ECE15: Lab #4

This lab is a cumulative wrap-up assignment for the entire course. As such, it relates to the material covered in Lecture Units 1 – 5 and 7 – 9 in class. Here are several instruction specific to this lab:

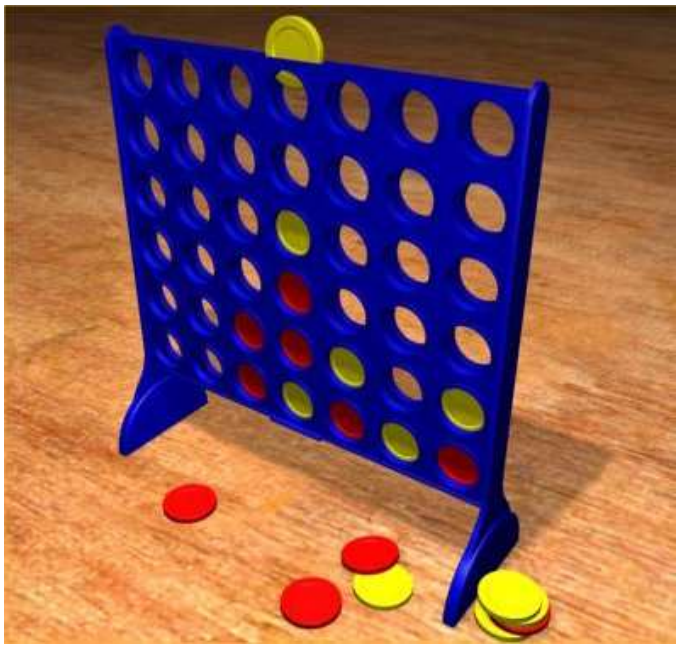
- In this lab, you are ***given*** the files `connect4.c` and `connect4_functions.h`, which you can download from the course website. The file `connect4.c` contains the **`main()`** function, while the file `connect4_functions.h` contains the forward declarations of the functions you are required to implement. You should implement these functions in the file `connect4_functions.c`, which is the ***only*** file you will be submitting in this lab. As a starting point, you should use the template for `connect4_functions.c`, provided on the course website at `http://ece15.ucsd.edu/Main/Homeworks.html`.
- Make sure that the file `connect4_functions.c` that you submit compiles and executes correctly with the files `connect4.c` and `connect4_functions.h` provided on the course website. You are ***not allowed*** to change (edit) these files in any way. In fact, you cannot change them: if you do, your changes will ***not*** really ***take effect***. The first step in grading the file `connect4_functions.c` that you submit will consist of compiling it together with the ***original versions*** of `connect4.c` and `connect4_functions.h`.
- This lab includes an extra-credit component. In order to be eligible for extra credit, you must implement the function **`best_move()`**. The function **`best_move()`** that you submit will be tested to verify that it satisfies several conditions described in detail on page 10. All the **`best_move()`** functions that pass these tests will play against each other in a round-robin competition, with the winner of the competition awarded 100 extra-credit points.
- In this lab, you ***can use*** any functions you like from the C standard library. The header files `<stdio.h>`, `<stdlib.h>`, `<time.h>`, and `<stdbool.h>` are already included in the file `connect4_functions.h`. If you would like to use standard library functions other than those declared in `<stdio.h>`, `<stdlib.h>`, `<time.h>`, you should **`#include`** the corresponding header files in the file `connect4_functions.c`.
- Try to make your code easy to follow and understand: choose meaningful names for identifiers, use proper indentation, and provide comments wherever they are useful. While this is not required, it could help you receive partial credit: C code that is not correct *and* is also difficult to understand will receive zero credit, even if it is partially correct.

This lab will be graded out of 100 points, with up to 100 extra-credit points. You should submit the lab by following the instructions posted at `http://ece15.ucsd.edu/Labs/TurnIn4.html`.

Problem 1.

