Programming Project

Coursework 1

Deadline: 10/3/2020 10:00 am

Submission: submit the report in pdf, and your entire git repository in a **single zip file** through Minerva. The compressed folder containing the repository must have either extension .zip or .gz. The folder in which you cloned the Gitlab repository contains a copy of the repository. Simply compress and submit this folder (again, in a single zip file).

Important notes on the submission:

- Write the program in standard C. If you write your code in any other language, it will not be assessed and you will get a zero mark.
- This is an individual project, and you are not supposed to work in groups or pairs with other students.
- Be aware that plagiarism in your code will earn you a zero mark and will have very serious consequences. If two (or more) students have large portions of their files nearly identical they will be accused of plagiarism or collusion. If found guilty, all parties involved will incur the penalty, regardless of who was the author of the code. For this reason, never show, or give access to, your code to anyone. Do not help a colleague by sharing your code, or you will both be found guilty of collusion.
- It is your responsibility to make sure that nobody has access to your code. Lock the session if you leave your computer unattended.
- Make sure to download and check your submission. Corrupted files, binary files, wrong versions, copies of your java project (over the years we have seen it all...), or anything other than what requested in this document will be considered an invalid submission.
- We will not accept submissions other than through Minerva.

Specification

Create a software for the management of a library, according to the specification below. Use the Gitlab repository created for this module to commit the required changes. If you do not have a Gitlab repository for this module, contact the instructor immediately.

Functionality

The software must have the following functionality:

- Register users **[5 marks]**
- User login **[4 marks]**
- Users can search for books **[5 marks]**
- Users can borrow and returns books. **[10 marks]**
- A special user, the librarian, can also add and remove books to the library. **[8 marks]**

- The state of the library (books, users, and loans) is saved to file and restored at a new execution **[8 marks]**

The software must have more than one module (C file). One module must implement the provided interface book_management.h.

[total: 40 marks]

Building

Create and keep up-to-date a CMakeLists.txt for the compilation of both the main program and the unit tests.

[5 marks]

Unit Testing

- Before implementing the functions in book_management.h, create unit tests for them using the test framework Unity. Make sure to test both the expected outcome and error conditions on all the functions. Commit the functions to the git repository, and tag the commit with "cw1_tests". Refer to https://git-scm.com/book/en/v2/Git-Basics-Tagging to learn how to tag a particular commit. This will make it easier to retrieve the tests developed at this stage. Describe your tests in the report. An example of the expected description is shown in the appendix at the end of this document.

  [10 marks]

- Create and maintain unit tests for other modules.

  [20 marks]

[total: 30 marks]

Modular Development

- Correct use of the static and const modifiers **[5 marks]**
- Code is modular and well structured **[10 marks]**
- Iterative improvements: create a call graph at any point in the development, annotated with a different colour per module, and add it to the report. Tag the git commit of the code at this stage with "cw1_before". Briefly comment on its structure, for example: are the modules highly coherent and loosely coupled? Are there functions doing too much that should be split in smaller functions? Are there functions doing too little that can be incorporated in other functions? Make a change based on your analysis, create the new call graph, and add it to the report. Tag the commit with the code after the change with "cw1_after". How has the change improved modularity? An example of how to structure the report can be found in the appendix.

  [5 marks]

[total: 20 marks]

Reflection

Write one paragraph on each of the following (maximum one a4 side in total).

1. What went well with this project? Include specific areas of the work, programming, design or testing.
2. What was the hardest part of this work? Why, and what will you do to address this for the future?

Please avoid generic statements about time-management. Focus on your C coding, design and testing processes.

[5 marks]

[grand total: 100 marks]

Appendix

Testing

Example of documentation of the test for a function in the restaurant software developed in class.

