C++ for Engineers and Scientists

Second Edition

Chapter 8
I/O File Streams and Data Files

Objectives

- I/O File Stream Objects and Methods
- Reading and Writing Character-Based Files
- Exceptions and File Checking
- Random File Access
- File Streams as Function Arguments

Objectives (continued)

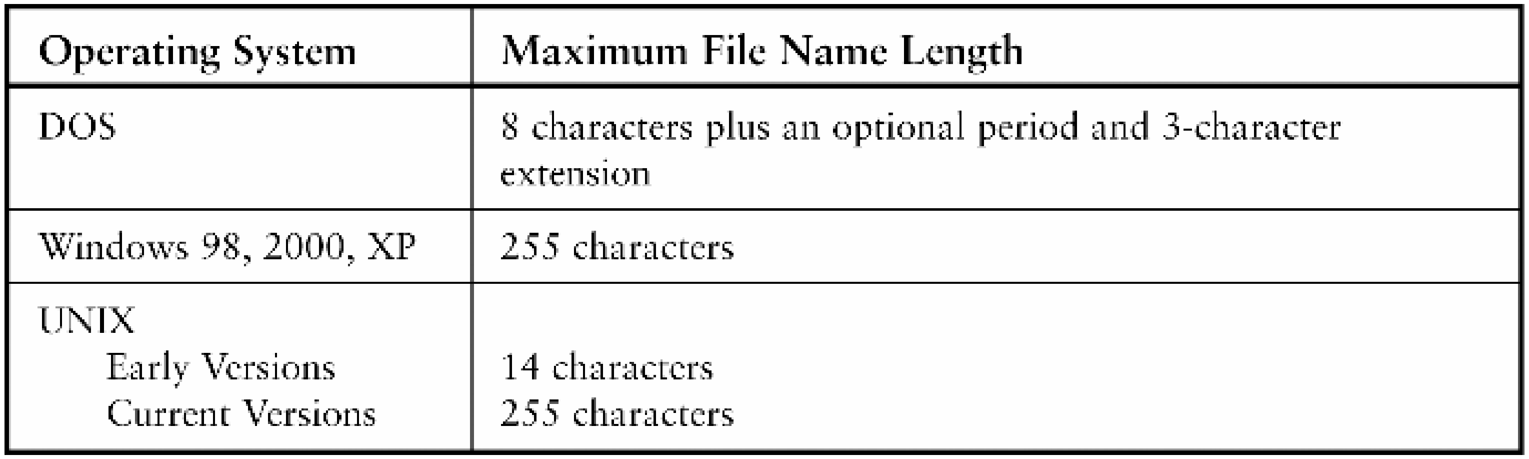
- Common Programming Errors
- The **`iostream`** Class Library

I/O File Stream Objects and Methods

- To store and retrieve data outside a program, you need:
  - A file
  - A file stream object
- **File**: a collection of data stored under a common name on an external storage unit (disk, CD-ROM, etc.)
- **External file name**: unique file name identifying the file to the operating system

I/O File Stream Objects and Methods (continued)

- External file name must conform to the name specifications of the O/S
- File extension can indicate the data type



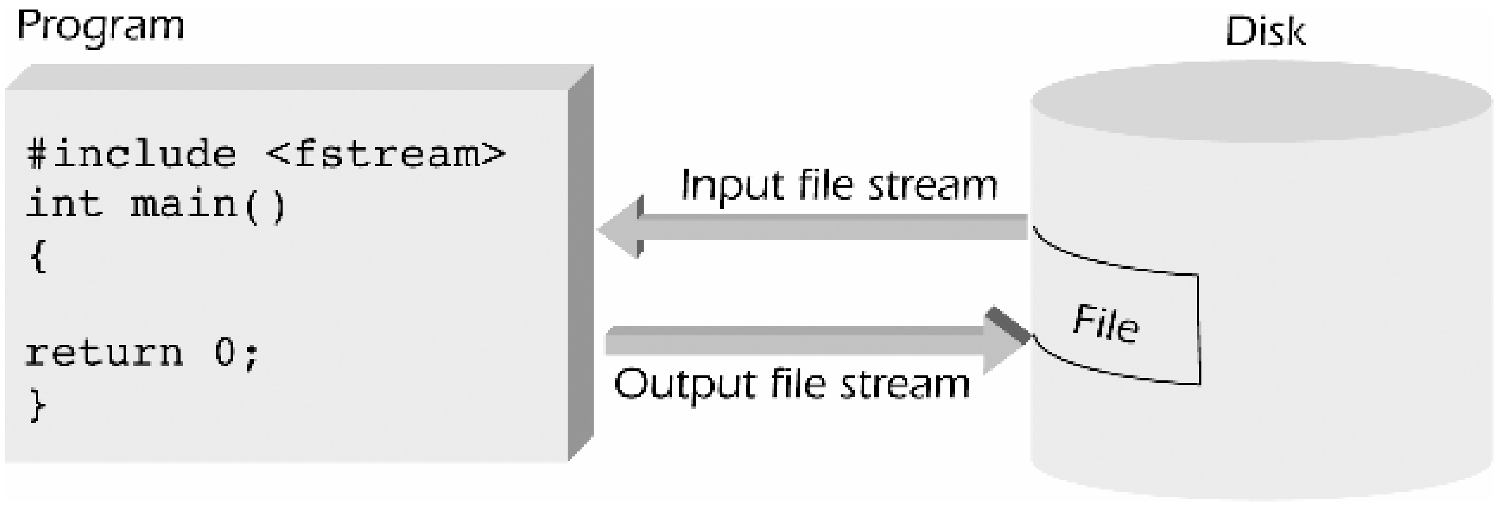
Operating System	Maximum File Name Length
DOS	8 characters plus an optional period and 3-character extension
Windows 98, 2000, XP	255 characters
UNIX Early Versions Current Versions	 14 characters 255 characters

I/O File Stream Objects and Methods (continued)

- Two basic data file types:
  - **Text**: character-based; stores characters using ASCII or UNICODE character codes
  - **Binary**: stores characters in binary form; numbers are in binary, strings are in ASCII or UNICODE form; more compact storage
- **File stream**: a one-way transmission path to connect a physical file to a program

I/O File Stream Objects and Methods (continued)

- File stream modes:
  - Input: moves data from a physical file into a program
  - Output: sends or writes data to a file from a program



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

return 0;
}
```

I/O File Stream Objects and Methods (continued)

- To read and write data to and from a file requires an input file stream and an output file stream
- Input file stream declared as type **`ifstream`**
- Output file stream declared as type **`ofstream`**
- File stream objects provide methods for connecting to a file, opening it, reading, writing, closing, etc.

I/O File Stream Objects and Methods (continued)

- **`open()`** method: requires the external file name as an argument

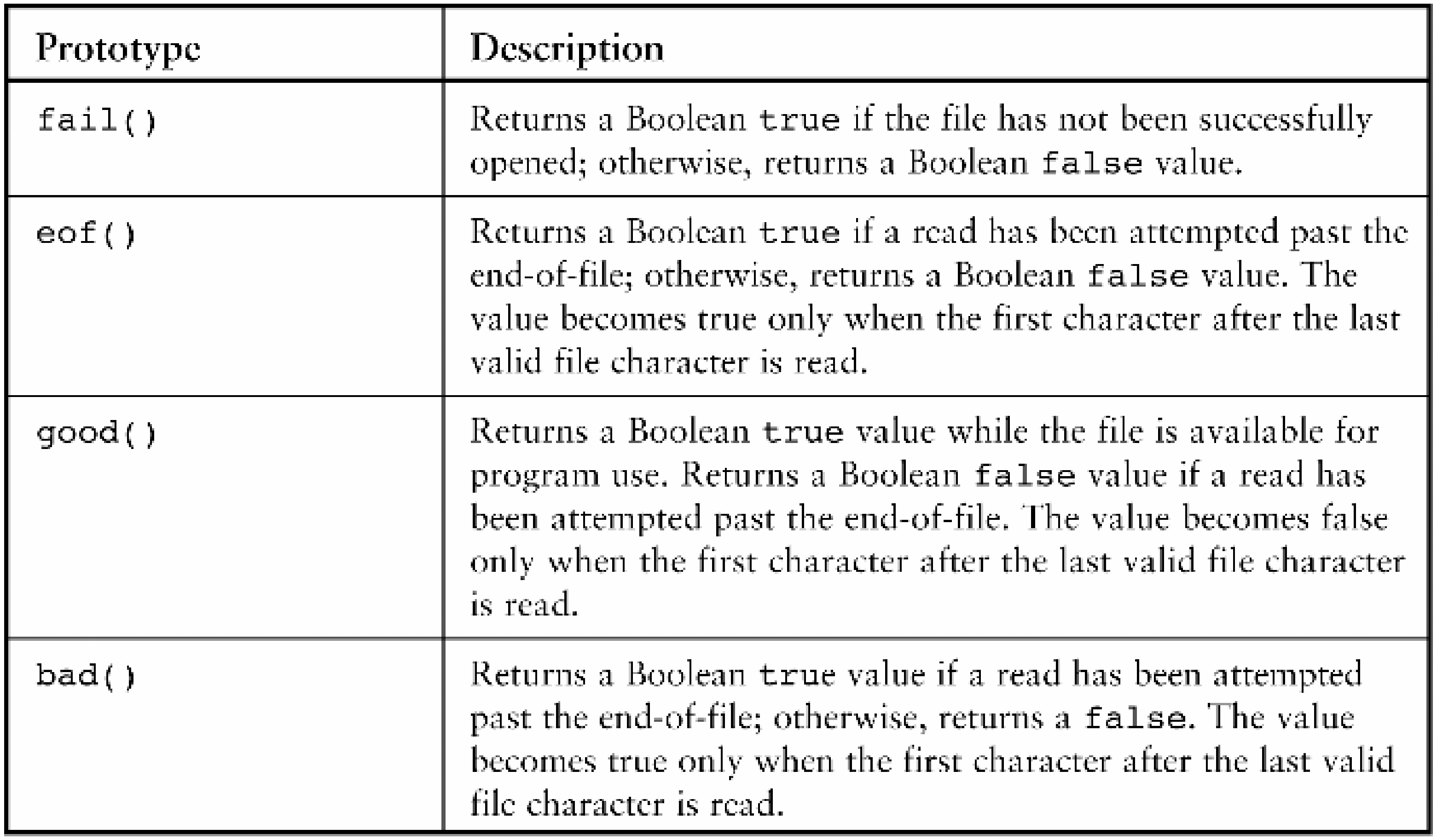
Syntax:

