CHAPTER 3

Expressions and Interactivity

TOPICS

3.1 The `cin` Object

CONCEPT: The `cin` object can be used to read data typed at the keyboard.

So far you have written programs with built-in data. Without giving the user an opportunity to enter his or her own data, you have initialized the variables with the necessary starting values. These types of programs are limited to performing their task with only a single set of starting data. If you decide to change the initial value of any variable, the program must be modified and recompiled.

VideoNote
Reading Input with `cin`

In reality, most programs ask for values that will be assigned to variables. This means the program does not have to be modified if the user wants to run it several times with different sets of data. For example, a program that calculates payroll for a small business might ask the user to enter the name of the employee, the hours worked, and the hourly pay rate. When the paycheck for that employee has been printed, the program could start over again and ask for the name, hours worked, and hourly pay rate of the next employee.

Just as `cout` is C++'s standard output object, `cin` is the standard input object. It reads input from the console (or keyboard) as shown in Program 3-1.

Program 3-1

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9     int length, width, area;
10
11    cout << "This program calculates the area of a ";
12    cout << "rectangle.\n";
13    cout << "What is the length of the rectangle? ";
14    cin >> length;
15    cout << "What is the width of the rectangle? ";
16    cin >> width;
17    area = length * width;
18    cout << "The area of the rectangle is " << area << ".\n";
19    return 0;
20 }
```

Program Output with Example Input Shown in Bold

```
This program calculates the area of a rectangle.
What is the length of the rectangle? 10 [Enter]
What is the width of the rectangle? 20 [Enter]
The area of the rectangle is 200.
```

Instead of calculating the area of one rectangle, this program can be used to get the area of any rectangle. The values that are stored in the `length` and `width` variables are entered by the user when the program is running. Look at lines 13 and 14:

```
cout << "What is the length of the rectangle? ";
cin >> length;
```

In line 13, the `cout` object is used to display the question "What is the length of the rectangle?" This question is known as a *prompt*, and it tells the user what data he or she should enter. Your program should always display a prompt before it uses `cin` to read input. This way, the user will know that he or she must type a value at the keyboard.

Line 14 uses the `cin` object to read a value from the keyboard. The `>>` symbol is the *stream extraction operator*. It gets characters from the stream object on its left and stores them in the variable whose name appears on its right. In this line, characters are taken from the `cin` object (which gets them from the keyboard) and are stored in the `length` variable.

Gathering input from the user is normally a two-step process:

1. Use the `cout` object to display a prompt on the screen.
2. Use the `cin` object to read a value from the keyboard.

The prompt should ask the user a question, or tell the user to enter a specific value. For example, the code we just examined from Program 3-1 displays the following prompt:

```
What is the length of the rectangle?
```

When the user sees this prompt, he or she knows to enter the rectangle's length. After the prompt is displayed, the program uses the `cin` object to read a value from the keyboard and store the value in the `length` variable.

Notice that the << and >> operators appear to point in the direction that data is flowing. In a statement that uses the `cout` object, the << operator always points toward `cout`. This indicates that data is flowing from a variable or a literal to the `cout` object. In a statement that uses the `cin` object, the >> operator always points toward the variable that is receiving the value. This indicates that data is flowing from `cin` to a variable. This is illustrated in Figure 3-1.

Figure 3-1

```
cout << "What is the length of the rectangle? ";
cin >> length;
```

Think of the << and >> operators as arrows that point in the direction that data is flowing.

```
cout ←"What is the length of the rectangle? ";
cin →length;
```

The `cin` object causes a program to wait until data is typed at the keyboard and the **[Enter]** key is pressed. No other lines in the program will be executed until `cin` gets its input.

`cin` automatically converts the data read from the keyboard to the data type of the variable used to store it. If the user types 10, it is read as the characters '1' and '0'. `cin` is smart enough to know this will have to be converted to an `int` value before it is stored in the `length` variable. `cin` is also smart enough to know a value like 10.7 cannot be stored in an integer variable. If the user enters a floating-point value for an integer variable, `cin` will not read the part of the number after the decimal point.

NOTE: You must include the `iostream` file in any program that uses `cin`.

Entering Multiple Values

The `cin` object may be used to gather multiple values at once. Look at Program 3-2, which is a modified version of Program 3-1.

Line 15 waits for the user to enter two values. The first is assigned to `length` and the second to `width`.

