EE273 - Engineering Design for Software Development 2

Week 7, Semester 1

Learning Outcomes:

- Ability to use pointers and references
- Ability to write and call functions within a C/C++ program and passing parameters by value, pointer and reference

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 to-date.

Part 1: Pointers

Create a program to implement a geometric progression (GP) where the *n*th term a_n is given by $a_n = a_1 r^{n-1}$ where *r* is the common ratio and a_1 is the starting term. To do this, implement a function that takes four arguments – the starting term, the common ratio and pointers to the 2nd and 3rd terms. The function should calculate the 2nd and 3rd terms and assign the values to the locations pointed at by the two pointers. (To do this, you will need to use the de-reference operator `*`).

The function you have written should be called from `main` (and so you should make use of the `&` operator in sending the correct values for the pointer arguments to your GP function). To verify that the function works, write some code in `main` to allow the user to choose the starting term and the common ratio and, also in `main`, to report the calculated 2nd and 3rd terms to the screen.

- Compile, Link and Run the program – demonstrate it working and with correct answers.

Part 2: Passing by Reference

Make your program from part 1 more general by using an array for different terms in the progression. To start with, in `main`, as well as the starting term and the common ratio, allow the user to choose how many terms to calculate in the progression. Use the `new` command to create an array of the correct size in which to store all the terms. (Refer back to earlier lectures for an example of use of `new`).

Modify your progression calculation function to accept four arguments, but this time where they are: the starting term, the common ratio, the array in which to store the terms (or, more strictly, a reference to it) and the size of the array. (Again, see earlier lectures for how to pass an array to a function). Use a loop to step through the array, calculating and assigning the relevant values of the progression each time. Modify `main` to show the resulting geometric progression on the screen.

- Compile, Link and Run the program.
- Use the test data from the earlier part and show that the results produced are correct. Can you find the limits of your code operation?

Part 3: Structures

Modify the solution to part 2, defining a structure called "`geometric`" that contains the starting value, the common ratio and the number of terms of the progression that will be calculated. The array to store the progression terms should remain a separate entity – created as in part 2. The program should be modified so that the user inputs the 3 key elements, a structure is created and initialised with the user input data. The programme then passes the structure to the progression calculation function should be modified to accept the `geometric` structure as an argument alongside the results array.

- Compile, Link and Run the program.
- Use the test data from the earlier part and show that the results produced are correct.
- Consider a revision to the programme where the `geometric` structure now includes a pointer to a results array so that the progression function only takes one input – a reference to a structure of type `geometric`.

Part 4: Save results to a file

Add some further code to main to ask the user for another integer value. Further modify the code in `main` that shows the resulting GP. In this extra code, write a message to the screen when the GP term is also divisible by the additional integer value entered by the user.

i) How will you test whether one value is divisible by another?
ii) Look up the `modf()`function at http://www.cplusplus.com/reference/clibrary/cmath/ Is this useful here? Might any problem arise as a consequence of rounding of a floating-point number, and how might you deal with it?

- Compile, Link and Run the program.
- Use the test data from the earlier part and show that the results produced are correct. Can you find the limits of your code operation?

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.
- 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 reflective comments about the laboratory and speak to the teaching staff to discuss your progress.