**File I/O in C++**

**Rizwan Rasheed**

**Why?**

In programs all your data remain in memory (RAM) because program is execute when it gets space in memory, as we all know memory (RAM) is volatile and contents are available until the continuity of power or till the end of program. When our program ends all the inputs/outputs which we have taken while program’s execution are vanishes. What if we want to store our outputs and results computed by program so that we can use them after program termination? Another problem is, when we have to take a number of inputs from user then it seems tedious task to enter large number of inputs from keyboard every time the program get executed.

C++ provides the solution of upper problems with file I/O. We can get inputs from physical files (stored on hard disk) and similarly can write results to physical files. (Note that, this seems close to real life, in daily life programs/software get their inputs (or load data) from files or from some database, and similarly, write their results to files or database). This tutorial only talks about file I/O in C++ and not about databases. Also, it is important to mention that there are different methods of file I/O in different languages but concept remains the same.

**File I/O:**

There are many methods to read/write from files even in C++, but in this tutorial we will cover the simplest one.

File I/O is reading from and writing to files. This tutorial will only cover text files, that is, files that are composed only of ASCII text.   
C++ has two basic classes to handle files, **ifstream** and **ofstream**. To use them, include the header file **fstream**. ifstream handles file input (reading from files), and ofstream handles file output (writing to files). The way to declare an instance of the ifstream or ofstream class is:

ifstream in;

ofstream out;

The beauty of the C++ method of handling files rests in the simplicity of the actual functions used in basic input and output operations. As every C++ developer is surely knows the usage of **cin** (to get input from keyboard) and **cout** (to display output on console), these two **cin** & **cout** are also streams. If you know the usage of << & >> operators (with **cout** and **cin** respectively) then congratulations that you will feel so easy with files I/O. Because C++ supports overloading operators, it is possible to use << and >> in front of the instance of the class (ofstream and ifstream) as if it were **cout** or **cin**. The only difference is that **cout** & **cin** are attached to console & keyboard streams by default, but, the instances of classes (ofstream and ifstream) will be attached to physical files. Once the physical files are opened then file streams (ofstream and ifstream) can be used exactly the same as **cout** and **cin**.

**ifstream** is used to read data from files as **cin** is used to read data from keyboard. Similarly,

**ofstream** is used to write data to files as **cout** is used to write data to console.

If you want read and write within a single instance then you can use an instance of class **fstream** (parent of both). But, this tutorial, only covers the usage of **ifstream & ofstream** (as these are easy, straightforward to use and reliable).

Now, we will see different example programs which show how to read/write integers, floats, characters, strings and even whole objects from/to files.

Before any operation read/write, it is necessary to open the dedicated file.

The constructor for both classes will actually open the file if you pass the name as an argument. As well, both classes have an open function (open()) and a close function (close()). You aren't required to use the close function as it will automatically be called when the program terminates, but if you need to close the file long before the program ends, it is useful.

**Writing:**

Its ends to your wait (as you may be get crazy to see the file I/O in action☺), the following program gets some input from user on console, and write the input to a file namely “MyFirstFile.txt”.

<prog_1.cpp>

The writing operation on files can be done in two modes (there are other modes as well) i.e,

* out mode (previous data washed away (deletes) when file opens)
* append mode (previous data keep intact and new data appends with the previous one)

The default mode for opening a file with ofstream's instance is **out** mode (as in prog\_1.cpp). If necessary, you can give a second argument that specifies how the file should be handled. This is listed below:

ofstream out;

out.open("MyFirstFile.txt",ios::app);

I’m leaving this as an exercise for you to make a program and check the behavior of **“ios::app”** mode.

**Reading:**

The following program gets data (input) from the file “MyFirstFile.txt” (which is created in <prog_1.cpp>) and displays on console.

<prog_2.cpp>

**Example Programs:**

The following programs elaborate the concepts.

<prog_3.cpp> reads some positive integers from a file namely “integers.txt” (on this file there are some integer values on each line and -1 indicates the end of values on one line) and writes the sum of integers of whole line on the corresponding line of another file namely “integerSums.txt”.

<prog_4.cpp> reads characters from file namely “chars.txt” and write their corresponding ASCII values to another file namely “charsASCII.txt”.

In <prog_1.cpp> you may noticed that when you read from a file in a buffer as

out << buffer ;

this reads only until first white space, now question comes here that what if white space is a part of our input, as real life names include space between first and last name. This problem can be handled with the usage of getline() method.

**getline(stream’s instance, buffer, ‘delimiter’)** takes three parameters where

**stream’s instance** is the instance with which the file is attached, as **in** in our examples.

**buffer** is the char array or string in which you want to keep data after reading.

**delimiter** is the character (in single quotes) to which you want to read, it can be any character say ‘,’ or ‘|’ or ‘ ’ (space) or ‘\n’ (new line) or anything else you want.

<prog_5.cpp> illustrates the use of **getline()**. This program reads some names from a file namely “strings.txt” and copy them into another file namely “stringsCopy.txt”. Input file use ‘|’ as delimiter and in output file ‘\n’ is delimiter.

**References:**

<http://www.cprogramming.com/tutorial/lesson10.html>

To read binary files and more about file I/O, please visit

<http://www.cplusplus.com/doc/tutorial/files/>