XJCO 1921- Programming Project - Coursework 2

This work is the second coursework for this module. It corresponds to **70**% of the overall assessment for this module. Submissions should be made via Gradescope.

This work has two submission deadlines:

Submission Deadlines:

1. *Planning Report (20 marks):* 11 AM GMT on Monday 19 April 2021
2. *Project implementation (50 marks):* 11 AM GMT on Monday 3 May 2021

Project Implementation [50 marks] – Due on Monday Week 10

You are asked to implement one of the four topics described below using the C programming language. Use a git repository host like GitHub or Gitlab for version control throughout the project life cycle.

Important notes:

- You should create a new repository for this coursework.
- There is code online for the project topics, some in C, some in other languages. If you choose to use some other code or libraries as a starting point, you must include the reference to this code in your planning report. Make sure you attribute any publicly available code that you use to its proper sources.
- You must also be clear what the code you have written and what is code that you found online during lab marking (e.g., by commenting on the source code).
- You are expected to apply the modular and testing programming techniques taught in the module. A good implementation should have a dedicate (regression) test suite.

Project Topics

Project 1 – A Chatbot

A chatbot is a program that tries to have a conversation with you. When you provide the Chatbot with some inputs in Natural Language (English, Chinese, etc.), it responds with something meaningful in that same language. A classical chatbot is Eliza – Try it at http://psych.fullerton.edu/mbirnbaum/psych101/Eliza.htm

What does it involve?

You will write a program that can have a conversation with the user. This does not have to be perfect - even the best chatbots are not very good at this.

There is an open problem in Artificial Intelligence called the Turing test, with a famous prize, if your Chatbot is perfect. There are 2 parts to the project:

1. Taking input from the user and trying to understand it
2. Writing a reply to the user that makes sense and display the reply.

In both cases, you will have to decide how to do this.

Notes: You can use String operations to parse the input and design your own functions to decide how to understand the user and input and how to respond.

You can make rules for your program or try to search for patterns in the user input.

Project 2 – Route finding

This project asks you to compute the best path between 2 points on a map.

You are given a large data set (to be downloaded from Minerva) that represents all the footpaths on the Leeds University Campus that you can use to test your application

What does it involve?

Data input from a file - the data file is quite complex, and you have to read it in

Data structures - you have to decide how to store the data, there are suggestions below

Algorithms - you have to create a function to find a path between 2 points, there are standard algorithms for this that you read about

More information

Given a data set in the form of a set of locations (Nodes) and paths connecting them (Links) you should implement an algorithm to compute the best route between 2 given locations.

"Best" could, for example, be the shortest route, but you may wish to consider other measures.

The data set is attached in the file 'Final_Map.map' and represents footpaths on campus. A jpeg image, also attached, shows what the path network looks like.

The following requirements are essential:

1. You should create a suitable data structure for the problem - a suitable candidate would be an Adjacency List which stores a list of points in the data, and for every point a list of points that are connected to it with edges.

https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs

2. You should consider how to import the data into your data structure.

3. You should consider a suitable visualisation of the data - for example, edges can be plotted in Gnuplot (see attached jpeg).

4. You should implement at least one algorithm that finds a path between 2 given points.

Notes

The data file contains several lists of different types of data.

The most basic is the "node" which is a point in space with coordinates defined by (lat,lon) - you can use them as (x,y). Each node has a defined "id".

The next is the "link" which is a line defined by 2 node id's.

Those 2 data-types are enough to define the map of the network. i.e. you can plot each link to get the picture I attach above.

There is extra data in that file that you can use or ignore.

There are several ways to read in the data file. One way is to use fgets() and sscanf().

Project 3 – The Game of Life

In this project, you will build the famous and fascinating system known as "Conway's Game of Life". Life is a "cellular automaton" - a system of cells that live on a grid, where they live, die and evolve according to the rules that govern their world. Life is simple - elegant rules give rise to astonishingly complex emergent behaviour.

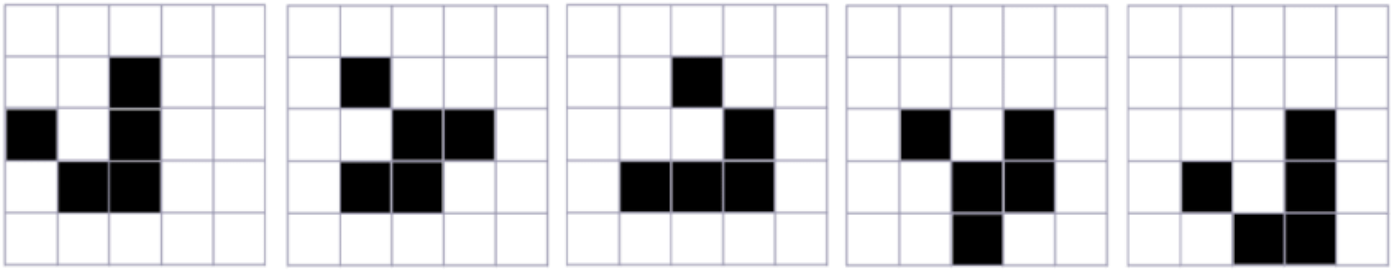
The Game is played on a 2-D grid, divided into "cells". Each cell is either "dead" or "alive" at a given "generation." The Game consists of a set of rules that describe how the cells evolve from generation to generation.

