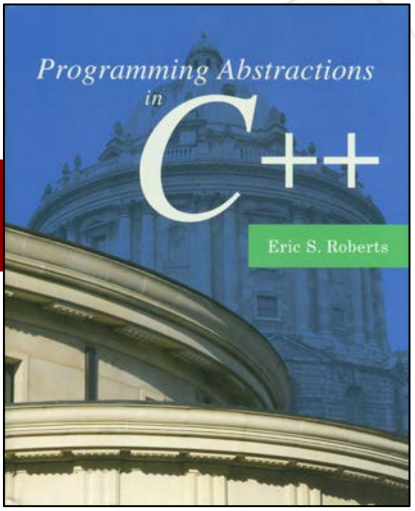
CHAPTER 4

Streams

We will not be satisfied until justice rolls down like waters and righteousness like a mighty stream.

—The Reverend Martin Luther King, Jr.,
I Have a Dream, August 28, 1963
(paraphrasing Amos 5:24)



Formatted Output: Insertion

- The standard technique to specify formatted output in C++ uses the *insertion operator*, which is written as <<. This operator takes an output stream on the left and an expression of any type on its right. The effect is to write the value of the expression to the output stream using the current format settings.
- The insertion operator returns the output stream as its value. The advantage of this interpretation is that output operations can be chained together, as in the following statement:

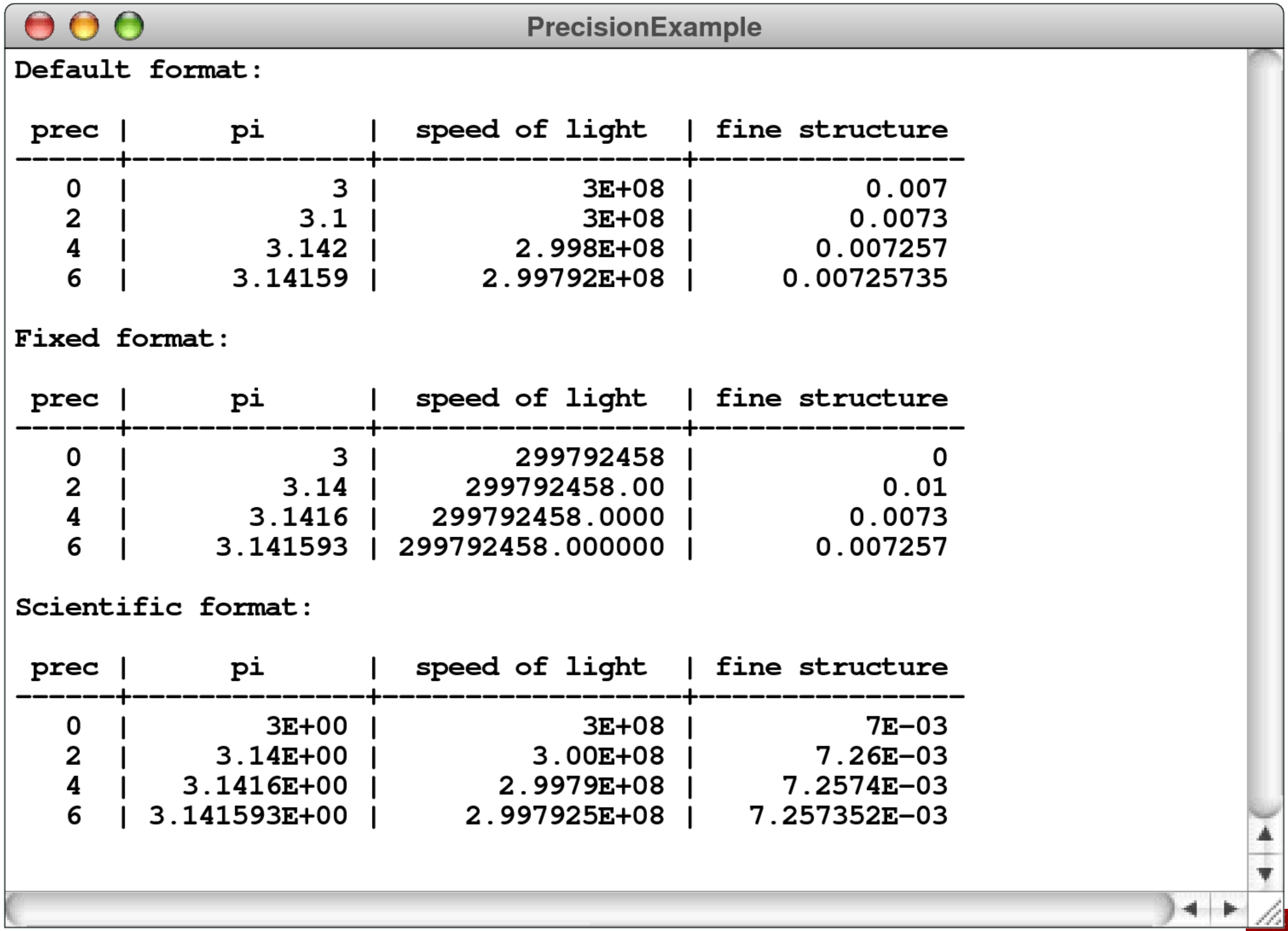
```
cout << "The total is " << total << endl;
```

