CHC5028 Software Development with C/C++

Coursework

Important Dates

Week 5 (7/11/2022-11/11/2022): Mandatory demonstration of Exercise 1, feedback with preliminary marks.

Week 10 (12/12/2022-16/12/2022): Mandatory demonstration of Exercise 2, feedback with preliminary marks.

Week 12 (26/12/2022-30/12/2022): Final demonstration and submission.

Background

"Text adventures", now called "interactive fiction", were among the first type of computer game ever produced. These games have no graphics; the player reads the story of the game in text, and decides what their character will do by typing commands at a prompt. Although less popular now, text adventures are still played and created, and developed into the original online RPGs (MUDs). You can play some sample modern text adventures here:

A Change in the Weather, Spider and Web, Slouching Towards Bedlam, 北大侠客行

These are playable online via a web browser. It is advisable to try out the games to get an understanding of how the games behave.

For this coursework, you will be creating a simple **game engine** for a text adventure. You are **not** required to write an actual adventure, only the back-end program code that would support one. You will need to add some material to the program in order to test it, but this may just be simple test material. You may add interesting descriptions or stories to your program if you want to, but there are no marks for doing so.

You are provided with a CLion project containing a very simple game harness which supports only two commands: going north (`north` or `n`), and quitting (`quit`). Extend it by doing the exercises below. Note that the later exercises are less explicitly described than the earlier ones, meaning that you must solve more problems yourself. This is intentional.

The coursework is written to be built using `gcc` through CMake and CLion. It is not recommended that you attempt to build it using Visual Studio or XCode.

Important: If you are building the sample coursework on a platform other than Windows, or on a machine which does not have the Windows API installed, you may get an error in the file *wordwrap.c*. This file calls a Windows specific function to find the width of the console. If you get this error, remove the `#include <windows.h>` from the top of the file, and edit the `initWordWrap()` function by deleting its contents and replacing them with `consoleWidth = 80; currentConsoleOffset = 0;`. You can change 80 here to any number that makes the output comfortably readable.

Exercise 1 (20% of the mark)

In the current system you can only move North. Extend the engine to allow movement in all four compass directions.

- Add properties to the `Room` class for storing east, south, and west exits. These properties will need accessor methods. (5%)
- Add code to the `gameLoop` method to understand the commands `east`, `south`, and `west` (and the abbreviations `e`, `s` and `w`) and to handle them in a similar way to `north`. (5%)
- Modify `initRooms` to create more rooms using the new exits in order to test your code. (5%)
- Find a more elegant way of implementing these exits which does not repeat code. [Hint: Traversing through map structures/strings, etc can be considered.] (5%)

Exercise 2 (40% of the mark)

A key part of most text adventure games is the ability to move objects around. Objects can be found in rooms and can be picked up and put down by the player. Add this capability to the game engine.

- Create an `GameObject` class. It should contain at least a short name, a long description, and a keyword (for the player to use when typing commands). (5%)
- Modify the `Room` class so that each Room includes a list of `GameObjects` in the room. (2%)
- Modify the `State` class to include a representation of a list of `GameObjects` the player is carrying, called `inventory`. (3%)
- Modify the `Room::describe()` method to also print out the short names of all the objects in the room, formatted as nicely as possible. (5%)
- Modify the `gameLoop` method to pay attention to the second word of the command the player enters, if there is one. (5%)

- Modify the `gameLoop` command to search through a) objects in the current room, and b) objects in the inventory, for an object with a keyword matching the second word of the command the player typed. (5%)
- Implement the player command `get` which, when typed with an object keyword, will move that object from the current room list into the inventory. It should display appropriate errors if the object is not in the room or the object is already in the inventory or the object does not exist. (5%)
- Implement the player command `drop` which, when typed with an object keyword, will move that object from the inventory into the current room list. It should display appropriate errors if the object is not in the inventory or already in the room, or does not exist, etc. (5%)
- Implement the player command `inventory` which will print out the short names of all the objects in the inventory. (2%)
- Implement the player command `examine` which, when typed with an object keyword, will print out the long description of that object. (3%)
- Modify `initRooms` to create some `GameObjects` and put them in the rooms. Use this to test your program. (No marks are assigned specifically for this task, but without it the ones above cannot be demonstrated.)

Exercise 3 (40% of the mark)

Since most players will not want to play an entire game at one sitting, most games include `save` and `load` (or `restore`) commands. Implement these commands. They should ask the user for a filename and then write or read the current game state, to or from that file.

Note that the layout and descriptions of rooms, and the list and descriptions of objects, are **not** part of the game **state** because they do not change during the game. These should **not** be included in the save file and saving them will lose marks.

A simple file open, load, and save does not guarantee full marks and may not guarantee "a good mark".

To this end, some important points to consider:

- The "game state" may also include the locations of objects the player has dropped in rooms. Would it be a good idea to restructure how object locations are stored?
- The `State` object stores the current room, and objects, using pointers. Pointers cannot safely be written to disk since addresses may be different when the program is reloaded. How can you enable this data to be safely saved and reloaded?
- It is worth ensuring to some degree that the user cannot readily cheat, or spoil the game, by reading or changing a save file. While it is not necessary to implement actual authentication or encryption but at the same time, the file does not have to be

just a text dump. This actually makes it harder to parse when loaded. So, for example, saving the required indexes into a static array of strings maybe be a better way than saving the strings themselves.