```
streamObject.open(“externalFileName”)
```

- File status methods provide status of file operations

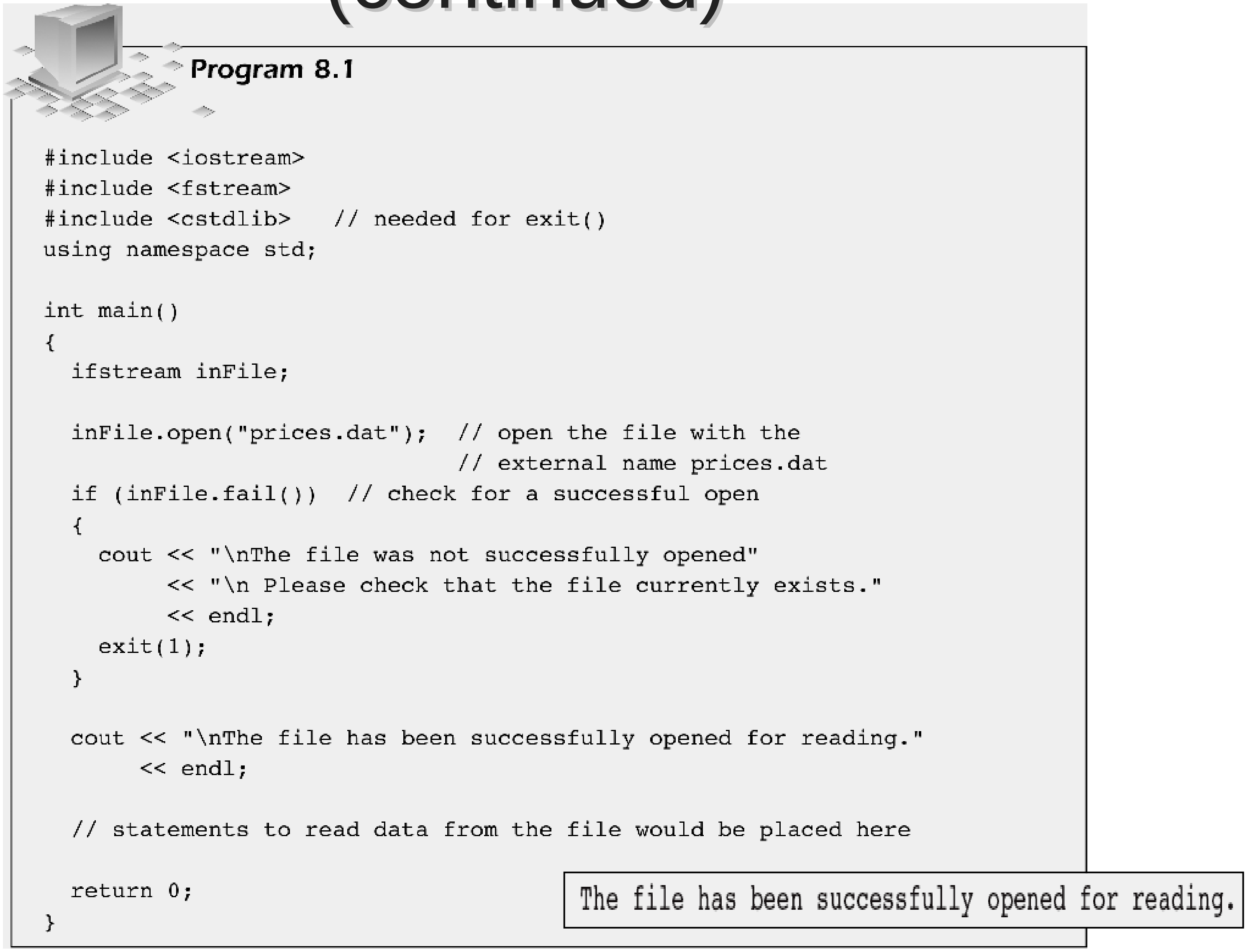
I/O File Stream Objects and Methods (continued)

File status methods:



Prototype	Description
`fail()`	Returns a Boolean `true` if the file has not been successfully opened; 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 a read has been attempted past the end-of-file; otherwise, returns a `false`. The value becomes true only when the first character after the last valid file character is read.

I/O File Stream Objects and Methods (continued)



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;
}
```

```
The file has been successfully opened for reading.
```

I/O File Stream Objects and Methods (continued)

- File existence must be checked before output; if it exists, data may be overwritten
- You can check for existence by trying to open the file
  - Open will fail if the file does not exist

I/O File Stream Objects and Methods (continued)

```
int main()
{
  ifstream inFile;
  ofstream outFile;

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

  char response;

  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);
  }
```

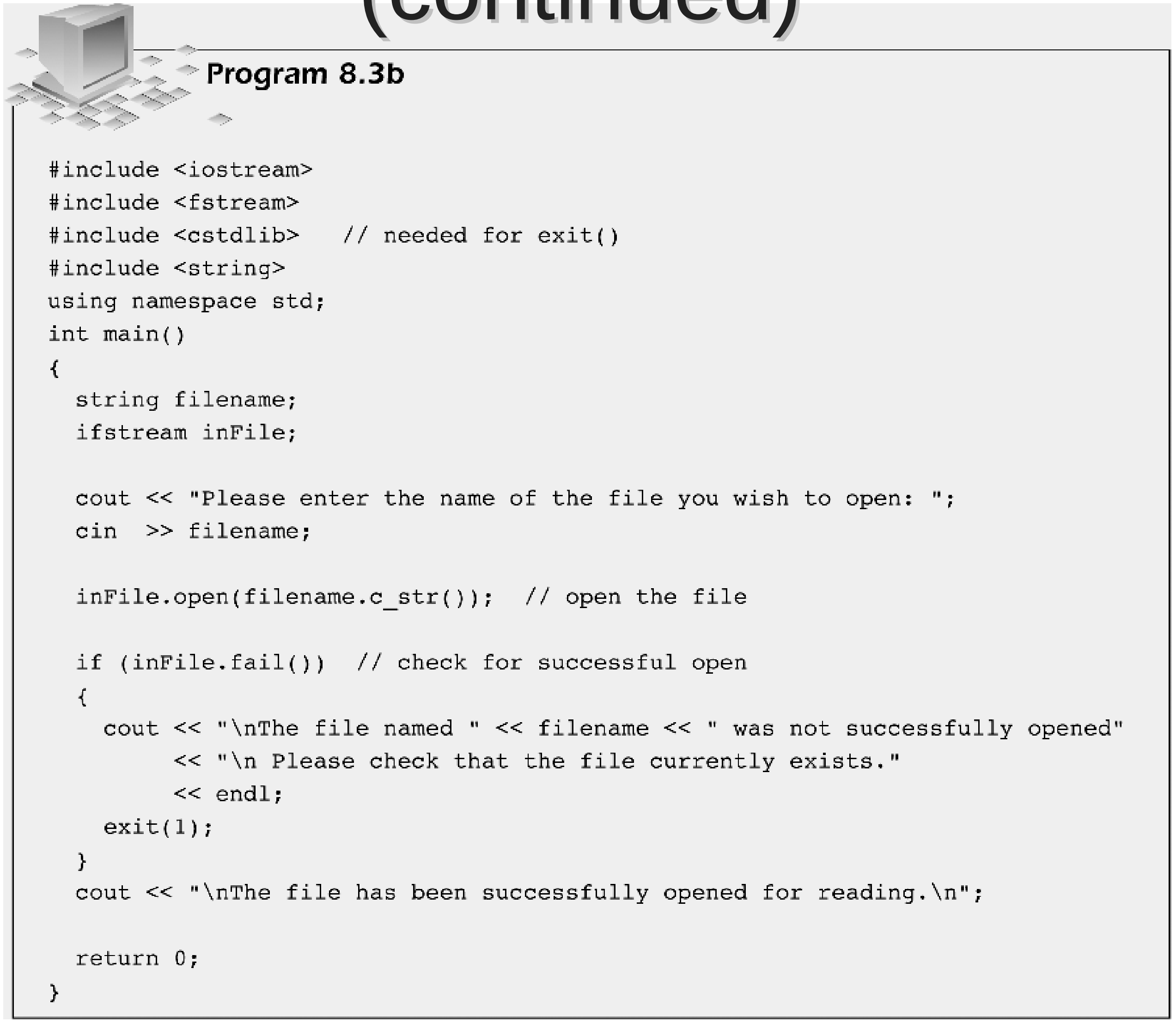
I/O File Stream Objects and Methods (continued)