Function: `int order_add_dish(struct Order * order,const char *dish_name);`

Assuming a return value `res`;

Expected behaviour:

- `res == 0`
- `order->order_length` after the invocation equals `order->order_length before the invocation + 1.`
- `order->dish[order->order_length – 1]` points to a dish with name `dish_name.`
- `order->array_size >= order→order_length.`

Checked Exceptions:

- res == 1 if dish_name does not contain the name of an existing dish

Assertions:

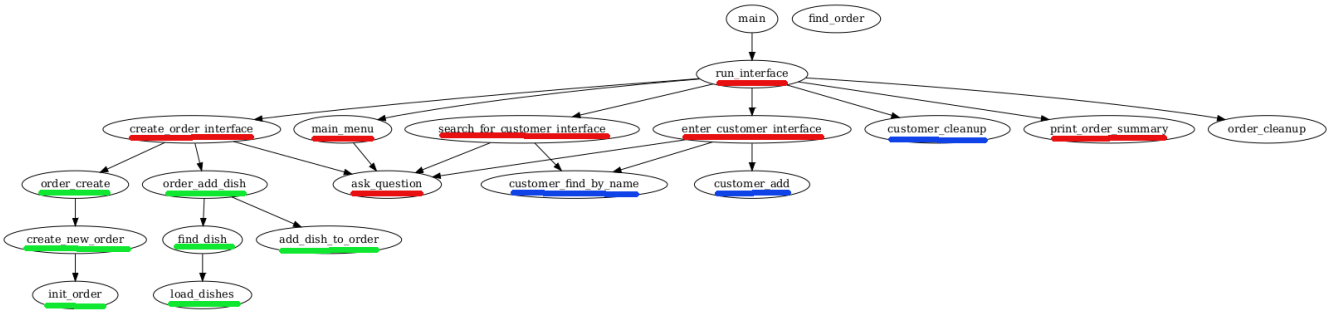
(*note: assertions are expected to be treated internally by the function, and do not have a corresponding test implemented in the test suite*)

- order and dish_name are both not null
- dish_name is a null terminated string

Call graph

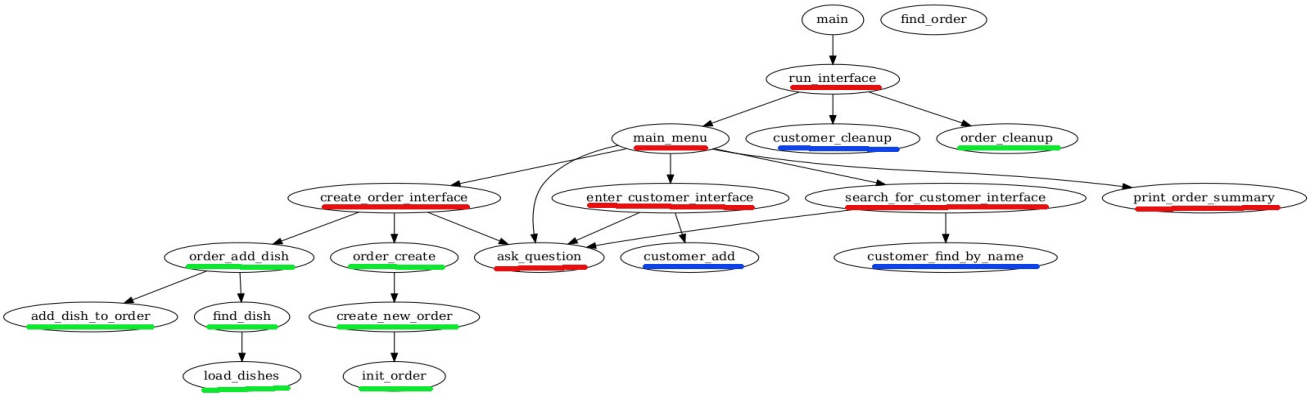
Example of the iteration of modular development using call graphs.

The call graph at commit with tag `cw1_before` is as follows:



The functions in the call graph form clear clusters, which suggests that the modules are coherent and loosely coupled. The function `run_interface` calls all the other interface functions, having many dependencies. This can be improved by braking it down further. Furthermore, `customer_find_by_name` is called by both `enter_customer_interface` and `search_for_customer_interface`. It could be called only by `search_for_customer_interface` reducing the dependencies to one.

The call graph after these changes, corresponding to the commit with tag cw1_after is shown below:



The function `run_interface` now calls fewer functions and the structure is more hierarchical with better distributed responsibilities between the functions. Furthermore, `customer_add` is only called by `enter_customer_interface`, while `customer_find_by_name` is only called by `search_for_customer_interface`, further reducing the dependences between the functions.