the end of the current line of input. This function takes two arguments. The first is the cin object, and the second is the string variable where the extracted data is stored.

The first example in this figure shows how this works. Here, the first statement defines the name variable, the second statement prompts the user to enter a full name, and the third statement passes the cin object and the name variable to the getline() function. As a result, this code stores whatever the user enters in the name variable, whether it's "Grace" or "Grace Hopper" or "Grace M. Hopper".

When reading strings and chars from the console, you sometimes need to ignore characters. To do that, you can use a special kind of function known as a *member function*, which is a function that's available from an object. For example, this figure summarizes the get() function and two versions of the ignore() function. All of these methods are members of the cin object. As a result, to call them, you start by coding the object name and the *dot operator* (.). Then, you code the function name and its arguments just as you would for any other function.

The examples below the syntax summary show how this works. Here, the first statement calls the get() member function of the cin object. This gets the next character in the stream. The second statement calls the ignore() member function of the cin object. Since this statement doesn't pass any arguments to the function, it just extracts and discards the next char in the input stream. However, the third statement calls the ignore() function and passes two arguments to it. These arguments indicate that the function should discard the next 100 characters or all characters until the next space character, whichever comes first.

The last example shows how to pause program execution until the user presses the Enter key. This is useful if the console for a program closes before the program is done. Here, the first statement discards all data that the user may have entered earlier, up to 1000 characters. This statement isn't always necessary, but it makes sure that this code will work even if there are some extra characters remaining in the input stream. Then, the second statement prompts the user to press Enter. Finally, the third statement extracts and ignores the newline character that's inserted into the input stream when the user presses Enter.

The getline() function of the iostream header

Function	Description
`getline(cin, var)`	Extracts an entire line of console input, including spaces, and assigns it to the specified variable.

How to use the getline() function to read a full name

```
string name;
cout << "Enter full name: ";
getline(cin, name);
cout << "Your name is " << name;
```

The console

```
Enter full name: Grace M. Hopper
Your name is Grace M. Hopper
```

Member functions of the cin object

Member function	Description
`get()`	Gets the next character in the input stream.
`ignore()`	Extracts and discards the next character in the input stream.
`ignore(n, delim)`	Extracts and discards characters in the input stream until either the number of characters discarded is n, or the character in delim is found.

How to call a member function of an object

Syntax

```
object_name.function_name(arguments);
```

Examples

```
char initial = cin.get(); // extract and return the next char
cin.ignore();             // extract and discard the next char
cin.ignore(100, ' ');     // extract and discard the next 100 chars
                          // or all characters up to the next space
```

Code that pauses until the user presses Enter to continue

```
cin.ignore(1000, '\n');   // discard any extra characters
                          // on the current line
cout << "\nPress [Enter] to close the terminal ...\n";
cin.ignore();
```

The console

```
Press [Enter] to close the terminal ...
```

Description

- Many objects in the standard library have *member functions*. To call a member function, code the name of the object, the dot operator (.), and the name of the member function.
- You pass arguments to and get return values from a member function just like you do a regular function.

Figure 2-16 How to read strings and chars from the console

How to fix a common problem with reading strings

In most cases, the getline() function works the way you want it to. Sometimes, though, you can run into a problem if you use getline() after code that uses an extraction operator to extract data from an input stream. Figure 2-17 begins by showing this problem. Here, the first example starts by using an extraction expression to read an account number. Then, it uses the getline() function to read a full name. However, the console doesn't give the user a chance to enter a full name.

What's causing this problem? Well, when a user types a value and presses the Enter key, C++ adds the value and the newline character to the input stream. But, the extraction expression only extracts the value from the stream, not the newline character. That's fine if you're using the extraction operator because this operator ignores a leading newline character. Unfortunately, the getline() function doesn't ignore this character. As a result, getline() reads the leading newline character into the name variable, and program execution continues to the statement that writes the output to the console.

To fix this problem, you can use the ignore() function of the cin object to extract and discard the newline character that's causing the problem as shown in the second example. Once you extract and discard this newline character, the getline() function works correctly.

Some programmers consider it a best practice to call the ignore() function after every extraction expression. That way, the leading newline character is always discarded. But, it's also common to only call the ignore() function when necessary.

