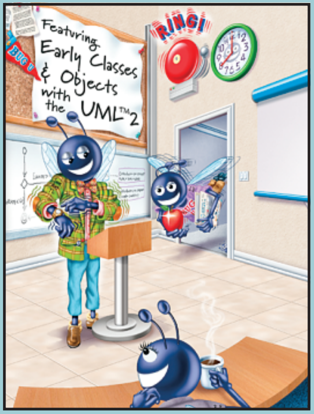
15



Stream Input/ Output

OBJECTIVES

In this chapter you will learn:

- To use C++ object-oriented stream input/output.
- To format input and output.
- The stream-I/O class hierarchy.
- To use stream manipulators.
- To control justification and padding.
- To determine the success or failure of input/output operations.
- To tie output streams to input streams.

Assignment Checklist

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities		
Matching	YES NO	
Fill in the Blank	10, 11, 12, 13, 14, 15, 16 ,17, 18	
Short Answer	19, 20	
Programming Output	21, 22, 23	
Correct the Code	24, 25	
Lab Exercises		
Lab Exercise 1 — ASCII Character Table	YES NO	
Lab Exercise 2 — `Complex` Input	YES NO	
Debugging	YES NO	
Labs Provided by Instructor		
1.		
2.		
3.		
Postlab Activities		
Coding Exercises	1, 2, 3	
Programming Challenges	1, 2	

Prelab Activities

Matching

Name: ____________________ **Date:** ____________________

Section: ____________________

After reading Chapter 15 of *C++ How to Program: Fifth Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

	Term		Description
___	1. Unformatted I/O	a)	`<<`.
___	2. Formatted I/O	b)	`endl`.
___	3. Parameterized stream manipulator	c)	Chaining together a series of outputs, such as `cout << a << b << "hi" << endl;`.
___	4. Stream-insertion operator	d)	Performed with the `read` and `write` member functions.
___	5. Stream-extraction operator	e)	Used to specify the kinds of formatting to be performed during I/O operations.
___	6. Stream	f)	`>>`.
___	7. Cascaded form	g)	Stream manipulator that takes an argument.
___	8. Format flags	h)	Requires capabilities declared in the header file `<iomanip>`.
___	9. Stream manipulator	i)	Sequence of bytes.

Prelab Activities

Name:

Fill in the Blank

Name: ____________________ **Date:** ____________________

Section: ____________________

Fill in the blank for each of the following statements:

10. In ____________ operations, bytes flow from a device (e.g., a keyboard, a disk drive, a network connection) to main memory.

11. In ____________ operations, bytes flow from main memory to a device (e.g., a display screen, a printer, a disk drive, a network connection).

12. The ____________ header declares services that are important file-processing operations.

13. Stream extraction causes the stream's ____________ to be set if data of the wrong type is input and causes the stream's ____________ to be set if the operation fails.

14. ____________ provide capabilities such as setting field widths, setting precision, setting and unsetting format flags, setting the fill character in fields, flushing streams, inserting a newline in the output stream and flushing the stream, inserting a null character in the output stream and skipping whitespace in the input stream.

15. The >> operator returns ____________ after end-of-file is encountered when reading from a stream.

16. The ____________ member function returns the error state of the stream.

17. C++ provides the ____________ member function to synchronize `istream` and `ostream` operations to ensure that outputs appear before subsequent inputs.

18. Member function ____________ restores a stream's state to "good," so that I/O may proceed on that stream.

Prelab Activities

Name:

Short Answer

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

In the space provided, answer each of the given questions. Your answers should be as concise as possible; aim for two or three sentences.

19. What is the difference between “low-level” and “high-level” I/O capabilities?

20. Explain the concept of type-safe I/O.

Prelab Activities

Name:

Programming Output

Name: ______________________ **Date:** ______________________

Section: ______________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

21. What is the output of the given program? Assume that the user enters the sentence "`This is my input sentence.`" when prompted for a sentence.

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int main()
{
   const int SIZE = 80;
   char buffer[ SIZE ];

   cout << "Enter a sentence: \n";
   cin.read( buffer, 15 );
   cout << "\nThe sentence entered was:\n";
   cout.write( buffer, cin.gcount() );
   cout << endl;
   return 0;
} // end main
```

Your answer:

22. What is the output of the following program?

