

# File Input Output

In this week

- C++ input/output streams
- Input/output file streaming objects
- Working with input/output file streaming objects
- Specifying the path of files
- End of file marker

# C++ Input/Output Streams

- So far we have been working with console input (keyboard) and output (screen) streams
- This was achieved with the **#include <iostream>** directive
- In addition to console input output, C++ supports input/output file streams
- This is supported by **#include <fstream>** directive

# C++ Input/Output Streaming Objects

- C++ input/output streaming objects are special objects in C++ that deliver inputs to our programs and outputs from our programs
- **cout** is an output streaming object while **cin** is an input streaming object
- These objects require the streaming in **>>** and streaming out **<<** operators in order to direct input and output
- In order to work with input/output files, we need C++ streaming objects that get connected to physical files in the hard drive of a computer

# C++ Input Output Streams

- The input/output file streamings are defined in the std namespace inside the **fstream** include directive
- Therefore we need to include the **fstream** directive and also use the namespace **std**
- Once input/output file streaming object is defined, it is used exactly as **cout** and **cin** streaming objects together with the input **>>** and output **<<** streaming operators

# File Output Stream

- In order to create an output file streaming object, we construct an object of type **ofstream**
- **ofstream** stands for output file stream
- **Syntax**

**ofstream fout;**

- Once an output file streaming object is constructed, we connect it to a file as follows

**fout.open("ExampleOutputFile.txt");**

# File Output Stream

- Construction of output file streaming object and connecting it to a file may also be done together in one C++ statement as

**ofstream fout("ExampleOutputFile.txt");**

- When we connect output file streaming object to a file in the hard drive, then a **new empty file will be created automatically**
- **If the same file name already exists, then it will be overwritten (i.e. the existing file will be deleted and a new empty file with the same file name will be created)**
- Moreover, in Visual C++ 2010 Express Edition, the file will be created in the folder where the **cpp** file is located

# File Output Stream

- Once the output file streaming object is constructed and connected to a file on the hard drive, then it will be used together with the output stream operator << in exactly the same way as cout object
- For cout, the output goes to the screen; while for fout the output goes to the file
- Once we have finished using an output file streaming object, we need to close it (disconnect it from the physical file in the hard drive)
- In order to close an output file stream object, we do  
**fout.close();**
- Once an output file streaming object is closed, it can no more be used to write output to a file because it is disconnected from the file; unless it is opened again and connected to a file.

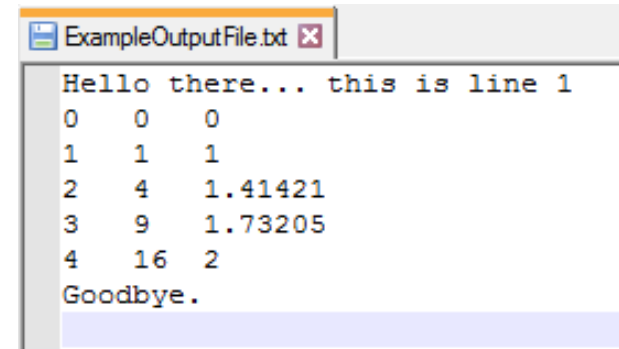
# File Output Stream

- The following program demonstrates, the construction, usage and closing of output file streaming object. The created text file is also shown.

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

int main()
{
    ofstream fout("ExampleOutputFile.txt");
    fout << "Hello there... this is line 1" << endl;
    for (int k = 0; k < 5; k++)
    {
        fout << k << "\t" << pow(k, 2.0) << "\t" << sqrt(1.0*k) << endl;
    }
    fout << "Goodbye." << endl;
    fout.close();

    system("Pause");
    return 0;
}
```



ExampleOutputFile.txt

```
Hello there... this is line 1
0    0    0
1    1    1
2    4    1.41421
3    9    1.73205
4    16    2
Goodbye.
```



# File Output Stream

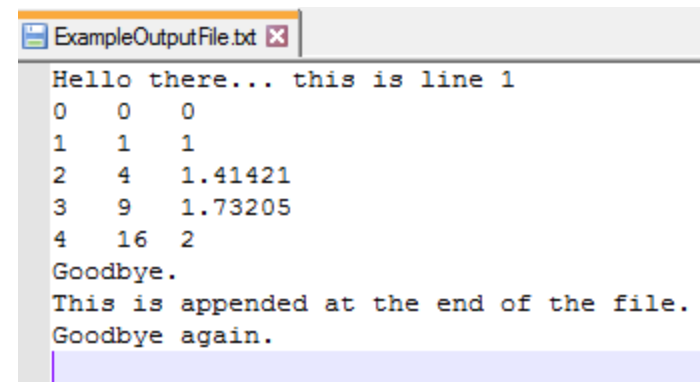
- Like we said earlier, when we open an output file streaming object; then by default the file is created from scratch (that is if the same file name exists, it will be overwritten)
- Sometimes, we may need to open an existing file but we don't want to overwrite it; instead we would like to append data to it
- This is done by informing C++ our intention to append when we open the file as shown below

# File Output Stream

- In order to append new information to an existing file, we do **ofstream fout("ExampleOutputFile.txt", ios::app);**
- Now what you write to fout will be appended to the contents of the file.
- The following program appends some data to an existing file. Assume the file exists in the same folder where the cpp file is located. The existing file is appended as expected as shown below.