- A string object can be used for the file name, but it must be converted to a C-string using **`c_str()`** method
- **`close()`** method closes a file and breaks the connection to the external file

Syntax:

```
streamObject.close()
```

I/O File Stream Objects and Methods (continued)



Program 8.3b

```
#include <iostream>
#include <fstream>
#include <cstdlib>   // needed for exit()
#include <string>
using namespace std;
int main()
{
  string filename;
  ifstream inFile;

  cout << "Please enter the name of the file you wish to open: ";
  cin  >> filename;

  inFile.open(filename.c_str());  // open the file

  if (inFile.fail())  // check for successful open
  {
    cout << "\nThe file named " << filename << " was not successfully opened"
         << "\n Please check that the file currently exists."
         << endl;
    exit(1);
  }
  cout << "\nThe file has been successfully opened for reading.\n";

  return 0;
}
```

Reading & Writing Character-Based Files

- Reading or writing character-based files is similar to input from keyboard and output to a display

Syntax:

```
outputFileStream << stringExpression
```

Reading & Writing Character-Based Files (continued)

```
int main()
{
  string filename = "prices.dat";  // put the filename up front
  ofstream outFile;

  outFile.open(filename.c_str());

  if (outFile.fail())
  {
    cout << "The file was not successfully opened" << endl;
    exit(1);
  }

  // set the output file stream formats
  outFile << setiosflags(ios::fixed)
          << setiosflags(ios::showpoint)
          << setprecision(2);

  // send data to the file
  outFile << "Mats " << 39.95 << endl
          << "Bulbs "  << 3.22 << endl
          << "Fuses " << 1.08 << endl;

  outFile.close();
  cout << "The file " << filename
       << " has been successfully written." << endl;

  return 0;
}
```

Reading & Writing Character-Based Files (continued)

- Use an **`ifstream`** object to read data from a text file

Syntax:

```
inputFileStream >> variableName
```

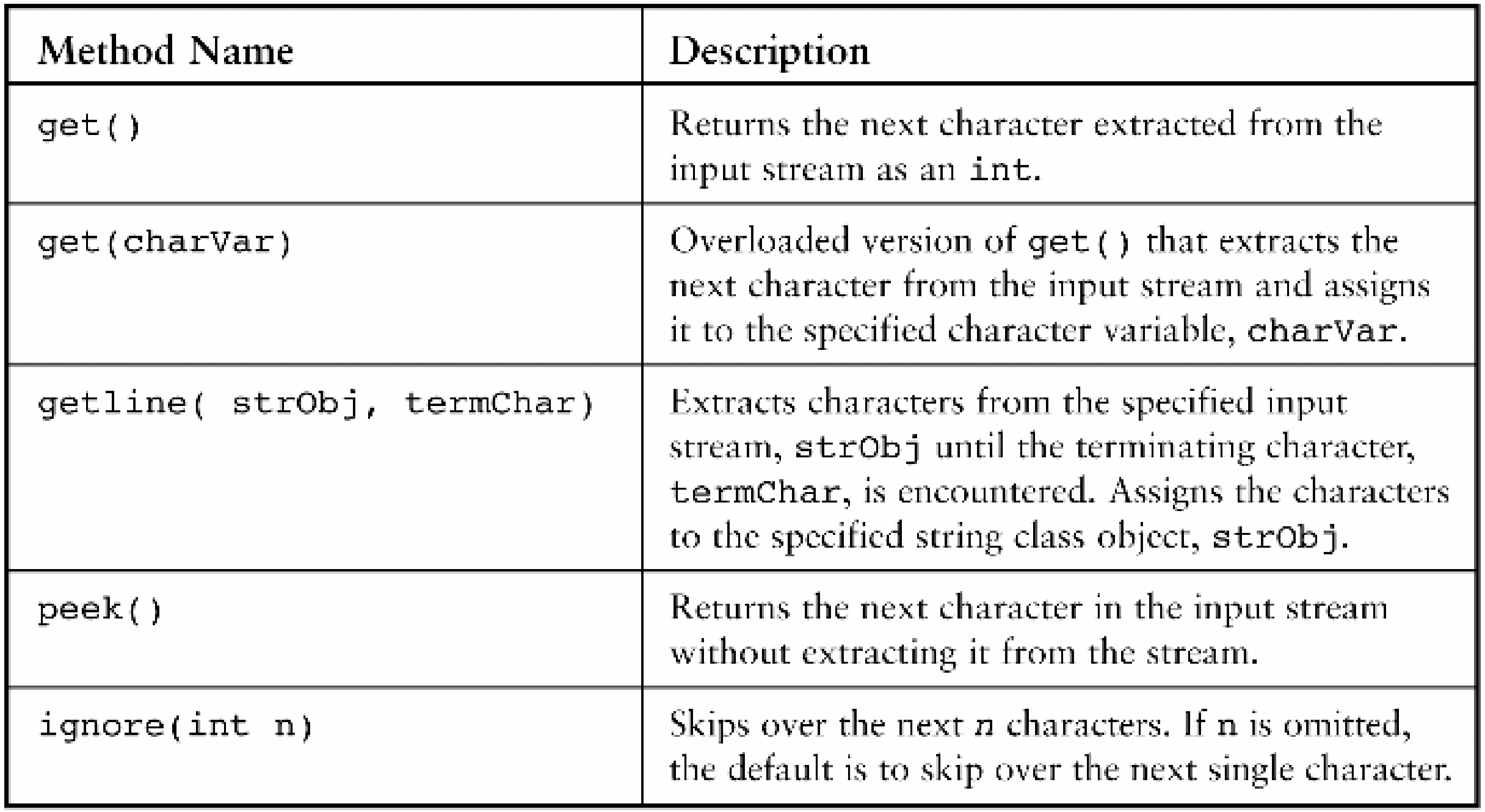
- The **`eof()`** method can be used to test if the end of file has been reached