These rules calculate the state of a cell in the next generation as a function of the states of its neighbouring cells in the current generation. In a 2-D world, ***a cell's neighbours*** *are those 8 cells* ***vertically, horizontally, or diagonally adjacent to that cell****.*

Conway's set of rules are summarised as:

1. Any live cell with 0 or 1 live neighbours becomes dead because of underpopulation
2. Any live cell with 2 or 3 live neighbours stays alive because its neighbourhood is just right
3. Any live cell with more than 3 live neighbours becomes dead because of overpopulation
4. Any dead cell with *exactly 3 live neighbours* becomes alive by *reproduction*

Example (the so-called "glider"):



What does it involve?

In this project, you are asked to implement Conway's Game of Life, with the minor restriction that our 2-D world is finite. The neighbours of a cell on the edge of the world that would be beyond the edge are assumed dead.

The initial state of the world should be read from a file, and the final state should be output to a file. You will need to decide the format of the file.

The size of the world should be configurable and is defined in the initial state file.

The number of steps could be taken from the user at the beginning of the program, or the program keeps evolving until it is terminate (hence no predefined numbers of steps).

The final state of your program should be saved into the state file so that it can be restored during the next run.

You will need to decide how to present the world and the evolutionary process.

A good (1st) implementation would require you to visualise the Game graphically and display the animations of each evolution step. The delay used for animations could be predefined in the initial state file.

Graphics library

The SDL library could be a good choice for graphic rendering, but you can use other graphic libraries too.

SDL - the Simple DirectMedia Layer

SDL is a high-level library giving access to the multimedia hardware on your device.

A Tutorial of SDL can be found at https://wiki.libsdl.org/Tutorials

There are many existing applications online that use SDL - if you choose to base your code on one, you must be able to clearly distinguish your work from the original code.

Note that many SDL applications use C++; **this course requires a C implementation.**

Installing SDL yourself

For Linux and Mac, it is easy to install SDL from source and compile on your machine.

It is described on the wiki and is quite simple.

Download the C source code and extract.

Run './configure','./make','./make install' (as root) and it should work.

Some extensions, e.g. SDL_image, have to be downloaded and built separately.

Building code with SDL:

They can be compiled directly into your code using the standard approach for libraries, as required

ie. `-lSDL2 -lSDL2_image -lSDL2_ttf`

Example: An example of SDL is provided on Minerva.

Planning Report [20 marks] – Due on Monday Week 8

As part of the project submission, you are required to submit a planning plan. Due to the limited time frame, plan your project carefully. The report submission deadline is designed to force you to consider the scope and design of the project carefully before implementing the project.

Specifically, you should write a design and test plan for your chosen project with the following structure:

Title

A chosen title for your project, your name, and your student number

Summary

A short description of your project. Which project choice it is? What your project will do?

List of the key modules and a single sentence describing their purpose (~300 words) .

Test plan

A clear statement of testing methodology, and how will you test your application? (~200 words).

What are the tests you can design for each iteration of your project? What are the tests for each module?

List of all the tests with description. There are a number of ways for documenting tests. An example of the expected description is shown in the appendix at the end of this document.

Schedule (~ 0.5 page)

It is important that you have a realistic plan for how much time you will spend on the project and on each iteration of the design.

Write a plan (in tabular form) for what will you do each week until the project is submitted. This should include time off required for revision and exams and your holidays.

For each week, write 1-2 sentences describing what you will do.

Notes:

The report should be written in the style of the technical report. Please be concise. I have indicated advisory page counts though these are not strict.

You do not need to include your code as part of the report. Snippets may be used if appropriate, but only if illustrating a specific point.

Submissions

There are two submissions for this coursework:

The planning report is due at 12 PM GMT on Monday, 19 April 2021.

The project implementation is due at 12 PM GMT on Monday, 3 May 2021.

You should follow the instructions below on how to prepare your submission. Late submissions are accepted up to 7 days late. Each day, or part of a day, will incur a 5% penalty.

Planning Report

You need to submit two copies of your report. Submit a hardcopy to the SWJTU lecturer/TA/teaching office and an electronic copy of your report as a single PDF file to the Turnitin submission portal on Minerva. You need to submit both copies by 12 PM GMT on Monday, 19 April 2021. Checks for plagiarism and collusion will be carried on for the report.

