Chapter 8:
I/O Streams and Data Files

C++ FOR ENGINEERS
AND SCIENTISTS

Objectives

- In this chapter, you will learn about:
  - I/O file stream objects and functions
  - Reading and writing character-based files
  - Random file access
  - File streams as function arguments
  - A case study involving pollen count file updates
  - The `iostream` class library
  - Common programming errors

I/O File Stream Objects and Functions

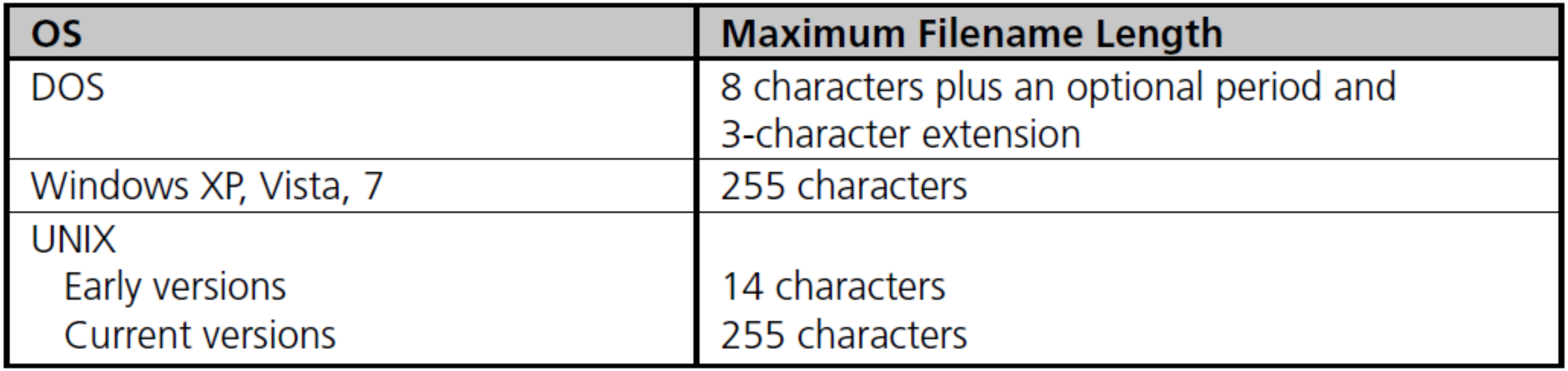
- To store and retrieve data outside a C++ program, two items are needed:
  - A file
  - A file stream object
- A file is a collection of data stored together under a common name, usually on disk, magnetic tape, USB drive, or CD
- Each file has a unique file name, referred to as file's **external name**

I/O File Stream Objects and Functions (continued)

- Choose filenames that indicate the type of data in the file
- Two basic types of files exist
  - **Text files** (also known as **character-based** files)
  - **Binary files**

Refer to page 463 for more explanations and examples

I/O File Stream Objects and Functions (continued)



OS	Maximum Filename Length
DOS	8 characters plus an optional period and 3-character extension
Windows XP, Vista, 7	255 characters
UNIX	
Early versions	14 characters
Current versions	255 characters

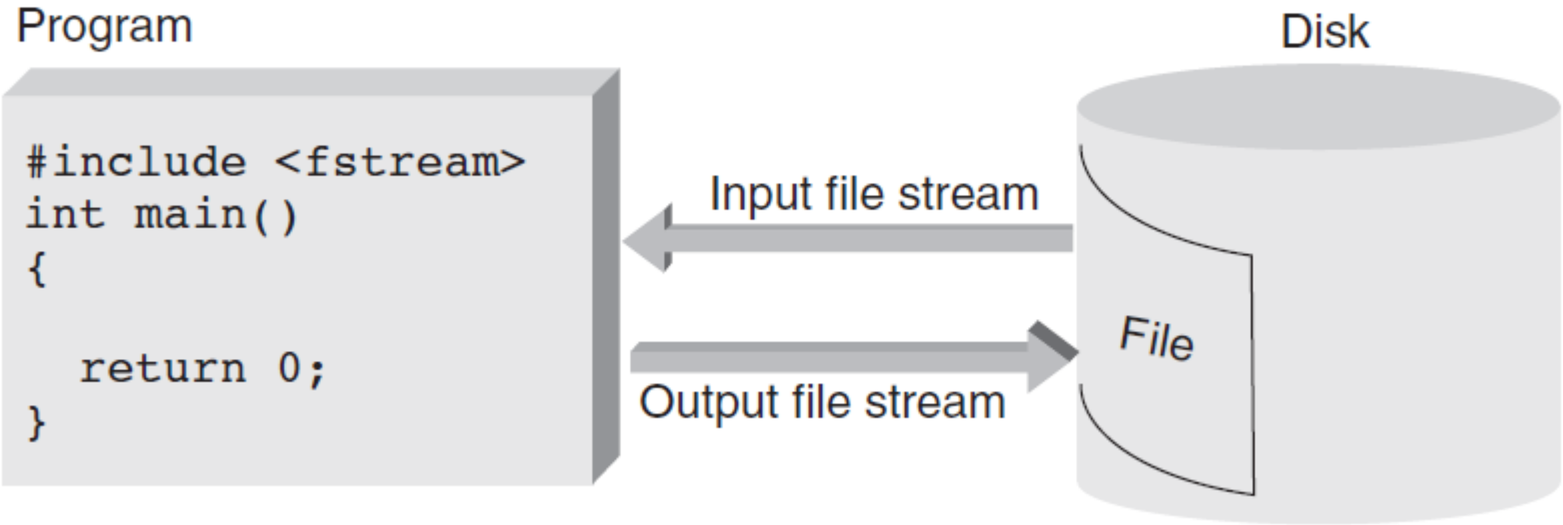
Table 8.1 Maximum Allowable Filename Characters

