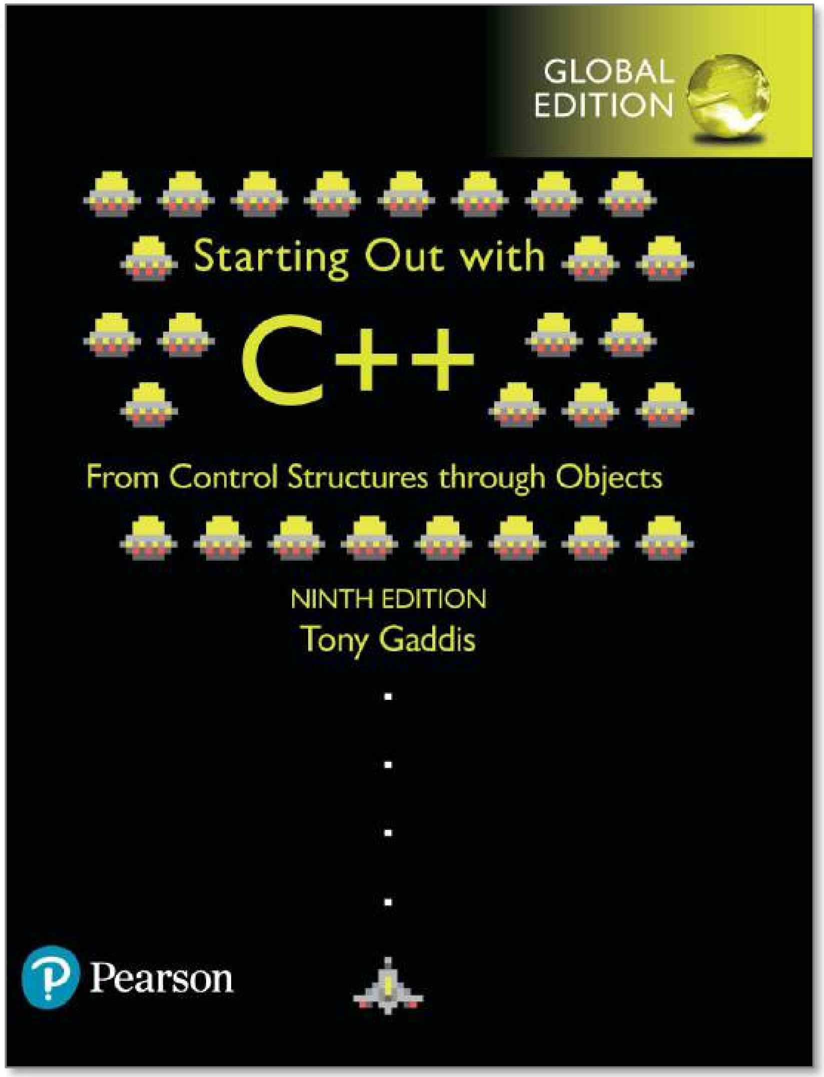
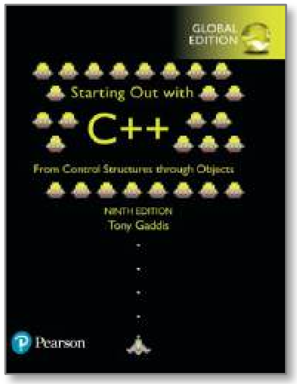
Chapter 5:

Files





5.11

Using Files for Data Storage

Using Files for Data Storage

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
  - *Open* the file
  - *Use* the file (read from, write to, or both)
  - *Close* the file

Files: What is Needed

- Use `fstream` header file for file access
- File stream types:

  `ifstream` for input from a file

  `ofstream` for output to a file

  `fstream` for input from or output to a file
- Define file stream objects:

```
ifstream infile;
ofstream outfile;
```

Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)
- Use the `open` member function:

```
    infile.open("inventory.dat");
    outfile.open("report.txt");
```

- Filename may include drive, path info.
- Output file will be created if necessary; existing file will be erased first
- Input file must exist for `open` to work

Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
if (!infile)
{
  cout << "File open failure!";
}
```

- Can also use the `fail` member function

Using Files

- Can use output file object and << to send data to a file:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
infile >> qtyInStock >>
qtyOnOrder;
```

Using Loops to Process Files

- The stream extraction operator `>>` returns `true` when a value was successfully read, `false` otherwise

- Can be tested in a `while` loop to continue execution as long as values are read from the file:

```
    while (inputFile >> number) ...
```

