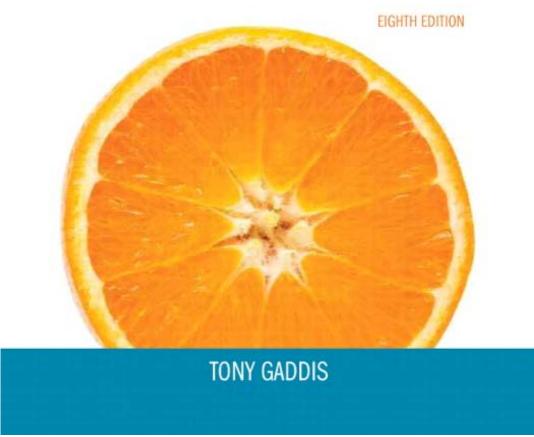
Chapter 12:

Advanced File Operations







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();
Addison-Wesley
is an imprint of
```



Default File Open Modes

- o 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:

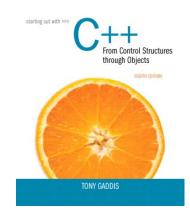
```
if (!gradeList) ...
```

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



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





File Output Formatting





File Output Formatting

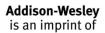
Use the same techniques with file stream objects as with cout: showpoint, setw(x), showprecision(x), etc.

Requires iomanip to use manipulators



Program 12-3

```
// This program uses the setprecision and fixed
   // manipulators to format file output.
   #include <iostream>
 4 #include <iomanip>
   #include <fstream>
   using namespace std;
   int main()
 9
10
      fstream dataFile;
11
      double num = 17.816392;
12
13
      dataFile.open("numfile.txt", ios::out); // Open in output mode
14
      dataFile << fixed; // Format for fixed-point notation
15
      dataFile << num << endl; // Write the number
16
17
      dataFile << setprecision(4); // Format for 4 decimal places
18
19
      dataFile << num << endl; // Write the number
20
21
      dataFile << setprecision(3); // Format for 3 decimal places
22
      dataFile << num << endl; // Write the number
23
```





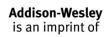
Program 12-3 (Continued)

```
24
       dataFile << setprecision(2); // Format for 2 decimal places
       dataFile << num << endl; // Write the number
25
26
27
       dataFile << setprecision(1); // Format for 1 decimal place
       dataFile << num << endl; // Write the number
28
29
3.0
       cout << "Done.\n";
                                     // Close the file
31
       dataFile.close();
32
       return 0;
3.3
Contents of File numfile.txt
17.816392
17.8164
17.816
17.82
17.8
```





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 &);
10
11
    int main()
12
13
    {
14
        fstream dataFile;
15
         if (openFileIn(dataFile, "demofile.txt"))
16
17
             cout << "File opened successfully.\n";
18
             cout << "Now reading data from the file.\n\n";
19
20
             showContents(dataFile);
             dataFile.close();
21
22
             cout << "\nDone.\n";</pre>
23
```



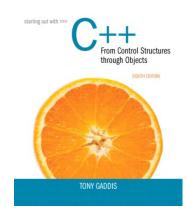
```
24
    else
25
        cout << "File open error!" << endl;
26
27
    return 0;
28 }
29
31 // Definition of function openFileIn. Accepts a reference
32 // to an fstream object as an argument. The file is opened
33 // for input. The function returns true upon success, false *
34 // upon failure.
35 //********************
36
37
  bool openFileIn(fstream &file, string name)
38 {
39
    file.open(name, ios::in);
40
    if (file.fail())
41
       return false;
42
    else
43
       return true;
44 }
45
46 //********************
47 // Definition of function showContents. Accepts an fstream
48 // reference as its argument. Uses a loop to read each name *
49 // from the file and displays it on the screen.
50 //********************
```



```
51
52 void showContents(fstream &file)
53 {
 54
       string line;
55
 56
       while (file >> line)
57
58
          cout << line << endl;
 59
       }
60 }
Program Output
File opened successfully.
Now reading data from the file.
Jones
Smith
Willis
Davis
Done.
```







More Detailed Error Testing

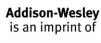




More Detailed Error Testing

- 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

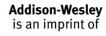
eof()	true if eofbit set, false otherwise
fail()	true if failbit or hardfail set, false otherwise
bad()	true if badbit set, false otherwise
good()	true if goodbit set, false otherwise
clear()	clear all flags (no arguments), or clear a specific flag