File Stream Objects

- **File stream:** A one-way transmission path used to connect a file stored on a physical device, such as a disk or CD, to a program
- Each file stream has its own mode that determines direction of data on transmission path
  - That is, whether path moves data from a file to a program or from a program to a file
- **Input file stream:** File stream that receives or reads data from a file to a program
- **Output file stream:** File stream that sends or writes data to a file

File Stream Objects (continued)

- For each file your program uses, regardless of file’s type, a distinct file stream object must be created



```
#include <fstream>
int main()
{

  return 0;
}
```

Figure 8.1 Input and output file streams

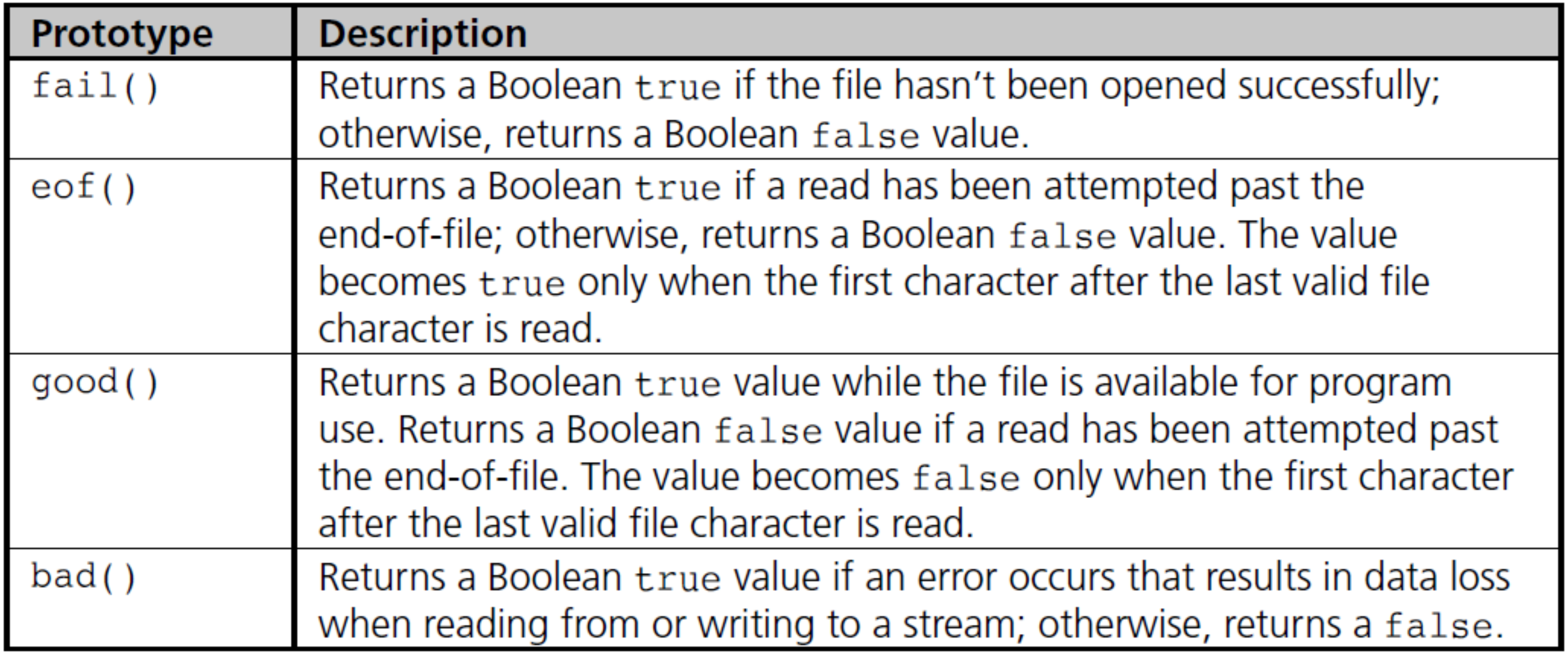
File Stream Functions

- Each file stream object has access to functions defined for its class
- Methods perform following functions:
  - Connecting stream object name to external filename: **opening a file**
  - Determining whether successful connection has been made
  - Closing connection: **closing a file**
  - Getting next data item into program from input stream
  - Putting new data item from program onto output stream

File Stream Functions (continued)

- When existing file is connected to input stream, file’s data is made available for input, starting with first data item in file
  - Called **read mode** or **input mode**
- File connected to output stream creates new file and makes file available for output
  - Called **output mode**
- When opening file for input or output, check that connection has been established before attempting to use file

