

CHAPTER 4

Character-Based Input and Output

In this chapter you will learn about

- Character-based output using `cout`
- Formatted output
- Character-based input using `cin`

- In Chapters 2 and 3, we introduced three GUI objects—`OKBox`, `IntTypeIn`, and `FloatTypeIn`.
- An alternative user interface of a program uses the predefined `istream` and `ostream` objects for input and output, respectively.
- The predefined `ostream` object `cout` is for displaying the output of a program, and the predefined `istream` object `cin` is for getting input into a program. Both of these objects are already declared in a file that comes with a compiler, and therefore, we do not have to declare them explicitly in our program.
- These `cin` and `cout` objects are for *Character-based user interface* (CUI) in which output and input operations are done with characters only.
- Because the CUI and GUI have completely different setup and programming models, they cannot coexist in the same program.¹ You cannot, for example, mix `cout` and `IntTypeIn` in the same program. Your program's interface supports either CUI (with `cout` and `cin`) or GUI (with our or somebody else's GUI objects), but never both.

1. This restriction does not hold for the UNIX environment. The GUI objects for the UNIX version may be used in conjunction with `cout` and `cin`.

Character-Based Output

- The purpose of an `ostream` object is similar to the purpose of an `OKBox` object. Here is a sample program for displaying a simple message.

```
//Program Hello: A simple program that displays a message  
//          by using the standard ostream object cout.
```

```
#include <iostream.h>  
  
void main()  
{  
    cout << "Hello, how are you?";  
}
```

← The brackets `<` and `>` are used to designate the header files that come with the compiler. All system header files are placed in the same directory.

- Executing the program will result in a window with the message displayed in it. We shall call this window a *cout window*. You terminate the program by closing the `cout` window.



- The standard `ostream` and `istream` objects `cout` and `cin` are both defined in the header file `iostream.h`. Notice that we use brackets `<` and `>` to surround the header file name, instead of the double quotes we have been using so far.

- Instead of

```
#include "iostream.h"
```

we have

```
#include <iostream.h>
```

- If you surround a header file name with double quotes, then the compiler begins the search for the designated header file from the local directory where the file that contains this `include` statement is stored. If you surround a header file with brackets, then the compiler begins the search from the specific directory where all the system header files are stored.
- The *system header files*, such as `iostream.h`, come with the compiler. They are all placed in one directory, and most commonly this directory is named `include`.
- To output a value to `cout`, we use the output operator `<<` (two less-than symbols put together). Whatever data we output to `cout` will get displayed on the `cout` window.
- An `OKBox` is intended solely for displaying a short message. With `cout`, a sequence of messages will stay on the window. (We will see a GUI object that has this capability later.)
- Consider the following program, where we output four strings to `cout` in succession.

```
//Program Hello2: A program that displays messages
//          using cout.

#include <iostream.h>

void main()
{
    cout << "Hello, ";
    cout << "My name is ";
    cout << "Akebono - ";
    cout << "Grand Champion of Sumo Wrestling.";
}
```

- The cout window looks like

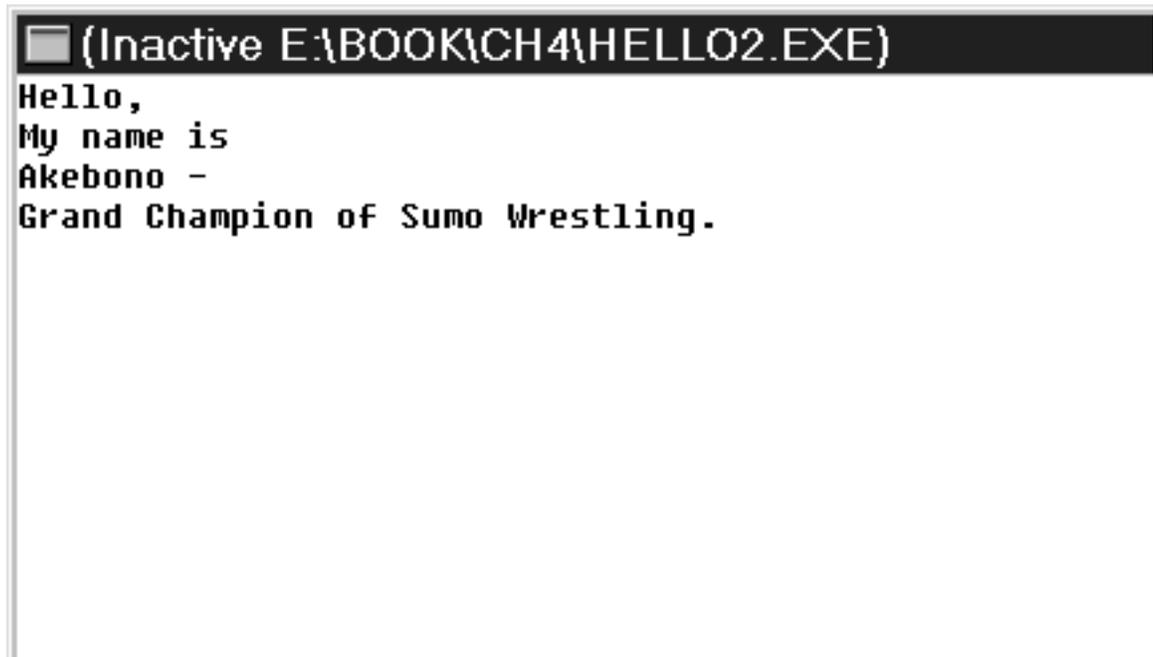


```
(Inactive E:\BOOK\CH4\HELLO2.EXE)
Hello, My name is Akebono - Grand Champion of Sumo Wrestling.
```