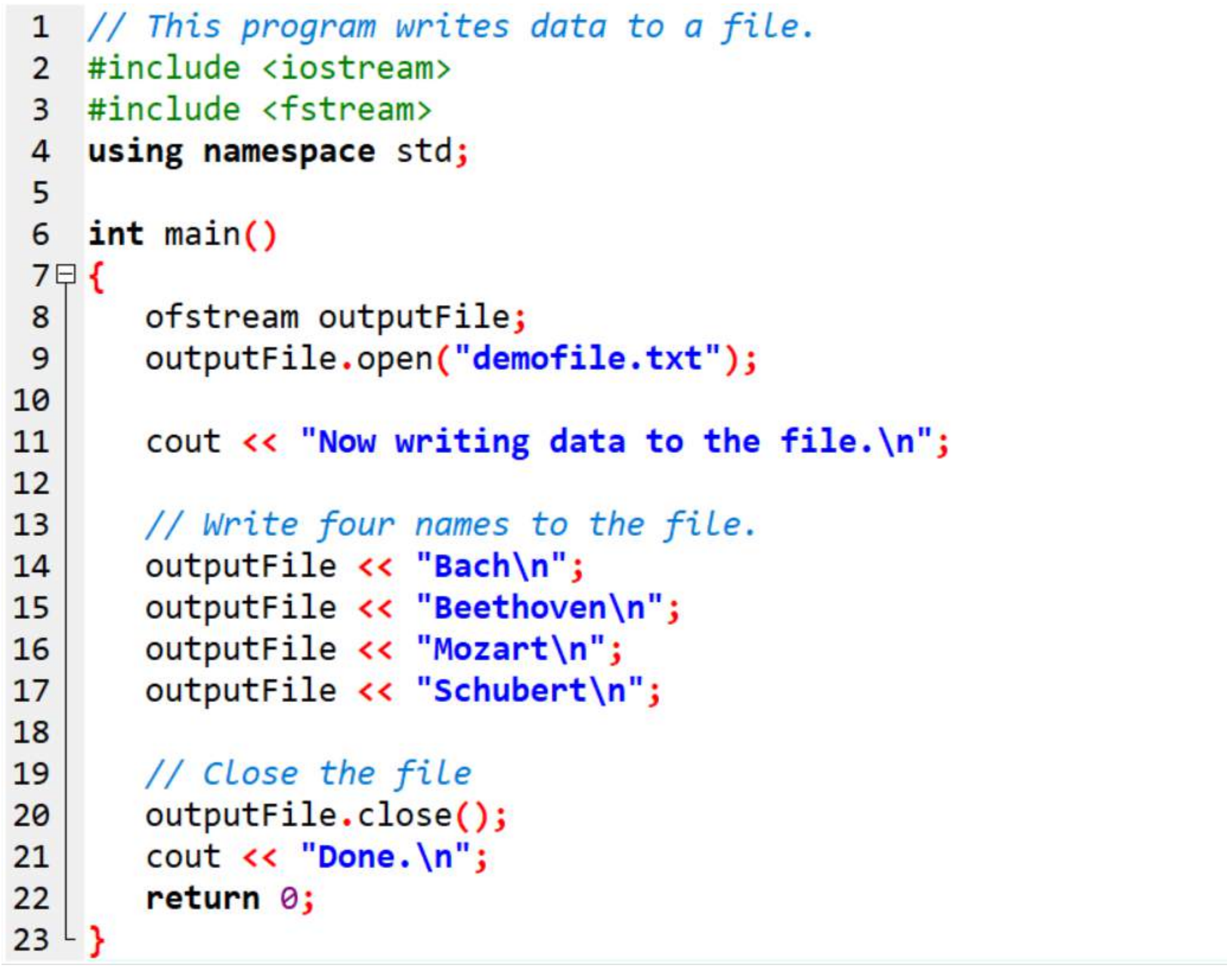
Closing Files

- Use the `close` member function:

```
infile.close();
outfile.close();
```

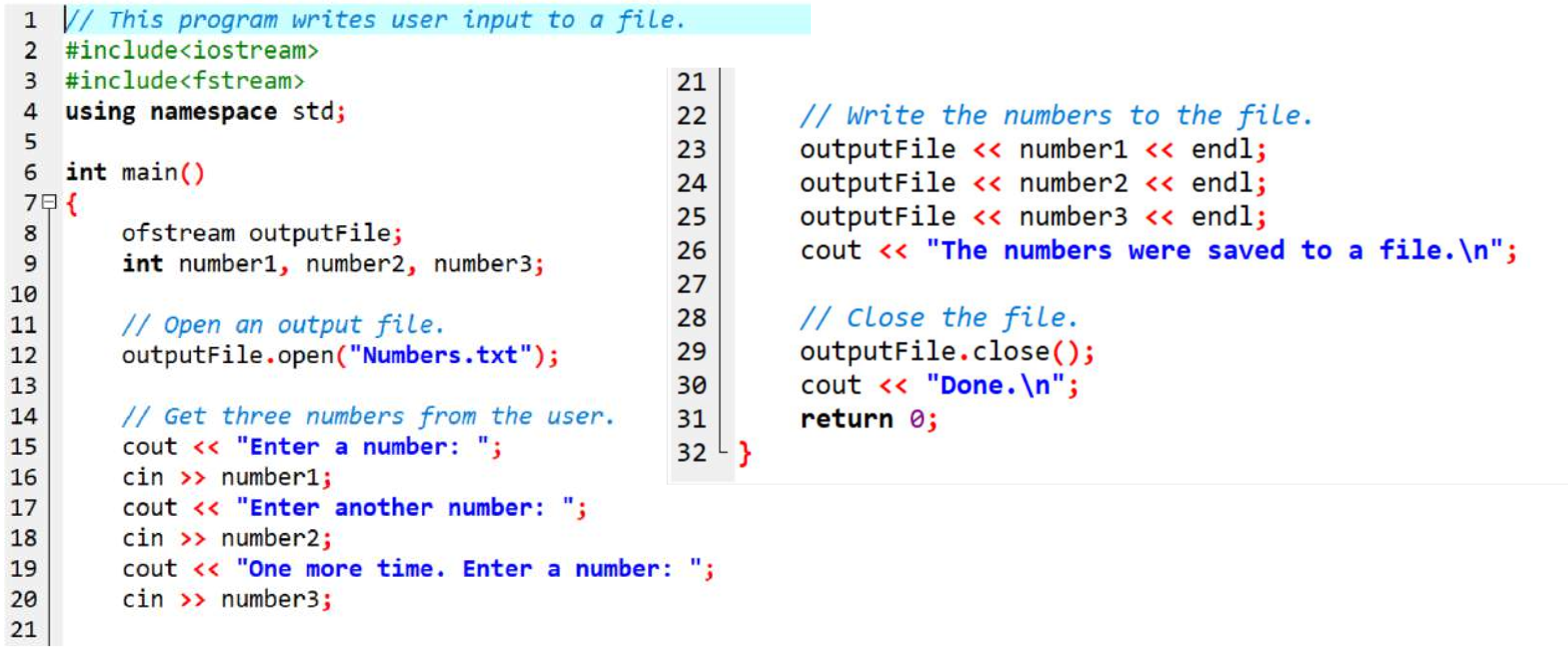
- Don’t wait for operating system to close files at program end:
  - may be limit on number of open files
  - may be buffered output data waiting to send to file