Example:

```
//while not at end of file
while(!inFile.eof())
```

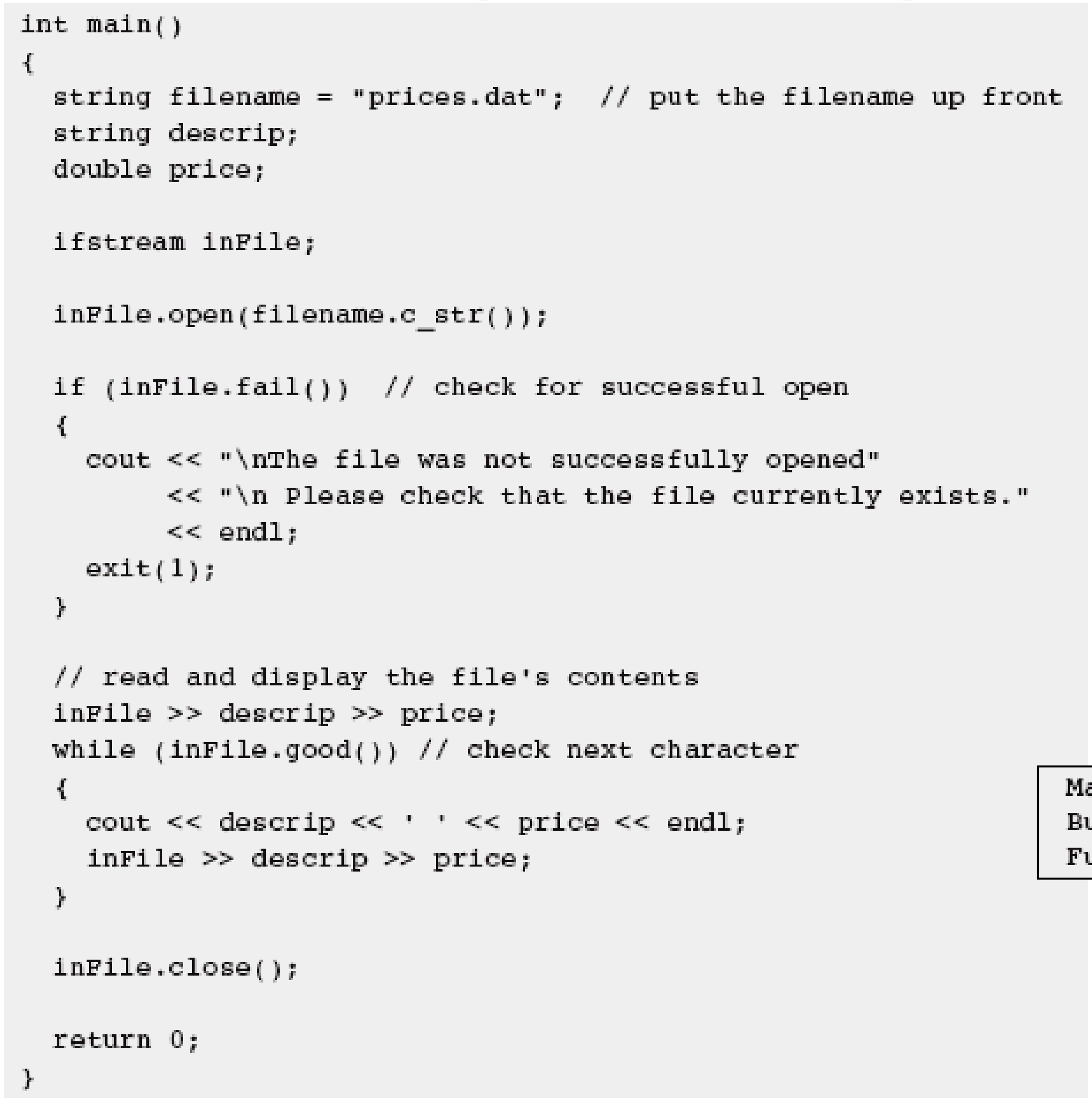
Reading & Writing Character-Based Files (continued)

File Stream Methods



Method 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( strObj, termChar)`	Extracts characters from the specified input stream, `strObj` 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.

Reading & Writing Character-Based Files (continued)



```
int main()
{
  string filename = "prices.dat";  // put the filename up front
  string descrip;
  double price;

  ifstream inFile;

  inFile.open(filename.c_str());

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

  // read and display the file's contents
  inFile >> descrip >> price;
  while (inFile.good()) // check next character
  {
    cout << descrip << ' ' << price << endl;
    inFile >> descrip >> price;
  }

  inFile.close();

  return 0;
}
```

```
Mats 39.95
Bulbs 3.22
Fuses 1.08
```

Reading & Writing Character-Based Files (continued)

- Extraction operation **>>** returns true if data was extracted from a stream and false otherwise; this can be used for testing end of file

Example:

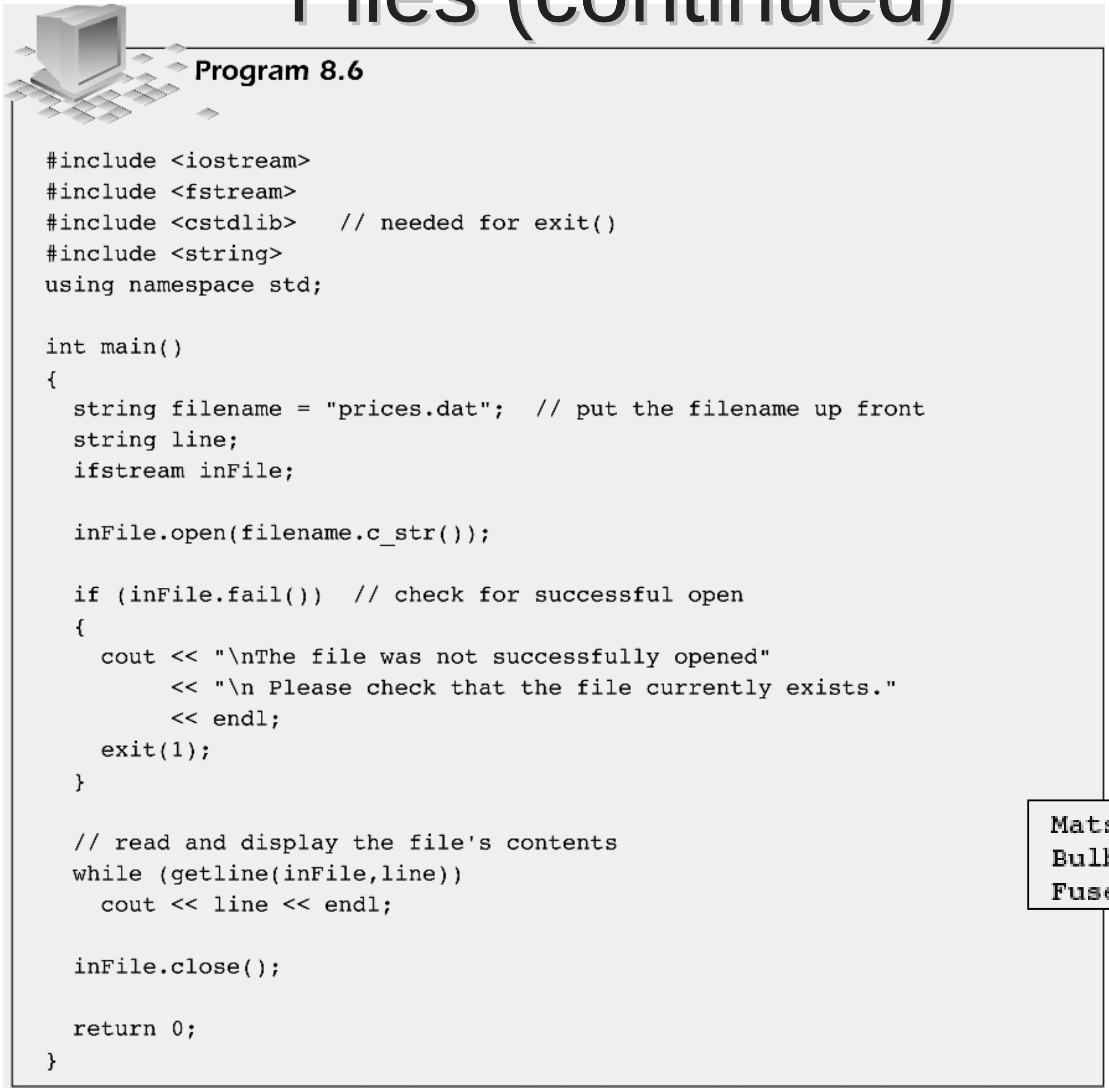
```
while (inFile >> descrip >> price)
```

- **`getline()`**can also be used to read data from a text file

Syntax:

```
getline(fileObject, strObj, terminateChar)
```

Reading & Writing Character-Based Files (continued)



Program 8.6

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

int main()
{
  string filename = "prices.dat";  // put the filename up front
  string line;
  ifstream inFile;

  inFile.open(filename.c_str());

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

  // read and display the file's contents
  while (getline(inFile,line))
    cout << line << endl;

  inFile.close();