The goal of this problem is to implement the ***Connect-Four*** game between a computer and a human player. Throughout this lab, we will refer to the human player as "Alice" for short. A brief description of the Connect-Four game follows. You can find much more information about this game on the web, for example at `http://mathworld.wolfram.com/Connect-Four.html`. You can actually play the game against a computer at `www.mathsisfun.com/games/connect4.html`.

Rules of the Game: Connect-Four is a two-player game played on a vertical board 7 columns across and 6 rows high. If you were to buy the game at a toy store, the board might look like this:



Each player begins the game with 21 identical *stones* (alternatively called pieces, chips, tokens, etc.). The two players alternate in making moves. Throughout this lab, the stones of the player that makes the first move are marked by **`X`**, while the stones of the player that moves second are marked by **`O`**. Each *move* consists of a player dropping a stone into one of the seven columns. Because the board is vertical, stones dropped into a given column always *slide down to the lowest unoccupied row* of that column. A column that already contains 6 stones is *full*, and no other stones can be placed in this column (attempting to do so constitutes an invalid move). The objective of each player is to place four of his/her stones on the board so that they form a *horizontal, vertical, or diagonal* sequence. The first player that is able to do so *wins*, and the game stops at this point. If all the $7 \times 6 = 42$ stones have been placed on the board without either player winning, then the game is considered a *draw*. □

In this lab, the function **`main()`** that implements the Connect-Four game between Alice and the computer is ***given to you*** in the file `connect4.c`, which you should download from the course website at `http://ece15.ucsd.edu/Main/Homeworks.html`. As you will see (on the next page), the function **`main()`** calls five other functions, namely:

```
print_welcome(void)
display_board(int board[][BOARD_SIZE_VERT])
random_move(int board[][BOARD_SIZE_VERT], int computer_num)
player_move(int board[][BOARD_SIZE_VERT], int player_num)
check_win_or_tie(int board[][BOARD_SIZE_VERT], int last_move)
```

You are required to implement these functions in the file `connect4_functions.c`. Note that forward declarations of these functions are also ***given to you***, in the file `connect4_functions.h` which you should download from the course website. Therefore, you *cannot change* the function prototypes — that is, the name of the function, its return type, and the type(s) of its parameter(s). Descriptions of these five functions follow shortly, but first here is the function **`main()`** from `connect4.c`.

```
int main()
{
   int board[BOARD_SIZE_HORIZ][BOARD_SIZE_VERT] = {{0}};
   int player_num, computer_num;
   int last_move;

   /* Ask Alice if she wants to go first */
   player_num = print_welcome();
   if (player_num == 1) computer_num = 2;
   else computer_num = 1;

   /* If Alice wants to go first, let her make a move */
   if (player_num == 1)
   {
      display_board(board);
      last_move = player_move(board,player_num);
      display_board(board);
   }

   /* The main loop */
   while (1)
   {
      /* Make a computer move, then display the board */
      last_move = random_move(board,computer_num);
      printf("Computer moved in column: %d\n", last_move);
      display_board(board);

      /* Check whether the computer has won */
      if (check_win_or_tie(board,last_move)) return 0;

      /* Let Alice make a move, then display the board */
      last_move = player_move(board,player_num);
      display_board(board);

      /* Check whether Alice has won */
      if (check_win_or_tie(board,last_move)) return 0;

   } /* end of while (1) */

} /* end of main() */
```

Note that symbolic constants **BOARD_SIZE_HORIZ** and **BOARD_SIZE_VERT** are defined in the file `connect4_functions.h` as 7 and 6. Now, here is a description of the functions called by **main()**.

`int print_welcome(void)`

This function does not take any input. It prints a welcome message for Alice, and then asks her if she would like to make the first move. Here are a couple of examples:

```
*** Welcome to the Connect Four game!!! ***
Would you like to make the first move [y/n]:  N
```

```
*** Welcome to the Connect Four game!!! ***
Would you like to make the first move [y/n]:  y
```

The function then reads from **`stdin`** the input typed by Alice, and clears the input buffer (in case she entered more than one character) using **`while (getchar() != '\n')`**. If she typed **`n`** or **`N`**, the function returns **`2`** indicating that Alice will be the second player to move. In all other cases, the function returns **`1`**, giving Alice the advantage of the first move (the computer is a gentleman).

