EE273 - Engineering Design for Software Development 2

Week 6, Semester 1

Learning Outcomes:

- Ability to open, read, write and close files.
- Ability to declare variables of specific type and recognise the role of typing within C/C++ programs.
- Ability to declare 2 dimensional arrays
- Ability to iterate through 2 dimensional arrays
- Ability to create arrays of arbitrary size
- 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 to-date.

Part 1: Matrix Multiplication Preparation

In this lab, the use of arrays for storing sets of data is introduced. You will also make use of what you practised in the previous lab on reading and writing files.

The main objective will be to use arrays for storing square matrices and then to write suitable code for multiplying two matrices together. This can be quite an awkward exercise to do, although it is a really quite a deceptively simple thing to state.

- Before you write any code, write an algorithm in your logbook for multiplying two square matrices. Show what loops you should use, what happens each time around the loop and how many times the loop should be executed.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}=\begin{pmatrix} 1a+2d+3g & 1b+2e+3h & 1c+2f+3i \\ 4a+5d+6g & 4b+5e+6h & 4c+5f+6i \\ 7a+8d+9g & 7b+8e+9h & 7c+8f+9i \end{pmatrix}$$

In solving the problems given over it may also be a good idea to develop a plan of how to solve then using intermediate steps – avoid the temptation to try and develop a final solution in one go….

Part 2: Matrix Multiplication Implementation

1. Create two text files, each containing 5 lines of 5 integers representing matrices (the integers should be separated by commas).
2. Write a program that
   - asks the user to specify the names of two files to be read;
   - reads in each file and assigns the values read to a two-dimensional array (a different array for each matrix). (The program can do this by assuming that the files contain 5 integers on each line of the file, the integers on each line separated by commas[1]).
   - treating each array as a matrix, multiplies the first matrix by the second one and assigns the result to a third two-dimensional array[2].
   - writes the third array to the console.
3. Check that your program is working correctly by creating more text files with different 5 x 5 matrices of integers in them, reading in pairs of matrices and multiplying them together. Have these files prepared in advance and ready to use. (Check that the multiplications are correct!) Run and test with files from a fellow student.

Part 3: Save results to a file

Modify the above program to save the multiplication result in a file. Note that the program should prompt the user to specify the file name – the destination file should **not** be hard coded. Compile, run and test the programme.

Part 3: Arbitrary matrix size and shape

Make the program more general by modifying it to read in any size (and shape) of matrix in each file (use the new operator to define the adequate array sizes). If the matrices have suitable dimensions – multiply them together; otherwise print an error message to the user.

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.

[1] A more flexible version of the program would read in matrices of any size and use the new function to create arrays of appropriate sizes.

[2] A more complete program would check that the matrices are of correct sizes to be multiplied together, and would give an error message if not.