  return 0;
}
```

```
Mats 39.95
Bulbs 3.22
Fuses 1.08
```

Reading & Writing Character-Based Files (continued)

- File Stream objects are logical file objects
- Physical file object is a stream that connects to a hardware device (e.g., keyboard, screen, printer, etc.)
- **Standard input file**: physical device assigned to your program for data entry; usually the keyboard
- **Standard output file**: physical device assigned to your program for data display; usually the screen

Reading & Writing Character-Based Files (continued)

- **`cin`** input stream is connected to the standard input device
- **`cout`** output stream is connected to the standard output device
- **`cerr`** object stream: standard error stream, connected to the terminal screen
- **`clog`** object stream: standard log stream, connected to the terminal screen

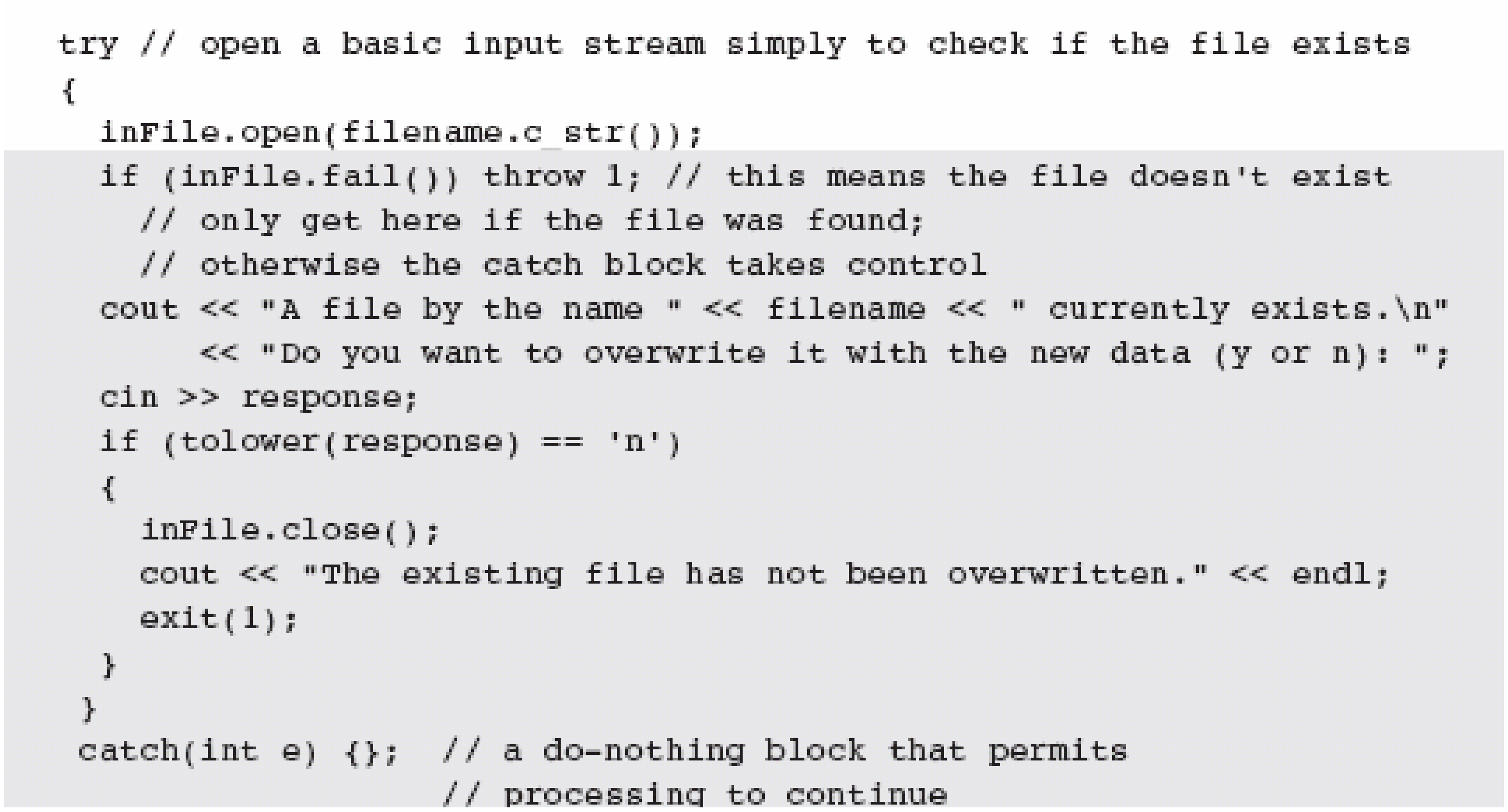
Reading & Writing Character-Based Files (continued)

- Other device names known to the system can be used, such as
  - **`prn`**: printer on IBM-compatible PCs

Exceptions and File Checking

- Place file operation statements in **`try`** blocks, and create **`catch`** blocks for the possible errors
- Opening a file for output will overwrite an existing file of the same name
- You can try to open the desired output file for input to determine if it already exists, before opening it for output

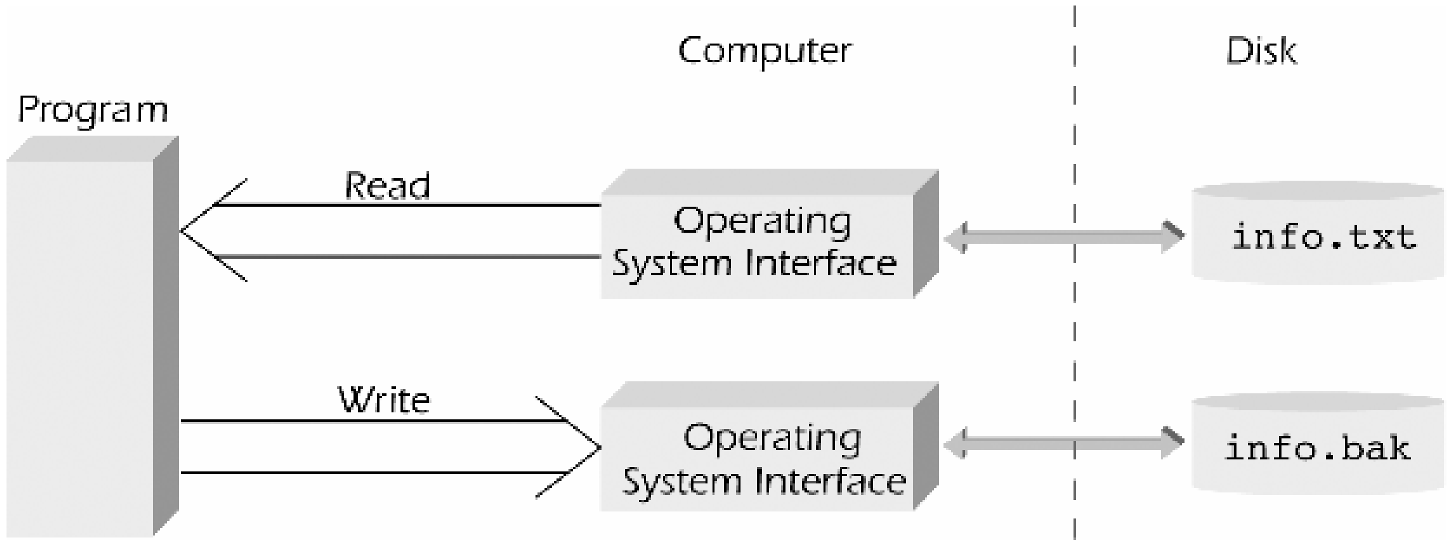
Exceptions and File Checking (continued)