Writes data to a file



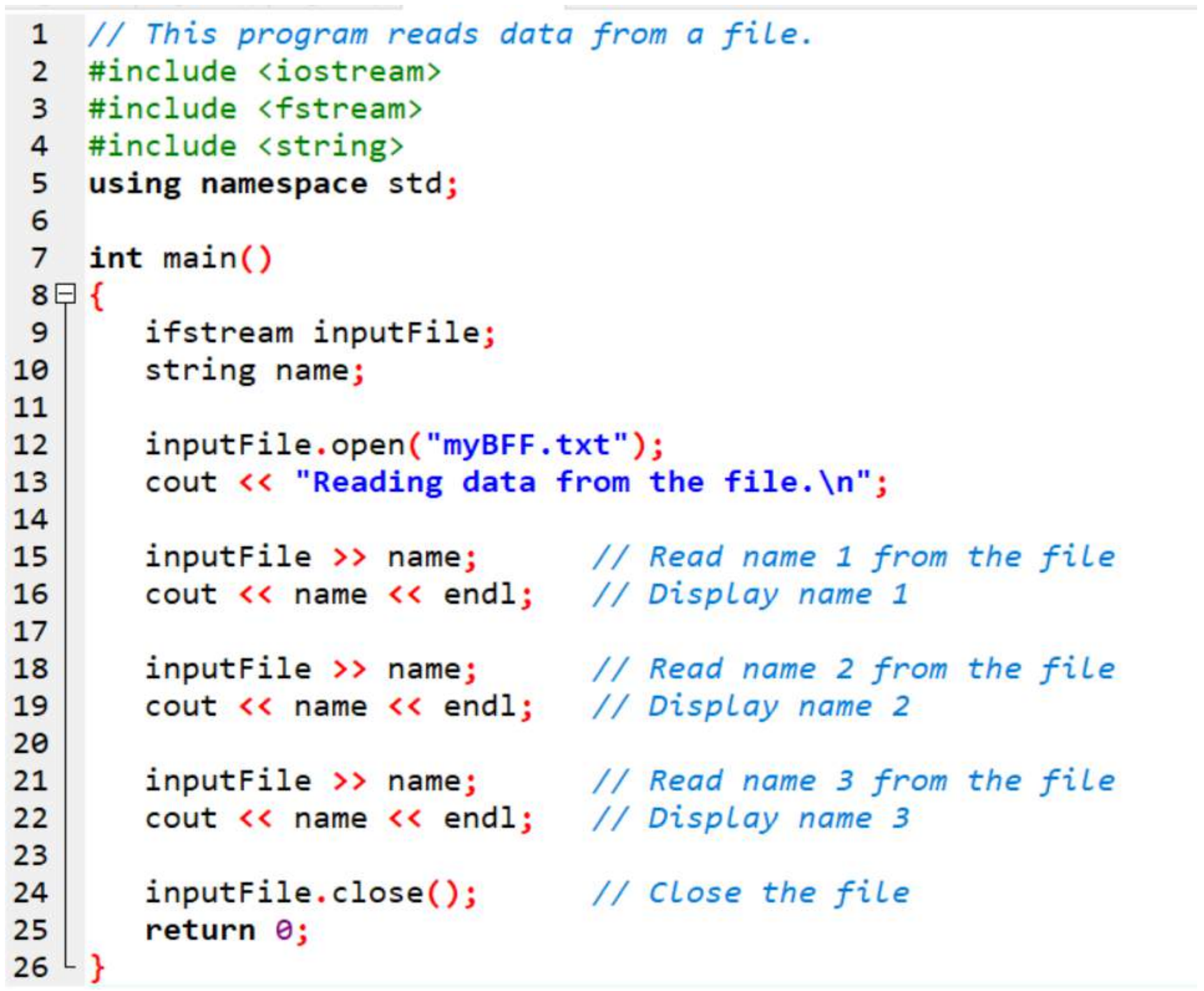
```cpp
// This program writes data to a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream outputFile;
    outputFile.open("demofile.txt");

    cout << "Now writing data to the file.\n";

    // Write four names to the file.
    outputFile << "Bach\n";
    outputFile << "Beethoven\n";
    outputFile << "Mozart\n";
    outputFile << "Schubert\n";

    // Close the file
    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Writes user input to a file



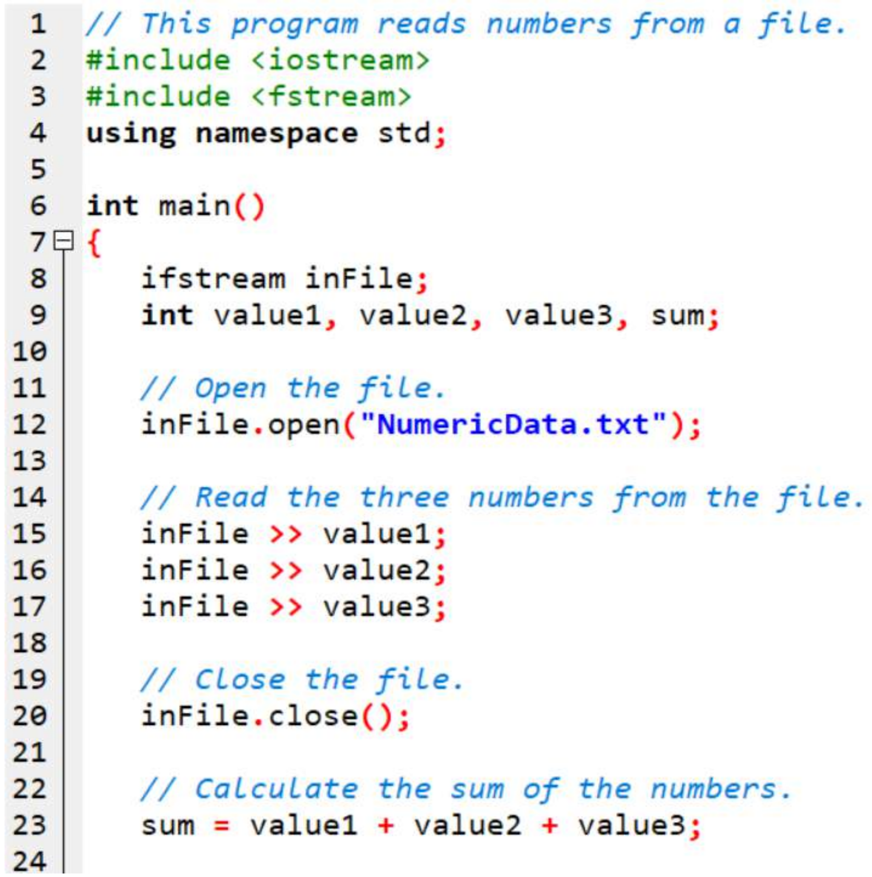
```cpp
// This program writes user input to a file.
#include<iostream>
#include<fstream>
using namespace std;

