Week 11 - Introductory Programming Lab

Exercise 10: Two-Dimensional Array

Background

Just like a one-dimensional array is an ordered sequence of values, a two-dimensional array is an ordered sequence of one-dimensional arrays. By convention, we think of these 1-D arrays as being stacked vertically.

Array Number	Array					
0	1-D array	0	1	2	3	4
1	1-D array	3	7	2	1	4
2	1-D array	4	2	8	5	6

Technical details

Using 2-D arrays:

```
#include <stdio.h>

// we MUST specify the size for a 2-D array
// (unless you use pointers)
void printArray(int values[5][3]) {
    int i, j;
    for (i=0; i<5; i++) {
        for (j=0; j<3; j++) {
            printf("%i ", values[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int array[5][3];
    int i, j;

    // fill array with some values
    for (i=0; i<5; i++) {
        for (j=0; j<3; j++) {
            array[i][j] = 3*i + j;
        }
    }
    printArray(array);

    getchar();
    return 0;
}
```

C does not check that you are only accessing an array within the bounds of the array. If you ask for `array[-1][-1]` then the computer will tell you what was at that memory location, even though that memory does not belong to that variable!

Your task...

Write a program which creates a "game board". We will use this in the next exercise to write a tic-tac-toe game (also known as "noughts and crosses", "tick tack toe", "X's and O's"). If you're not familiar with the game, see Wikipedia's page on tic-tac-toe, or ask a lab instructor.

- Your game board should be 3x3. Each square in the board can either be empty (print a dot "."), or have an "X":

  ```
  . X .
  X . .
  . . .
  ```

- You *must* use a two-dimensional array to represent the game board in memory. (You can't have variables like `int squareUpperLeft, squareBottomMiddle; ...`)
- The board begins complete clear (all blank squares `"."`)
- The player may select any square by entering a number from the *computer keypad*. (your program should read it as an `int`, and you may assume that it is between 1 and 9 inclusive)
- Before each player takes a move, print out the current game board. You can remove board from the screen by doing `printf("\n\n\n\n\n\n\n\n\n\n");` 2 or 3 times.
- If the player tries to select a square that is currently occupied, print a warning message and ask them to choose again.
- Continue playing the game until all squares are filled.
- You *must* create formulae to deal with "number <=> row and column". You *may not* use `if` statements or a `switch...case` statement for these formulae, but you can use them elsewhere in the assignment.

- Hint: create formulae to deal with "number <=> row and column" for these three cases:

  ```
  (a)          (b)           (c)
  easy         mobile        computer keypad
  0 1 2        1 2 3         7 8 9              < row 0
  3 4 5        4 5 6         4 5 6              < row 1
  6 7 8        7 8 9         1 2 3              < row 2

  ^ ^ ^        ^ ^ ^         ^ ^ ^
  0 1 2        0 1 2         0 1 2
  column       column        column
  ```

  Your assignment needs to work with the *computer keypad*, but it is strongly recommended that you learn how to solve (a) and (b) before trying to do (c).

  - You may find the division `/` and modulus `%` (or "remainder") operators useful.
  - Solve case (a) first.
  - After solving (a), figure out how to transform the numbers in (b) into the numbers in (a).
  - Do the same for (c) into (b).

Show your work to a demonstrator/GTA.

Exercise 11: Tic-Tac-Toe

Background

This lab combines everything we've done so far. If you didn't understand any of the previous exercises, you may want to ask questions about them before beginning this one!

Technical details

Nothing new.

Your task...

Write a tic-tac-toe game (also known as "noughts and crosses", "tick tack toe", "X's and O's"). If you're not familiar with the game, see Wikipedia's page on tic-tac-toe, or ask a lab instructor.

- Use your "game board" from exercise 10. The same constraints apply, namely:
  - You *must* use a two-dimensional array to represent the game board in memory.
  - You *must* create formulae to deal with "number <=> row and column". You *may not* use `if` statements or a `switch...case` statement. You can use those commands elsewhere in the assignment.
- Modify your "print the game board" function to print out `.XO` as required.
- Your game should be human-vs-computer, and human moves first.

- Computer moves randomly. Note that it must select an unoccupied square; you cannot simply pick a number from 1-9 at random without checking!
- Before each player takes a move, print out the current game board. You can remove board from the screen by doing
  `printf("\n\n\n\n\n\n\n\n\n\n\n");` 2 or 3 times.
- After each player takes a move, you must check to see if anybody won the game.

  (hint: write a function that checks if player `X` won the game, then call this function twice, for player 1 and player 2)
  (another hint: there are 8 possible winning combinations; a player wins if one of 8 combinations of 3 square all have his mark. You can either write out all 8 combinations in full, or use three `for` loops -- test the three vertical wins, three horizontal wins, and two diagonal wins.)
  (final hint: begin work on this point by changing the rules of the game. Pretend that you can only win by going horizontally or diagonally across the middle. Solve this problem first; once you have it working for the "fake rules", just add in the other 6 winning combinations.)

(optional: instead of having the computer move randomly, try to make it play intelligently.)

Show your work to a demonstrator/GTA.