Dear Jing (2010573@Swansea.ac.uk),

Thank you for your coursework submission. Below I've detailed your feedback and marks. I apologise again that, due to my wrist injury, I'm delayed in getting these to you.

Your Total Mark is 90.0 (out of 100.0).

Feedback on your coursework

I have broken down your feedback into two sections: firstly, I've made some comments on the tests on Autograder, and secondly I've passed through your coursework and made a qualitative assessment of your code.

Feedback from automated marking (59/65)

You received 59 completeness marks (out of 65). I've reviewed how your program performed against each of the automated tests run on the CS Autograder service. For deciding the weighting of each test (i.e., number of marks), each test was assessed for its relative difficulty based on the module content and how much work was involved.

You passed all the provided Catch2 tests, scoring all (22 / 22) marks possible. This is obviously a great outcome and demonstrates your capability to read and understand unit tests, and implement code to pass them.

With the additional Catch2 tests designed to test the edge cases and advanced parts of the coursework, your code passed most of the tests (24 / 30), which given these tests were designed to be challenging, is a great result. Many of these tests focused on following the description in the function block comments more closely or testing your code with the additional datasets. The marks for these tests were not evenly distributed (i.e., some tests were worth more marks than others) due to their difficulty).

Your program correctly passed the first unit test, thus you received 2 marks. This test was designed to ease you in to working with the Catch2 unit tests on a relatively simple function.

There was also a hidden test determining whether you processed the action argument case-insensitively (which you did!), earning you another 1 mark.

Your program correctly passed the unit tests surrounding the Date class, which earned you 2 marks. These were designed to get you thinking about dealing with unexpected and invalid inputs as well as overloading the equals and less than operator.

There was also a hidden test determining whether reasonable looking, but still invalid dates (e.g. 2024-04-31) would still throw an appropriate exception and if leap years are correctly handled (e.g. 2024-02-29). Your program passed these tests and you earned 2 marks. Great job identifying and handing these edge cases!

There was also a hidden test to check if dates are being correctly padded with zeroes (e.g. 2024-02-02 vs 2024-2-2). While the documentation mentioned that dates should be formatted as YYYY-MM-DD, you did not pass these tests.

Your program also passed all three tests focusing on the three main containers in your program (Task, Project and Todolist). The implementation of these functions was similar across each of them. Perhaps in a bigger application, there could have been some generalisation introduced (e.g., creating a template class and using this), however, this was way beyond the scope of the coursework.

I ran two additional tests on your Project class.

Your implementation of Project::newTask() worked as expected, returning an existing object instead of creating a new one when asked to create a second Task with the same identifier as an existing Task. Trying to reduce pointless duplication of data is something C++ gives us the power to do, so nice work doing this.

Your Project class was also tested to determine whether it merges Task objects instead of replacing them when given a new Task object with the same identifier as an existing Task.

I also ran two additional tests on your TodoList class, following the same sort of pattern as the Project tests.

Your implementation of TodoList::newProject() also worked as expected in the same was as Project::newTask().

Your TodoList class also merges Project objects instead of replacing them when given a new Project object with the same identifier as an existing Project.

Loading the data from JSON also worked in your program. As I said in the coursework description (and many times!), this coursework was as much about your programming as it was about testing your ability to read library documentation and write code in response to this. This is often just as difficult (if not more so, at times) than writing code from scratch. So, well done pulling it off.

As expected, you also managed to go in the opposite direction and convert your data structure back to JSON.

Your implementations for the == overload operators for Task, Project, and TodoList all passed another test.

The last three tests focused on testing your main program when the action argument was set to json, create and delete.

Your program worked with all three of these. If the rest of your program worked well, passing these tests mostly required calling into the right functions and writing some error handling code.

I also tested your implementation to see if you could add multiple levels of data at once (i.e., a project, task and a tag)—your code worked!

No test was provided for the update mechanism—this was by far the hardest part of the coursework and unfortunately your code did not pass this test.

!!!INVESTIGATE EXTENDED 9!!!

Your program also correctly doesn't output null for empty data (this was something quite common and only uncovered if you tested your code with empty datasets).

You passed all the provided output tests, scoring (13 / 13) marks possible. These were designed to test the basic output of your program in response to the commands. Passing all the output tests is a great outcome.