- C++ allows you to control the output by including items in the output chain called *manipulators*, which affect the way in subsequent values are formatted. A list of the most common output manipulators appears on the next slide.

Output Manipulators

`endl`	Moves cursor to the next line.
`setw(`*n*`)`	Sets the width of the next value to *n* characters.
`setprecision(`*digits*`)`	Sets how many digits should appear.
`setfill(`*ch*`)`	Sets the fill character used to pad values.
`left`	Aligns the value at the left edge of the field.
`right`	Aligns the value at the right edge of the field.
`fixed`	Sets fixed-point output (no scientific notation).
`scientific`	Sets scientific-notation output.
`showpoint/noshowpoint`	Controls whether a decimal point must appear.
`showpos/noshowpos`	Controls appearance of a plus sign.
`uppercase/nouppercase`	Controls whether uppercase is used in hex.
`boolalpha/noboolalpha`	Controls whether `bool`s appear as `true/false`.

Precision Example



```
Default format:

 prec |      pi       |  speed of light  | fine structure
------+---------------+------------------+---------------
   0  |             3 |            3E+08 |          0.007
   2  |           3.1 |            3E+08 |         0.0073
   4  |         3.142 |        2.998E+08 |       0.007257
   6  |       3.14159 |      2.99792E+08 |     0.00725735

Fixed format:

 prec |      pi       |  speed of light  | fine structure
------+---------------+------------------+---------------
   0  |             3 |        299792458 |              0
   2  |          3.14 |     299792458.00 |           0.01
   4  |        3.1416 |   299792458.0000 |         0.0073
   6  |      3.141593 | 299792458.000000 |       0.007257

Scientific format:

 prec |      pi       |  speed of light  | fine structure
------+---------------+------------------+---------------
   0  |         3E+00 |            3E+08 |          7E-03
   2  |       3.14E+00 |        3.00E+08 |       7.26E-03
   4  |     3.1416E+00 |      2.9979E+08 |     7.2574E-03
   6  | 3.141593E+00 |     2.997925E+08 |   7.257352E-03
```

Formatted Input

- For input, C++ includes the >> operator, which is called the *extraction operator.* The >> operator is symmetrical to the << operator and reads formatted data from the stream on the left into the variables that appear on the right.
- Up to now, you might not have any idea how C++ represents streams inside the computer or how you might manipulate the characters that make up a stream. At the same time, the fact that you don't know those things has not compromised your ability to use streams effectively because you have been able to think of streams holistically as if they were a primitive type.

Data Files

- A *file* is the generic name for any named collection of data maintained on the various types of permanent storage media attached to a computer. In most cases, a file is stored on a hard disk, but it can also be stored on removable medium, such as a CD or flash memory drive.
- Files can contain information of many different types. When you compile a C++ program, for example, the compiler stores its output in an *object file* containing the binary representation of the program. The most common type of file, however, is a *text file*, which contains character data of the sort you find in a string.

Text Files *vs.* Strings

touching lives

Although text files and strings both contain character data, it is important to keep in mind the following important differences between text files and strings:

1. *The information stored in a file is permanent.* The value of a string variable persists only as long as the variable does. Local variables disappear when the method returns, and instance variables disappear when the object goes away, which typically does not occur until the program exits. Information stored in a file exists until the file is deleted.