```
try // open a basic input stream simply to check if the file exists
{
  inFile.open(filename.c_str());
  if (inFile.fail()) throw 1; // this means the file doesn't exist
    // only get here if the file was found;
    // otherwise the catch block takes control
  cout << "A file by the name " << filename << " currently exists.\n"
       << "Do you want to overwrite it with the new data (y or n): ";
  cin >> response;
  if (tolower(response) == 'n')
  {
    inFile.close();
    cout << "The existing file has not been overwritten." << endl;
    exit(1);
  }
 }
 catch(int e) {};  // a do-nothing block that permits
                   // processing to continue
```

Exceptions and File Checking: Opening Multiple Files

- File copying requires an input stream for the source and an output stream for the destination
- **`try`** blocks can be nested, so that the inner operation is not tried if the outer operation fails

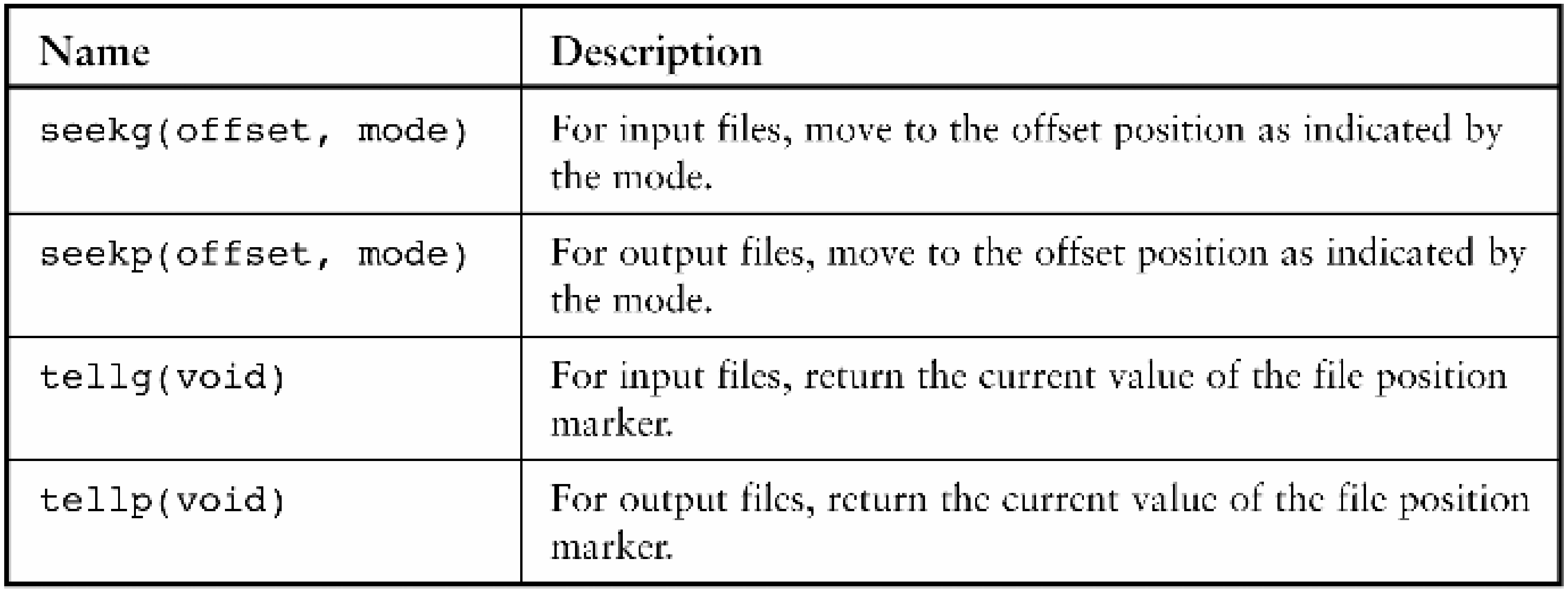


Random File Access

- Two types of file access:
  - **Sequential access**: characters are stored and retrieved in a sequential manner
  - **Random access**: characters can be read or written directly at any position
- For random access, the **`ifstream`** object creates a long integer file position marker representing the offset from the beginning of the file

Random File Access (continued)

File Position Marker Functions



Name	Description
`seekg(offset, mode)`	For input files, move to the offset position as indicated by the mode.
`seekp(offset, mode)`	For output files, move to the offset position as 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.

Random File Access (continued)

- **`seek()`** functions allow you to move to any position in the file
- Character position in a data file is zero-relative
- Arguments to **`seek()`** functions are the offset into the file and the mode (where the offset is to be calculated from)
- Mode alternatives:
  - **`ios::beg`**: start from beginning of file
  - **`ios::cur`**: start from current position
  - **`ios::end`**: start from end of file

Random File Access (continued)

- **`Seek()`** function examples:

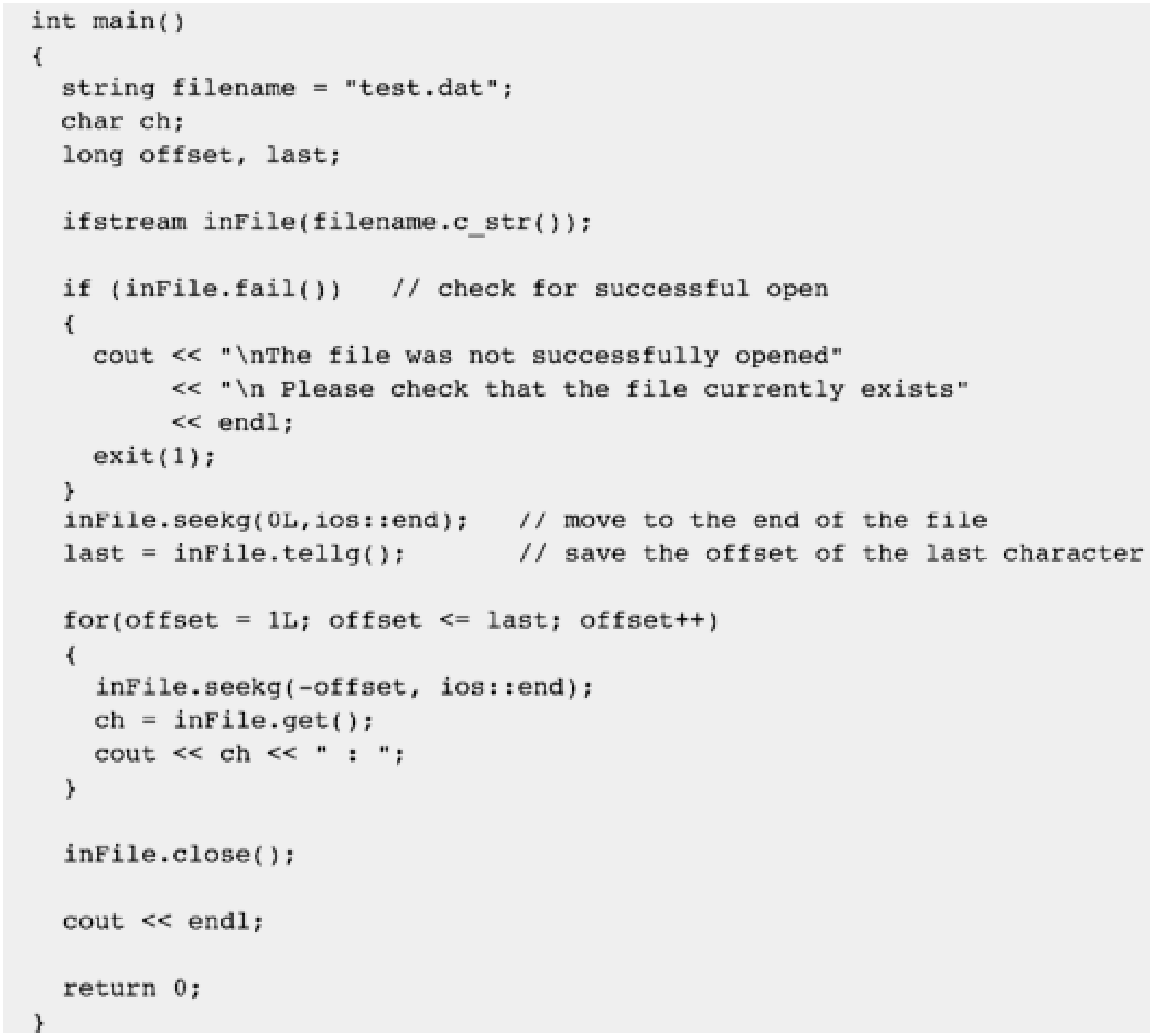
```
inFile.seekg(4L,ios::beg); //go to 5th char
inFile.seekg(4L,ios::cur); //move ahead 5
  characters
inFile.seekg(-4L,ios::cur); //move back 5
  characters
```

- **`tell()`** functions return the offset value of the file position marker

Example:

```
  inFile.tellg();
```

Random File Access (continued)

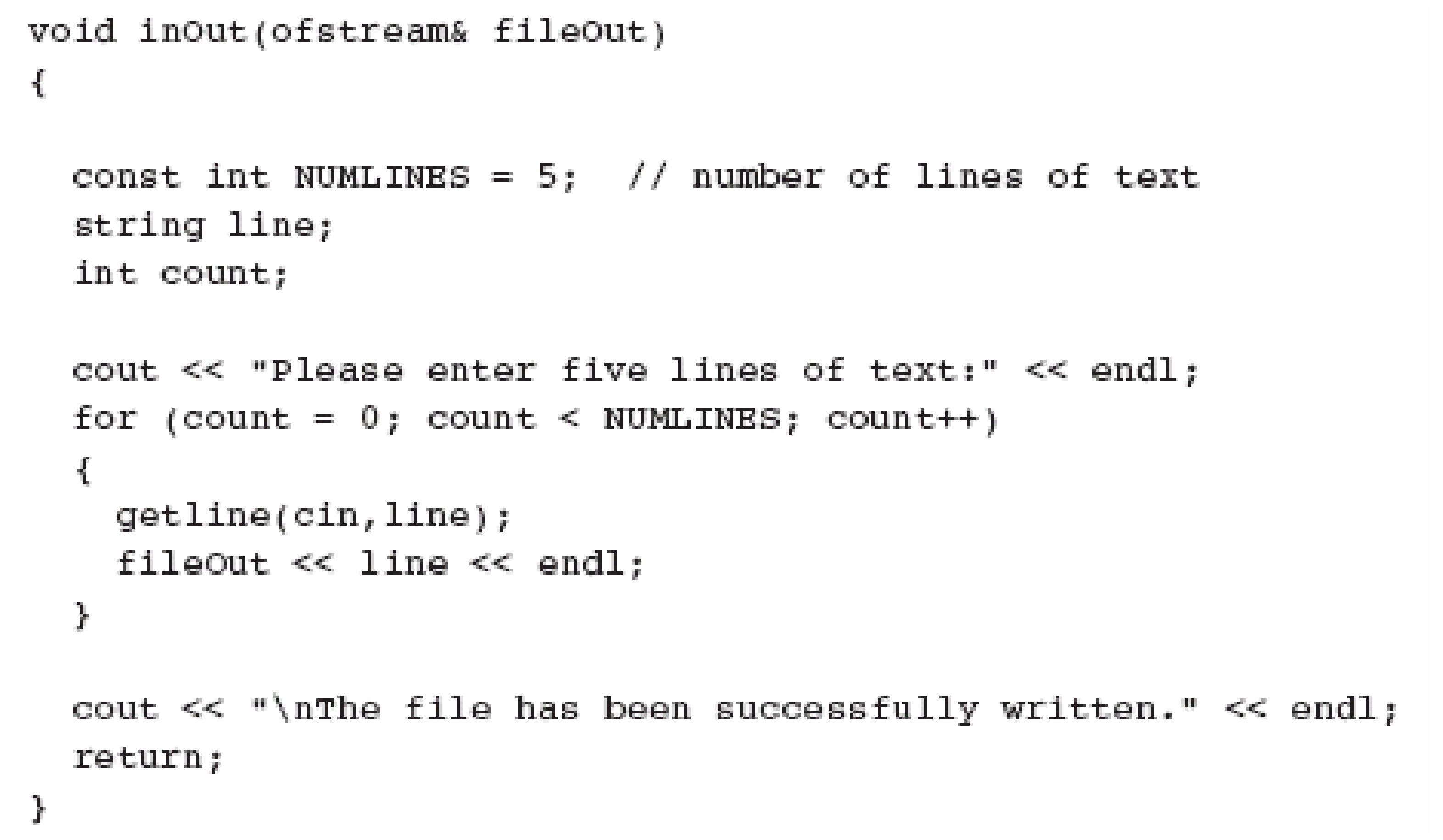