int main()
{
    ofstream outputFile;
    int number1, number2, number3;

    // Open an output file.
    outputFile.open("Numbers.txt");

    // Get three numbers from the user.
    cout << "Enter a number: ";
    cin >> number1;
    cout << "Enter another number: ";
    cin >> number2;
    cout << "One more time. Enter a number: ";
    cin >> number3;

    // Write the numbers to the file.
    outputFile << number1 << endl;
    outputFile << number2 << endl;
    outputFile << number3 << endl;
    cout << "The numbers were saved to a file.\n";

    // Close the file.
    outputFile.close();
    cout << "Done.\n";
    return 0;
}
```

Read data from a File

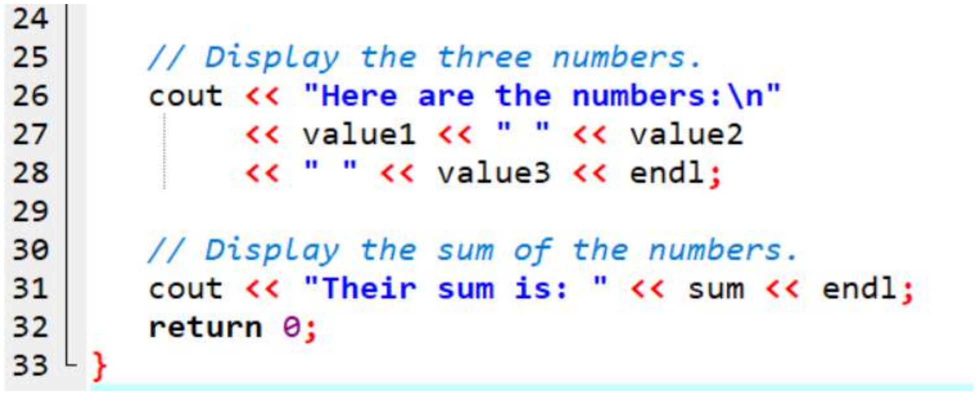


```cpp
// This program reads data from a file.
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
    ifstream inputFile;
    string name;

    inputFile.open("myBFF.txt");
    cout << "Reading data from the file.\n";

    inputFile >> name;        // Read name 1 from the file
    cout << name << endl;     // Display name 1

    inputFile >> name;        // Read name 2 from the file
    cout << name << endl;     // Display name 2

    inputFile >> name;        // Read name 3 from the file
    cout << name << endl;     // Display name 3

    inputFile.close();        // Close the file
    return 0;
}
```

Read numbers from a File

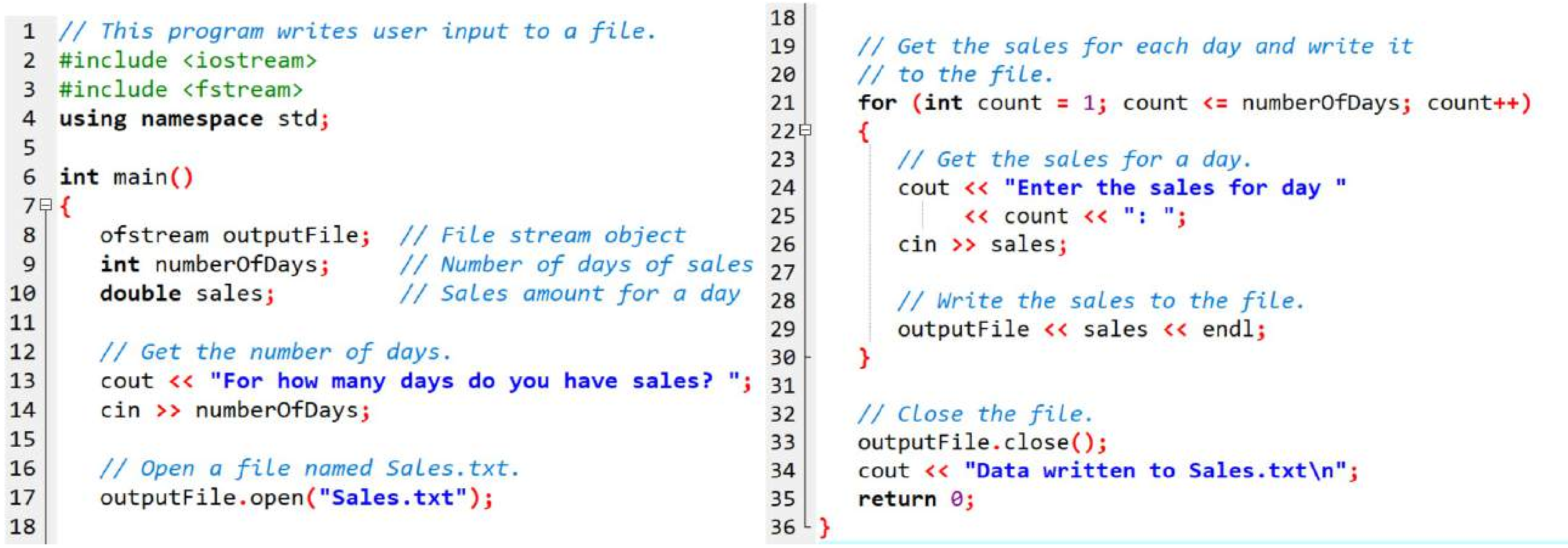