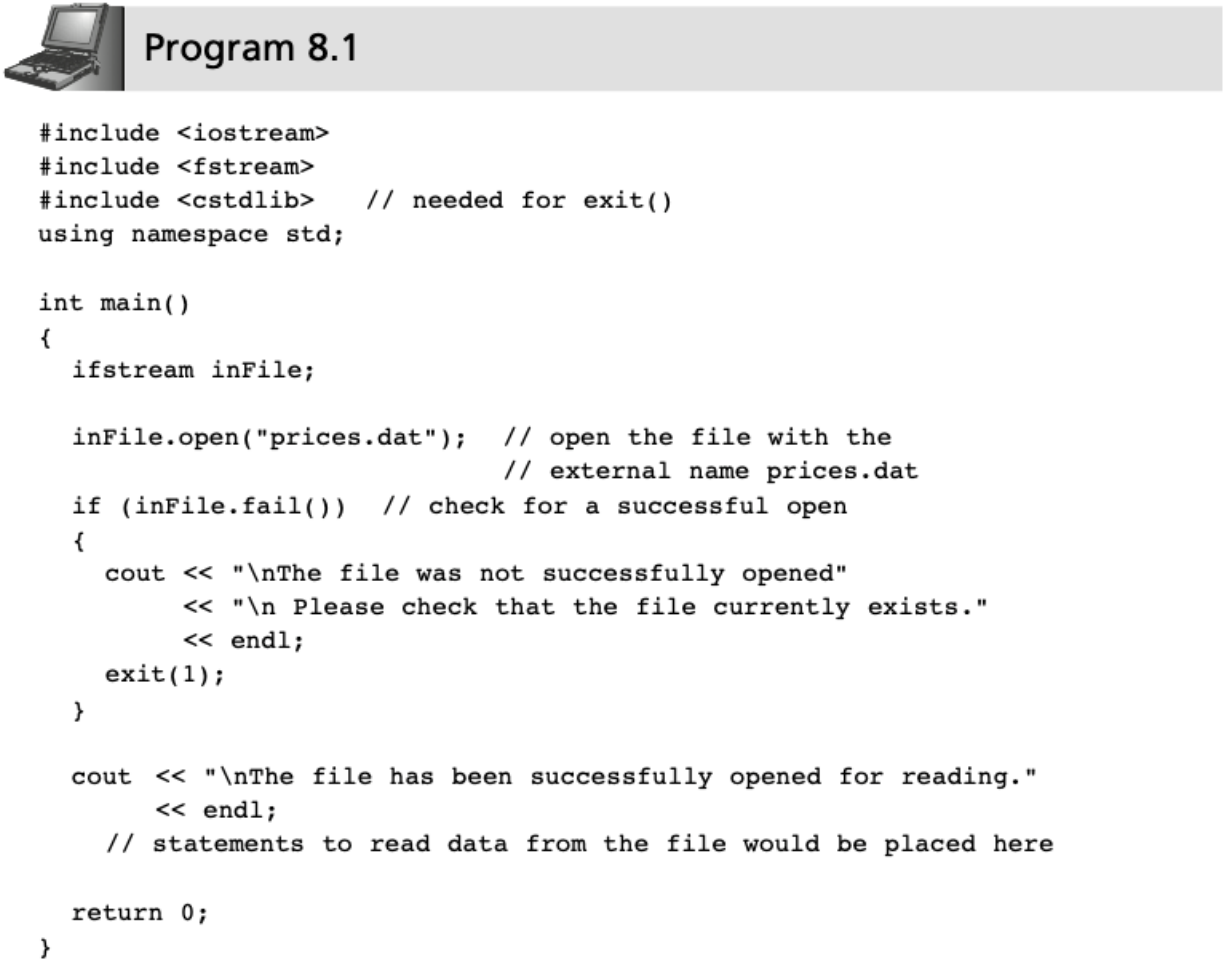
File Stream Functions (continued)



Prototype	Description
`fail()`	Returns a Boolean `true` if the file hasn't been opened successfully; otherwise, returns a Boolean `false` value.
`eof()`	Returns a Boolean `true` if a read has been attempted past the end-of-file; otherwise, returns a Boolean `false` value. The value becomes `true` only when the first character after the last valid file character is read.
`good()`	Returns a Boolean `true` value while the file is available for program use. Returns a Boolean `false` value if a read has been attempted past the end-of-file. The value becomes `false` only when the first character after the last valid file character is read.
`bad()`	Returns a Boolean `true` value if an error occurs that results in data loss when reading from or writing to a stream; otherwise, returns a `false`.

