

Chapter 24 – Input from Text Files

1. Input from a Text File

Input from a text file is very similar to input from the keyboard with the object *cin*. Input from a text file is accomplished via a programmer declared object of class *ifstream*. In order to make class *ifstream* available for use in a program, the file *fstream* must be included.

2. ifstream Objects

An *ifstream* object is declared in the same manner as *ofstream* objects, except that the *open* method uses the mode *ios::in*.

```
ifstream object-name;  
object-name.open(string-of-path-to-use-for-input, ios::in);
```

where *object-name* is the *ifstream* object *string-of-path-to-use-for-input* is a valid path in the operating system of the computer or workstation in a string literal or variable
ios::in is the mode for input from a file

The *ifstream* object is then used for input from the text file in the same manner as the *cin* object is used for input from the keyboard. When an *ifstream* object is declared, the associated file is opened. For the declaration of an *ifstream* object to be effective, the file must exist before the declaration. The file will be read in order starting from its beginning.

Text files accessed via an *ifstream* object should be closed when the program has completed reading from the file. This will happen automatically when a program terminates. However, it is a good idea to close files no longer needed to reduce demand on limited system resources, especially in larger programs that may manipulate many files. The programmer may close a file with the *close()* member function of the *ifstream* object. This has the same syntax and usage as the *close* member function of *ofstream* class.

The one potential problem with opening a class for input that must be addressed before producing an example program is whether or not the file exists. It does not matter that a file does not exist when opening a file for output because the file will be created. Input, however, will open the file and attempt to read from it whether or not the file exists at all! To get around this possible trap, an if statement can be constructed by using the *ifstream* object as a Boolean that returns true if the file exists or false if it does not exist. (If the file does not exist, the statement return can be used to halt processing in the function.) Here is an example of a simple test to see if an input file exists.

```

    if (!inFile) {
        cout << "No Such File!" << endl;
        return;
    }

```

Here is a brief example of input from a text file located at the path “a:\mystuff.txt” with the contents:

Elton Thibadeaux 234-5678

```

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

void main( )
{
    char phone[15], fname[15], lname[15];
    ifstream inFile;
    inFile.open("a:\mystuff.txt", ios::in);    // change to match an actual
                                              // file

    if (!inFile) {
        cout << "No Such File!" << endl;
        return;
    }
    inFile >> fname >> lname >> phone;
    inFile.close( );
    cout << "The phone number of " << fname << " " << lname
         << " is " << phone << endl;
}

```

When run, this program outputs the following to the monitor screen:

The phone number of Elton Thibadeaux is 234-5678

3. Reading Until the End-of-File

While the exact length or number of data elements in a file may be known to the programmer, it is most often the case that the number of data elements in a file will be unknown to the programmer. The number of data elements may even change from time to time between runs of the program that is to read the file. For example, a word processing program must read in files of various lengths. While the length of the file (in bytes) can be discovered easily enough, the length of the file and the text contained in it do not correspond exactly. For instance, a word processing file contains information about page setup, fonts used, character style and size, and many other bits of information that will only be included when the word processor user elects to place these options in a file. Such

a file with a lot of text but few stylistic instructions may be relatively small, while a file with little text but a lot of stylistic instructions may be relatively large. The only solution for the programmer of the word processor is to write a program that will read whatever data file is selected all the way to the end of the file.

All object of class *ifstream* can be used as a Boolean test to determine if data read from a file is valid (*true* or *1*, not the end-of-file) or invalid (*false* or *0*, the end-of-file has been reached).

All objects of class *ifstream* also have a member function *eof()*, which is a member function that returns *false* or *0* when the end-of-file has not been reached and returns *true* or *1* when the end-of-file has been reached.

Here is an example of a program that reads an unknown number of integers from a data file and sums them up. Note that there are two inputs from file. One before the loop, which is necessary for the *eof()* function or *ifstream* object to work properly in the Boolean expression of the while statement, and one as the last instruction of the loop, which insures that the last thing read – the end-of-file – is not included in the total of the counter.

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

void main()
{
    long sum=0, number;
    ifstream numSource;
    numSource.open("a:\\number.fil", ios::in);

    if (!numSource) {
        cout << "No such file!" << endl;
        return;
    }

    numSource >> number;
    while (numSource) // or while (!numSource.eof())
    {
        sum += number;
        numSource >> number;
    }
    numSource.close();
    cout << "Total : " << sum << endl;
}
```

If the file “a:\number.fil” has the contents

```
1 2 3
4 5
```

The output of the program will be

Total: 15

If the file “a:\number.fil” has the contents

```
1 2 3 4 5 6 7 8
9 10
```

The output of the file will be

Total: 55

Programming Assignment 24.1

Using the file created in programming assignment 23.1, write a program to read your name from the file and display the information on the monitor screen.

Programming Assignment 24.2

Using the file created in programming assignment 23.2, write a program to read the entire contents of the file and display it on the monitor screen.

Programming Assignment 24.3

Using the file of heights created in programming assignment 23.3, write a program that sums all of the heights in the file and displays the average height. *Note: This will require a counter for the number of heights and the sum of the heights.*

Programming Assignment 24.4

Using `getline()`, write a program that will display the contents of any text file path input by the user.

Programming Assignment 24.5

Alter the program from programming assignment 23.4 so that the information about CDs (certificates of deposit) that have been stored in files may be retrieved and the information displayed.

Programming Assignment 24.6

Create and use a class that keeps track of high and low temperatures, storing them in a file or files. The class should provide a means to add daily information (*see appending to a file below*), to display the information, and to display the current average hi and low.

Appending to a File

To append to a existing file without destroying the file's contents, the file must be opened with the *open()* member function using the mode *ios::app*.

```
ofstream fileObject;  
fileObject.open(string-of-path-to-file, ios::app);
```

The file is closed as normal.