`void display_board(int board[][BOARD_SIZE_VERT])`

This function receives (a pointer to) the **`board`** array as input, and then prints the current state of the board to **`stdout`**. The function expects the value of every cell in the **`board`** array to be either **`0`**, **`1`**, or **`2`**, where **`1`** denotes stones of the first player (which should be printed as **`X`**) while **`2`** denotes stones of the second player (which should be printed as **`O`**). A **`board`** cell whose value is **`0`** corresponds to a place on the board that is not occupied by a stone of either player.

The print-out of the board to **`stdout`** should be formatted as follows. The width of every cell is three characters, and the stone occupying this cell (if any) is the middle character. Vertical lines separate between the cells in the same row, while the rows themselves are separated by a line of hyphens along with '+' characters. Right under the board, the function should print the indices of the columns, with each such index centered in its column. Here is an example:

```
+---+---+---+---+---+---+---+
| O |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| X |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
| O |   |   | X | X |   |   |
+---+---+---+---+---+---+---+
| X |   |   | X | X | O |   |
+---+---+---+---+---+---+---+
| O |   |   | O | O | X |   |
+---+---+---+---+---+---+---+
| O | X |   | O | X | O |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

`int random_move(int board[][BOARD_SIZE_VERT], int computer_num)`

This function receives (a pointer to) the **`board`** array and a player number (either **`1`** or **`2`**) as input. It then makes a *valid random move*. To this end, the function should generate uniformly at random an integer **`m`** in the range **`1,2,...,BOARD_SIZE_HORIZ`** using an (appropriately normalized) call to the **`rand()`** standard library function. It should then verify that this integer **`m`** constitutes a valid move, by calling the function **`is_column_full()`**. If **`m`** is a valid

move, the function should return **m**. If not (that is, if the **m**-th column is full), the function should repeat the process until a valid move is generated by **rand()**. Note that the function assumes that at least one cell in the array **board** is **0**; otherwise it enters into an infinite loop! Prior to returning **m**, the function should also update the state of the board by making the function call **update_board(board,m,computer_num)**.

int player_move(int board[][BOARD_SIZE_VERT], int player_num)

This function receives (a pointer to) the **board** array and a player number (either **1** or **2**) as input. The function prompts Alice to enter her move, reads from **stdin** the input entered by Alice, then clears the input buffer using **while (getchar() != '\n')**. If Alice enters anything other than an integer in the range **1,2,...,BOARD_SIZE_HORIZ**, the function should say "**Not a valid move. Enter a column number!**," then prompt Alice again to enter her move. If Alice enters an integer **m** in the range **1,2,...,BOARD_SIZE_HORIZ**, the function should verify that the corresponding column is *not* full by calling the function **is_column_full()**. If the column *is* full, the function should say "**This column is full. Try again!**" and again prompt Alice to enter a move. If Alice does enter a valid move **m**, the function updates the state of the board via the call **update_board(board,m,player_num)**, and then returns **m**. Here is an example of a call to **display_board()** followed by a call to **player_move()**:

```
+---+---+---+---+---+---+---+
| O |   |   | O |   |   |   |
+---+---+---+---+---+---+---+
| X |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
| O |   |   | X | X |   |   |
+---+---+---+---+---+---+---+
| X |   |   | X | X | O |   |
+---+---+---+---+---+---+---+
| O |   |   | O | O | X |   |
+---+---+---+---+---+---+---+
| O | X |   | O | X | O |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
Please enter your move: ???
Not a valid move. Enter a column number!

Please enter your move: -5 3
Not a valid move. Enter a column number!

Please enter your move: 4
This column is full. Try again!