Table 8.2 File status methods

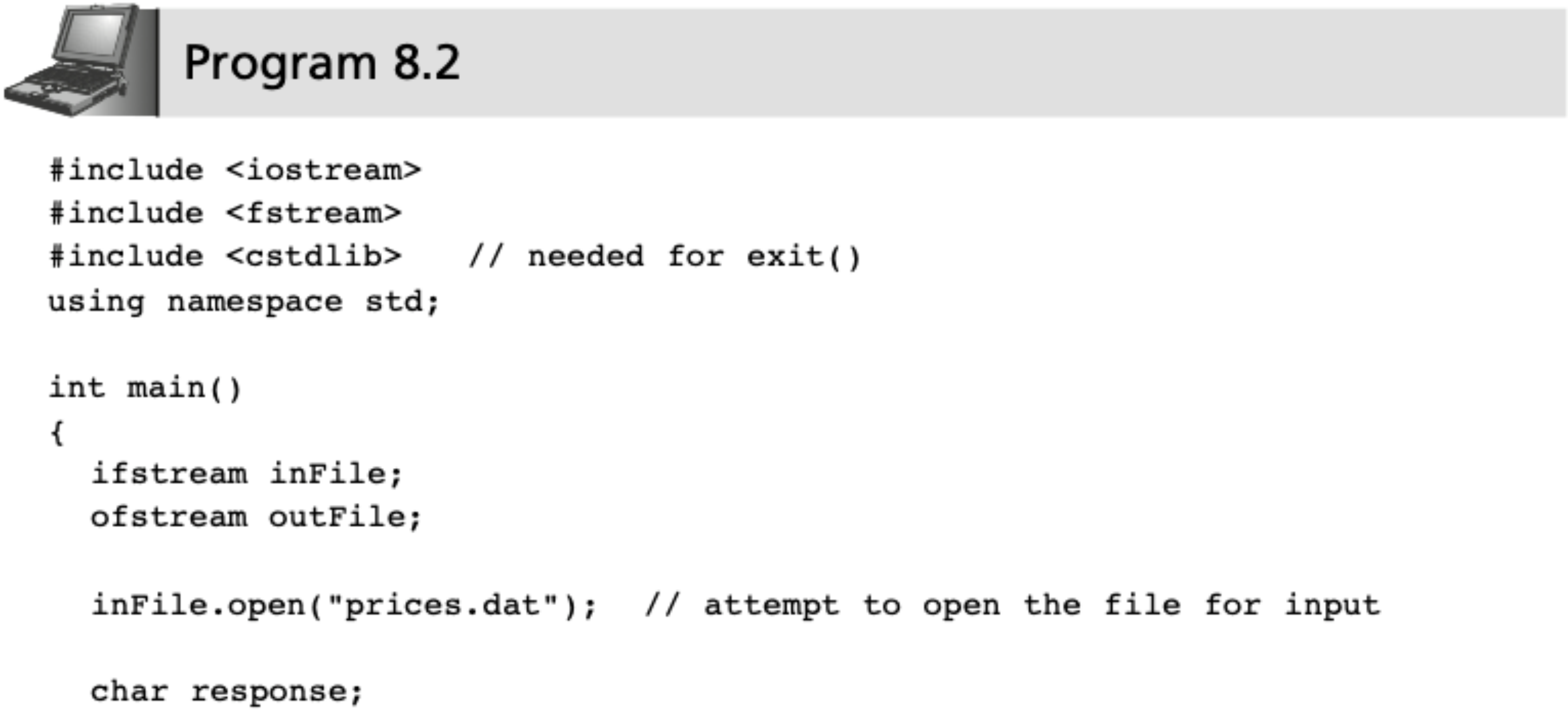
Program 8.1



Program 8.1

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // needed for exit()
using namespace std;

int main()
{
  ifstream inFile;

  inFile.open("prices.dat");  // open the file with the
                              // external name prices.dat
  if (inFile.fail())  // check for a successful open
  {
    cout << "\nThe file was not successfully opened"
         << "\n Please check that the file currently exists."
         << endl;
    exit(1);
  }

  cout  << "\nThe file has been successfully opened for reading."
        << endl;
    // statements to read data from the file would be placed here

  return 0;
}
```

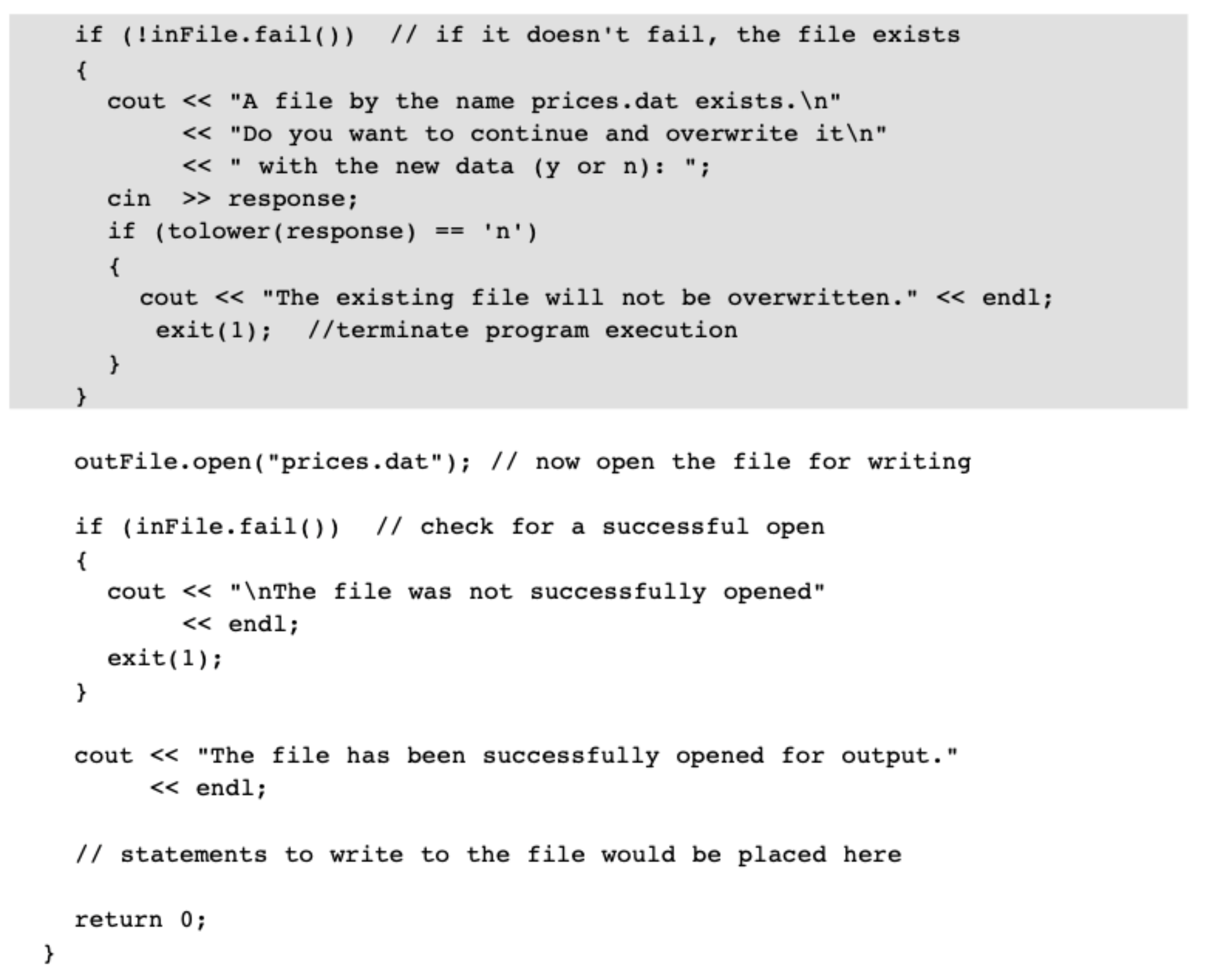
Program 8.2



Program 8.2

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // needed for exit()
using namespace std;

int main()
{
  ifstream inFile;
  ofstream outFile;

  inFile.open("prices.dat");  // attempt to open the file for input

  char response;
```

