EE273 - Engineering Design for Software Development 2

Week 4, Semester 1

Learning Outcomes:

- Ability to debug and test simple functional programs with loops.
- Ability to declare variables of specific type and recognise the role of typing within C/C++ programs.
- Ability to use logical and arithmetic operations
- Ability to write and call simple functions within a C/C++ program.
- Ability to use keyboard and screen input/output functions.

Preface

- The lab sheets are released one week in advance of the appropriate laboratory session.
- It is key that students prepare in advance of the actual laboratory sessions; reading through the lab sheets, mapping out draft solutions and indeed trying out sample code.
- Students are expected at arrive at each lab session fully prepared for the lab exercises; the lab exercises are not designed to be completed solely within the lab sessions but are designed to be completed by the end of each lab session if appropriate preparation takes place.
- Students are expected to maintain a logbook as part of the laboratory work – taking notes, recording code and reflective comments about their solutions and progress todate.

Part 1: Practising with Loops

These first, simple, exercises give you practise with the ideas introduced in lecture 3.

1) Create a simple program that uses a for loop to go through the 5 times table up to and including 5 x 10 and writes each value to console using `cout`. Remember that your program must have one main function. (It is up to you whether you implement the rest of the functionality inside main or inside another function).
   a) Compile the program with GCC and run it via the console.
   b) Using Visual studio, compile and run the program, ensuring that (when running the program) the output can be seen easily on the monitor.
   c) Do the task in question 1 again but this time using a `while` loop instead of a `for` loop.
2) Create a simple programme that is capable of printing (on screen) in DESCENDING order all numbers that is exactly divisible by 7 within the range 0-100. Run as previously using GCC and Visual Studio.

Part 2: Advanced Exercises

3) Modify the previous programme (part 2) such that it can out all numbers divisible by “A” within the range “B” to “C” in either ascending or descending order, where variables A, B, and C are set by the user at runtime using `cin`.
4) Taking the example developed in lecture 3, design, write and test a programme that can calculate the total number of prime numbers that exists between 0 and a given upper limit “X” where “X” can be set by the user at run-time. Determine the overall performance of your programme in terms of correctness, upper limits and run-time.

Part 4: Main Task - Preparation

You are required to write a program to play a “20 questions” game to identify a number thought of by the program’s user.

In a game of “20 questions”, one player has to guess what the other player has thought of – usually an object or a job or something, but in this case a number – by asking questions to which the second player can only reply ‘yes’ or ‘no’. The aim is to guess the number in as few questions as possible.

Before you start writing any code, you should think about the algorithm to implement the above. Write your algorithm in your logbook in words or as a flow chart.

Look at what you’ve written down and test the algorithm you have suggested just on paper to see whether it works. One of the things the program should do is make an initial guess and then make a new guess based on some the user’s response of ‘yes’ or ‘no’ to a question that the program poses.

- What kind of question should the program ask that (a) has just two possible answers: ‘yes’ or ‘no’, and (b) allows that answer to be used to narrow down the possibilities of what number the user might have thought of?
- If the right kind of question is asked, the program can use the answer to update what it thinks is the possible range of numbers the user has thought of, and then to repeat the same process to narrow down the range of possibilities further. In other words, to put the program’s main action in a loop that is simply repeated until the range of possibilities is narrowed down to just one number (which ought to be the number the user thought of!)

The program will need to keep track of the possible range of numbers and to update it based on the answers the user gives to each question. The upper and lower limits of the range might be tracked as the values of two different variables. An `if` statement can be used to perform different actions to change either one of these variables.

The program must keep track of how many questions have been asked, since only 20 are allowed and the program must exit after the 20th.

Part 4: Implementation

Only when you are pretty sure that your algorithm will work should you start coding it. When start coding the tasks, implement and test in stages. Develop a framework for the solution and compile, run and test in stages. Do NOT write in 5,10 or 20.. lies of code and expect them to compile, run or operate correctly straight away.

Your program MUST be written with at least two functions – the usual main function that a C++ program must have, and another that implements the main search and returns an integer representing the number of questions asked.

The program should ask the user to think of an integer between 0 and 100 and then, in the second function you write, the program should try to guess the number the user has thought of.

- The search function should make use of variables and loops – `for`, `while` or `do … while` – it is up to you which. (However, recall that the program should stop asking more questions after it has asked 20. It also doesn’t need to ask any more when it has guessed the number!)
- The program should identify questions to ask the user and ask the user to respond – via the keyboard using `cin` – ‘y’ or ‘n’. For example “Is the number you thought of equal to or less than 50?” Based on the answer, the program will ask another question (so you will need to use an if statement) until finally the program makes a guess at the number.

5) You should try to write your code to be as concise as possible, i.e. with as few lines of code as you can while still making the code readable and easy to follow. You should minimise repetition of lines of code and make full use of loops.

Recording Progress

- Print out listings of your programs and put in your logbook. Make a note in your logbook of anything that you found difficult but learned how to do (so that you can look it up again later in case you forget it). Make a note of the answers to all key questions.
- Check that your program can run under both Visual Studio and GCC. After this lab session, the formal requirement to run code under both the VS IDE and GCC will not be restated. Students should be however able to use both tools if required.
- Demonstrate the operation of your code with a colleague and test the program. How well does your program perform compared with theirs?
- Write some general comments about the laboratory.