SOFT153
Algorithms, Data Structures and Mathematics

Coursework

Hand-in Date: 12 noon, Monday 14th May, 2018

Hand in via the "Algorithms Coursework Submission" Option on the SOFT153 DLE

Module Leader: Thomas Wennekers
thomas.wennekers@plymouth.ac.uk
School of Computing, Electronics and Mathematics

This assignment accounts for 50% of the module coursework mark; the other 50% are for the Maths part.

The assignment is to be done individually.

Format of coursework to be handed in:

Hand in a single file. This can be a zip-file, if more than one document needs to be included.

For tasks requiring text or figures use .doc, .docx or .pdf document formats. As long as they are clear and legible, figures can be drawn by hand and scanned, or using any graphical tool available. They should be in PDF format or directly included in a document file.

For tasks requiring coding provide **a single C#** file containing your program, *suitably laid out and commented*. Don't submit a full C#-solution (compare with the examples handed out in the tutorials). The code should run without modification, otherwise marks will be reduced.

Don't code graphical user interfaces -- use a console application. GUIs will not attract more marks, but very likely make the code less readable. This would reduce marks.

Don't use programming constructs on top of the very basic "C-core".
Keep your code simple and clear. There are plenty examples in the lectures and practicals. If in doubt ask in the practicals.

Late assignments policy
You are reminded about the University's policy for late assignments. You can submit for up to 24 hours after the hand-in date, but in this case the assignment can only receive a maximum of pass-level marks (40%). Later assignments will automatically be awarded zero. You are, therefore, encouraged to submit the assignments prior to the hand-in date. The SCOLAR system allows to replace previously submitted files as long as the hand-in date has not passed.

This assignment assesses the following learning outcomes:

- Recognise and explain the importance of algorithmic design in optimising use of computing resources.
- Identify suitable structures and algorithms to implement programming tasks.
- Synthesize the solution to a real-world task as a combination of two or more standard algorithms

Plagiarism: students are warned that the University takes plagiarism very seriously. You are advised to read the section in your student handbook on Examination and Assessment Offences. As a brief guide, plagiarism is the deliberate use of another person's work without attribution or acknowledgement. DO NOT construct your assignments from code taken from another student or from the Internet. There is no objection to you discussing your solution with other people, but the code you hand in must be your own.

Please read carefully the specifications for the tasks given below – you will loose marks if your programs do not do what is specified.

Mastermind is a code-breaking game for two players. If you don't know it, read the Wikipedia page about it or ask questions in the practicals. This assignment is about implementing mastermind in software.

In the original real-world game one player A selects 4 pegs out of 6 colours and puts them in a certain fixed order; multiples of colours are possible (for example, red green red green). His opponent B does not know the colours or order, but has to find out the secret code. To do so B makes a series of guesses, each evaluated by the first player. A guess consists of an ordered set of colours which B believes is the code. The first player A evaluates the guess and feeds back to B how many positions and colours are correct. A position is correct ("black") if the guess and the secret code have the same

colour at it. Additional colours are correct ("white"), if they are in the guess and the code, but not at the same location. For example
secret: red green red green
guess: red blue green purple
results in one correct position ("black = 1") for the red peg at position one and one additional correct colour ("white=1") for the green peg in the guess. Note that one "green" in the guess matches only one of the two "greens" in the secret code.

There are versions of mastermind with more or less positions than four, and more or less colours than six. Instead of colours use numbers in your code; ie., the codes should be stored in N-element arrays of integers in the range from 1 to M.

The task: Write code that implements mastermind with N positions and M colours (encoded by numbers from 1 to M). N and M should be parameters of the code; the code should work for any positive choice of N and M (up to some maximum value of, say, M=9). The code should take the role of player A in the description above, such that the user has to guess a secret code (role B); ask the user for N and M at the beginning; and then enter a main loop that iteratively starts new games until the user indicates an exit-condition (the code may ask the user if he wants to play again).

In a game the computer generates a random secret according to the chosen values of N and M, and then iteratively asks the user for guesses. A guess would be evaluated and the results printed, i.e., the number of correct positions "black", and number of additional correct 'colours' "white". The program would recognise if the secret code has been correctly identified, print a message, and start gain.

1a) Draw a structogram of how you want to solve the problem. The structogram is part of the assignment. Then implement the code.

Hint: Start with a coarse-level structogram just outlining the main loop, and rough blocks like user input, evaluation of the whites and blacks, and output.

If you get this program working using well structured, commented and laid out code, and a reasonable structogram this would reach pass-level.

1b) Add a "history" to the code in a) that stores the guesses in each step as a queue and prints them out at the beginning of each step for inspection and at the end of the game if the secret code has been broken.

Because it subsumes task 1a, if you solve 1b it suffices to submit only the program and structogram(s) for 1b.