- If we want to print the four string values in the program Hello2 on four separate lines, we output the “new line” command `endl` (for *end line*) to `cout` as

```
cout << "Hello, " << endl;  
cout << "My name is " << endl;  
cout << "Akebono - " << endl;  
cout << "Grand Champion of Sumo Wrestling.";
```

which results in



The screenshot shows a Windows command prompt window titled "(Inactive E:\BOOK\CH4\HELLO2.EXE)". The output of the program is displayed as follows:

```
Hello,  
My name is  
Akebono -  
Grand Champion of Sumo Wrestling.
```

- The command `endl` is called a *manipulator* because it manipulates the appearance of output. We will learn more about other manipulators later in the section.
- The output operator accepts any primitive data type, for example:

```
x = 5 + 7;
cout << x;
```

or more concisely

```
cout << 5 + 7;
```

- Instead of using one output operator for each output value, such as

```
x = 145.67;
y = 3343.77
cout << x;
cout << y;
```

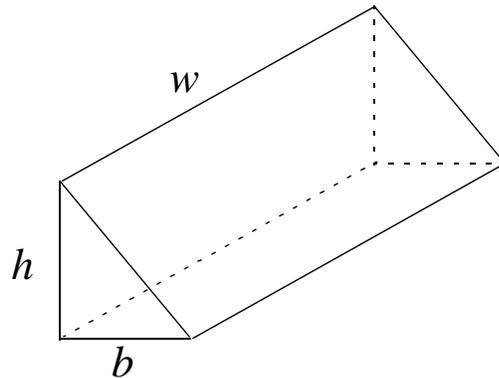
we can cascade the operators as

```
x = 145.67;
y = 3343.77;
cout << x << y;
```

- We can output different types of data. For example, we can say

```
cout << "Value of x is " << x
```

- To illustrate the use of `cout`, let's write a simple program that computes the volume of a triangular prism.



$$volume = \frac{1}{2}bhw$$

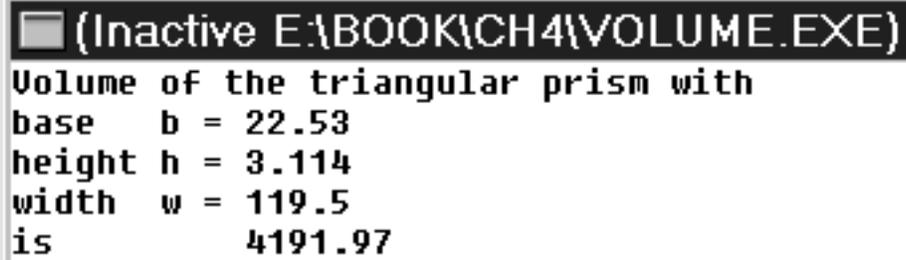
```
// Program Volume:   A program that computes the volume of
//                   a triangular prism using the formula
//                   (1/2) * b * h * w.
#include <iostream.h>
void main()
{
    float b = 22.53;
    float h = 3.114;
    float w = 119.5;
    float volume;

    volume = 0.5 * b * h * w;

    cout << "Volume of the triangular prism with" << endl;
    cout << "base   b = " << b << endl;
    cout << "height h = " << h << endl;
    cout << "width  w = " << w << endl;
    cout << "is           " << volume;
}

```

- The output of the program will be



```
(Inactive E:\BOOK\CH4\VOLUME.EXE)
Volume of the triangular prism with
base b = 22.53
height h = 3.114
width w = 119.5
is 4191.97
```

