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:

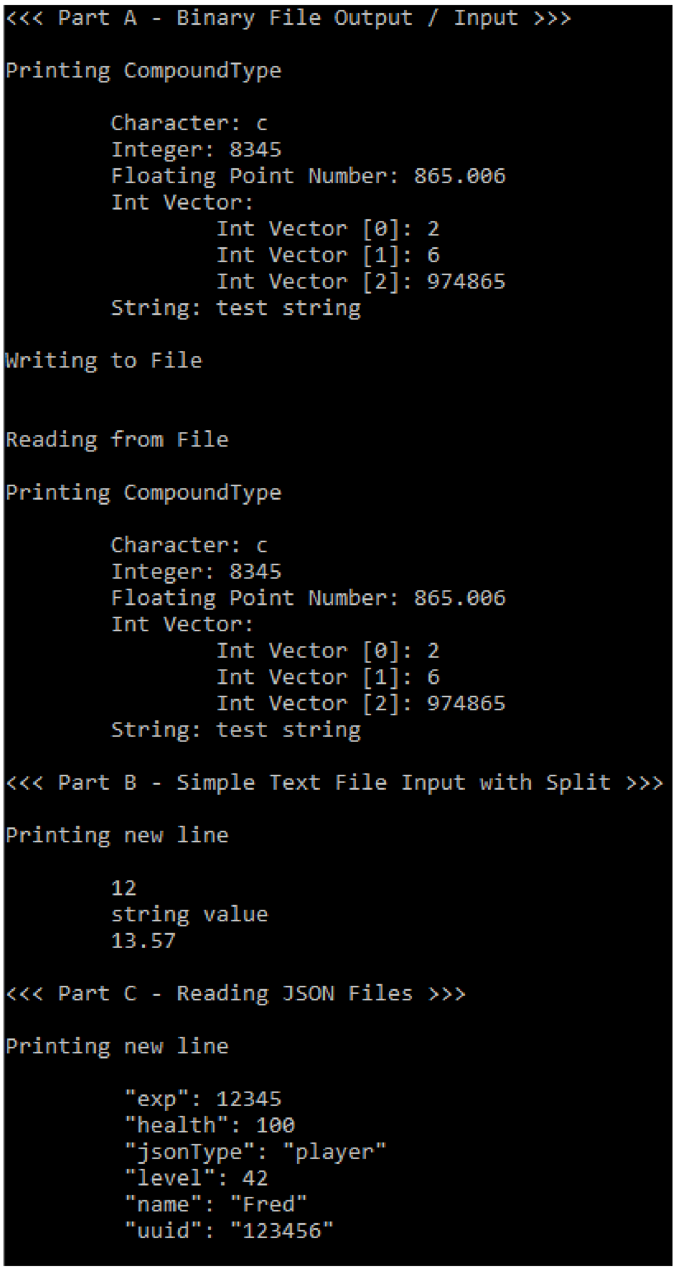
  - ios::in – open file for input operations.
  - ios::out – open file for output operations.
  - ios::binary – open file in binary mode.
  - ios::ate – set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
  - ios::app – all output 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);
  ```

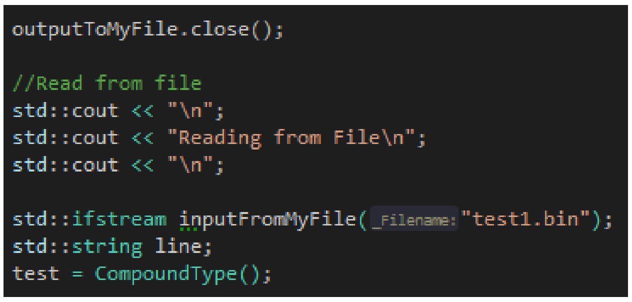
  Input modes can be combined in the parameters of open() using the | (OR) operator.The default mode of the ofstream (output file stream) class (if no modes are declared in open()) is “ios::out”; ifstream (input file stream) defaults to “ios::in”; and fstream (file stream; allows input and output) defaults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



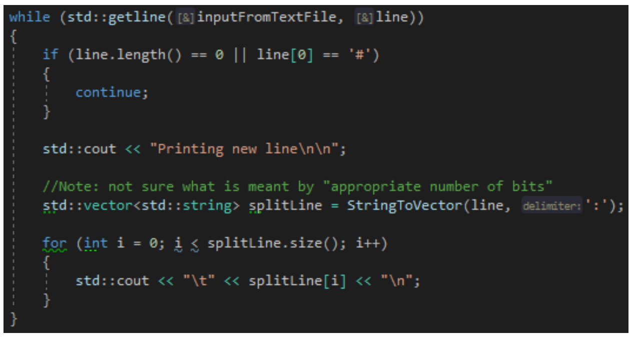
```
<<< Part A - Binary File Output / Input >>>

Printing CompoundType

        Character: c
        Integer: 8345
        Floating Point Number: 865.006
        Int Vector:
                Int Vector [0]: 2
                Int Vector [1]: 6
                Int Vector [2]: 974865
        String: test string

Writing to File


Reading from File

Printing CompoundType

        Character: c
        Integer: 8345
        Floating Point Number: 865.006
        Int Vector:
                Int Vector [0]: 2
                Int Vector [1]: 6
                Int Vector [2]: 974865
        String: test string

