Project [6]: Functions

Due Tuesday, 10/30/2018, 11:59 pm

Project Goals:

The goals of this project are to:

- Get students familiar with the use of functions
- Show students how simple it can be to implement complicated-looking functions.

Important Notes:

1. **Formatting:** Make sure that you follow the precise recommendations for the output content and formatting: for example, do not change the text of the problem from "`Player 1 enter your selection [row, col]: `" to "`Player 1 selection: `". Your assignment will be auto-graded and any change in formatting will result in a loss in the grade.
2. **Comments:** Header comments are required on all files, for each function, and recommended throughout the rest of the program. Points will be deducted if no header/function comments are included.
3. **Restriction:** The use of `goto` statements anywhere within this program is prohibited. Points will be deducted if `goto` is used.

Problem 1

Write a program that implements the game of Tic-Tac-Toe where you can play against the computer. Player 1 will be the user and player 2 will be the computer. Your program should go through the following steps:

1. Generate an empty Tic-Tac-Toe board (3x3 array)
2. Run a loop until one of the players places three in a row (a player has won) or the table is full (stalemate). This loop should:
    a. Display the current layout of the table
        i. Blank spaces are displayed as an underscore
        ii. O for player 1's moves (user)
        iii. X for player 2's moves (computer)
        iv. Put spaces between the squares
    b. If it is player 1's turn (the user)
        i. Ask the user to enter their selection (the location on the board where the O should be placed (row, col))
        ii. Check to make sure that the row and column the user entered is valid
        iii. If the row or column player 1 entered was invalid (outside the bounds of the board or a space that is already occupied) the program should ask the user to enter the option again.
    c. If it is player 2's turn (the computer)
        i. Randomly generate a move (row, col) in the board that is not currently occupied (if the computer selects and occupied space generate a new move)
        ii. To generate a random move: generate two random integers, one for the row and one for the column, each between 0 and 2
        iii. Print out player 2's move

iv. Update the board with player 2's move

3. If the above loop ends because one of the players has placed three in a row then the program should print out a winning message of that player (see below). Else, it should print out the follow: "`Game over, no player wins.`"

The program should function as follows (items underlined are to be entered by the user):

```
The current state of the game is:
_ _ _
_ _ _
_ _ _
Player 1 enter your selection [row, col]: 1,2
The current state of the game is:
_ O _
_ _ _
_ _ _
Player 2 has entered [row, col]: 1,3
The current state of the game is:
_ O X
_ _ _
_ _ _
Player 1 enter your selection [row, col]: 2,2
The current state of the game is:
_ O X
_ O _
_ _ _
Player 2 has entered [row, col]: 2,1
The current state of the game is:
_ O X
X O _
_ _ _
Player 1 enter your selection [row, col]: 3,2
The current state of the game is:
_ O X
X O _
_ O _
Congratulations, Player 1 wins!
```

Your program should implement and use the following functions:

- Function name: `display_table`
  - Return:
    - Nothing
  - Parameters:
    - The board as a 3x3 array
  - Requirements:
    - Prints out the following message:
      - "`The current state of the game is:`"
    - Prints out the current status of the board (as shown above)
      - Print out an underscore '_' for an empty cell
- Function name: `clear_table`
  - Return:
    - Nothing

    - Parameters:
        - The board as a 3x3 array
    - Requirements:
        - Clear the board for the beginning of the game by making every position in the array an empty cell
- Function name: `check_table_full`
    - Return:
        - True if the board is full
        - False if the board is not full
    - Parameters:
        - The board as a 3x3 array
    - Requirements:
        - Checks to see if the board is full or not
- Function name: `update_table`
    - Return:
        - Nothing
    - Parameters:
        - The board as 3x3 array
        - The move (row and column)
        - The players token (either an X or an O)
    - Requirements:
        - Updates the board with the given player token (either an X or an O) at the given move (row and column)
- Function name: `check_legal_option`
    - Return:
        - True if the given move is legal
        - False if the given move is illegal
    - Parameters:
        - The board as a 3x3 array
        - The possible move (row and column)
    - Requirements:
        - Checks to see if the given move is within the bounds of the board
        - Checks to see if the given move is on an empty cell
- Function name: `generate_player2_move`
    - Return:
        - Nothing
    - Parameters:
        - The board as a 3x3 array
        - The move (row and column)
    - Requirements:
        - If the game is not over:
            - Generate a valid, random move for player 2
            - Update the board with the generated move
            - Print out the generated move (as seen above)
            - Print out the current state of the board
- Function name: `check_three_in_a_row`
    - Return:
        - Zero if no one has three in a row

