**Lab:** Task 11

**Title:** File Input Output

**Author:** Sam Huffer, 101633177

# Part A: Binary File Output / Input Questions

* *Q: There are different file open modes. What are they?*

File opening allows the following modes:

* + ios::in – open file for input operations.
  + ios::out – open file for output operations.
  + ios::binary – open file in binary mode.
  + ios::ate – set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
  + ios::app – all output operations are at the end of the file, appending the content to the current content of the file.
  + ios::trunc – If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new content.

They are used in the following manner:

std::ofstream myfile;

myfile.open(“example.bin”, ios::out | ios::app | ios::binary);

Input modes can be combined in the parameters of open() using the | (OR) operator.The default mode of the ofstream (output file stream) class (if no modes are declared in open()) is “ios::out”; ifstream (input file stream) defaults to “ios::in”; and fstream (file stream; allows input and output) defaults to “ios::in | ios::out”.

Note: if you didn’t declare “using namespace std;”, format modes as “std::ios::[MODE]”.

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

* *Q: What happens if you don’t “close” the file? Is it something we need to worry about?*

When a file is “closed”, the operating system is notified and the resources being spent on the open file become available again. Once the file is closed, the stream object can be re-used to open other files, and the file that was open can now be opened by other processes.

Consequently, until you close the file, you cannot open other files with the same stream object, and the resources being spent on the file won’t be freed. If many files are opened without being closed (via multiple stream objects), all of them will be taking up resources together, which can impact performance.

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

* *Q: How many bytes are in the file? Is this expected based on the size of the variable types?*

The size of the file created is 18B. The data types written to the file are that of a char, integer, and float, which have the sizes 1B, 4B, and 4B respectively. That adds up to 9B. If one assumes extra data is taken to format the binary file, then the file size is not inconsistent with the sizes of the variables output to the file.

Source: <https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=vs-2019>

# Screenshots

Figure 4: false starts with reading and printing JSON files for part C. The recommended JSON library needed updated documentation, and other libraries didn’t want to play nice.

Figure 5: the code that did read the JSON file and print it, albeit without storing it as a JSON object.

Figure 2: closing of the file output stream, opening of the file with an input stream, and overwriting the test variable with a new blank object during part A.

Figure 1: results of parts A (writing values to a binary file, then re-reading them and printing them), B (reading lines from a text file, splitting them into values, and printing them) and C (reading lines from a JSON file and printing them).

Figure 3: splitting of string into vector of strings for part B, with a note about the instructions being unclear on one point.