```
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::hex;
using std::dec;
using std::oct;
using std::setbase;

```

Prelab Activities

Name:

Programming Output

```
int main()
{
   int n = 17;

   cout << n << " in hexadecimal is: "
        << hex << n << endl
        << dec << n << " in octal is: "
        << oct << n << endl
        << setbase( 10 ) << n << " in decimal is: "
        << n << endl;
   return 0;
} // end main
```

Your answer:

23. What is the output of the following program?

```
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::ios;
using std::setprecision;
using std::fixed;
using std::setw;
using std::setfill;
using std::showpos;

int main()
{
   double n = 83;

   cout << fixed << showpos
        << setw( 10 ) << setprecision( 3 ) << setfill( '-' )
        << n << endl;
   return 0;
}
```

Your answer:

Prelab Activities

Name:

Correct the Code

Name: ______________________ **Date:** ______________________

Section: ______________________

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic, syntax or compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write "no error." [*Note:* It is possible that a program segment may contain multiple errors.]

24. The following program should print a table of numbers:

```
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

int main()
{
   int n[ 3 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

   // display each number in a field width of 10
   cout << setw( 10 );

   for ( int i = 0; i < 3; i++ )
   {
      cout << endl;

      for ( int j = 0; j < 3; j++ )
         cout << n[i][j];
   } // end for

   return 0;
} // end main
```

Your Answer:

Prelab Activities

Name:

Correct the Code

25. This program should output the following results:

```
32 in hexadecimal is: 20
32 in octal is: 40
32 in decimal is: 32
```

```
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::hex;
using std::dec;
using std::oct;

int main()
{
   double n = 32;

   cout << n << " in hexadecimal is: "
        << hex << n << endl
        << n << " in octal is: "
        << oct << n << endl
        << n << " in decimal is: "
        << dec << n << endl;
   return 0;
} // end main
```

Your Answer:

Lab Exercises

Lab Exercise 1 — ASCII Character Table

Name: ____________________ **Date:** ____________________

Section: ____________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into five parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 15.1)
5. Problem-Solving Tip

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tip as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 15 of *C++ How To Program: Fifth Edition*. In this lab, you will practice

- Varying stream format states
- Displaying integers as characters

Problem Description

Write a program that uses a `for` statement to print a table of ASCII values for the characters in the ASCII character set from `33` to `126`. The program should print the decimal value, octal value, hexadecimal value and character value for each character. Use the stream manipulators `dec`, `oct` and `hex` to print the integer values.

Sample Output

```
Decimal   Octal    Hexadecimal     Character
     33     041           0x21             !
     34     042           0x22             "
     35     043           0x23             #
     36     044           0x24             $
     37     045           0x25             %
     38     046           0x26             &
     39     047           0x27             '
     40     050           0x28             (
...
    118    0166           0x76             v
    119    0167           0x77             w
    120    0170           0x78             x
    121    0171           0x79             y
    122    0172           0x7a             z
    123    0173           0x7b             {
    124    0174           0x7c             |
    125    0175           0x7d             }
    126    0176           0x7e             ~
```

Lab Exercises

Name:

Lab Exercise 1 — ASCII Character Table

Template

```cpp
// Lab 1: ASCII.cpp
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::dec;
using std::hex;
using std::oct;
using std::setw;
using std::showbase;

int main()
{
   // display column headings and set field lengths
   cout << setw( 7 ) << "Decimal" << setw( 9 ) << "Octal " << setw( 15 )
      << "Hexadecimal " << setw( 13 ) << "Character" << showbase << '\n';

   // loop through ASCII values 33-126 and display corresponding
   // integer, octal and hexadecimal values
   /* Write a for header that will iterate from 33 through 126
   /* Write a cout statement to output the current ASCII value in
      decimal, octal, hexadecimal and character formats; follow the
      spacing convention established above */

   return 0;
} // end main
```

Fig. L 15.1 | ASCII.cpp.

Problem-Solving Tip

1. Use an `int` variable as the counter in your `for` loop. To display a character value cast the `int` to a `char` by using the `static_cast` operator.

Lab Exercises

Name:

Lab Exercise 2 — `Complex` Input

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into five parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 15.2–Fig. L 15.4)
5. Problem-Solving Tips

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 15 of *C++ How To Program: Fifth Edition*. In this lab, you will practice

- Checking streams for error states.
- Reading and validating input for a user-defined type.

Problem Description

Write a program that accomplishes each of the following:

a) Create a user-defined class `Complex` that contains the private integer data members `real` and `imaginary` and declares stream insertion and stream extraction overloaded operator functions as `friend`s of the class.

b) Define the stream insertion and stream extraction operator functions. The stream extraction operator function should determine whether the data entered is valid, and, if not, it should set `failbit` to indicate improper input. The input should be of the form