A common problem with reading strings

The code

```
int account_num;
cout << "Enter account number: ";
cin >> account_num;          // extracts data but leaves the newline character

string name;
cout << "Enter full name: ";
getline(cin, name);          // reads the newline character left in the stream

cout << "Name: " << name << " | Account: " << account_num;
```

The console

```
Enter account number: 1234
Enter full name:
Name:  | Account: 1234
```

The data in the cin object after the user enters "1234"

1	2	3	4	\n

How to fix the problem

The code

```
int account_num;
cout << "Enter account number: ";
cin >> account_num;

string name;
cout << "Enter full name: ";
cin.ignore();          // discards the newline character left in the stream
getline(cin, name);    // reads the next line

cout << "Name: " << name << " | Account: " << account_num;
```

The console

```
Enter account number: 1234
Enter full name: Mary Delamater
Name: Mary Delamater | Account: 1234
```

Description

- When the user presses the Enter key at the console, C++ inserts a newline character into the input stream.
- When you use the extraction operator to extract data, it skips over any leading whitespace characters (such as spaces and newline characters) and extracts the data up to the next whitespace character, leaving the whitespace character in the stream.
- When you use the getline() function to read data, it doesn't skip over a leading newline character. Instead, it returns an empty string. To skip over a leading newline character, you can call the ignore() member function of the cin object to ignore the leading newline character.

Figure 2-17 How to fix a common problem with reading strings

The Guest Book program

Figure 2-18 presents another program that reads input from the console and writes output to the console. This program accepts string input from the user and displays formatted output.

Like the Gallon Converter program, the Guest Book program starts by including the iostream header. It also includes the string header. In addition, it specifies a using directive for the std namespace so the code doesn't need to use fully qualified names for the objects in the iostream and string headers.

The main() function starts by writing the title of the program to the console. Note that this statement uses the newline character rather than the endl manipulator to add line breaks. Here, the first newline character (\n) starts a new line and the second adds a blank line after the program title.

After the title, this code prompts the user to enter a first name. Then, it uses an extraction expression to get a first name from the user. Next, it calls the ignore() function of the cin object to discard the leading newline character left in the stream by the previous extraction expression as well as any other characters left in the stream, up to 100 characters.

After the first name, the code prompts the user to enter a middle initial. Then, it uses the get() member function to get the middle initial. Next, it calls the ignore() function to discard any characters left in the stream, including the leading newline character.

After the middle initial, the code prompts the user for a last name. Here, the code uses the getline() function to get the last name, so the user can enter a last name that contains spaces. This code continues by using the getline() function to get the user's city and country.

After getting all of the input from the user, the code formats the data and writes it to the console. To do that, it uses the newline character (\n) to start new lines. In addition, it uses the + operator to create a full name by concatenating the first name, a space character, the middle initial, a period, a space character, and the last name. It also uses the + operator to create the line that contains the city and country.

As you review this code, note that each prompt uses a tab character (\t) to align the user's entry. In addition, this code encloses special characters such as the tab character in double quotes if the special character is part of a string. Otherwise, this code encloses the special character in single quotes so it's treated as a char type. That's true for other single characters as well, such as a space. As you learned earlier in this chapter, that makes your code slightly more efficient.

Also, note that this program only extracts the first name that's entered before any whitespace. As a result, if a user enters "Emmy Lou" as the first name, the program only stores "Emmy" as the first name. Similarly, this program only extracts the first character that's entered for the middle initial. As a result, if a user enters "Wiliford" for the middle initial, the program only stores "W". If that's not what you want, you can use the getline() function instead of the extraction operator to get the first name and middle initial. And if you do that, you don't need to use the ignore() function.

The console

```
Guest Book

First name:      Dave
Middle initial: Williford
Last name:       Von Ronk
City:            New York
Country:         United States

ENTRY
Dave W. Von Ronk
New York, United States
```

The code

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

using namespace std;

int main()
{
    cout << "Guest Book\n\n";

    string first_name;
    cout << "First name:\t";
    cin >> first_name;              // get first string only
    cin.ignore(100, '\n');          // discard leftover chars and newline

    char middle_initial;
    cout << "Middle initial:\t";
    middle_initial = cin.get();     // get first char only
    cin.ignore(100, '\n');          // discard leftover chars and newline

    string last_name;
    cout << "Last name:\t";
    getline(cin, last_name);        // get entire line

    string city;
    cout << "City:\t\t";
    getline(cin, city);             // get entire line

    string country;
    cout << "Country:\t";
    getline(cin, country);          // get entire line

    cout << "\nENTRY\n"             // display the entry
         << first_name + ' ' + middle_initial + ". " + last_name + '\n'
         << city + ", " + country + "\n\n";
}
```

Figure 2-18 The Guest Book program