```
int main()
{
    ofstream fout("ExampleOutputFile.txt", ios::app);
    fout << "This is appended at the end of the file." << endl;
    fout << "Goodbye again." << endl;
    fout.close();

    system("Pause");
    return 0;
}
```



```
ExampleOutputFile.txt
Hello there... this is line 1
0  0  0
1  1  1
2  4  1.41421
3  9  1.73205
4  16  2
Goodbye.
This is appended at the end of the file.
Goodbye again.
```

# File input stream

- We can also open an input stream object and read data from a file to our program

- **Syntax**

- ifstream fin;**

- fin.open("ExampleInputFile.txt");**

- OR combining these two statements in one

- ifstream fin("ExampleInputFile.txt");**

- Then we can use fin together with the input streaming operator >> just like we do with cin

# File input stream

- For example, suppose there is a file named `ExampleInputFile.txt` in the same folder where our `cpp` file is located
- Moreover suppose the file contains five integer numbers separated either by spaces, tabs or each number on its own line
- Then the following program will read the five numbers one by one using a loop and compute the minimum number and display it onto the screen

# File input stream

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    ifstream fin("ExampleInputFile.txt");
    int x, m;
    fin >> m;    //Read the first number
    for (int i = 0; i < 4; i++)
    {
        fin >> x;
        if (x < m)
            m = x;
    }
    fin.close();
    cout << "The minimum number in the file is " << m << endl;

    system("Pause");
    return 0;
}
```

ExampleInputFile.txt

4  
2  
-5  
7  
3

# File input stream

- Extra care needs to be followed when working with file input stream objects
  - Firstly we should be certain the physical file from which to read exists
  - Secondly, we should know the type of the data in the file in order to read to proper data type variables
  - Thirdly, we should ascertain there is enough data in the file to read

# Specifying Path of Files

- Sometimes we may need to give an explicit path to a file for input or output streaming objects
- This is done by giving the full path when we construct or open the file streaming object
- Example

```
ofstream fout("C:/Users/Me/Desktop/ExampleOutputFile.txt");
```

- This creates the file on the desktop

# Checking IO stream

- It is important we check if an input/output file stream object is opened correctly before using it
- Sometimes, output stream object may fail because we don't have the privilege to create a new file on the computer we are using
- An input file stream may fail because there is no file with the given file name in the folder we are specifying
- In order to check the success of input/output file stream objects we proceed as follows:

```
ofstream fout("Test.txt")
if (fout.fail())
    //File opening failed
else
    //File opening succeeded
```

```
ifstream fin("Test.txt")
if (fin.fail())
    //File opening failed
else
    //File opening succeeded
```



# Checking IO stream

- In the following example, we give a complete example to create a new file, write data on it and then read the data from the file and print it to the screen
- We also demonstrate the full path of input/output files specification in order to read from or write to any folder of our choice
- We also check the success of our file input output stream objects before we make use of the objects

# File IO Example

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    //Open output file stream
    ofstream fout("C:/Users/user_name/Desktop/TestOutputFile.txt");
    if (fout.fail())
    {
        cout << "Output file stream failed." << endl;
        return 0;
    }
    //If we reach here the output file is opened successfully
    for (int i = 0; i < 10; i++)
    {
        fout << rand() % 20 << " ";
    }
    fout.close();
}
```

# File IO Example

```
//Now open the file for reading
ifstream fin("C:/Users/user_name/Desktop/TestOutputFile.txt");
if (fin.fail())
{
    cout << "Input file stream failed." << endl;
    return 0;
}
//If we reach here the input file is opened successfully
int x;
for (int i = 0; i < 10; i++)
{
    fin >> x;
    cout << x << " ";
}
cout << endl;
fin.close();

system("Pause");
return 0;
}
```

# More General Complete Example

- Create a text file named **ClassList.txt** manually using notepad and edit it as follows and save it somewhere on the computer

<b>John Walter</b>	<b>20</b>	<b>19</b>	<b>45</b>
<b>Sara Gill</b>	<b>16</b>	<b>15</b>	<b>35</b>
<b>Mark Black</b>	<b>23</b>	<b>24</b>	<b>50</b>
<b>Jess Paul</b>	<b>10</b>	<b>20</b>	<b>25</b>
<b>Joe Nash</b>	<b>14</b>	<b>18</b>	<b>44</b>

Think of these as students' full names and their assessment marks for exercises, projects and final exam. Write a C++ program that reads this file (ClassList.txt) and that creates a new file named **Report.txt** on the same folder with the same content as the input file together with the letter grades of the students on the same line for each student. For the letter grades, use the settings  $[90, 100] = A$ ,  $[75, 90) = B$ ,  $[65, 75) = C$ ,  $[50, 65) = D$  and  $[0, 50) = F$ .

# More General Complete Example

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
    ifstream fin("U:/Users/user_name/Desktop/ClassList.txt");
    if (fin.fail())
    {
        cout << "Input file not found." << endl;
        system("Pause");
        return 0;
    }
    ofstream fout("U:/Users/user_name/Desktop/Report.txt");
    if (fout.fail())
    {
        cout << "Output file not created." << endl;
        system("Pause");
        return 0;
    }
}
```