As you'll have discovered while looking at these tests, often our program needs to output slightly different output to users as opposed to internal exception messages. Think about the experiences you've had with computer programs: when they crash, do you always want some internal stack trace or cryptic message? Programming is as much about designing code for other programmers as it is about designing usable programs for people.

Feedback on your code (31.0 / 35.0)

I firstly tested whether your code compiled well. Your code compiled without issuing any warnings when using the -pedantic and -Wall flags. In large projects, it is often frowned upon to push code to a repository that has warnings as this can be a sign of sloppy code.

Your coding style was very good and was consistent throughout! Coding style is made up of many things, including indentation, variable naming, commenting, and general presentation of your code.

Your use of indentation and general code style is consistent throughout.

You have used commenting well throughout your coursework solution. Comments help readers of your code understand complex code blocks.

In terms of naming convention, you have used good naming of variables etc. in your code. Good naming removes the need for many comments because it allows people to read and make sense of code without explanatory comments.

You have some great use of pass-by-reference in your code too, showing you've taken away key points from the lectures.

You haven't used any pointers in your codebase to pass in and out of functions, which is excellent. Returning and passing in pointers is what we did in C, and while this can work, it does also open us up to many issues (including null pointers). We really should be trying to make use of either smart pointer classes, references, and RAII to avoid having pointers in our codebase as these open us up to many issues (e.g., segmentation faults).

Your program doesn't handle the case where there are no program arguments. This was something you should have easily caught with some testing.

You didn't use 'using namespace std' in your code, which I strongly discouraged in lectures. This both adds extra legwork to your compiler, and also pollutes your main namespace.

You ensured the functions that did not change member variables were all marked as const, which is always good practice.

Looking at your code, you also used const with variables where it made sense. Compilers can use this information to reason about our code and potentially optimise it.

You've shadowed a variable. This is where you create a variable inside a given scope, when it has been always declared in a higher scope or as a parameter. The first place I spotted this was in project.cpp on line 16.

You've not used a switch statement without a default value. This isn't always bad but can be considered problematic in some cases, so including a default case is generally considered good practice.

You've not used a cast to remove qualifiers (e.g., a const) -- this is often done to allow developers to deal with situations of poor program design and are rarely the correct solution to the problem.

Looking through all your comparisons and if statements, these seem all good and reasonable.

All your functions were declared in header files. This is also typically expected in industrial or open-source projects, so this requirement wasn't not just me being difficult.

You used the Standard Library vector container, which was an OK choice for this coursework. However, a map offers needs of the functions you had to implement, proffering quick access to items and allowing easy expansion/deletion of items. Vectors store data contiguously so as you add data, you would have invoked expensive copy operations. Think about the situations where you might be accessing/editing/deleting a project in the middle of the vector: this would be O(n), where as a map would typically be O(log n).

You haven't used new/delete anywhere, which is great. As I covered in lectures, this introduces challenges, e.g., when we start to also mix in things like exceptions which sidestep the typical flow of execution.

Your loading of JSON is a good approach (passing the stream into the JSON library), although there were other more involved and elegant approaches provided by the library for converting objects to/from JSON. I actually used your approach, as this worked well, but you could have tried making use library's from_json and to_json functions.

You've also implemented some error handling for when opening and parsing JSON files, but I think there could be a little more. Be sure to think through what happens when the file doesn't exist, isn't readable, or has invalid code, and how can we provide appropriate and distinct error notices in each case.

Your JSON generation is a very manual process, whereas you really should have looked into the JSON library's generation functions—these would have made the process really simple (i.e., their dump() function).

You consistently used const when catching exceptions, which is good practice that I covered in lectures. You did catch them all as references, which means you avoided copies of the exception objects being made when they are thrown (2.0 / 2.0 marks).

Taking a holistic look, it also seems like you managed to avoid redundant code throughout your program. As I stated in the coursework specification and the coursework video, I wanted to award marks for better quality code rather than completeness too -- this is, after all, a module where you are expected to learn a good understanding of why and how we develop OOP code the way we do.

You did not implement any bonus features, which is fine as this wasn't expected.

Thank you for your comments in the README.md file. They are a useful documentation of the nuances of your implementation.

Summary

This is an excellent coursework solution, Jing. You have produced some nice code, and there is very little room for improvement.