Program 8.2(continued)



```
  if (!inFile.fail())  // if it doesn't fail, the file exists
  {
    cout << "A file by the name prices.dat exists.\n"
         << "Do you want to continue and overwrite it\n"
         << " with the new data (y or n): ";
    cin  >> response;
    if (tolower(response) == 'n')
    {
      cout << "The existing file will not be overwritten." << endl;
       exit(1);  //terminate program execution
    }
  }

  outFile.open("prices.dat"); // now open the file for writing

  if (inFile.fail())  // check for a successful open
  {
    cout << "\nThe file was not successfully opened"
         << endl;
    exit(1);
  }

  cout << "The file has been successfully opened for output."
       << endl;

  // statements to write to the file would be placed here

  return 0;
}
```

Embedded and Interactive Filenames

- Programs 8.1 and 8.2 have two problems
  - External filename is embedded in program code
  - There's no provision for user to enter filename while program is running
- As both programs are written, if filename is to change, programmer must modify external filename in call to `open()` and recompile program
- Both these problems can be avoided by assigning filename to string variable

Refer to page 470,473 for more explanations and examples

Closing a File

- File is closed using `close()` method
- This method breaks connection between file’s external name and file stream, which can be used for another file
- Because all computers have limit on maximum number of files that can be open at one time, closing files no longer needed makes good sense
- Any open files existing at end of normal program execution are closed automatically by OS

Reading and Writing Character-Based Files

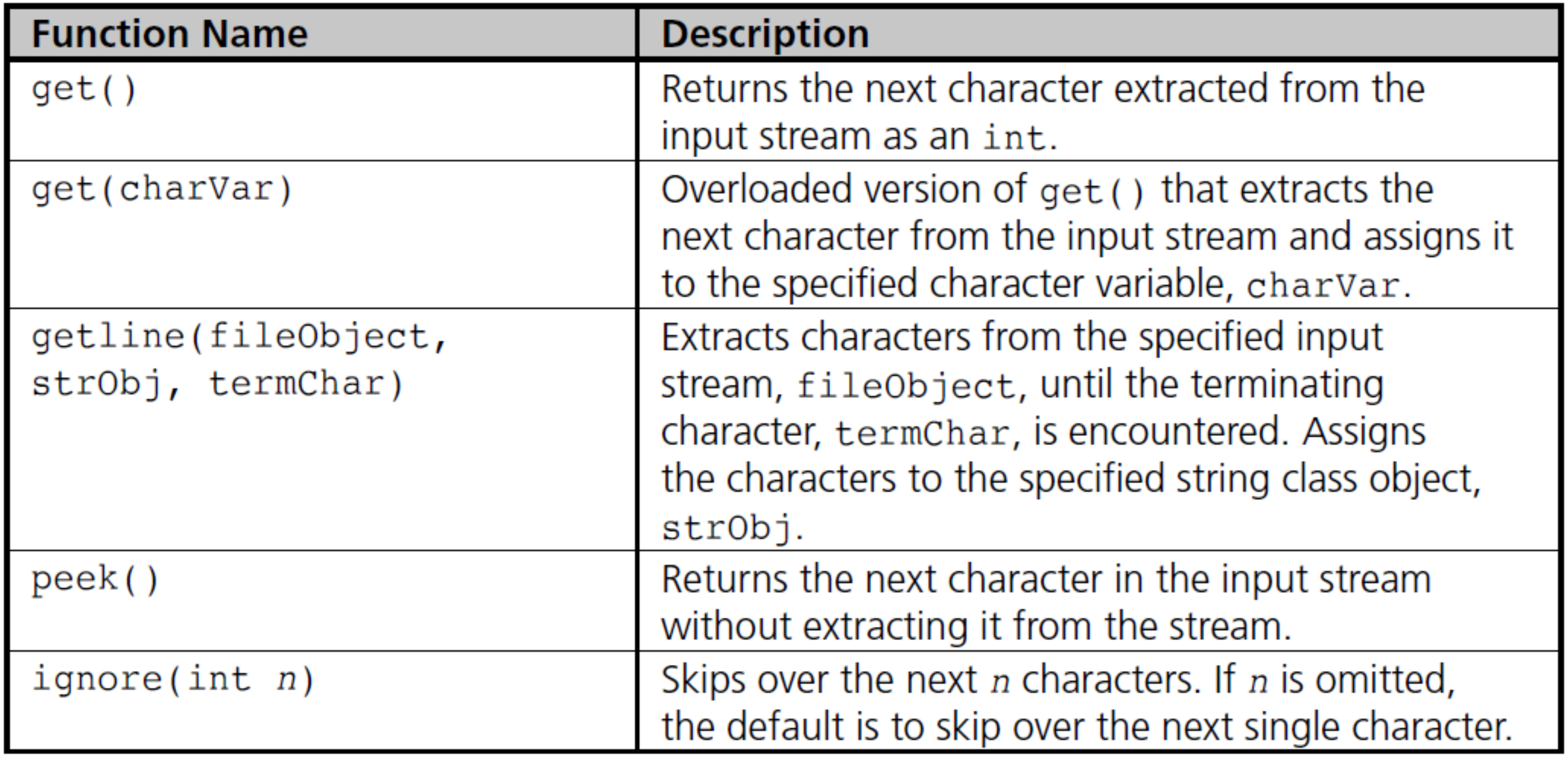
- Reading or writing character-based files involves almost identical operations for reading input from keyboard and writing data to screen
- For writing to a file, `cout` object is replaced by `ofstream` object name declared in program

Reading from a Text File

- Reading data from text file is almost identical to reading data from standard keyboard, except `cin` object is replaced by `ifstream` object declared in program

Refer to page 476,480,483 for more explanations and examples

Reading from a Text File (continued)



Function Name	Description
get()	Returns the next character extracted from the input stream as an int.
get(charVar)	Overloaded version of get() that extracts the next character from the input stream and assigns it to the specified character variable, charVar.
getline(fileObject, strObj, termChar)	Extracts characters from the specified input stream, fileObject, until the terminating character, termChar, is encountered. Assigns the characters to the specified string class object, strObj.
peek()	Returns the next character in the input stream without extracting it from the stream.
ignore(int *n*)	Skips over the next *n* characters. If *n* is omitted, the default is to skip over the next single character.