```
int main()
{
  string filename = "test.dat";
  char ch;
  long offset, last;

  ifstream inFile(filename.c_str());

  if (inFile.fail())   // check for successful open
  {
    cout << "\nThe file was not successfully opened"
         << "\n Please check that the file currently exists"
         << endl;
    exit(1);
  }
  inFile.seekg(0L,ios::end);   // move to the end of the file
  last = inFile.tellg();       // save the offset of the last character

  for(offset = 1L; offset <= last; offset++)
  {
    inFile.seekg(-offset, ios::end);
    ch = inFile.get();
    cout << ch << " : ";
  }

  inFile.close();

  cout << endl;

  return 0;
}
```

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 (**`ifstream&`** or **`ofstream&`**)

File Streams as Function Arguments (continued)



```
void inOut(ofstream& fileOut)
{

  const int NUMLINES = 5;  // number of lines of text
  string line;
  int count;

  cout << "Please enter five lines of text:" << endl;
  for (count = 0; count < NUMLINES; count++)
  {
    getline(cin,line);
    fileOut << line << endl;
  }

  cout << "\nThe file has been successfully written." << endl;
  return;
}
```

Common Programming Errors

- Using a file’s external name instead of the internal file stream object name
- Opening a file for output without first checking if a file with the same name already exists
- Not understanding that end of file is detected only after the EOF sentinel has been read or passed over
- Attempting to detect end of file using character variables for the EOF marker

Common Programming Errors (continued)

- Using an integer argument with the **`seekg()`** and **`seekp()`** functions instead of a long integer

Summary

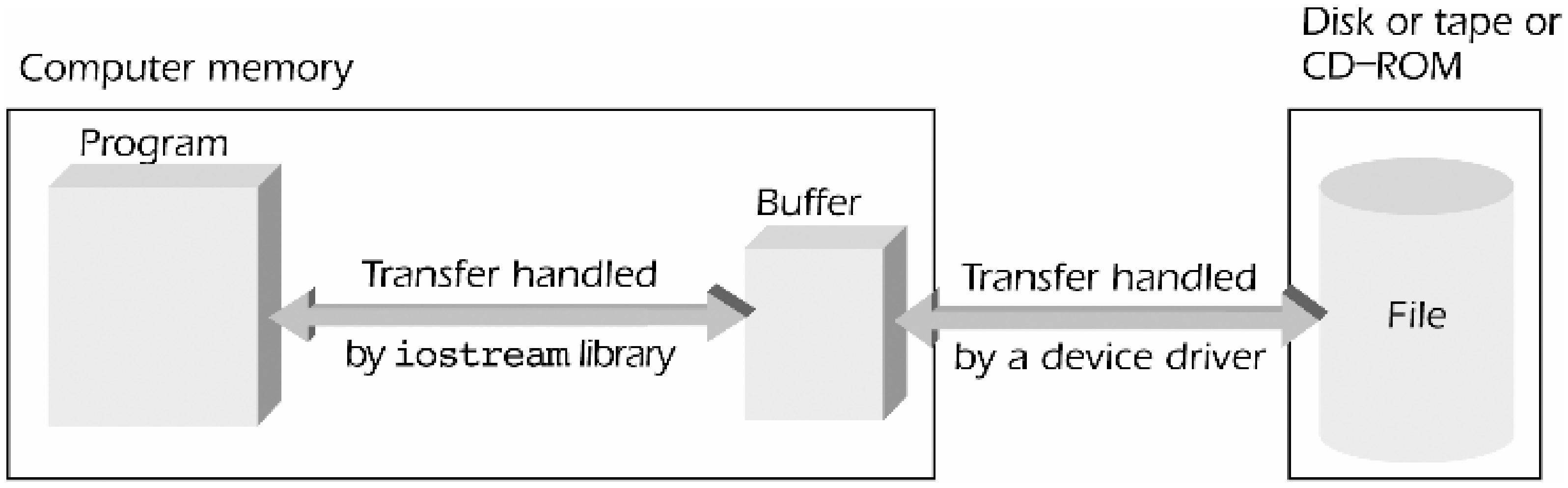
- Data file: any collection of data stored together in an external storage medium under a common name
- **open()** method of a file stream connects a data file’s external name to the internal stream object name
- File can be opened in input mode to read its contents, or in output mode to write into it
- Opening a file that already exists in output mode will cause its contents to be overwritten

Summary (continued)

- Standard stream objects **`cin`**, **`cout`**, **`cerr`** are automatically declared and opened when a program is run