```cpp
// This program reads numbers from a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream inFile;
    int value1, value2, value3, sum;

    // Open the file.
    inFile.open("NumericData.txt");

    // Read the three numbers from the file.
    inFile >> value1;
    inFile >> value2;
    inFile >> value3;

    // Close the file.
    inFile.close();

    // Calculate the sum of the numbers.
    sum = value1 + value2 + value3;

```



```cpp

    // Display the three numbers.
    cout << "Here are the numbers:\n"
         << value1 << " " << value2
         << " " << value3 << endl;

    // Display the sum of the numbers.
    cout << "Their sum is: " << sum << endl;
    return 0;
}
```

Using Loops to Process Files



```cpp
// This program writes user input to a file.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream outputFile;    // File stream object
    int numberOfDays;       // Number of days of sales
    double sales;           // Sales amount for a day

    // Get the number of days.
    cout << "For how many days do you have sales? ";
    cin >> numberOfDays;

    // Open a file named Sales.txt.
    outputFile.open("Sales.txt");

    // Get the sales for each day and write it
    // to the file.
    for (int count = 1; count <= numberOfDays; count++)
    {
        // Get the sales for a day.
        cout << "Enter the sales for day "
             << count << ": ";
        cin >> sales;

        // Write the sales to the file.
        outputFile << sales << endl;
    }

    // Close the file.
    outputFile.close();
    cout << "Data written to Sales.txt\n";
    return 0;
}
```

Letting the User Specify a Filename

- In many cases, you will want the user to specify the name of a file for the program to open.
- In C++ 11, you can pass a `string` object as an argument to a file stream object’s `open` member function.

Letting the User Specify a Filename in Program 5-24

Program 5-24

```cpp
// This program lets the user enter a filename.
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
    ifstream inputFile;
    string filename;
    int number;

    // Get the filename from the user.
    cout << "Enter the filename: ";
    cin >> filename;

    // Open the file.
    inputFile.open(filename);

    // If the file successfully opened, process it.
    if (inputFile)
```

Continued…

Letting the User Specify a Filename in Program 5-24

```cpp
    {
        // Read the numbers from the file and
        // display them.
        while (inputFile >> number)
        {
            cout << number << endl;
        }

        // Close the file.
        inputFile.close();
    }
    else
    {
        // Display an error message.
        cout << "Error opening the file.\n";
    }
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter the filename: ListOfNumbers.txt [Enter]
100
200
300
400
500
600
700
```

Using the `c_str` Member Function in Older Versions of C++

- Prior to C++ 11, the `open` member function requires that you pass the name of the file as a null-terminated string, which is also known as a *C-string*.
- *String literals are stored* in memory as null-terminated C-strings, but *string objects* are **not**.

Using the `c_str` Member Function in Older Versions of C++

- `string` objects have a member function named `c_str`
  - It returns the contents of the object formatted as a null-terminated C-string.
  - Here is the general format of how you call the `c_str` function:

```
stringObject.c_str()
```

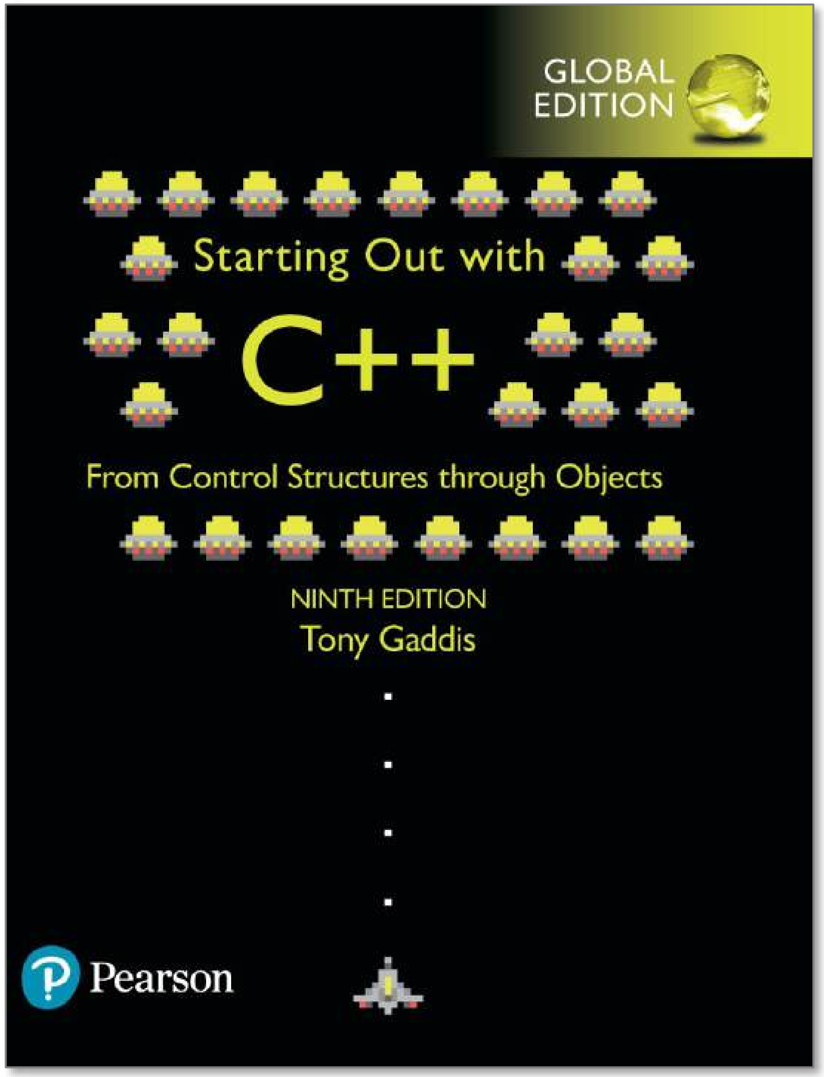
  - Line 18 in Program 5-24 could be rewritten in the following manner:

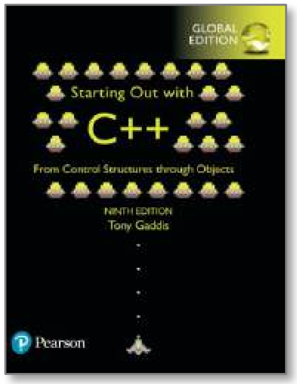
```
inputFile.open(filename.c_str());
```

Chapter 12:

Advanced

File Operations





12.1

File Operations

File Operations

- File: a set of data stored on a computer, often on a disk drive
- Programs can read from, write to files
- Used in many applications:
  - Word processing
  - Databases
  - Spreadsheets
  - Compilers

Using Files

1. Requires `fstream` header file
   - use `ifstream` data type for input files
   - use `ofstream` data type for output files
   - use `fstream` data type for both input, output files
2. Can use `>>`, `<<` to read from, write to a file
3. Can use `eof` member function to test for end of input file

`fstream` Object

- `fstream` object can be used for either input or output
- Must specify mode on the `open` statement
- Sample modes:

  `ios::in` – input

  `ios::out` – output
- Can be combined on `open` call:

```
dFile.open("class.txt", ios::in | ios::out);
```

File Access Flags

Table 12-2

File Access Flag	Meaning
`ios::app`	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
`ios::ate`	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
`ios::binary`	Binary mode. When a file is opened in binary mode, data is written to or read from it in pure binary format. (The default mode is text.)
`ios::in`	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
`ios::out`	Output mode. Data will be written to the file. By default, the file's contents will be deleted if it already exists.
`ios::trunc`	If the file already exists, its contents will be deleted (truncated). This is the default mode used by `ios::out`.

Using Files - Example

```
// copy 10 numbers between files
// open the files
fstream infile("input.txt", ios::in);
fstream outfile("output.txt", ios::out);
int num;
for (int i = 1; i <= 10; i++)
{
   infile >> num;     // use the files
   outfile << num;
}
infile.close();       // close the files
outfile.close();
```

Default File Open Modes

- `ifstream`:
  - open for input only
  - file cannot be written to
  - `open` fails if file does not exist
- `ofstream`:
  - open for output only
  - file cannot be read from
  - file created if no file exists
  - file contents erased if file exists

More File Open Details

- Can use filename, flags in definition:

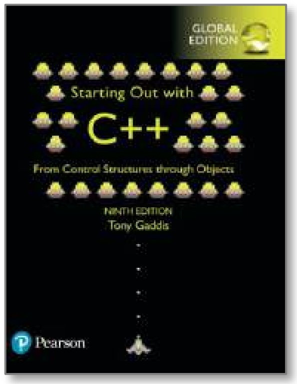
```
ifstream gradeList("grades.txt");
```

- File stream object set to `0` (`false`) if open failed:

```
if (!gradeList) ...
```

- Can also check `fail` member function to detect file open error:

```
if (gradeList.fail()) ...
```



12.2

File Output Formatting

File Output Formatting

- Use the same techniques with file stream objects as with `cout`: `showpoint`, `setw(x)`, `showprecision(x)`, etc.
- Requires `iomanip` to use manipulators

Program 12-3

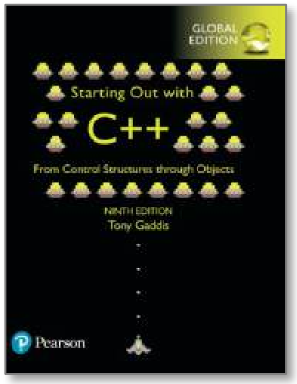
```
// This program uses the setprecision and fixed
// manipulators to format file output.
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

int main()
{
   fstream dataFile;
   double num = 17.816392;

   dataFile.open("numfile.txt", ios::out);   // Open in output mode

   dataFile << fixed;              // Format for fixed-point notation
   dataFile << num << endl;        // Write the number

   dataFile << setprecision(4);   // Format for 4 decimal places
   dataFile << num << endl;        // Write the number

   dataFile << setprecision(3);   // Format for 3 decimal places
   dataFile << num << endl;        // Write the number

```

Program 12-3 (Continued)

```
    dataFile << setprecision(2);   // Format for 2 decimal places
    dataFile << num << endl;       // Write the number

    dataFile << setprecision(1);   // Format for 1 decimal place
    dataFile << num << endl;       // Write the number

    cout << "Done.\n";
    dataFile.close();              // Close the file
    return 0;
}
```

Contents of File `numfile.txt`