Formatted Output

- We can use manipulators available in the header file `<iomanip.h>` to format the output so their decimal points get aligned. The following version of program Volume formats the output.

```
// Program Volume2: A program that computes the volume of a
//                 triangular prism using the formula
//                 (1/2) * b * h * w and
//                 aligns the output values.
```

```
#include <iostream.h>
#include <iomanip.h>
```

Manipulators `setprecision`, `setw`, and others are defined in this header file.

```
void main()
{
    float b = 22.53;
    float h = 3.114;
    float w = 119.5;
    float volume;

    volume = 0.5 * b * h * w;

    cout << setiosflags(ios::fixed);
    cout << setprecision(3);

    cout << "Volume of the triangular prism with" << endl;

    cout << "base   b = " << setw(10) << b << endl;
    cout << "height h = " << setw(10) << h << endl;
```

```
cout << "width w = " << setw(10) << w << endl;
cout << "is          " << setw(10) << volume;
}
```

- Now the output of the program will be

```
(Inactive E:\BOOK\CH4\VOLUME.EXE)
Volume of the triangular prism with
base b = 22.53
height h = 3.114
width w = 119.5
is 4191.97
```

- The manipulator `setw` (set width) determines the number of spaces occupied by the value that follows the `setw` manipulator.
- The function `setprecision` determines the number of decimal places for the values that follow the manipulator. In the above program, we are setting three decimal places. Since the whole width is 10, we have enough spaces to represent numbers up to 999999.999. There are six spaces for the whole part, three spaces for the fractional part, and one space for the decimal point.

- Notice that we have to specify `setw(10)` in front of all values because the `setw` command applies only to the value that immediately follows the `setw` command. The `setprecision` and all other manipulators apply to all of the values that come after the manipulators. Thus we have only one `setprecision` specified in the above program.
- The manipulator `setiosflags` (set input and output stream flags) specifies various settings for the output values. There is one flag, or setting, specified in the program—`ios::fixed`.
- The flag `ios::fixed` forces all real numbers to be displayed with the fixed-point format, that is, all the numbers are displayed with the same number of decimal places. Let's discuss this and other flags with examples. Consider the statements

```
float x = 5.0, y = 21.5555;

cout << "[" << x << "]" << endl;
cout << "[" << y << "]" << endl;
```

The output will be

```
[5]
[21.5555]
```

- The decimal point did not appear for the `x` value, since the numbers are displayed in the floating-point format where the number of decimal places is not fixed.
- To display the decimal points, we set the `ios::fixed` flag as

```
cout << setiosflags(ios::fixed);
```

```
cout << "[" << x << "]" << endl;
cout << "[" << y << "]" << endl;
```

and now the output will be

```
[5.000000]
[21.555500]
```

- All real numbers (`float`, `double`, `long double`) are displayed with six decimal places (six is the *default* value—a value preset by the compiler) after the `ios::fixed` flag is set. To change the default six decimal places, we set the precision as

```
cout << setiosflags(ios::fixed);
cout << setprecision(3);
cout << "[" << x << "]" << endl;
cout << "[" << y << "]" << endl;
```

and now the output will be

```
[5.000]
[21.556]
```

- We can align the decimal points by specifying the width using the `setw` manipulator

```
cout << setiosflags(ios::fixed);
cout << setprecision(3);
cout << "[" << setw(8) << x << "]" << endl;
cout << "[" << setw(8) << y << "]" << endl;
```

and then the output will be

```
[ 5.000]
[ 21.556]
```

- The table summarizes the common manipulators defined in the header file `<iomanip.h>`.

Manipulators	Descriptions
<code>setw(N)</code>	Display the next value using <code>N</code> spaces. Default value is normally 1 for most compilers. When the value to be displayed requires more than the designated <code>N</code> spaces, the number will be displayed using the minimum spaces necessary to fully display the value.
<code>setprecision(N)</code>	Display the following values using <code>N</code> decimal places. Default value is normally 6 for most compilers.
<code>setiosflags(FLAGS)</code>	Set the flags in <code>FLAGS</code> , which is a list of flags separated by the vertical bar (<code> </code>). The following are the common flags.

Flags

<code>ios:: fixed</code>	Display the value in fixed-point format.
<code>ios:: scientific</code>	Display the value in scientific notation.
<code>ios:: left</code>	Display the value left-justified in the allocated spaces.
<code>ios:: right</code>	Display the value right-justified in the allocated spaces.