Project Implementation

Submit your entire git repository (containing your code, **regression test suite**, and a Makefile or CMakeList.txt for building the program), along with a **ReadMe.txt** file containing (1) the URL of your git repository, (2) a screenshot of your git commit history and (3) instructions on how to run your program, **all in in a single zip (.zip or .gz) file** through Gradescope.

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 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 Gradescope.

Project Demonstration

- The project implementation will be marked during a lab session after the submission deadline.
- The projects will be evaluated on the correctness of code and ingenuity.
- During marking, you will be asked to reflect on your project development. What challenges you faced and how you addressed them and what you learned from this exercise?
- You will be asked to demonstrate your work ***from your Gradescope submission***.
- You need to demonstrate your work to a member of the course team, failing to do so will result in 0 marks.
- You need to come to the marking session with your exercise completed. We will not be able to provide support for this exercise during marking.
- You should be able to explain what you have done clearly, to show that you understand the concepts introduced.
- Checks for plagiarism and collusion will be carried out on all work.

Outline Marking Schemes

Outline marking schemes for the planning report and project implementation and demonstration are given below.

Planning Report [Total: 20 marks]

Report Section	Mark distribution (out of 20)	Criteria
Summary	2	• Project aims and objectives are clearly defined project deliverables properly listed
Testing Plan	10	• A proper testing and evaluation plan was devised and clearly documented • Test cases have covered the different functionalities of the program • Test cases should also cover the edge cases and error handling • Each test case is clearly defined with the input and expected output and behaviour
Schedule	5	• The project schedule is realistic, with clear milestones for measuring progress
Writing Quality	3	• Referencing is valid, and the numbering of figures and tables is correct and consistent. • The language is sound, and the sentences are clear and easy to understand. • The report is free of grammatical and typographical errors.

Project Implementation (Marking for Lab Demo) [Total: 50 marks]

Assessment Criteria	Mark Distribution (out of 50)	Level of attainment (circle)				Notes
Code Compilation	5	A	B		F	A: Program can be built and compiled using make or cmake (100%) B: Program has to be manually compiled and linked (60%) F: Code does not compile (0%)
Functionalities	20	A	B	C	F	A: Program can run and meet all requirements set by the project description (100%) B: does not meet some of the requirements but fail on some of the edge cases (60%) C: a fair attempt, but many problems (30%) F: Not working (0%)
Testing	10	A	B	C	F	A: Have a well-designed, dedicated test program set/inputs (100%); Have a regression test suite; correctly using static and const modifiers for static checking; B: Test cases only cover standard cases but do not cover special edge cases that many (60%); The test suite only contains very few test cases. C: Very few test cases hard-coded in the source code (30%); Don't have a regression test suite. F: No test case is implemented (0%)
Quality of Design	5	A	B	C	F	A: elegant and complete design, e.g. follow modular development by correctly using static and const modifiers and make sure the code is modular and well structured; clear and meaningful messages for error handling (100%). B: okay and complete design but the code structure can be improved; programming style is generally good; error messages can some times be confusing (60%). C: complete or almost complete design, but not very elegant; e.g., does not apply modular design (30%) F: incomplete solution, i.e., key functionality not implemented (0%)

Version Control	5	A B C F	A: Use version control throughout with clear code revision history throughout the project duration: meaningful commit messages and issue control (100%) B: Use version control, but code revisions only happened within the final week of the submission deadline (60%) C: Have a git repository but only pushed 1 or 2 code commits, or the code repository is incomplete (30%) F: Did not use version control (0%)
Critical reflection	5	A B C F	A: High level of self-reflection, able to discuss in detail the pros and cons (100%) B: good level of self-reflection, sees strengths and points for improvement (60%) C: limited self-reflection, possible to tease out interesting points with questioning (30%) F: cannot answer (0%)

Appendix

Example test case description

Using Coursework 1 as an example

Function: `int load_books (FILE *file, BookArray* a);`

Expected behaviour:

- result == 0
- Loads the database of books from file and into an array in the program

Assertions:

- The file pointer "file" is not NULL
- The Bookarry buffer is not NULL

Test cases:

C1:

Input:

A normal file handler and a benign BooKarray buffer pointer,

Expected results:

Stored the data into BookArray buffer, return 0

C2:

Input:

A NULL file handler and a benign BooKarray buffer pointer,

Expected results:

Display an error message and return -1

C3:

Input:

A NULL BooKarray buffer pointer and a benign file handler

Expected results:

Display an error message and return -1

C4:

Input:

A NULL BooKarray buffer pointer and a NULL file handler

Expected results:

Display an error message and return -1