- Random file access is done using the **`seekg()`** and **`tellg()`** methods for input file streams, and the **`seekp()`** and **`tellp()`** methods for output file streams

Appendix: The `iostream` Class Library

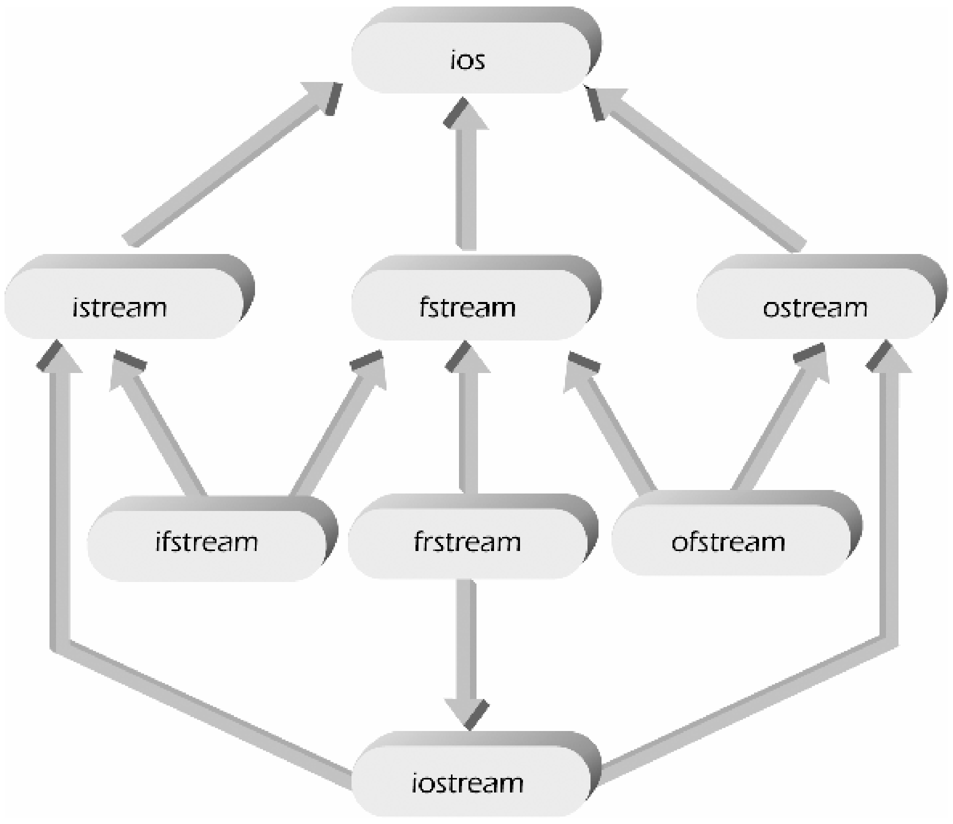
- File stream transfer mechanism involves an intermediate file buffer in computer memory
- File buffer is managed by a device driver
- **Device driver**: part of O/S code for accessing a hardware device

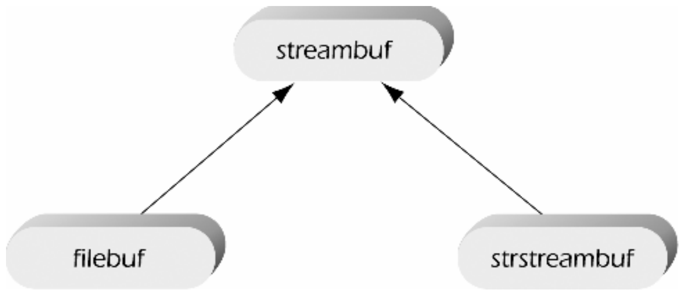


Appendix: The **`iostream`** Class Library (continued)

- **`iostream`** class library contains two primary base classes:
  - **`streambuf`** class: provides the file buffer and routines for transferring binary data
  - **`ios`** class: contains a pointer to the file buffers and routines for transferring text data

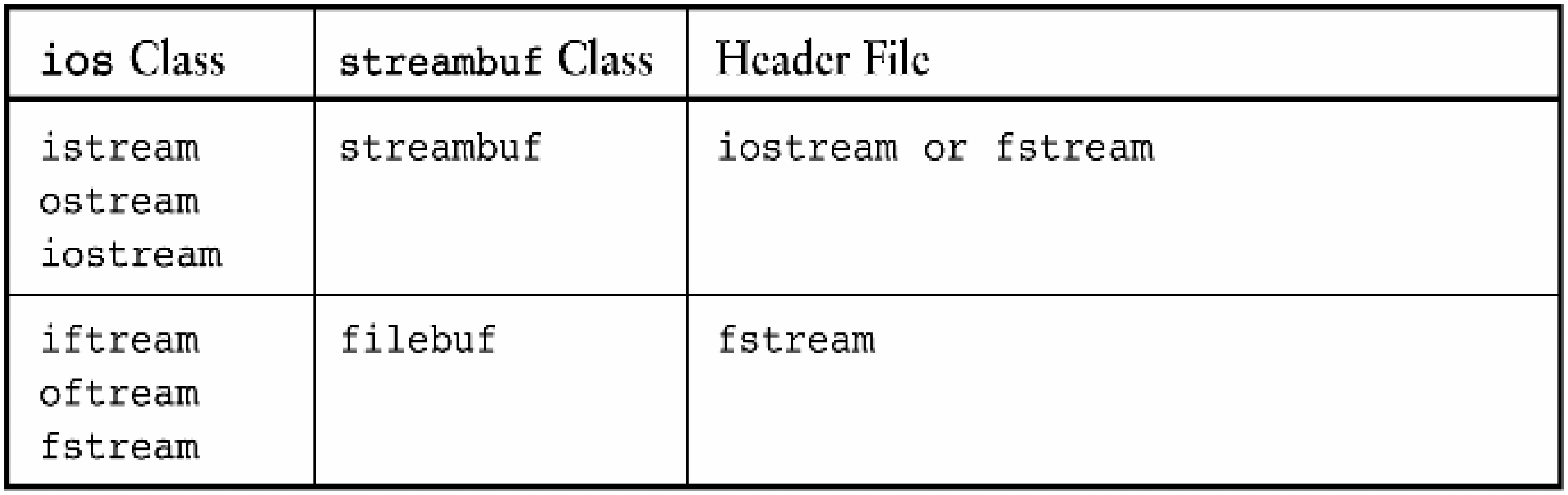
Appendix: The **`iostream`** Class Library (continued)





Appendix: The **`iostream`** Class Library (continued)

Correspondence between **`ios`** and **`streambuf`** Classes



`ios` Class	`streambuf` Class	Header File

`istream`
`ostream`
`iostream`
	`streambuf`	`iostream or fstream`

`iftream`
`oftream`
`fstream`
	`filebuf`	`fstream`

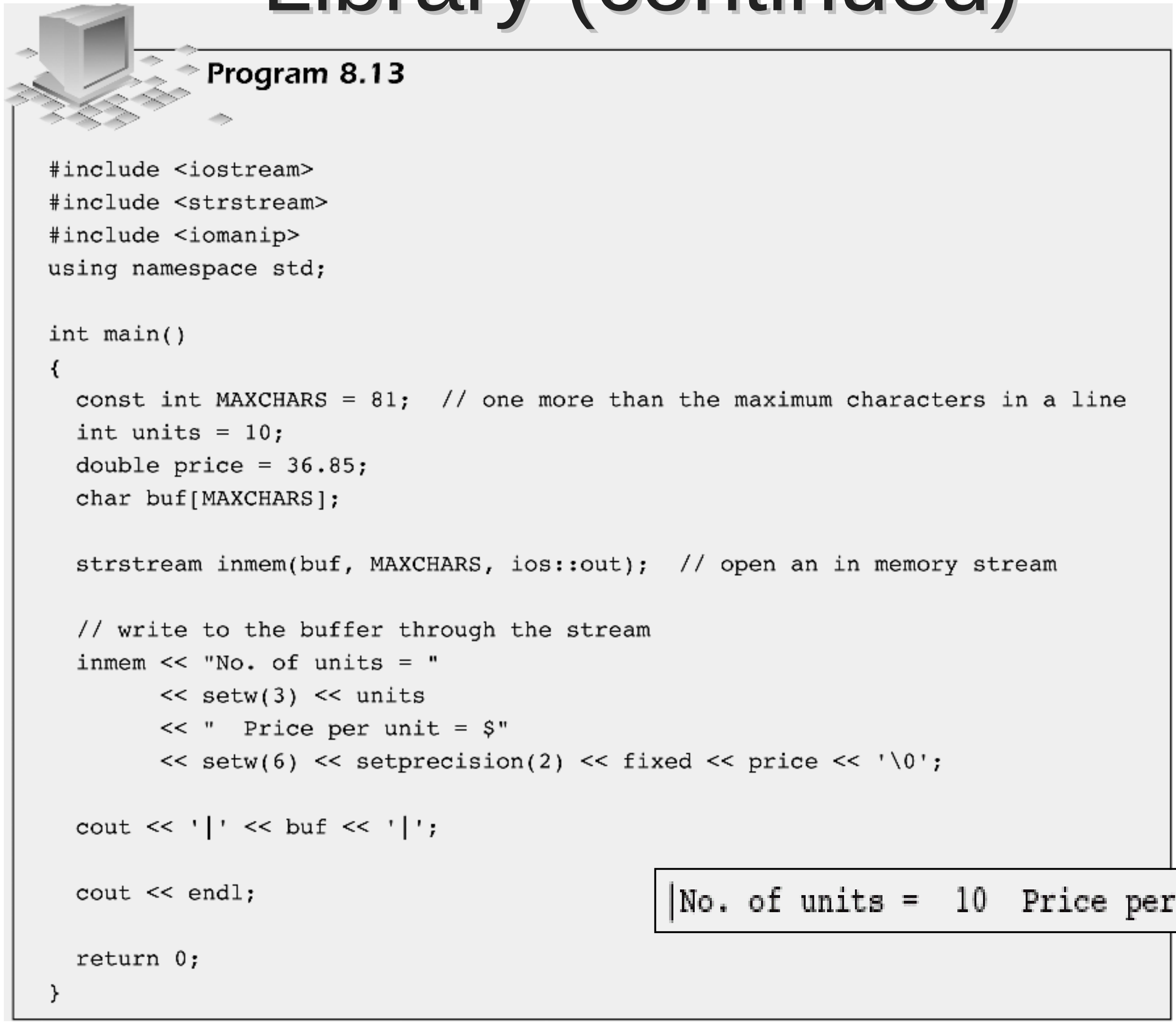
Appendix: The **`iostream`** Class Library (continued)

- **`ios`** class also contains **`strstream`** class for writing and reading strings to and from in-memory-defined streams
- Usually used to “assemble” a string from smaller pieces before writing it to **`cout`** or to a file

Example:

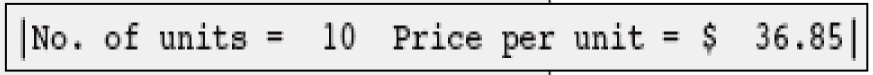
```
strstream inmem(buf, 72, ios::out);
```

Appendix: The **`iostream`** Class Library (continued)



Program 8.13

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

int main()
{
  const int MAXCHARS = 81;  // one more than the maximum characters in a line
  int units = 10;
  double price = 36.85;
  char buf[MAXCHARS];

  strstream inmem(buf, MAXCHARS, ios::out);  // open an in memory stream

  // write to the buffer through the stream
  inmem << "No. of units = "
        << setw(3) << units
        << "  Price per unit = $"
        << setw(6) << setprecision(2) << fixed << price << '\0';

  cout << '|' << buf << '|';

  cout << endl;

  return 0;
}
```



```
|No. of units =  10  Price per unit = $  36.85|
```