COMP0002 Principles of Programming

C Coursework

Submission: The coursework must be submitted online using Moodle by 4pm Wednesday 11th November 2020.

Aim: To design and implement a longer C program of up to several hundred lines of source code in length.

Feedback: The coursework will be marked and returned by 2nd December 2020.

The coursework is worth 20% of the overall module mark, and will be marked according to the UCL Computer Science Marking Criteria and Grade Descriptors.

On this scheme a mark in the range 50-59 is considered to be satisfactory, which means the code compiles and runs, does more or less the right things and has a reasonable design using functions. Marks in the ranges 60-69 and 70-79 represent better and very good programs, while the range 40-49 denotes a less good program that shows some serious problems in execution and/or design. A mark of 80-89 means a really outstanding program, while 90+ is reserved for something exceptional. A mark below 40 means a failure to submit of sufficient merit. This follows the department's Marking Criteria and Grade Descriptors document, a copy of which is on Moodle .

To get a mark of 70 or better you need to submit a really very good program. Credit will be given for novelty as well as quality.

Getting a good mark: Marking will take into account the quality of the code you write. In particular pay attention to the following:

- Proper declaration and use of functions, variables and data structures.
- The layout and presentation of the source code.
- Appropriate selection of variable and function names.
- Appropriate use of comments. Comments should add information to the source code, not duplicate what the code already says (i.e., no comments like "This is a variable"!).
- As much as possible your code should be fully readable without having to add comments.
- Selection of a suitable design to provide an effective solution to the problem in question.

Clean straightforward and working code, making good use of functions, is considered better than longer and more complex but poorly organised code.

Development Advice:

- Keep things straightforward!

- Keep things straightforward! (Very important so it is repeated!)
- Straightforward does not mean trivial.
- First brainstorm/doodle/sketch to get a feel for the program you need to write and what it should do.
- Don't rush into writing C code if you don't fully understand what variables or functions are needed. Don't let the detail of writing code confuse your design thinking.
- How is the behaviour of the program implemented in terms of functions calling each other?
- Role play or talk through the sequence of function calls to make sure everything makes sense.
- Are your functions short and cohesive?
- Can't get started? Do a subset of the problem or invent a simpler version, and work on that to see how it goes. Then return to the more complex problem.

What to Submit

Your coursework should be submitted on Moodle, via the upload link for the C Coursework. The upload will permit a single file to be uploaded, so you should create a zip archive file containing all the files you intend to submit and upload that. Please use the standard .zip file format only, *don't* use any other variant or file compression system.

The zipfile should be named COMP0002CW1.zip.

The zipfile should contain:

- a directory named src containing the application source code (including any sub-directories used).
- a directory containing any data files your program might use (if needed). Make sure that the data files can be accessed use file names relative to the location where the executable file is created. Don't use absolute path names in your code (e.g., C:\Documents\MyProject\ etc., as this will only work on your machine unless the code is edited).
- a directory named Documentation a readMe file (see below).

Submit source code files, any data files, and the readMe file only, don't submit compiled code (binary code) such as .o files or executable programs.

The readMe should include the following:

- A concise description of what the application does, and how to use it. You can use screenshots to help explain your program.
- The commands needed to compile and run the program.

This should be 2 pages at the very most.

Plagiarism

This is an individual coursework, and the work submitted must be the results of your own work. You can ask questions and get help at the Lab sessions in Week 5. Also you can make use of the example PhoneBook code provided on Moodle without referencing it, but you should not simply submit a copy with just a small number of modifications.

Using comments in your source code, you should clearly reference any code you copy and paste from other sources, or any non-trivial algorithms or data structures you use.

See the UCL guidelines at https://www.ucl.ac.uk/ioe-writing-centre/reference-effectively-avoid-plagiarism/plagiarism-guidelines.

What Program to Write

Choose *ONE* of the questions below to answer (not two or more of them!).
The question descriptions are meant to be fairly open-ended, providing scope for the idea to be extended as you feel able. Beware of getting too ambitious too early. Get something basic working first and progressively extend it.

A good answer to a standard level question can certainly get a good mark, such as 70 plus. Of course, you still need to design and write a good quality program, and go beyond the basic specification given in the questions. Intermediate and challenge level questions also need good answers to get higher marks, just because the problems are harder doesn't automatically mean a higher mark. The risk is that the problems are too hard to complete at this stage of learning to program, or you end up with poor quality code.

Standard level questions
The first two questions are deliberately similar in style to the example Phone Book program that is on Moodle. It is entirely acceptable to use that code as a basis for answering these questions.

1. Write a program to manage the inventory for a small business. It should be possible to at least enter an item (name, barcode, price), record the number in stock, update the number in stock, search for an item, produce a report of all items in stock and delete an item. The data should be stored in one or more files.

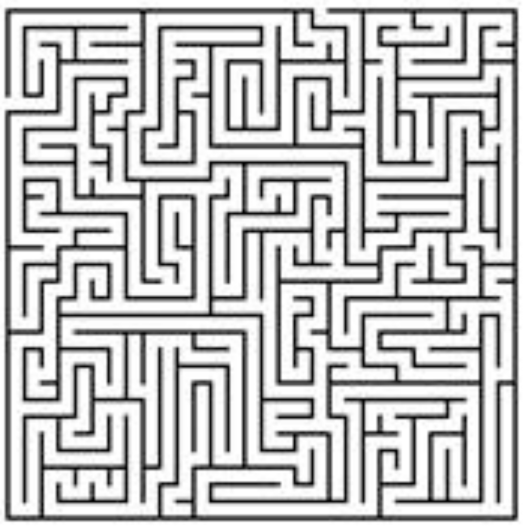
2. Write a diary program, organised by day, week, month and year. It should be possible to add appointments to any day and allow for repeated appointments. The program should also be able to display a week or month neatly. It should also be possible to search the diary. The data should be stored in one or more files.

Intermediate level questions
3. Write a program that can parse and evaluate mathematical expressions, written in a textual form (e.g., SQRT(10 * 11) + SQR(5)). Simple expressions might make use of numbers and functions such as log or sin, while more complicated expressions might involve variables.

4. Write a program to perform Matrix operations, including addition, subtraction, multiplication, and any other matrix operations you want to add. You will need to identify a data structure to represent a matrix, most likely based on a 2D array, or array or arrays. It should be possible to load and save matrices to and from data files, with the data in text format.

5. Write a drawing program that can read a data file and draw a graph to display the data in the file. The data file should specify the kind of graph (line graph, bar graph, pie chart, etc.), the size, how the axes are labelled, how data points are marked and the data set to be displayed. Support a range of styles and formatting, for example 3D charts.

6. Write a drawing program that can generate mazes, for example one like this:



There should be options that allow the size, complexity, visual style and other attributes to be selected. It should also be possible to show the path through a maze.

Challenge questions

7. Write a program to find the route between any two stations on the London Underground network. You can extend this to include London Overground, the DLR, Thameslink, and other services if you have time. The user should be able to select a start and destination station and the program then finds the route or routes between them, printing out travel instructions such as which line to take, where to change lines, and so on. You might want to include travel time, the fastest time, the journey with the fewest changes, and so on.

8. Write a Unix shell program, like bash, adding as many features as you can. A shell program is what runs inside a Unix terminal window to display the prompt and interpret commands when they are typed in. See the bash documentation online to find out what a shell is and does. Bash also supports scripting via the bash scripting language (like a programming language), which you might want to add.