Task 10 - Lab: File Input Output

The aim of this lab is to improve (or refresh) your file input/output skills in C++. The lab in divided into three parts.

We take a quick look at reading and writing to binary files in Part A, then read from a simple text file (splitting lines) in Part B, and finally make use of the common JSON file format in Part C. There are many other file I/O related topics to explore, and some extension ideas are included that you may wish to explore in and include as "extension" work in your final portfolio.

Output: Lab Notes (points, URLs, question answers) to Doubtfire as PDF. Code in repo and demo to your tutor.

Tip: Remember to tidy up/clean your code when you finish the lab, and commit that code to your repo (perhaps with the appropriate message of "code clean up"). Remove unnecessary code – you should have a commit history of it anyway!

There is no set format for the "lab notes" for this lab but you do need to take "notes" as you go and answer some questions.

You will need to create the required programs and demo to your tutor. Because you need to demonstrate to your tutor anyway, making a lab notes document that you commit to your repo is a very good way to help you do this.

Note: For all of the following steps, you could take a very object-oriented approach (with classes and methods), or a simple reusable function (callable code blocks), or just in-line all your code in main. It's up to you, but whatever you do try to be clean and tidy so that others can easily understand your code.

Part A: Binary file Output / Input

1. Create a new document for your lab notes. (Add "Part A" section heading?)
2. Write a C++ program that has
    a. a basic struct or class ("compound type") that has at least 3 simple variables: char, int, float;
    b. create (or have created) an instance of your compound type, then
    c. set the value of each variable in your instance, to something other than a zero value.
3. Write a reusable routine (function) to print/show the values to screen.
4. Compile and run. Add and commit doc + code to repo!

5. Modify your code to
    a. open a binary file in "write" mode (such as "test1.bin"), then
    b. write the three different values to the binary file, and finally
    c. close the file.

    Q: There are different file open modes: What are they? (Answer in lab notes!)
    Q: What happens if you don't "close" the file? Is it something we need to worry about? (Answer in lab notes!)
6. Compile and run.
7. Confirm that a file was created.
8. Using an appropriate viewing program (hex viewer or similar), inspect the file and confirm that something is being written to the file.

    Q: How many bytes are in the file? Is this expected based on the size of the variable types? (Answer in lab notes!)
9. Commit to repo!

10. Modify your code:
    a. Disable (comment/don't call) your existing "save to file" code,
    b. open the file you saved earlier, in binary read (only) mode,
    c. set the values of your instance variables to those read from the file (in correct order), then
    d. call your reusable print/show code values of the variables
11. Compile and run, confirm that it worked! Commit to repo!

Extensions: (If you do these, brag about it in your Lab Notes document!)

- Write a variable number of int values (from example from a vector or array), by first saving the "length" and then each value. Write the corresponding Read routine.
- Demonstrate that you can read the binary data into inappropriate order, or with unrelated variable types.
- Demonstrate how to write and read a string (c-style or std) to a binary file.

Part B: Simple Text File Input with Split

1. Add new section heading “Part B” to your Lab Notes?
2. With a text editor, create and save a simple text file (such as “test2.txt”) that contains three lines similar to the following, with the last line being the actual line of useful data and using a full-colon separator character:

```
# This is a comment line. The next line is deliberately blank.

12:string value:13.57
```

3. Create a new program that is able to
   a. Open the file (text mode, read only),
   b. Print each line to screen, one at a time
4. Compile and run. Confirm that it works
5. Modify your code so that it can (**required**!)
   a. Ignore any blank line (“strip” whitespace first?),
   b. Ignore a line that starts with the hash “#” character (treats it as a single line comment),
   c. Splits all other lines, checking that is has the appropriate number of “bits”, and
   d. Prints each split line to screen, on bit at a time. (The “bits” are just strings in this case.)
6. Compile and run. Confirm that it works as expected. Commit to repo!

Note: You should have code from an earlier lab to “split” a string. ☺ However, a split function that gives you a string broken up into a collection of substrings is very useful, so you might want to research this topic in more detail.

Extensions:

- Modify this so that it works as a basic “INI” file format reader, with “key=value” lines saved into a dictionary (of strings). Don’t worry about identifying value types to start with.
- Write a INI file “writer” (class with support methods?), as we have only written a “reader”.

Part C: Reading JSON Files

There are many good reasons for using established file formats, and to also use well-tested code. Now we get some experience with the JSON format in C++ using a popular JSON library.

1. Add new “Part C” section to Lab Notes?
2. With a text editor create a basic JSON text file (such as “test3.json”) with the following content (or similar) for player character details:

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

3. Go to https://github.com/nlohmann/json, read and download the JSON library. We suggest the “Trivial integration” version (search the README.md) which is a single `hpp` file in plain C++11, but it’s up to you. You could use another JSON library if you want, but this one is popular and well designed.
4. Create a simple JSON test program in C++ that, using the JSON library, opens your JSON file and then print the contents to screen. (Simple to write – but might take you a bit of effort. Use similar steps to the ones earlier in this lab.)

Final Step: Update repo, update Doubtfire status, show your tutor!

When your repository is up-to-date (files you have created for this lab are saved, added/committed and your local repo has been pushed to your bitbucket repo), update your Doubtfire status and show your tutor!