Table 8.3 Stream Input Class Functions

Standard Device Files

- **Logical file object:** Stream that connects a file of logically related data to a program
- **Physical file object:** Stream that connects to hardware device such as keyboard, screen, or printer
- Actual physical device assigned to your program for data entry is formally called **standard input file**
  - `cin` method calls are routed to this standard input file
  - `cout` method calls are written to a device that has been assigned as standard output file

Other Devices

- Keyboard, display, error, and log streams are connected automatically to stream objects `cin`, `cout`, `cerr`, and `clog` when `iostream` header file is included in program

Random File Access

- **File access:** Refers to process of retrieving data from a file
- Two types of file access
  - Sequential file access
  - Random file access
- **File organization:** Refers to the way data is stored in a file
- The files you have used and will continue to use have a sequential organization, meaning characters in file are stored in a sequential manner

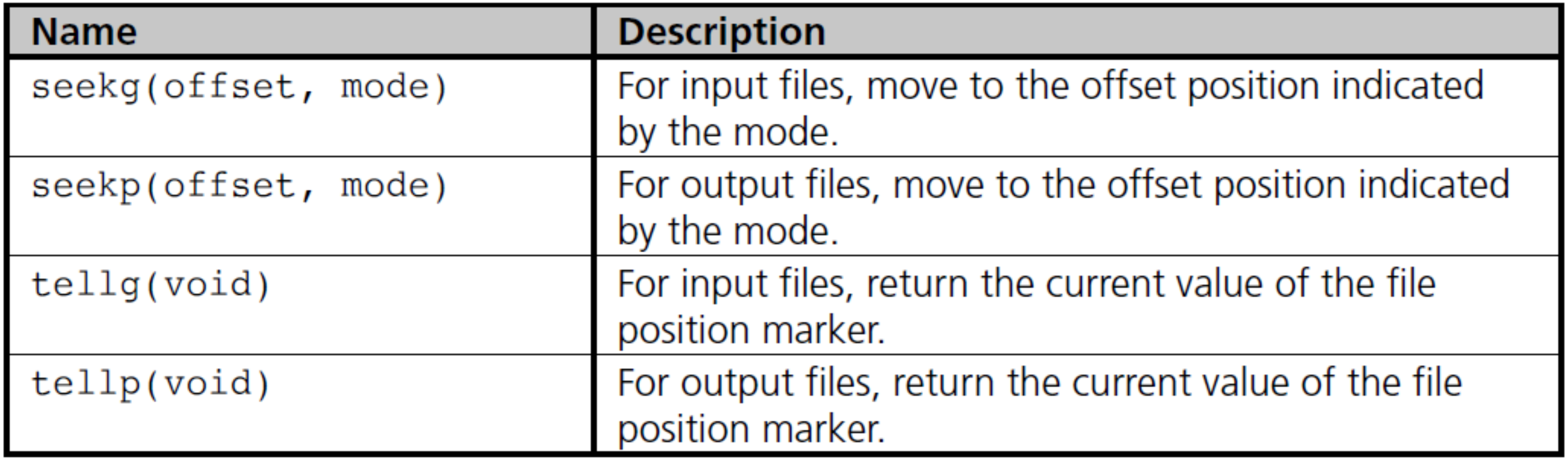
Random File Access (continued)

- Each open file has been read in a sequential manner, meaning characters are accessed one after another, which is called **sequential access**
  - Although characters are stored sequentially, they don’t have to be accessed in same way

Random File Access (continued)

- In **random access**, any character in opened file can be read without having to read all characters stored ahead of it first
  - To provide random access, each `ifstream` object creates a file position marker automatically
  - This marker is a long integer representing an offset from the beginning of file

Random File Access (continued)



Name	Description
`seekg(offset, mode)`	For input files, move to the offset position indicated by the mode.
`seekp(offset, mode)`	For output files, move to the offset position indicated by the mode.
`tellg(void)`	For input files, return the current value of the file position marker.
`tellp(void)`	For output files, return the current value of the file position marker.

Table 8.4 File Position Marker Functions

Random File Access (continued)

- `seek()` method allows programmer to move to any position in file
- Character's position is referred to as its **offset** from the start of file

Refer to page 489 for more explanations and examples

File Streams as Function Arguments

- A file stream object can be used as a function argument
- The function's formal parameter must be a reference to the appropriate stream, either `ifstream&` or `ofstream&`
  - Examples: `inOut()`,`getOpen()`

Refer to page 491 ,492 for more explanations and examples

A Case Study: Pollen Count File Update

- After a data file has been created, application programs are typically written to read and update the file with current data
- In this case study, a file is used as a database storing the ten most recent polling counts, which are used in the summer as allergy “irritability” measures
  - Analyze the problem
  - Develop a solution
  - Code the solution
  - Test and correct the program

Refer to page 494-499 for more explanations and examples

A Closer Look: The `iostream` Class Library

- Classes in `iostream` class library access files by using entities called streams
- For most systems the data bytes transferred on a stream represent ASCII characters or binary numbers
- Mechanism for reading a byte stream from a file or writing a byte stream to a file is hidden when using a high level language like C++