Please enter your move: 7
```

bool check_win_or_tie(int board[][BOARD_SIZE_VERT], int last_move)

This function receives as input (a pointer to) the **board** array and an integer **last_move**, which is interpreted as the index of the column (in the range **1,2,...,BOARD_SIZE_HORIZ**) where the most recent stone was played. The function calls **check_winner()** to determine whether the game has been won by either player. If so, the function should print "**Player X won!**" or "**Player O won!**" and return **true**. If there is no winner, the function checks whether the game is a draw (no spaces left on the board). If so, the function should print "**Tie game!**" and return **true**. Otherwise, the function returns **false**, indicating that the game is not yet over.

You have probably noticed that (some of) the five functions described above call upon other functions, specifically the functions:

`is_column_full(int board[][BOARD_SIZE_VERT], int m)`

`update_board(int board[][BOARD_SIZE_VERT], int m, int player_num)`

`check_winner(int board[][BOARD_SIZE_VERT], int last_move)`

This is a common practice in programming, wherein a given task is broken into smaller and smaller components. Concretely, this means that you are required to implement the three functions above, in the file `connect4_functions.c`. The forward declarations of these functions are already given in the header file `connect4_functions.h`. Here is their description.

`bool is_column_full(int board[][BOARD_SIZE_VERT], int m)`

This function receives as input (a pointer to) the **`board`** array and an integer **`m`** which is expected to be in the range **`1,2,...,BOARD_SIZE_HORIZ`**. The function returns **`true`** if the **`m`**-th column of the board is full (already contains **`BOARD_SIZE_VERT`** stones), and **`false`** otherwise.

`void update_board(int board[][BOARD_SIZE_VERT], int m, int player_num)`

This function receives as input (a pointer to) the **`board`** array, an integer **`m`** which is expected to be in the range **`1,2,...,BOARD_SIZE_HORIZ`**, and an integer **`player_num`** which should be either **`1`** or **`2`**. It then updates the board by changing the appropriate entry in the **`m`**-th column from **`0`** to **`player_num`**. Note that the **`m`**-th column of the board is stored in **`board[m-1][]`**. Also note that the function must determine which *row* in the **`m`**-th column to update (using the rule that a stone dropped into a given column always *slides down to the lowest unoccupied row*).

`int check_winner(int board[][BOARD_SIZE_VERT], int last_move)`

This function receives as input (a pointer to) the **`board`** array and an integer **`last_move`**, which is interpreted as the index of the column (in the range **`1,2,...,BOARD_SIZE_HORIZ`**) where the most recent stone was played. The function then checks whether the placement of this most recent stone results in a win (four stones on the board forming a consecutive horizontal, vertical, or diagonal sequence). If so, the function returns the player number (either **`1`** or **`2`**) of the winning player. If there is no winner, the function returns **`0`**.

In addition to the eight functions described above, you are welcome to declare and implement other functions, although you are not required to do so. For example, you might wish to break down the function **`check_winner()`** into three functions that check for a winning sequence in each of the three possible directions: **`check_horizontal()`**, **`check_vertical()`**, and **`check_diagonal()`**. If you do implement additional functions, you should write both the definitions and the *forward declarations* of these functions in the file `connect4_functions.c`.

Compilation Instructions: Put all the files, namely `connect4.c`, `connect4_functions.h`, and `connect4_functions.c`, in the same directory (also called "folder" on some systems). If you are compiling with gcc, you should compile *only* the `.c` files. Use:

```
gcc -Wall -std=c99 connect4.c connect4_functions.c -o connect4
```

If you are using an integrated development environment (IDE), simply add all the files to your project. For example, in Pelles C, assuming your project is called `connect4.exe`, you should right-click on `connect4.exe` in the source tree on the right side of the IDE. Select the option "**`Add files to project`**" to add both `connect4_functions.c` and `connect4_functions.h` to the project.

Here is a sample run of the entire program, assuming that the executable file is called `connect4`. This run illustrates a situation where Alice wins the game. The user input is underlined, as usual.