```
3 + 8i
```

c) The values can be negative or positive, and it is possible that one of the two values is not provided. If a value is not provided, the appropriate data member should be set to 0. The stream-insertion operator should not be able to display the complex number if an input error occurred. For negative imaginary values, a minus sign should be printed rather than a plus sign.

d) Write a `main` function that tests input and output of user-defined class `Complex`, using the overloaded stream extraction and stream insertion operators.

Lab Exercises

Name:

Lab Exercise 2 — `Complex` Input

Sample Output

```
Input a complex number in the form A + Bi:
7 - 777i
Complex number entered was:
7-777i
```

Template

```
// Lab 2: Complex.h
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>
using std::ostream;
using std::istream;

class Complex
{
   // overloaded input and output operators
   /* Write friend declarations for the stream insertion
      and extraction operators */

public:
   Complex( void ); // constructor
private:
   /* Write declarations for data members real and imaginary */
}; // end class Complex

#endif
```

Fig. L 15.2 | `Complex.h`.

```
// Lab 2: Complex.cpp
// Member-function definition of class Complex.
#include <iostream>
using std::ios;
using std::istream;
using std::ostream;

#include <iomanip>
using std::showpos;

#include "Complex.h"

// default constructor
Complex::Complex( void ):
   real( 0 ),
   imaginary( 0 )
{
   // empty body
} // end Complex constructor
```

Fig. L 15.3 | `Complex.cpp`. (Part 1 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — `Complex` Input

```cpp

// overloaded output (<<) operator
ostream &operator<<( ostream &output, const Complex &c )
{
   output << c.real << showpos << c.imaginary << "i\n" << showpos;
   return output; // return ostream reference
} // end overloaded output (<<) operator

// overloaded input (>>) operator
istream &operator>>( istream &input, Complex &c )
{
   int number;
   int multiplier;
   char temp; // temporary variable used to store input

   input >> number; // get input

   // test if character is a space
   if ( /* Write a call to the peek member function to
           test if the next character is a space ' ' */ ) // case a + bi
   {
      c.real = number;
      input >> temp;

      multiplier = ( temp == '+' ) ? 1 : -1;

      // set failbit if character not a space
      if ( input.peek() != ' ' )
         /* Write a call to the clear member function with
            ios::failbit as the argument to set input's fail bit */
      else
      {
         // set imaginary part if data is valid
         if ( input.peek() == ' ' )
         {
            input >> c.imaginary;
            c.imaginary *= multiplier;
            input >> temp;

            if ( /* Write a call to member function peek to test if the next
                    character is a newline \n */ ) // character not a newline
               input.clear( ios::failbit ); // set bad bit
         } // end if
         else
            input.clear( ios::failbit ); // set bad bit
      } // end else
   } // end if
   else if ( /* Write a call to member function peek to test if
                the next character is 'i' */ ) // test for i of imaginary number
   {
      input >> temp;

      // test for newline character entered
      if ( input.peek() == '\n' )
      {
         c.real = 0;
```

Fig. L 15.3 | `Complex.cpp`. (Part 2 of 3.)

Lab Exercises

Name:

Lab Exercise 2 — Complex Input

```
            c.imaginary = number;
         } // end if
         else
            input.clear( ios::failbit ); // set bad bit
      } // end else if
      else if ( input.peek() == '\n' ) // set real number if it is valid
      {
         c.real = number;
         c.imaginary = 0;
      } // end else if
      else
         input.clear( ios::failbit ); // set bad bit

      return input;
   } // end overloaded input (>>) operator
```

Fig. L 15.3 | Complex.cpp. (Part 3 of 3.)