<<< Part B - Simple Text File Input with Split >>>

Printing new line

        12
        string value
        13.57

<<< Part C - Reading JSON Files >>>

Printing new line

         "exp": 12345
         "health": 100
         "jsonType": "player"
         "level": 42
         "name": "Fred"
         "uuid": "123456"
```

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).



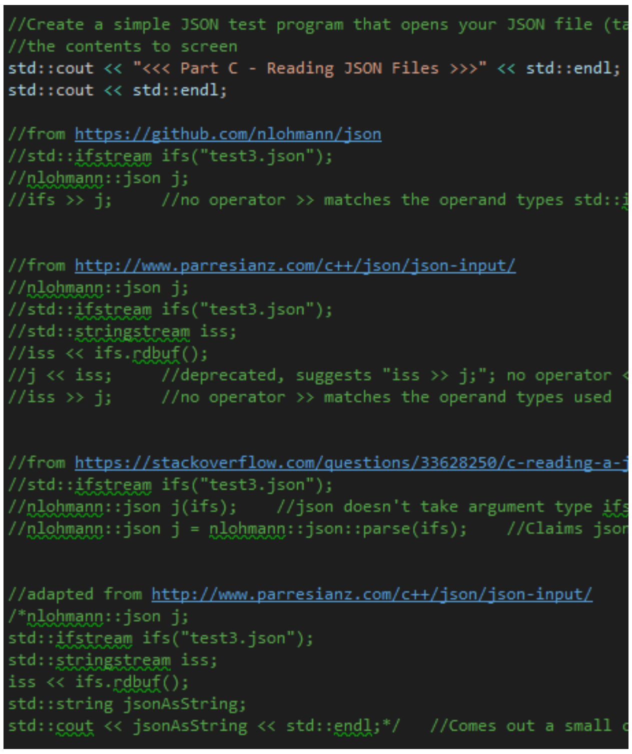
```
outputToMyFile.close();

//Read from file
std::cout << "\n";
std::cout << "Reading from File\n";
std::cout << "\n";

std::ifstream inputFromMyFile(_Filename: "test1.bin");
std::string line;
test = CompoundType();
```

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.



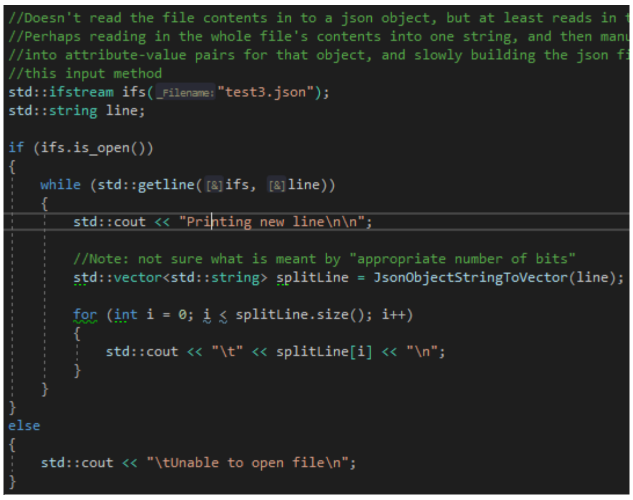
```
while (std::getline([&]inputFromTextFile, [&]line))
{
    if (line.length() == 0 || line[0] == '#')
    {
        continue;
    }

    std::cout << "Printing new line\n\n";

    //Note: not sure what is meant by "appropriate number of bits"
    std::vector<std::string> splitLine = StringToVector(line, delimiter: ':');

    for (int i = 0; i < splitLine.size(); i++)
    {
        std::cout << "\t" << splitLine[i] << "\n";
    }
}
```

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



```
//Create a simple JSON test program that opens your JSON file (ta
//the contents to screen
std::cout << "<<< Part C - Reading JSON Files >>>" << std::endl;
std::cout << std::endl;

//from https://github.com/nlohmann/json
//std::ifstream ifs("test3.json");
//nlohmann::json j;
//ifs >> j;     //no operator >> matches the operand types std::i


//from http://www.parresianz.com/c++/json/json-input/
//nlohmann::json j;
//std::ifstream ifs("test3.json");
//std::stringstream iss;
//iss << ifs.rdbuf();
//j << iss;     //deprecated, suggests "iss >> j;"; no operator <
//iss >> j;     //no operator >> matches the operand types used


//from https://stackoverflow.com/questions/33628250/c-reading-a-j
//std::ifstream ifs("test3.json");
//nlohmann::json j(ifs);    //json doesn't take argument type ifs
//nlohmann::json j = nlohmann::json::parse(ifs);    //Claims json


//adapted from http://www.parresianz.com/c++/json/json-input/
/*nlohmann::json j;
std::ifstream ifs("test3.json");
std::stringstream iss;
iss << ifs.rdbuf();
std::string jsonAsString;
std::cout << jsonAsString << std::endl;*/   //Comes out a small c
```

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.



```
//Doesn't read the file contents in to a json object, but at least reads in t
//Perhaps reading in the whole file's contents into one string, and then manu
//into attribute-value pairs for that object, and slowly building the json fi
//this input method
std::ifstream ifs(_Filename: "test3.json");
std::string line;

if (ifs.is_open())
{
    while (std::getline([&]ifs, [&]line))
    {
        std::cout << "Printing new line\n\n";

        //Note: not sure what is meant by "appropriate number of bits"
        std::vector<std::string> splitLine = JsonObjectStringToVector(line);

        for (int i = 0; i < splitLine.size(); i++)
        {
            std::cout << "\t" << splitLine[i] << "\n";
        }
    }
}
else
{
    std::cout << "\tUnable to open file\n";
}
```

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