From Program 12-6

```
68
    void showState(fstream &file)
69
       cout << "File Status:\n";
7.0
71
       cout << " eof bit: " << file.eof() << endl;
       cout << " fail bit: " << file.fail() << endl;
72
       cout << " bad bit: " << file.bad() << endl;
73
74
       cout << " good bit: " << file.good() << endl;
7.5
       file.clear(); // Clear any bad bits
76
```







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

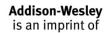
```
1 // This program uses the getline function to read a line of
 2 // data from the file.
 3 #include <iostream>
 4 #include <fstream>
 5 #include <string>
 6 using namespace std;
   int main()
      string input; // To hold file input
10
      fstream nameFile; // File stream object
11
12
13
      // Open the file in input mode.
14
      nameFile.open("murphy.txt", ios::in);
15
16
      // If the file was successfully opened, continue.
17
      if (nameFile)
18
      {
19
          // Read an item from the file.
20
          getline(nameFile, input);
21
```



```
22
          // While the last read operation
23
          // was successful, continue.
          while (nameFile)
24
25
          {
26
             // Display the last item read.
27
             cout << input << endl;
28
29
             // Read the next item.
30
             getline(nameFile, input);
31
          }
32
33
          // Close the file.
34
          nameFile.close();
35
36
      else
37
38
         cout << "ERROR: Cannot open file.\n";
39
40
      return 0;
41 }
```

Program Output

Jayne Murphy 47 Jones Circle Almond, NC 28702



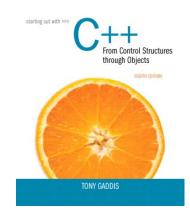


Single Character I/O

oget: 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);





Working with Multiple Files





Working with Multiple Files

Can have more than 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

```
1 // This program demonstrates reading from one file and writing
 2 // to a second file.
 3 #include <iostream>
 4 #include <fstream>
 5 #include <string>
 6 #include <cctype> // Needed for the toupper function.
 7 using namespace std;
 9 int main()
10 {
11
      string fileName; // To hold the file name
12
      char ch;
                  // To hold a character
      ifstream inFile; // Input file
13
14
15
     // Open a file for output.
      ofstream outFile("out.txt");
16
17
18
      // Get the input file name.
19
      cout << "Enter a file name: ";
20
      cin >> fileName;
21
22
      // Open the file for input.
23
     file.open(name, ios::in);
24
25
      // If the input file opened successfully, continue.
```

```
26
      if (inFile)
27
      {
28
           // Read a char from file 1.
29
           inFile.get(ch);
30
31
          // While the last read operation was
          // successful, continue.
32
33
          while (inFile)
34
           {
35
              // Write uppercase char to file 2.
36
              outFile.put(toupper(ch));
37
38
              // Read another char from file 1.
39
              inFile.get(ch);
40
41
42
          // Close the two files.
43
           inFile.close();
          outFile.close();
44
45
          cout << "File conversion done.\n";
46
      }
47
      else
48
         cout << "Cannot open " << fileName << endl;</pre>
49
      return 0;
50 }
```



Program Screen Output with Example Input Shown in Bold

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

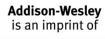
Contents of hownow.txt

how now brown cow. How Now?

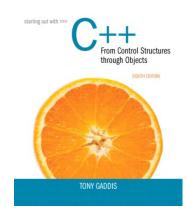
Resulting Contents of out.txt

HOW NOW BROWN COW.

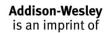
HOW NOW?







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. how many bytes to The read function expects read from the file to read chars // 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



is an imprint of



12.8

Creating Records with Structures





Creating Records with Structures

Can write structures to, read structures from files

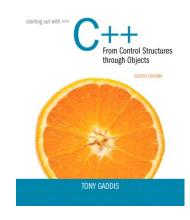
- To work with structures and files,
 - ouse ios::binary file flag upon open
 - ouse read, write member functions



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</pre>
 (&oneTest), sizeof(oneTest));
```





12.9

Random-Access Files





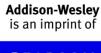
Random-Access Files

- 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

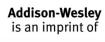


Random Access Member Functions

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

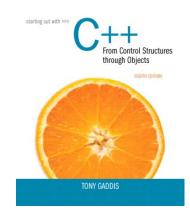
tellg member function: return current byte position in input file

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

tellp member function: return current byte position in output file

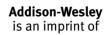
```
whereAmI = outData.tellp();
```





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:

Can also use ios::binary flag for binary data