```
// Lab 2: ComplexInput.cpp
// Complex test program.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include "Complex.h"

int main()
{
   Complex complex; // create Complex object

   // ask user to enter complex number
   cout << "Input a complex number in the form A + Bi:\n";
   cin >> complex; // store complex number

   if ( /* Write a call to member funciton fail to determine if the
           stream operation failed, then negate it to test if input
           was valid */ ) // display complex number entered by user if valid
      cout << "Complex number entered was:\n" << complex << endl;
   else
      cout << "Invalid Data Entered\n";

   return 0;
} // end main
```

Fig. L 15.4 | ComplexInput.cpp.

Problem-Solving Tips

1. Use the peek member function of istream to check the next character in the stream before taking it out of the stream.
2. Recall that the clear member function of istream can be used to set error bits as well as clear them.

Lab Exercises

Name:

Debugging

Name: ____________________________ Date: ____________________________

Section: ____________________________

The program in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code is corrected.

Sample Output

[*Note:* Be careful when comparing your output with the one shown here. Confirm that your output is formatted identically.]

```
Enter a number: 2.3456
Enter a number: 0.895
The value of x is: +2.346000000
The value of y is:    8.95e-001
```

Broken Code

```
// Debugging: debugging.cpp
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>
using std::setw;
using std::setprecision;
using std::fixed;
using std::left;
using std::right;
using std::scientific;

double readNumber();
void printFormatted( double, double );

int main()
{
   double x, y;

   x = readNumber();
   y = readNumber();
   printFormatted( x, y );
   return 0;
} // end main

```

Fig. L 15.5 | `debugging.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Debugging

```
// function readNumber definition
double readNumber()
{
   double number = 0;
   double place = 10;

   cout << "Enter a number: ";
   number = cin.getline() - '0';

   while ( cin.peek() != '.' && cin.peek() != '\n' )
      number *= 10 + atof( cin.get() );

   while ( cin.peek() != '.' )
   {
      number += static_cast< double >( cin.get() ) / place;
      place *= 10;
   } // end while

   cin.ignore();

   return number;
} // end function getNumber

// function printFormatted definition
void printFormatted( double x, double y )
{
   char buffer[] = "The value of x is: ";

   for ( int i = 0; buffer[ i ] != '\n'; i++ )
      cout.put( buffer[ i ] );

   cout << setw( 12 ) << setprecision( 3 ) << setfill( '0' )
        << ios::fixed
        << left << x << endl;

   cout.write( "The value of y is: " );

   cout << setprecision( 2 )
        << ios::scientific << ios::right
        << y << endl;
} // end function printFormatted
```

Fig. L 15.5 | `debugging.cpp`. (Part 2 of 2.)

Postlab Activities

Coding Exercises

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For each of the following problems, write a program or a program segment that performs the specified action.

1. Write a program to test the inputting of integer values in decimal, octal and hexadecimal format. Output each integer read by the program in all three formats. Test the program with the following input data: 10, 010, 0x10.

Postlab Activities

Name:

Coding Exercises

2. Write a program that prints the value `100.453627` rounded to the nearest digit, tenth, hundredth, thousandth and ten thousandth.

3. Write a program that converts integer Fahrenheit temperatures from `0` to `212` degrees to floating-point Celsius temperatures with `3` digits of precision. Use the formula

   ```
   celsius = 5.0 / 9.0 * ( fahrenheit - 32 );
   ```

   to perform the calculation. The output should be printed in two right-justified columns and the Celsius temperature should be preceded by a sign for both positive and negative

Postlab Activities

Name:

Programming Challenges

Name: ____________________ **Date:** ____________________

Section: ____________________

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a C++ program for each of the problems in this section. The answers to these problems are available at `www.deitel.com` and `www.prenhall.com/deitel`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. Write a program to test the results of printing the integer value `12345` and the floating-point value `1.2345` in various-sized fields. What happens when the values are printed in fields containing fewer digits than the values?

Hints:

- Use field sizes ranging from `0` to `10`.
- Let the output values be right-justified within the fields, which is the default setting.

2. Write a program to show that the `getline` and three-argument `get` `istream` member functions both end the input string with a string-terminating null character. Also, show that `get` leaves the delimiter character on the input stream, whereas `getline` extracts the delimiter character and discards it. What happens to the unread characters in the stream?

Hints:

- After a `get` or `getline` call places the input into a `char` array, output that `char` array using the stream insertion operator to demonstrate that the string-terminating null character is already in the array at the end of the input.
- Specify that `*` be used as the delimiter character so that the delimiter character will be visible and identifiable in the output.
- Follow each `get` or `getline` call with a stream extraction operation on `cin` to demonstrate whether the preceding `istream` member function left the delimiter character in the input stream or removed it from the input stream.