Input/output Processing – Using file data

CONCEPT: Program input can be read from a file and program output can be written to a file
Setting up your program for file input/output is normally a five-step process:
1. Request the preprocessor to include the header file fstream. The file fstream contains all the definitions necessary for file operations. It is included with the following statement. #include <fstream>
2. Use declaration statements to declare the file streams we will use.

The next step in setting up a program to perform file I0 is to define one or more file stream objects. They are called stream objects because a file can be thought of as a stream of data. File stream objects work very much like cin and cout objects. A stream of data can be sent to cout, which causes values to be displayed on the screen. A stream of data can be read from the keyboard by cin and stored in variables. Likewise, streams of data can be sent to a file stream object, which writes the data to a file. Data that is read from a file flows from a file stream object into other variables.

The fstream header file contains definitions for the data types ofstream, ifstream, and fstream. Before a C++ program can work with a file, it must define an object of one of these data types. The object will be associated with an actual file stored on some secondary storage medium, and the operations that may be performed on that file depend on which of these three data types you pick for the file stream object. Table below lists and describes file stream data types.

File Stream Data Types

ofstream - Output file stream. This data type can be used to open output files and write data to them. If the file does not yet exist, the open operation will automatically create it. If the file already exists, the open operation will destroy it and create a new, empty file of the same name in its place. With the ofstream data type, data may only be copied from variables to the file, but not vice versa.

ifstream - Input file stream. This data type can be used to open existing input files and read data from them into memory. With the ifstream data type, data may only be copied from the file into variables, not but vice versa.

fstream - File stream. This data type can be used to open files, write data to them, and read data from them. With the fstream data type, data may be copied from variables into a file, or from a file into variables.

Here are example statements that define ofstream and ifstream objects:
ofstream outputFile;
ifstream inputFile;

These two file stream objects, outputFile and inputFile, could have been named using any legal C++ identifier names. However, as is good programming practice, they were given descriptive names that clarify their use. The outputFile object is of the of stream type, so data can be written to any file associated with it. The inputFile object is of the ifstream type, so data can be read from any file it is associated with.

3. Prepare each file for reading or writing by using a function named open

Before data can be written to or read from a file, the file must be opened. Outside of the C++ program, a file is identified by its name. Inside a C++ program, however, a file is identified by a stream object. The object and the file name are linked when the file is opened.

Files can be opened through the open function that exists for file stream objects. Assume inputFile is an if stream object, defined as ifstream inputFile;

The following statement uses inputFile to open a file named customer.dat:
inputFile.open(”customer.dat’); // Open an input file

The argument to the open function in this statement is the name of the file. This links the file customer.dat with the stream object inputFile. Until inputFile is associated with another file, any operations performed with it will be carried out on the file customer.dat.

It is also possible to define a file stream object and open a file all in one statement. Here is an example:
ifstream inputFile(“customer.dat’);

In our example open statement, the customer.dat file was specified as a simple file name, with no path given. When no path is given, the program will look for the file in a default directory. If the program is being executed from the command line, the default directory is the current directory. If

the program is being executed from within an integrated development environment (IDE), the default directory depends on the particular compiler you are using. If the file you want to open is not in the default directory, you will need to specify its location as well as its name.

Run-Time Input of File Names (Program InputFIleName.cpp)

The open function associated with the ifstream data type requires an argument that specifies the name of the actual data file. By using a literal string, as in the preceding example, the file name is fixed at compile time. Therefore, the program works for only this particular file.

We often want to make a program more flexible by allowing the file name to be determined at run time. A common technique is to prompt the user for the name of the file, read the user's response into a variable, and pass the variable as an argument to the open function. In principle, the following code should accomplish what we want. Unfortunately, the compiler does not allow it.

```
ifstream inFile;
string fileName;

cout << "Enter the input file name:
cin >> fileName;

inFile .open (fileName) // Compile-time error
```

The problem is that the open function does not expect an argument of type string. Instead, it expects a C string. A C string (so named because it originated in the C language, the forerunner of C++) is a limited form of string whose properties is discussed a later in this chapter. A literal string, such as "datafile.dat", happens to be a C string and thus is acceptable as an argument to the open function.