        - One if player 1 has three in a row
        - Two if player 2 has three in a row
    - Parameters:
        - The board as a 3x3 array
    - Requirements:
        - Returns the ID of the player that has three in a row or zero if no one has three in a row
- Function name: `check_end_of_game`
    - Return:
        - True if the game has ended
        - False if the game hasn't ended
    - Parameters:
        - The board as a 3x3 array
    - Requirements:
        - Returns true or false depending on if the game is over or not
- Function name: `get_player1_move`
    - Return:
        - Nothing
    - Parameters:
        - The board as a 3x3 array
        - The move (row and column)
    - Requirements:
        - If the game is not over:
            - Get a possible move from the user (as seen above)
            - If the given move is not valid get another move from the user until you have a valid move
            - Update the board with the given, valid move
            - Print out the current state of the board
- Function name: `print_winner`
    - Return:
        - Nothing
    - Parameters:
        - The board as 3x3 array
    - Requirements:
        - If a player has won prints out the victory message (as seen above)
        - If the game is a stalemate prints:
            - "`Game over, no player wins.`"

This is the main function. Copy this into your code and write the ten functions from above in order to make the program work.

```
int main ()
{
   //Declare the tic-tac-toe board
   char board[SIZE][SIZE];

   //The row and column of the move for either player 1 or 2
```

```
   int row, col;

   //Clear the table
   clear_table(board);

   //Display the table
   display_table(board);

   do
   {
      //Have player 1 enter their move
      get_player1_move(board, row, col);

      //Generate player 2 move
      generate_player2_move(board, row, col);

   //Do this while the game hasn't ended
   }while(check_end_of_game(board) == false);

   //After the game is over, print who won
   print_winner(board);

   return 0;
}
```

Notes:

- You are NOT allowed to use global variables
- You are NOT allowed to change the main function
- You cannot add parameters to any function other than the ones described above
- `SIZE` is defined as 3 using a `#define` at the top of the program

Save your program as `tictactoe.c`

Challenge for problem 1 (10 extra credit points):

Make your program run in a loop. At the end of the game your program should print "`Would you like to play again (Y/N): `". The user will then enter either a '`Y`' indicating they do want to play again or an '`N`' indicating they do not want to play again and the program should end. If the user does want to play again the board should be reset and a new game started.

Note:

- You may change the main function for the challenge

Save your challenge separately as `tictactoe_c.c`

Grading Rubric

Grading will be done for each problem as follows:

Correctly-named file	5%
Header comment	2%
Program compiles	5%
Correctly-reading data from terminal	28%
Correct result printed	60%

Submission details

To submit your project, you will have to use the submission script. You do this by either:

1. Working on an ECC machine
2. Working on the provided VMware
3. Secure Copying your files (See Mac Support for information)

To Submit your project:

- Have a directory called "project6"
- Save your *.c files in that directory
- To submit: **(don't type the '>' symbols)**
  > cd project6
  > submit

The submission script copies all files in the current directory to our directory. You may submit as many times as you like before the deadline, we only keep the last submission.

Academic Honesty

Academic dishonesty is against university as well as the system community standards. Academic dishonesty includes, but is not limited to, the following:

Plagiarism: defined as submitting the language, ideas, thoughts or work of another as one's own; or assisting in the act of plagiarism by allowing one's work to be used in this fashion.

Cheating: defined as (1) obtaining or providing unauthorized information during an examination through verbal, visual or unauthorized use of books, notes, text and other materials; (2) obtaining or providing information concerning all or part of an examination prior to that examination; (3) taking an examination for another student, or arranging for another person to take an exam in one's place; (4) altering or changing test answers after submittal for grading, grades after grades have been awarded, or other academic records once these are official.

Cheating, plagiarism or otherwise obtaining grades under false pretenses" constitute academic dishonesty according to the code of this university. Academic dishonesty will not be tolerated and

penalties can include cancelling a student's enrolment without a grade, giving an F for the course, or for the assignment. For more details, see the University of Nevada, Reno General Catalog.