2. *Files are usually read sequentially.* When you read data from a file, you usually start at the beginning and read the characters in order, either individually or in groups that are most commonly individual lines. Once you have read one set of characters, you then move on to the next set of characters until you reach the end of the file.

THE UNIVERSITY OF ALABAMA
DEPARTMENT OF COMPUTER SCIENCE

Using Text Files

- When you want to read data from a text file as part of a C++ program, you need to take the following steps:

    1. Construct a new `ifstream` object that is tied to the data in the file by calling the `open` method for the stream. This phase of the process is called *opening the file*. Note that the argument to `open` is a C string rather than a C++ string.

    2. Call the methods provided by the `ifstream` class to read data from the file in sequential order. The text of the file can be read in several ways, including character by character or line by line.

    3. Break the association between the reader and the file by calling the stream's `close` method, which is called *closing the file*.

Opening an Input File

```
/*
 * File: filelib.h
 * ---------------
 * This file exports a standardized set of tools for working with
 * files . . .
 */

#ifndef _filelib_h
#define _filelib_h

/*
 * Function: promptUserForFile
 * Usage: string filename = promptUserForFile(stream, prompt);
 * -----------------------------------------------------------
 * Asks the user for the name of a file.  The file is opened
 * using the reference parameter stream, and the function
 * returns the name of the file.  If the requested file cannot
 * be opened, the user is given additional chances to enter a
 * valid file.  The optional prompt argument provides an input
 * prompt for the user.
 */
```

Opening an Input File

```
string promptUserForFile(ifstream & stream, string prompt) {
   while (true) {
      cout << prompt;
      string filename;
      getline(cin, filename);
      openFile(stream, filename);
      if (!stream.fail()) return filename;
      stream.clear();
      cout << "Unable to open that file.  Try again." << endl;
      if (prompt == "") prompt = "Input file: ";
   }
}
```

The library "filelib.h" and other Stanford C++ libraries used throughout this textbook are freely available as open-source at

http://cs.stanford.edu/people/eroberts/StanfordCPPLib/

Reading Characters

- You can read characters from an input stream by calling the `get` method, which comes in two forms:
  - If you supply no arguments, `get()` reads and returns the next character value as an `int`, which is `EOF` at the end of the file.
  - If you instead pass a character variable by reference, `get(ch)` reads the next character into that variable. This form of `get` returns a value that acts like `false` at the end of the file.
- The second form is less conventional but typically more convenient. The general pattern for reading a stream called `infile` character by character looks like this:

```
char ch;
while (infile.get(ch)) {
    . . . Perform the necessary operations using the character . . .
}
```

Reading a File Character by Character

```
/*
 * File: ShowFileContents.cpp
 * --------------------------
 * This program displays the contents of a file chosen by the user.
 */

#include <iostream>
#include <fstream>
#include <string>
#include "filelib.h"
using namespace std;

int main() {
   ifstream infile;
   promptUserForFile(infile, "Input file: ");
   char ch;
   while (infile.get(ch)) {
      cout.put(ch);
   }
   infile.close();
   return 0;
}
```

Reading Lines from a File

- You can also read lines from a text file by calling the free function `getline`, which takes an `ifstream` and a `string` as reference parameters. The effect of `getline` is to store the next line of data from the file into the string variable after discarding the end-of-line character.
- If you try to read past the end of the data, `getline` sets the "fail" indicator, which means that it is interpreted as `false`.
- The following code fragment uses the `getline` method to determine the length of the longest line in the stream `infile`:

```
int max = 0;
string line;
while (getline(infile, line)) {
   if (line.length() > max) max = line.length();
}
```

Reading a File Line by Line

```
/*
 * File: ShowFileContents.cpp
 * --------------------------
 * This program displays the contents of a file chosen by the user.
 */

#include <iostream>
#include <fstream>
#include <string>
#include "filelib.h"
using namespace std;

int main() {
   ifstream infile;
   promptUserForFile(infile, "Input file: ");
   string line;
   while (getline(infile, line)) {
      cout << line << endl;
   }
   infile.close();
   return 0;
}
```

The End