# More General Complete Example

```
//Now we have both input and output files ready to use
string fname, lname;
int exercise, project, final, total;
char grade;
for (int i = 0; i < 5; i++)
{
    fin >> fname >> lname >> exercise >> project >> final;
    total = exercise + project + final;
    if (total >= 90)
        grade = 'A';
    else if (total >= 75)
        grade = 'B';
    else if (total >= 65)
        grade = 'C';
    else if (total >= 50)
        grade = 'D';
    else
        grade = 'F';
    fout << fname << " " << lname << "\t" << exercise << "\t" << project << "\t" << final << "\t" << grade << endl;
}
//Close the file stream objects
fin.close();
fout.close();

system("Pause");
return 0;
}
```

# End of File Marker

- Sometimes we may wish to read input data from an input file but we may not know the amount of data that is stored in the file
- In such cases, we need to loop reading until all the data in the file is read
- That is, until we reach the END OF THE FILE
- C++ input streaming objects can test end of file using **eof()** member function that returns true when the end of file is reached

# End of File Marker

- In order to demonstrate the **eof** member function, assume there is an input text file named `TestInputFile.txt` on the desktop of the computer that we are working on
- Assume that this input text file contains some integer numbers (but also assume that we don't know how many numbers are in the file)
- Then the following program will read all the numbers in the file and print the minimum and maximum numbers in the file



# End of File Marker

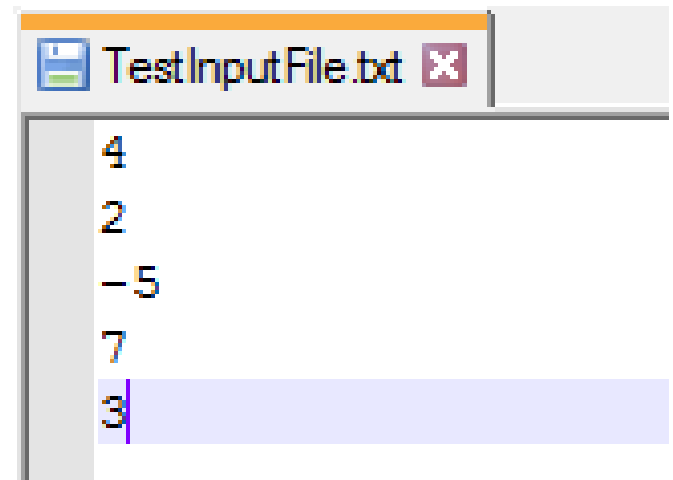
```
int main()
{
    ifstream fin("C:/Users/user_name/Desktop/TestInputFile.txt");
    if (fin.fail())
    {
        cout << "Input file not found. Exiting program." << endl;
        return 0;
    }
    else
    {
        //Read the first number input
        int a;
        fin >> a;

        //At this point, this first number is both the min and the max
        int minimum = a, maximum = a;

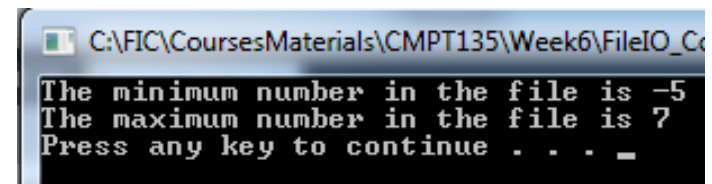
        //Now read all the remaining numbers using eof marker
        while (fin.eof() == false)
        {
            fin >> a;
            if (a > maximum)
                maximum = a;
            if (a < minimum)
                minimum = a;
        }
        fin.close();

        //Print the minimum and maximum
        cout << "The minimum number in the file is " << minimum << endl;
        cout << "The maximum number in the file is " << maximum << endl;
    }
    system("Pause");
    return 0;
}
```

Input File Example



Output of the program



# End of File Marker

- When using eof() member function of the input file streaming objects, it is important not to have any extra empty lines at the end of the input file we are reading from; for otherwise empty lines will be read before we can reach the end of file marker
- **Practice Question:-** Given an input file that contains several integers, write a C++ program that reads the numbers from the file and prints them to the screen in reverse order (that is the first number read is printed last and last number read is printed first)

Hint:- You can do this in two ways

- One way is to make use of your SmartArray class
- Another way is to use recursion which will require no arrays

Of course you can not use a static or dynamic array because it is not known how many numbers the file contains; unless you scan the file twice: once to count how many numbers there are in the file and then to read the numbers in to a dynamic array