File Stream Transfer Mechanism

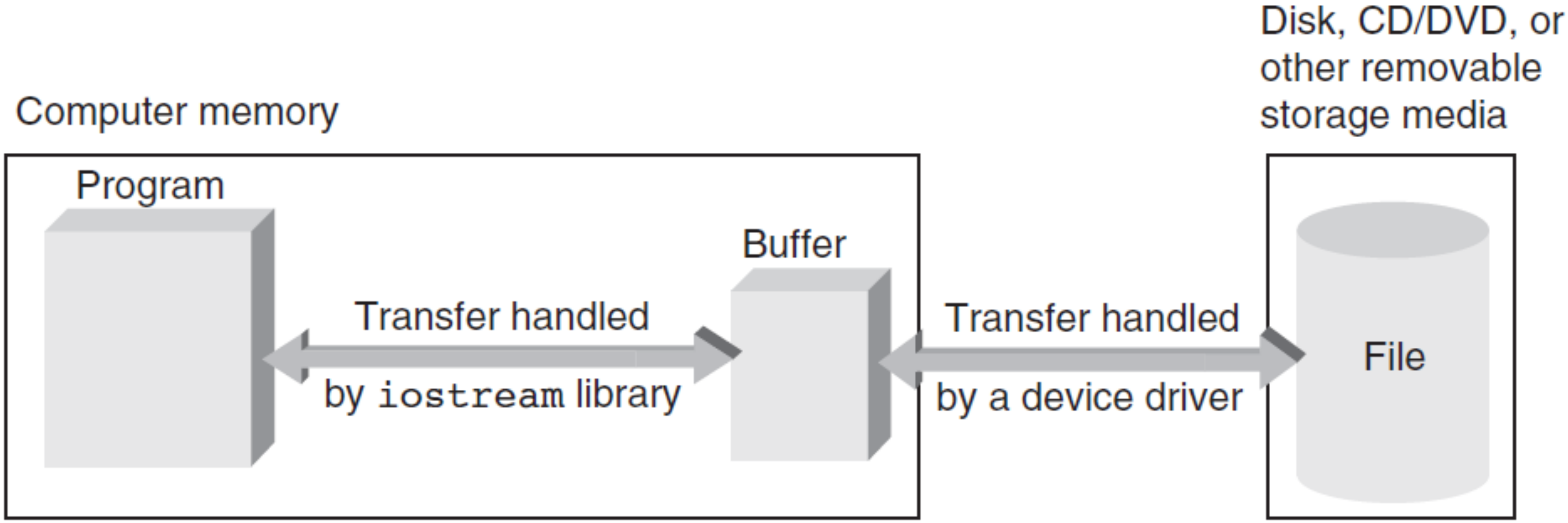


Figure 8.5 The data transfer mechanism

Components of the `iostream` Class Library

- `iostream` class library consists of two primary base classes
  - `streambuf`
  - `ios`
- `streambuf` class provides the file buffer
- `ios` class contains pointer to the file buffers provided by `streambuf` class and general routines for transferring text data

Components of the `iostream` Class Library (continued)

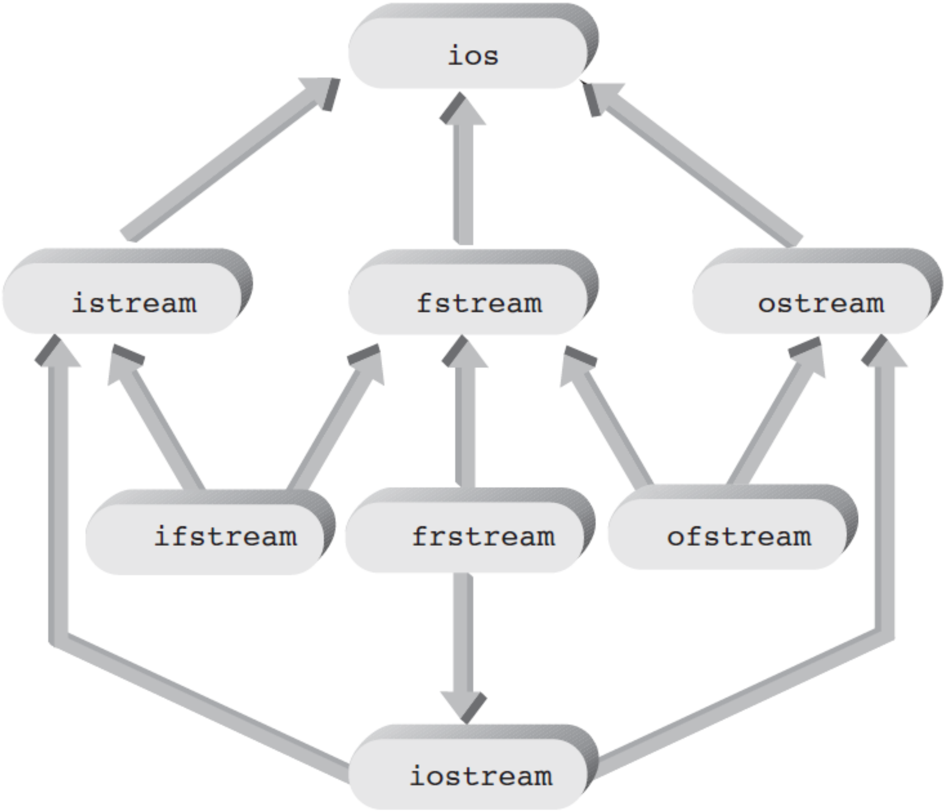


Figure 8.6 The base class `ios` and its derived classes

Components of the `iostream` Class Library (continued)

