National University of Singapore
School of Continuing and Lifelong Education
TIC1002: Introduction to Computing and Programming II
Semester II, 2019/2020

Tutorial 1
Structure, Nested Loop, Composite Data Structures

1. [Structure] Use the given partially completed file "structStruct.cpp" and attempt the following:
   a. Complete the **readLine()** and **printLine()** functions **using the given readPoint()** and **printPoint()** function.

   b. Trace the **readLine()** function by showing the memory snapshot for the following execution:

```
//other code not shown
int main()
{
     Line l;

      readLine(...... //use the correct parameter here );

}
```

   c. Write a new function for the **Line** structure that returns the length of the line. Give **two versions**:

Version 1 (pass by address) double length(Line* l)
Version 2 (pass by reference) double length2(Line& l)

   d. Using (b), write a function **compareLine()** that takes in two lines {L1, L2} and return result as follows {-1: L1 is shorter, 0: same length, 1: L2 is longer}.

```
int compareLine( Line* L1, Line* L2 )
```

   Show how you'll use the two versions in (c) differently.

2. [Nested Loop] Let us rise to the challenge posed in "Challenge: Matrix Multiplication" from lecture 2. For simplicity, let us assume **M = 2, P = 3, N = 2**. So, matrix A is M x P matrix (i.e. 2 x 3 2D array), matrix B is P x N matrix and the result matrix C is M x N matrix respectively.

   a. Write a function that calculates **one element of** the result matrix C . The function header is:

   ```
   void matmul_one_element( int A[][P], int B[][N], int C[][N], int i, int j)
   ```

   So, to calculate the result of $C_{1,0}$ (as shown in the lecture slide), we call:

   ```
   //Assume we declared the matrix as matA, matB and matC
   matmul_one_element( matA, matB, matC, 1, 0);
   ```

   b. Use (a) to write a complete matrix multiplication function.

   ```
   //You can assume M, N, P are predefined values
   void matmul ( int A[][P], int B[][N], int C[][N])
   ```

   [Hint: remember to printout meaningful debug message(s) at the right places to understand the working of your code].

   c. Identify the main obstacle (in terms of programming syntax) for generalizing the solutions in (a) and (b) to any M, N and P.

3. [Multi-Dimensional Array] Let us tackle the "Minesweeper Problem" from lecture for real. In the given "**minesweeper.cpp**", complete the following tasks:
   a. Complete the **print_field()** function, such that the entire playing field is shown. When a mine is printed, it should be printed as 'M' on screen. You can decide how a mine is represented in the array. [Note: your decision may be influenced by how you code (b) / (c)].

   b. Observe the **plant_mine()** function on how random number is generated in a C/C++ program. Proceed to complete the **update_neighbor()** function, which increase the mine count in the 8 adjacent cells. Note that this function does not use the invisible border technique.

   c. Write a **plant_mine_halo()** function to utilize the invisible border technique (also called the ***Halo*** technique) discussed in the lecture. Additional challenge: You need to keep the declaration of the field the same for the function parameter, i.e.

      ```
      void plant_mine_halo( int field[][MAXCOL], int nrow, int ncol)
      ```

      This design hides the internal design from the caller of the function, i.e. they are not aware that we are using a slightly bigger array to solve the problem **internally.** (hint: may need to transfer the field…..)