To make this code work correctly, we need to convert a string variable to a C string. The string data type provides a value-returning function named c_str that is applied to a string variable as follows:

```
fileName.c_str()  //(see runtime_file_name.cpp for an example)
```

4. Specify the name of the file stream in each input or output statement.

Writing information to a file (see Program write_to_file.cpp)

You already know how to use the stream insertion operator (<<) with the cout object to write information to the screen. It can also be used with file stream objects to write information to a file. Assuming outputFile is a file stream object, the following statement demonstrates using the << operator to write a string to a file:

```
outputFile << "C++ programming is fun";
```

As you can see, the statement looks like a cout statement, except the file stream object name replaces cout. Here is a statement that writes both a string and the contents of a variable to a file:

```
outputFile << '"Price: "<< Price;
```

This statement writes the stream of information to outputFile exactly as cout would write it to the screen.

Output files have two common uses. The first is to hold computer program output. When a program writes output to a file, rather than to the computer screen, it can be saved for later viewing and printing. This is often done with reports that would not easily fit on one screen or that must be printed more than once. The second common use of output files is to hold data generated by one program that will later be read in by another program. Because these are simple text files, they can be viewed and printed with any text editor.

Reading information from a file (see Program read_from_file.cpp)

The >> operator not only reads user input from the cm object, but it can also be used to read data from a file. Assuming inFile is a file stream object, the following statement shows the >> operator reading data from the file into the variable name: inFile >> name;

Data is read from files in a sequential manner. When a file is first opened, the file stream object's read position is at the first byte of the file. The first read operation extracts data starting at the first byte. As data is read, the file stream object's read position advances through the file. When the >> operator extracts data from a file, it expects to read pieces of data that are separated by whitespace characters (spaces, tabs, or newlines).

A data file can be created with any text editor (such as Windows WordPad or NotePad). This is often done when a program has a substantial amount of input. Placing the data in a text file ahead of time and then having the program read the data from the file saves the user having to enter the data when the program is run.

5. Close the file (see Program close_file.cpp)

The opposite of opening a file is closing it. Although a program's files are automatically closed when the program shuts down, it is a good programming practice to write statements that explicitly close them. Here are two reasons a program should close files when it is finished using them:

• Most operating systems temporarily store information in a file buffer before it is written to a file. A file buffer is a small holding section of memory that file-bound information is first written to. When the buffer is filled, all the information stored there is written to the file. This technique improves the system's performance. Closing a file causes any unsaved information that may still be held in a buffer to be saved to its file. This means the information will be in the file if you need to read it later in the same program.

• Some operating systems limit the number of files that may be open at one time. When a program keeps open files that are no longer being used, it uses more of the operating system's resources than necessary.

Calling the file stream object's close function closes a file. Here is an example: outputFile.close();

```
//*************************************************************
// Program InputFileName.cpp demonstrates run-time input of file names.
//*************************************************************
#include <fstream>
#include <iostream>
#include <string>
#include<cstdlib>

using namespace std;
int main ()
{
  ifstream inFile;
  string   fileName;
  string inputString;
  cout << "Enter the input file name: ";
  cin >> fileName;
  inFile.open(fileName.c_str());

  getline(inFile, inputString);
  cout << "First line of file: " << inputString << endl;
system("PAUSE");
  return 0;
}

//use sample data file testData.in

Oh! Its a beautiful day in the neighborhood
said Mr. Rogers
```

Source code samples featured:

write_to_file.cpp and demofile.txt - Uses the << operator to write information to a file.
read_from_file.cpp and demofile.txt - uses the >> operator to read information from a file.
read_close_file.cpp and dimensions.txt - uses the >> operator to read rectangle dimensions from a file. It demonstrates that, as with cin, more than one value can be read in from a file with a single statement.

```
// Program IODemo demonstrates how to use files

#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   cout  << fixed  << showpoint;

   float val1, val2, val3, val4;   // declares 4 variables
   ifstream inData;                // declares input stream
   ofstream outData;               // declares output stream

   inData.open("Data.In");
   // binds program variable inData to file "Data.In"
   outData.open("Data.Out");
    // binds program variable outData to file "Data.Out"

   inData  >> val1 >> val2 >> val3 >> val4;            // inputs 4 values
   outData  << val4  << endl;
   outData  << val3  << endl;
   outData  << val2  << endl;
   outData  << val1  << endl;   // outputs 4 values
   return 0;
}

/*contents of Data.In

5.5
6.6
7.7
8.8
*/


/* FOR LECTURE DEMONSTRATION - Using Files - Storing and Retrieving Numbers

For this assignment you will write two programs:

Program 1: Write a program that asks the user to enter five floating-point numbers. The program should create a file and save all five numbers to the file.

Program 2: Write a program that opens the file created by Program 1, reads the five numbers, and displays them. The program should also calculate and display the sum of the five numbers. */
```