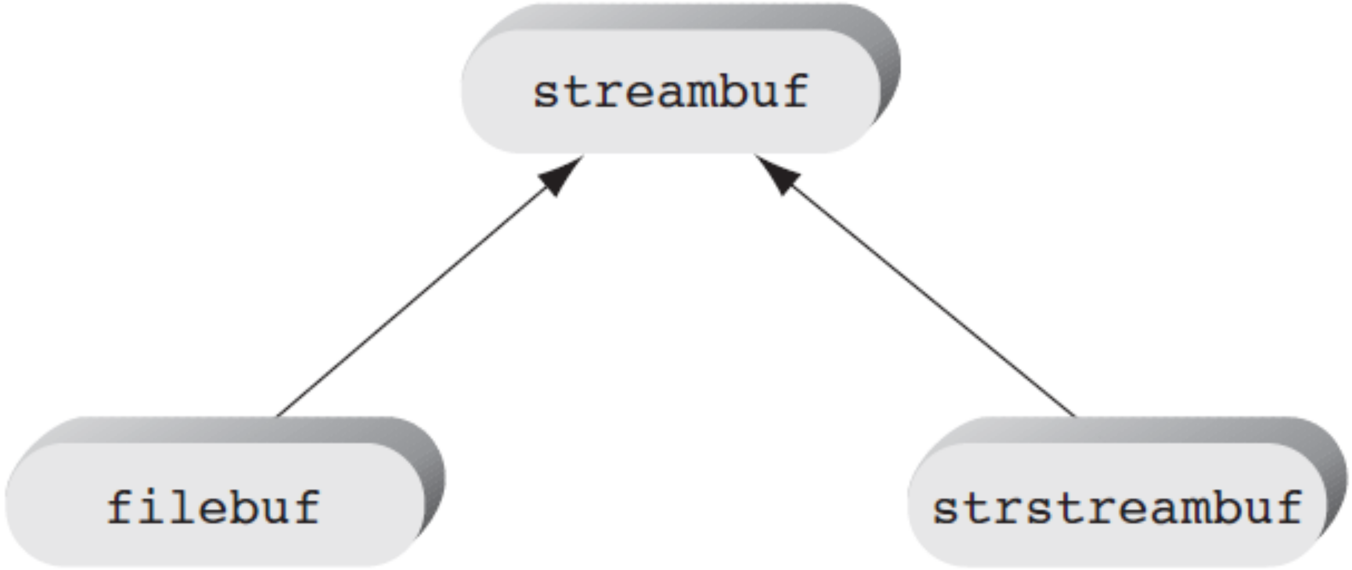
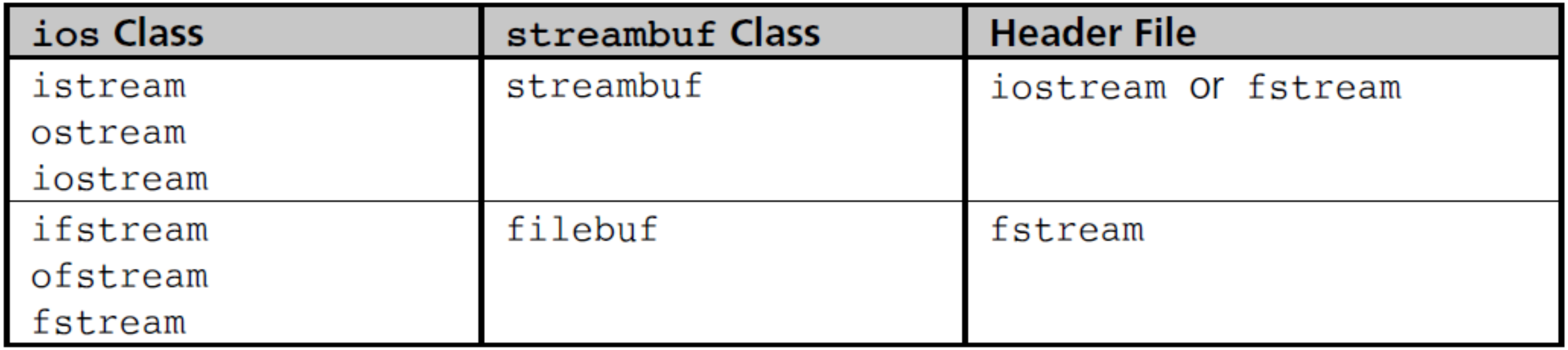


Figure 8.7 The base class `streambuf` and its derived classes

Components of the `iostream` Class Library (continued)



ios Class	streambuf Class	Header File
istream ostream iostream	streambuf	iostream or fstream
ifstream ofstream fstream	filebuf	fstream

Table 8.5 Correspondence Between Classes in Figures 8.6 and 8.7

In-Memory Formatting

- In addition to the classes shown in Figure 8.7, a class named `strstream` is derived from `ios` class
  - Uses `strstream` class shown in Figure 8.7, requires `strstream` header file, and provides capabilities for writing and reading to and from in-memory defined streams
- As output, these streams are typically used to "assemble" a string from a smaller pieces until a complete line of characters is ready to be written to `cout` or to a file

Refer to page 503,504 for more explanations and examples

In-Memory Formatting (continued)

- `strstream` object can also be opened in input mode
  - This stream is used as working storage area, or buffer, for storing complete line of text from file or standard input
  - After buffer has been filled, and extraction operator is used to “disassemble” the string into component parts and convert each data item into its designated data type

Common Programming Errors

- Forgetting to open file before attempting to read from it or write to it
- Using file’s external name in place of internal file stream name when accessing a file
- Opening file for output without first checking that file with the same name already exists
  - Opening existing file for output overwrites that file
- Not understanding that end of file is detected only after `EOF` marker has been read and passed over

Common Programming Errors (continued)

- Attempting to detect end of file by using character variables for `EOF` marker
  - Any variable used to accept `EOF` marker must be declared an integer variable
- Using integer argument with `seekg()` and `seekp()` functions
  - This offset must be long integer constant or variable

Summary

- Data file is any collection of data stored together in an external storage medium under a common name
- Data file is connected to file stream by using `fstream` `open()` function
- File can be opened in input and output mode
- All file streams must be declared as objects of `ifstream` or `ofstream` class
- In addition to any files opened in a function, standard stream objects `cin`, `cout`, and `cerr` are declared and opened automatically when a program runs

Homework

- P508, exercise 2
- P508, exercise 3