Marking scheme for this section:

- 5% for basic correct structure of I/O.
- 5% for handling errors appropriately.
- 10% for the file format designed for storing the saved game.
- 10% for the code that performs the save.
- 10% for the code that performs the load.

Assessment Rules

Code will be assessed by a demonstration and viva in week 12. You will be asked to demonstrate your code and to explain how it works. There is no hard division of marks between code and viva.

If you cannot explain your code sufficiently well to satisfy the assessor that it is your own work, they have the right to award 0 marks for that exercise, regardless of the quality of the code.

The fact that your code works does not guarantee full marks. All code is expected to also be readable, maintainable, and efficient. You are not required to exactly follow the steps in the exercises above. Alternative designs are also acceptable if they can be justified in the viva. However, designs which substantially reduce efficiency or other desirable properties without corresponding benefit will lose marks.

In addition to final submission in Week 12, there will be two mandatory feedback sessions as follows:

- **Week 5(7/11/2022-11/11/2022):**
  Exercise 1 will need to be demonstrated.

- **Week 10(12/12/2022-16/12/2022):**
  Exercise 2 will need to be demonstrated.

During these two weeks, you must demonstrate your coursework and you will be given a preliminary mark, and detailed feedback. This preliminary mark does not count towards the module, but **your final mark for the first two exercises will be capped at double the preliminary mark (explained below**). This is to encourage you to attend the session and to work on the coursework over time rather than at the last minute.

- **The deadline for submission of the coursework is Week 12.**
- **In Week 12 you will also be required to demonstrate the final version of your work, and verbal feedback will be given.**

Exercise 2 needs to be demonstrated in Week 10, and not in Week 5, although general queries can be discussed.

Exercise 3 does not need to be not demonstrated in Weeks 5 or 10, although you may demonstrate it for feedback if you wish. Its contribution to the mark is not subject to the cap, although it will be difficult to score highly on Exercise 3 if you did not complete Exercises 1 and 2.

Capping Rule for Exercises 1 and 2

Assume that your preliminary mark for Exercise 1 is X/20, and that for Exercise 2 is Y/40. During your final demo in Week 12, your marks for Exercises 1 and 2 will be capped at Z/60 as follows:

$$Z = \min(2*(X+Y),60),$$

Example: if you score 20/60 on exercises 1 and 2 at the interim assessment, then at the final assessment the most you can score for exercises 1 and 2 is 40/60. You could also still get full marks (40/40) on exercise 3, meaning your total mark would be 80%. Also note that the mark is *capped*, not *scaled down*, so to increase your score to 40/60 on exercises 1 and 2 you would only need to get an unadjusted score of 40/60.

This method is more generous than a separate mark for the interim assessment in more than 70% of cases. We do not recommend attempting to limit your work to score no more than a given maximum mark, and staff will not advise you on how to do this.

Notice on presentation and submission

You do not need to give a presentation nor submit a report for either section of the coursework. This coursework's focus is on the quality of your final code and on your ability to understand it, not your software engineering process (which is not expected to be standard when you are learning the language).

Standard rules on plagiarism apply to this coursework.

The Code should be your own work and must not be copied from the internet or any other source. If you have difficulty with the coursework, you should approach your practical tutor in the first instance. Posting questions about the coursework on Stack Overflow, Quora, or similar sites may be treated as an incitement to plagiarism. Posting parts of your answer to the coursework on the publicly available internet where other students may access it will be treated asan incitement to plagiarism. Soliciting or obtaining answers to the coursework in exchange for money and any other consideration will be treated as serious academic misconduct. Asking for coursework answers from any party outside of the University is itself attempted plagiarism and you should not do it; if that third party commits any of theoffenses in this section on your behalf, you may be held responsible, even if you were not directly aware they would do so (because you should not have asked them in the first place).

Assignment Data

Contact person	Leon Liang, leon@zy.cdut.edu.cn
Learning outcomes	See below.
Formative deadlines	Week 5 (7/11/2022-11/11/2022): Exercise 1 (Demonstration) Week 10 (12/12/2022-16/12/2022): Exercise 2 (Demonstration)
Formative feedback	Week 5 (7/11/2022-11/11/2022): Exercise 1 (Spoken interactive) Week 10 (12/12/2022-16/12/2022): Exercise 2 (Spoken interactive)
Summative deadline	Week 12 (26/12/2022-30/12/2022) Demonstration and Submission
Summative feedback	Week 12 (26/12/2022-30/12/2022) Spoken interactive Final marks after assessment committees
Assignment Weighting	50% of module

Learning Outcomes

- Understand the fundamental concepts of C and C++ programming for object manipulation, data structuring and input/output control.
- Refine a problem specification into a collection of C++ classes.
- Create a software artefact specified in terms of C++ objects and their interrelations.
- Research the techniques for safe and efficient programming in C and C++.