```
/home/userXYZ/ECE15/Lab4> connect4
*** Welcome to the Connect Four game!!! ***
Would you like to make the first move [y/n]:  *?
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
Please enter your move: 4
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
Computer moved in column: 5
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X | O |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
Please enter your move: 4
```

```
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X | O |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Computer moved in column: 3
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   | O | X | O |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Please enter your move: ?
Not a valid move. Enter a column number!

Please enter your move: *4
Not a valid move. Enter a column number!

Please enter your move: 9 4
Not a valid move. Enter a column number!

Please enter your move: -4 4
Not a valid move. Enter a column number!

Please enter your move: 4 -4

+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   | O | X | O |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7
```

```
Computer moved in column: 3
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   | O | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   | O | X | O |   |   |
+---+---+---+---+---+---+---+
  1   2   3   4   5   6   7

Please enter your move: 4

+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   |   | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   | O | X |   |   |   |
+---+---+---+---+---+---+---+
|   |   | O | X | O |   |   |
+---+---+---+---+---+---+---+
   1   2   3   4   5   6   7

*****************************
* Player X won!!! Game over *
*****************************
```

Notes:

- Note that in **`board[m][n]`**, the first number **`m`** represents the column whereas the second number **`n`** represents the row. It is recommended to identify **`board[0][0]`** with the upper-left cell of the board, so that **`board[6][5]`** is the lower-right cell. Observe that **`board[m][n]`** should contain **`1`** if player **`X`** has a stone in this cell, **`2`** if player **`O`** has a stone in this cell, and **`0`** otherwise.
- The return type of the functions **`check_win_or_tie()`** and **`is_column_full()`** is **`bool`**. This data type was covered in Lecture Unit #3. It is defined in the standard library header file `<stdbool.h>`, which is included in `connect4_functions.h`. This header file also defines the constants **`true`** and **`false`** as **`1`** and **`0`**, respectively.

Extra Credit (up to 100 points)

To earn extra credit in this lab, you should implement a *winning strategy* for the Connect-Four game. The better your strategy, the more extra-credit points you are likely to receive. Specifically, you need to implement the following function:

```
int best_move(int board[][BOARD_SIZE_VERT], int computer_num)
```

in the file `connect4_functions.c` that you submit. A forward declaration for this function is already provided in the file `connect4_functions.h`. The function should be identical in its functionality to the function **`random_move()`** described on pp. 4 – 5, with one important difference: instead of generating a random move, the function should return the ***best move*** (that you can come-up with) in each position. Of course, if you would like your **`best_move()`** function to call other functions, you are welcome to do so: declare and define these additional functions in the file `connect4_functions.c`.

The function **`best_move()`** that you submit will be, first, tested to verify that it produces valid moves and correctly updates the state of the board (using the **`update_board(board,m,computer_num)`** function call). It will be then tested against the **`random_move()`** function a 100 times, with the requirement that it wins at least 66 of the 100 games. Another requirement is that you should *not* use global variables anywhere in `connect4_functions.c`, since global variables make cause a collision when functions written by different students are jointly compiled together.

All the **`best_move()`** functions that pass these tests will play against each other in a *round-robin competition*. Since making the first move gives a distinct advantage in the Connect-Four game, each pair of **`best_move()`** functions A and B will play each other twice: once with A making the first move, and once with B making the first move. The round-robin competition will be scored as in chess: for each game played, a win is 1 point, a draw is 0.5 points for both players, a loss is 0 points.

The **`best_move()`** function that takes first place (or ties for first place) in the round-robin competition will be awarded 100 extra-credit points. The **`best_move()`** function that places last in the competition will not receive any extra credit. All the other **`best_move()`** functions will earn extra-credit points prorated linearly according to their rank in the competition. For example, if 21 students take part in the competition and there are no ties, then first place will earn 100 points, second place will earn 95 points, third place 90 points, ..., 19-th place 10 points, 20-th place 5 points, and last place 0 points.