```
17.816392
17.8164
17.816
17.82
17.8
```



12.4

More Detailed Error Testing

More Detailed Error Testing

- Can examine error state bits to determine stream status
- Bits tested/cleared by stream member functions

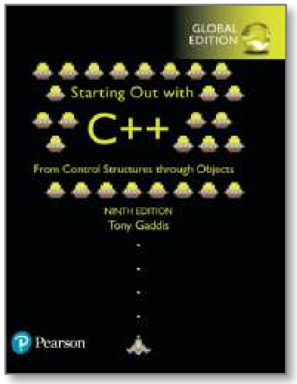
`ios::eofbit`	set when end of file detected
`ios::failbit`	set when operation failed
`ios::hardfail`	set when error occurred and no recovery
`ios::badbit`	set when invalid operation attempted
`ios::goodbit`	set when no other bits are set

Member Functions / Flags

`eof()`	true if `eofbit` set, false otherwise
`fail()`	true if `failbit` or `hardfail` set, false otherwise
`bad()`	true if `badbit` set, false otherwise
`good()`	true if `goodbit` set, false otherwise
`clear()`	clear all flags (no arguments), or clear a specific flag

From Program 12-6

```
void showState(fstream &file)
{
   cout << "File Status:\n";
   cout << "  eof bit: " << file.eof() << endl;
   cout << "  fail bit: " << file.fail() << endl;
   cout << "  bad bit: " << file.bad() << endl;
   cout << "  good bit: " << file.good() << endl;
   file.clear();  // Clear any bad bits
}
```



12.5

Member Functions for Reading and Writing Files

Member Functions for Reading and Writing Files

- Functions that may be used for input with whitespace, to perform single character I/O, or to return to the beginning of an input file
- Member functions:
  - `getline`: reads input including whitespace
  - `get`: reads a single character
  - `put`: writes a single character

The `getline` Function

- Three arguments:
  - Name of a file stream object
  - Name of a `string` object
  - Delimiter character of your choice
  - Examples, using the file stream object `myFile`, and the `string` objects `name` and `address`:

    ```
    getline(myFile, name);
    getline(myFile, address, '\t');
    ```

  - If left out, `'\n'` is default for third argument

Program 12-8

```
// This program uses the getline function to read a line of
// data from the file.
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
   string input;     // To hold file input
   fstream nameFile; // File stream object

   // Open the file in input mode.
   nameFile.open("murphy.txt", ios::in);

   // If the file was successfully opened, continue.
   if (nameFile)
   {
       // Read an item from the file.
       getline(nameFile, input);

```

```
        // While the last read operation
        // was successful, continue.
        while (nameFile)
        {
           // Display the last item read.
           cout << input << endl;

           // Read the next item.
           getline(nameFile, input);
        }

        // Close the file.
        nameFile.close();
    }
    else
    {
       cout << "ERROR: Cannot open file.\n";
    }
    return 0;
}
```

Program Output

```
Jayne Murphy
47 Jones Circle
Almond, NC 28702
```

Single Character I/O

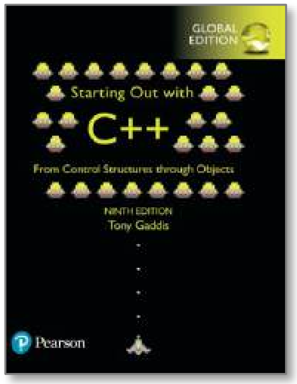
- `get`: read a single character from a file

```
char letterGrade;
gradeFile.get(letterGrade);
```

  Will read any character, including whitespace

- `put`: write a single character to a file

```
reportFile.put(letterGrade);
```



12.6

Working with Multiple Files

Working with Multiple Files

- Can have more than one file open at a time in a program
- Files may be open for input or output
- Need to define file stream object for each file

Program 12-12

```
// This program demonstrates reading from one file and writing
// to a second file.
#include <iostream>
#include <fstream>
#include <string>
#include <cctype> // Needed for the toupper function.
using namespace std;

int main()
{
   string fileName;      // To hold the file name
   char ch;              // To hold a character
   ifstream inFile;      // Input file

   // Open a file for output.
   ofstream outFile("out.txt");

   // Get the input file name.
   cout << "Enter a file name: ";
   cin >> fileName;

   // Open the file for input.
   file.open(name, ios::in);

   // If the input file opened successfully, continue.
   if (inFile)
   {
      // Read a char from file 1.
      inFile.get(ch);

      // While the last read operation was
      // successful, continue.
      while (inFile)
      {
         // Write uppercase char to file 2.
         outFile.put(toupper(ch));

         // Read another char from file 1.
         inFile.get(ch);
      }

      // Close the two files.
      inFile.close();
      outFile.close();
      cout << "File conversion done.\n";
   }
   else
      cout << "Cannot open " << fileName << endl;
   return 0;
}
```

Program Screen Output with Example Input Shown in Bold

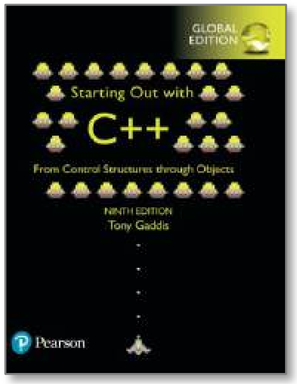
```
Enter a file name: **hownow.txt [Enter]**
File conversion done.
```

Contents of `hownow.txt`

```
how now brown cow.
How Now?
```

Resulting Contents of `out.txt`

```
HOW NOW BROWN COW.
HOW NOW?
```



12.9

Random-Access Files

Random-Access Files

- Sequential access: start at beginning of file and go through data in file, in order, to end
  - to access 100th entry in file, go through 99 preceding entries first
- Random access: access data in a file in any order
  - can access 100th entry directly

Random Access Member Functions

- `seekg` (seek get): used with files open for input
- `seekp` (seek put): used with files open for output
- Used to go to a specific position in a file

Random Access Member Functions

- `seekg,seekp` arguments:

  offset: number of bytes, as a `long`

  mode flag: starting point to compute offset

- Examples:

```
inData.seekg(25L, ios::beg);
// set read position at 26th byte
// from beginning of file
outData.seekp(-10L, ios::cur);
// set write position 10 bytes
// before current position
```

Important Note on Random Access

- If `eof` is true, it must be cleared before `seekg` or `seekp`:

```
gradeFile.clear();
gradeFile.seekg(0L, ios::beg);
// go to the beginning of the file
```

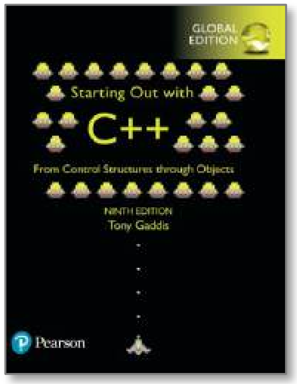
Random Access Information

- `tellg` member function: return current byte position in input file

```
    long int whereAmI;
    whereAmI = inData.tellg();
```

- `tellp` member function: return current byte position in output file

```
    whereAmI = outData.tellp();
```



12.10

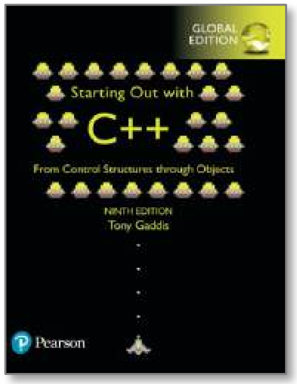
Opening a File for Both Input and Output

Opening a File for Both Input and Output

- File can be open for input and output simultaneously
- Supports updating a file:
  - read data from file into memory
  - update data
  - write data back to file
- Use `fstream` for file object definition:

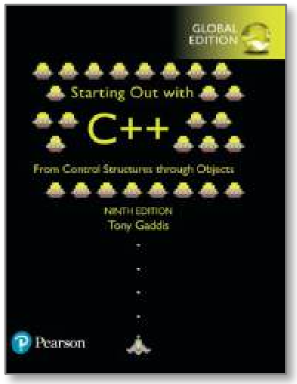
```
fstream gradeList("grades.dat",
          ios::in | ios::out);
```

- Can also use `ios::binary` flag for binary data


Starting Out with C++ From Control Structures through Objects NINTH EDITION Tony Gaddis Pearson GLOBAL EDITION

KIV Section

The following has not yet been covered because the prerequisite chapter will come in the second half semester



12.3

Passing File Stream Objects to Functions

Passing File Stream Objects to Functions

- It is very useful to pass file stream objects to functions
- Be sure to always pass file stream objects by reference

Program 12-5

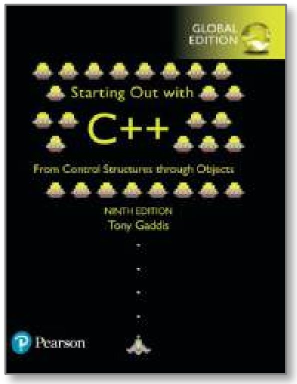
```
// This program demonstrates how file stream objects may
// be passed by reference to functions.
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

// Function prototypes
bool openFileIn(fstream &, string);
void showContents(fstream &);

int main()
{
    fstream dataFile;

    if (openFileIn(dataFile, "demofile.txt"))
    {
        cout << "File opened successfully.\n";
        cout << "Now reading data from the file.\n\n";
        showContents(dataFile);
        dataFile.close();
        cout << "\nDone.\n";
    }
```

```
   else
       cout << "File open error!" << endl;

   return 0;
}

//*************************************************************
// Definition of function openFileIn. Accepts a reference   *
// to an fstream object as an argument. The file is opened   *
// for input. The function returns true upon success, false *
// upon failure.                                             *
//*************************************************************

bool openFileIn(fstream &file, string name)
{
   file.open(name, ios::in);
   if (file.fail())
      return false;
   else
      return true;
}

//*************************************************************
// Definition of function showContents. Accepts an fstream   *
// reference as its argument. Uses a loop to read each name *
// from the file and displays it on the screen.              *
//*************************************************************
```

```cpp

void showContents(fstream &file)
{
   string line;

   while (file >> line)
   {
      cout << line << endl;
   }
}
```

Program Output

```
File opened successfully.
Now reading data from the file.

Jones
Smith
Willis
Davis

Done.
```



12.7

Binary Files

Binary Files

- Binary file contains unformatted, non-ASCII data
- Indicate by using `binary` flag on open:

```
inFile.open("nums.dat", ios::in |
ios::binary);
```

Binary Files

- Use `read` and `write` instead of `<<, >>`

```
char ch;
// read in a letter from file
inFile.read(&ch, sizeof(ch));
```

address of where to put the data being read in. The `read` function expects to read `char`s

how many bytes to read from the file

```
// send a character to a file
outFile.write(&ch, sizeof(ch));
```

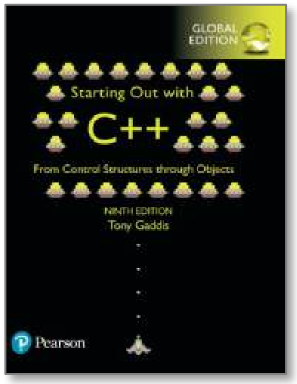
Binary Files

- To `read`, `write` non-character data, must use a typecast operator to treat the address of the data as a character address

```
int num;
// read in a binary number from a file
inFile.read(reinterpret_cast<char *>&num,
                            sizeof(num));
```

treat the address of `num` as the address of a `char`

```
// send a binary value to a file
outf.write(reinterpret_cast<char *>&num,
                            sizeof(num));
```



12.8

Creating Records with Structures

Creating Records with Structures

- Can write structures to, read structures from files

- To work with structures and files,
  - use `ios::binary` file flag upon open
  - use `read`, `write` member functions

Creating Records with Structures

```
struct TestScore
{
  int studentId;
  double score;
  char grade;
};
TestScore oneTest;
...
// write out oneTest to a file
gradeFile.write(reinterpret_cast<char *>
  (&oneTest), sizeof(oneTest));
```