```
cin >> length >> width;
```

Program 3-2

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9     int length, width, area;
10
11    cout << "This program calculates the area of a ";
12    cout << "rectangle.\n";
13    cout << "Enter the length and width of the rectangle ";
14    cout << "separated by a space.\n";
15    cin >> length >> width;
16    area = length * width;
17    cout << "The area of the rectangle is " << area << endl;
18    return 0;
19 }
```

Program Output with Example Input Shown in Bold

```
This program calculates the area of a rectangle.
Enter the length and width of the rectangle separated by a space.
10 20 [Enter]
The area of the rectangle is 200
```

In the example output, the user entered 10 and 20, so 10 is stored in `length` and 20 is stored in `width`.

Notice the user separates the numbers by spaces as they are entered. This is how `cin` knows where each number begins and ends. It doesn't matter how many spaces are entered between the individual numbers. For example, the user could have entered

```
10          20
```

NOTE: The **[Enter]** key is pressed after the last number is entered.

`cin` will also read multiple values of different data types. This is shown in Program 3-3.

Program 3-3

```
1  // This program demonstrates how cin can read multiple values
2  // of different data types.
3  #include <iostream>
4  using namespace std;
5
```

```
 6  int main()
 7  {
 8     int whole;
 9     double fractional;
10     char letter;
11
12     cout << "Enter an integer, a double, and a character: ";
13     cin >> whole >> fractional >> letter;
14     cout << "Whole: " << whole << endl;
15     cout << "Fractional: " << fractional << endl;
16     cout << "Letter: " << letter << endl;
17     return 0;
18  }
```

Program Output with Example Input Shown in Bold

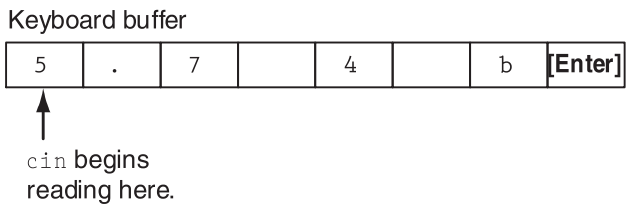
```
Enter an integer, a double, and a character: 4 5.7 b [Enter]
Whole: 4
Fractional: 5.7
Letter: b
```

As you can see in the example output, the values are stored in their respective variables. But what if the user had responded in the following way?

```
Enter an integer, a double, and a character: 5.7 4 b [Enter]
```

When the user types values at the keyboard, those values are first stored in an area of memory known as the *keyboard buffer*. So, when the user enters the values 5.7, 4, and b, they are stored in the keyboard buffer as shown in Figure 3-2.

Figure 3-2



When the user presses the Enter key, `cin` reads the value 5 into the variable `whole`. It does not read the decimal point because `whole` is an integer variable. Next it reads .7 and stores that value in the `double` variable `fractional`. The space is skipped and 4 is the next value read. It is stored as a character in the variable `letter`. Because this `cin` statement reads only three values, the b is left in the keyboard buffer. So, in this situation the program would have stored 5 in `whole`, 0.7 in `fractional`, and the character '4' in `letter`. It is important that the user enters values in the correct order.

Checkpoint

myprogramminglab *www.myprogramminglab.com*

3.1 What header file must be included in programs using `cin`?

3.2 TRUE or FALSE: `cin` requires the user to press the **[Enter]** key when finished entering data.

3.3 Assume `value` is an integer variable. If the user enters 3.14 in response to the following programming statement, what will be stored in `value`?

```
cin >> value;
```

A) 3.14

B) 3

C) 0

D) Nothing. An error message is displayed.

3.4 A program has the following variable definitions.

```
long miles;
int feet;
float inches;
```

Write one `cin` statement that reads a value into each of these variables.

3.5 The following program will run, but the user will have difficulty understanding what to do. How would you improve the program?

```
// This program multiplies two numbers and displays the result.
#include <iostream>
using namespace std;

int main()
{
   double first, second, product;

   cin >> first >> second;
   product = first * second;
   cout << product;
   return 0;
}
```

3.6 Complete the following program skeleton so it asks for the user's weight (in pounds) and displays the equivalent weight in kilograms.

```
#include <iostream>
using namespace std;

int main()
{
   double pounds, kilograms;

   // Write code here that prompts the user
   // to enter his or her weight and reads
   // the input into the pounds variable.

   // The following line does the conversion.
   kilograms = pounds / 2.2;

   // Write code here that displays the user's weight
   // in kilograms.
   return 0;
}
```