Character-Based Input

- We use the predefined `istream` object `cin` for getting input values into a program. The `cin` object accepts input values from the keyboard.
- Whatever characters pressed by the user on the keyboard they are displayed in the cout window, so the user can see what he has entered. We call this process *echo printing*.
- The `istream` object `cin` accepts any primitive data type. The input operator `>>` (two greater-than symbols put together) is used with the `cin` object to input values.
- Let's modify the program Volume from the previous section to illustrate the use of the `cin` object. Instead of having assigned values for the base, height, and width within the program, this program will read in those values. Here is the program:

```
// Program Volume3: A program that computes the volume of
//           a triangular prism using the formula
//           (1/2) * b * h * w. The values for b, h,
//           and w are read from the cin object.
```

```
#include <iostream.h>
void main()
{
    float b, h, w, volume;

    cin >> b; //read b,
    cin >> h; //      h,
    cin >> w; // and w
```

These can be stated equally as
`cin >> b >> h >> w;`

```

volume = 0.5 * b * h * w;

cout << "Volume of triangular prism with" << endl;
cout << "base   b = " << b << endl;
cout << "height h = " << h << endl;
cout << "width  w = " << w << endl;
cout << "is           " << volume;
}

```

- The following window is the result of executing the program with the input 12.34, 9.87, and 45.6.

```

(Inactive E:\BOOK\CH4\VOLUME3.EXE)
12.34 9.87 45.6
Volume of triangular prism with
base   b = 12.34
height h = 9.87
width  w = 45.6
is           2776.94

```

- Notice that the three inputs are separated by a single space and the <Return> key is pressed after the last number. The separator doesn't have to be a single space. You can put as many spaces as you wish between the numbers. Instead of spaces between the values, it is also acceptable to press the <Return> key (you may press more than once). The space and the <Return> key are

called *separators* because they are used as markers to separate input values.

- If you press the <Return> key once after each number, then the window content will look like

```
(Inactive E:\BOOK\CH4\VOLUME3.EXE)
12.34
9.87
45.6
Volume of triangular prism with
base b = 12.34
height h = 9.87
width w = 45.6
is 2776.94
```

- We recommend prompting the user with an appropriate message so that he or she knows which value to enter. We can modify the input portion of the above program with such prompts.

```
cout << "Enter the value for the base: ";
cin >> b;
```

```
cout << "Enter the value for the height: ";
cin >> h;
```

```
cout << "Enter the value for the width: ";
cin >> w;
```

```
cout << endl << endl;
```

- We terminate the prompts with colons so the user can tell that the program is expecting a data value to be entered. Executing the modified program (and assuming that the <Return> key is pressed after entering each of the three inputs) will result in

```
(Inactive E:\BOOK\CH4\VOLUME3A.EXE)
Enter the value for the base: 12.34
Enter the value for the height: 9.87
Enter the value for the width: 45.6
Volume of triangular prism with
base    b = 12.34
height  h = 9.87
width   w = 45.6
is      2776.94
```

- We can cascade input operators for different data types in a single input statement. For example, the following code reads two integers and two real numbers.

```
int i, j;
float x, y;
cin >> i >> x >> j >> y;
```

We must enter values in the sequence of `int`, `float`, `int`, and `float`. Since we can assign an integer to a float variable, entering four integer values would also be acceptable.

Sample Program

- The following sample program computes the sum and average of given real numbers. Since we have not yet mastered the technique necessary to write a program that will compute the sum and average of N ($N \geq 1$) real numbers, we will restrict this program to five real numbers.
- The overall flow of the program for the restricted version is as follows:

1. Display the opening message.
2. Ask the user to enter five real numbers v , w , x , y , and z .
3. Compute the sum.

$$sum = v + w + x + y + z$$

4. Compute the average.

$$avg = \frac{sum}{5}$$

5. Print out the results.

- Here is the complete program.

```
//Program Statistic: A program that computes the sum and
//                          average of five real numbers.
```

```
#include <iostream.h>

void main()
{
    float v, w, x, y, z;
    float avg, sum;

    //opening message
    cout << "This program will compute the sum and average"
         << endl;
    cout << "of five real numbers you enter." << endl;
    cout << endl << endl;

    cout << "Now, please enter five numbers" << endl;
    cout << "Press the <Return> key after " << endl;
    cout << "you entered five numbers" << endl;

    cin >> v >> w >> x >> y >> z;

    //compute the sum
    sum = v + w + x + y + z;

    //compute the average
    avg = sum / 5;

    //print out the results
    cout << endl << endl;
    cout << "The sum is      " << sum << endl;
    cout << "The average is " << avg;
}
```