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

/*  Task: Lab 10
*   Title: File Input Output
*   Author: Thomas Horsley 103071494

    Part A
    TODO: 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.
    TODO: 4. Compile and run. Add and commit doc + code to repo

    Part B
    - 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.
            TODO: There are different file open modes: What are they? (Answer in lab notes!)
            TODO: What happens if you don⁄t ⁄close⁄ the file? Is it something we need to worry about? (Answer in lab notes!)
*/

class CompoundType {
private:
    char _char;
    int _int;
    float _float;

public:
    CompoundType(char character = ' ', int integer = 0, float floating_pt = 0.0) {
        _char = character;
        _int = integer;
        _float = floating_pt;
    }

    char getChar() { return _char; }
    int getInt() { return _int; }
    float getFloat() { return _float; }
    void setChar(char character) { _char = character; }
```

```cpp
    void setInt(int integer) { _int = integer; }
    void setFloat(float floating_pt) { _float = floating_pt; }

    void show() {
        std::cout << "Character: " << _char << std::endl;
        std::cout << "Integer: " << _int << std::endl;
        std::cout << "Float: " << _float << std::endl;
    }
};

int main() {
    CompoundType write_tester('C', 1, 3.14);
    CompoundType read_tester;
    std::ofstream writer;
    std::ifstream reader;

    if (writer) {
        writer.open("test.dat", std::ios::app | std::ios::binary);
        writer.write(reinterpret_cast<char*>(&write_tester), sizeof(CompoundType));
        writer.close();
    }

    reader.open("test.dat", std::ios::binary);
    while (reader.read(reinterpret_cast<char*>(&read_tester), sizeof(CompoundType))) {
        read_tester.show();
    };
    reader.close();

    return 0;
}
```

```cpp
#include <iostream>
#include <fstream>
#include <algorithm>
#include <string>
#include <vector>
#include <json.hpp>

using json = nlohmann::json;

/*  TODO: 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:

    - 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!

    Part B Notes:
        - Use ifstream objects with stream extraction operators
        - Open the file, verify, process data and close
        - Use loops to read entire file


    TODO: 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.
```

```cpp
    - 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.)

    Part C Notes:
        - Should probably make the FileReader and JSONReader subclasses of a Reader parent or implement an interface containing
          the declarations for Splitting lines, checking comments and delimiting strings based off of values. Would be nice for
          JSON formatting which is currently non-existent... but hey... it prints and that's enough documentation crawling for
          tonight.

*/


class CompoundType {
private:
    int _int;
    std::string _string;
    float _float;

public:
    CompoundType(int integer = 0, std::string string_data = " ", float floating_pt = 0.0) {
        _int = integer;
        _string = string_data;
        _float = floating_pt;
    }

    void setInt(int integer) { _int = integer; }
    void setString(std::string string_data) { _string = string_data; }
    void setFloat(float floating_pt) { _float = floating_pt; }
    void setAll(int integer, std::string string_data, float floating_pt) {
        _int = integer;
        _string = string_data;
        _float = floating_pt;
    }

    void show() {
        std::cout << "Integer: " << _int << std::endl;
        std::cout << "String: " << _string << std::endl;
        std::cout << "Float: " << _float << std::endl;
    }
};

//Does operations and such to read from the file, can be expanded for writing too
```

```cpp
class FileReader {
private:
    std::string _file_name;
    std::ifstream _reader;

    std::vector<std::string> splitLine(std::string string_data, char delimiter) {
        std::vector<std::string> split_strings;
        int start_idx = 0, end_idx = 0;

        for (int i = 0; i <= string_data.size(); i++) {
            if (string_data[i] == delimiter || string_data[i] <= string_data.size()) {
                std::string delimited_data;
                end_idx = i;

                delimited_data.append(string_data, start_idx, end_idx - start_idx);
                split_strings.push_back(delimited_data);
                start_idx = end_idx + 1;
            }
        }

        return split_strings;
    }

    bool isComment(std::string string_data) {
        remove(string_data.begin(), string_data.end(), ' ');

        if (string_data[0] == '#') { return true; }
        return false;
    }

    std::vector<std::string> processString(std::string string_data) {
        std::vector<std::string> delimited_data;
        std::string line = string_data;

        // Remove Whitespace
        remove(line.begin(), line.end(), ' ');

        // If the comment isn't a line, split the string and add each element to our
        // delimited_data vector.
        if (!isComment(line) && line != "") {
            delimited_data = splitLine(line, ':');
        }

        return delimited_data;
    }
```

```cpp

public:
    FileReader(std::string file_name) {
        _file_name = file_name;
        _reader.open(file_name);
    }

    ~FileReader() {
        if (_reader.is_open()){ _reader.close(); }
    }

    // Return all of our delimited string data in a vector.
    // When working on this vector it is assumed that the data will
    // come in the format int:string:float so that proper typecasting can
    // occur on the return vector.
    std::vector<std::string> readLines() {
        std::string line;
        std::vector<std::string> formatted_strings;

        if (!_reader.is_open()) { _reader.open(_file_name); }

        while (std::getline(_reader, line)) {
            for (auto it : processString(line)) {
                formatted_strings.push_back(it);
            }
        }

        return formatted_strings;
    }
};

// Needs to be able to read in each line of the JSON script and print those
// to the terminal preferably using a for loop and iterators.
class JSONReader {
private:
    std::string _file_name;
    std::ifstream _reader;
    json _json_data;

public:
    JSONReader(std::string file_name) {
        _file_name = file_name;
        _reader.open(file_name);
        _json_data = json::parse(_reader);

        /* For testing.
        _json_data = {{"exp" , 12345},
                      {"health", 100},
```

```
                        {"jsonType", "player"} ,
                        {"level", 42},
                        {"name" , "Fred"},
                        {"uuid", "123456" } };*/
    }

    ~JSONReader() {
        if (_reader.is_open()) { _reader.close(); }
    }

    void printJSON() {
        // Cool function, not sure how efficient it is but it's pretty cool.
        auto json_string = to_string(_json_data);
        std::cout << json_string;
    }

    void printJSONButPretty() {
        auto json_string = to_string(_json_data);


    }

};

//We're assuming that the data will be formatted correctly in format int:string:float
//  - Go through each line of the file saving the line to a string
//      * If the line begins with '#' then ignore
//  - Remove whitespace from the string
//  - Split the string into it's components using ':' as a delimeter and return a vector
//    containing the strings
//  - Print results
int main() {
    CompoundType tester;
    FileReader text_reader("input.txt");
    JSONReader json_reader("test.json");

    // Part B
    if (true) {
        std::vector<std::string> formatted_strings = text_reader.readLines();

        for (auto it : formatted_strings) {
            std::cout << it << std::endl;
        }

    }
```

```

    // Part C
    if (false) {
        json_reader.printJSON();
    }

    return 0;
}
```