ECE15: Homework 8 - FINAL

In all homeworks, the dialogue boxes show the system prompt in \((\sim)\) $, the user input in red, and the computer output in black. Unless otherwise indicated, assume that the user always enters a valid input.

Note: This is the final version of the homework, uploaded on Saturday 11/12. The homework will be due Saturday 11/19 at 5pm.

Problem 0. Read the following Zybooks sections and solve the problems there in.
Sections 8.5, and 8.7 to 8.14 by 8pm Tuesday November 15th.
Sections 9.1 to 9.3 by 8pm Thursday November 17th.

Problem 1. Write a program root.c that uses the Newton-Raphson algorithm to find a root of the function \( f(x) = x \times \cos^2(x^{3/2}) \). Use the math library define \( f(x) \) and its derivative \( f'(x) \). The program asks the user for the initial \textit{double} value of \( x_1 \), and iterates until the first \( i \) that \( |x_i - x_{i-1}| < 1E-7 \), a value defined as a macro.

At every iteration the program prints the value of \( x_i \) to a 6-digit accuracy. Note that the root obtained by the algorithm will vary based on the initial guess. You may assume that the initial guess will be greater than or equal to 1. You may also assume that the initial guess such that the derivative \( f'(x) \) will never be zero.

\begin{verbatim}
(\sim) $ a.out
Initial guess: 12
Iteration 1: 12.000000
Iteration 2: 12.108824
Iteration 3: 12.135967
Iteration 4: 12.148858
Iteration 5: 12.155221
Iteration 6: 12.158390
Iteration 7: 12.159973
Iteration 8: 12.160764
Iteration 9: 12.161159
Iteration 10: 12.161357
Iteration 11: 12.161456
Iteration 12: 12.161505
Iteration 13: 12.161530
Iteration 14: 12.161542
Iteration 15: 12.161548
Iteration 16: 12.161552
Iteration 17: 12.161553
Iteration 18: 12.161554
Iteration 19: 12.161554
Iteration 20: 12.161554
The root is 12.161554
\end{verbatim}
Initial guess: 3
Iteration 1: 3.000000
Iteration 2: 2.902183
Iteration 3: 2.856739
Iteration 4: 2.833934
Iteration 5: 2.822416
Iteration 6: 2.816616
Iteration 7: 2.813704
Iteration 8: 2.812245
Iteration 9: 2.811515
Iteration 10: 2.811149
Iteration 11: 2.810966
Iteration 12: 2.810875
Iteration 13: 2.810829
Iteration 14: 2.810807
Iteration 15: 2.810795
Iteration 16: 2.810789
Iteration 17: 2.810787
Iteration 18: 2.810785
Iteration 19: 2.810784
Iteration 20: 2.810784
Iteration 21: 2.810784
The root is 2.810784
Problem 2.

In the Mine game, hidden mines are placed on squares of a board and a player tries to determine their locations by clearing all un-mined squares. Write a program mine.c that plays this game.

The program asks the player for the board size, number of mines, and a random seed, and randomly selects the location of the mines. It then prints the initial board with all squares empty, and repeatedly asks the user the integer row and column of a square to clear, and does the following:

- If the square was previously selected, prints “Already cleared!” and asks for another square.
- If the square is out-of-range, prints “Out of bounds!” and asks the player for another square.
- If the square does not contain a mine, displays the board with the number of mines neighboring each cleared square. Note that a square has up to 8 neighbors.
- If the square contains a mine, displays the board with the number of mines neighboring each cleared square, as well as the mines—indicated by a ‘*’, and prints “You lost!”
- If all un-mined squares have been cleared, displays the board with the number of mines neighboring each cleared square, as well as the mines—indicated by a ‘*’, and prints “You won!”

Define at least the following functions (you can use more if you wish):

- **int main()**: Asks the player for the board size size and the number numMines of mines. Defines a square board int board[size][size] to store the board state. Calls initializeBoard to initialize the board. Keeps prompting the user for moves, and for each one: calls validMove to see if the square is out of bounds or was already cleared, checks if the square contains a mine, calls numMinesNear to determine the number of mines neighboring the checked square, calls printBoard to display the board and mines, and prints whether the player lost or won. You can assume that size ≤ 10 and numMines ≤ size². If the numMines == size², displays the board with all the mines and prints “You won!”.

- **void initializeBoard(int size, int board[][size], int numMines)**: Gets the board size, the un-initialized board, and the number of mines. Prompts the user for a seed, and calls srand with this seed. Determines the location of the numMines mines using the following algorithm (converted to proper C code). Please follow this algorithm exactly to ensure that your output matches our test program.

        i = 0;
        while (i < numMines) {
            row = rand() % size;
            col = rand() % size;
            if board[row][col] already has a mine then continue
            else assign board[row][col] as a mine and increment i
        }

    Note that you will need to include stdlib.h. Also, if size is large and numMines is close to size² the process may require many tries, you can assume that the numbers will be sufficiently small that this won’t happen.

- **bool validMove(int size, int board[][size], int row, int column)**: gets the current board and the row and column of the selected square and returns true if it is valid, else it returns false. A move is valid if it is within the range of size of the board. Note that you will need to include stdbool.h.
• `int numMinesNear(int size, int board[][size], int row, int column)`: Returns the number of mines among the (up to 8) neighbors of the square at location `(row, column)`.

• `void printBoard(int size, int board[][size], bool showMines)`: Prints the board on the screen. Displays the number of mines neighboring each cleared square. If `showMines` is `true` also shows the location of the mines—indicated by “*”.

Hints: You can define just one `board`: use a code (such as $-1$ or $-2$) to indicate whether a square contains a mine, was not previously cleared, or the number of neighboring mines it has.

```
(-)$ a.out
Board size [1.to.10]: 3
Number of mines [0.to.9]: 3
Random seed: 123
  0 | 1 | 2
  0 |   |   |   |
  1 |   |   |   |
  2 |   |   |   |
Row and column [0.to.2] each: 0.2
  0 | 1 | 2
  0 |   |   |   |
  1 |   |   |   |
  2 |   |   |   |
Row and column [0.to.2] each: 1.0
  0 | 1 | 2
  0 |   |   |   |
  1 |   |   |   |
  2 |   |   |   |
Row and column [0.to.2] each: 3.0
Out of bounds!
Row and column [0.to.2] each: 0.3
Out of bounds!
Row and column [0.to.2] each: 1.0
Already cleared!
Row and column [0.to.2] each: 2.0
  0 | 1 | 2
  0 |   |   |   |
  1 |   |   |   |
  2 |   |   |   |
You lost!
(-)$
```
Board size [1 to 10]: 2
Number of mines [0 to 4]: 3
Random seed: 8194
Row and column [0 to 1] each: 1.0
You won!

Board size [1 to 10]: 1
Number of mines [0 to 1]: 0
Random seed: 1212
Row and column [0 to 0] each: 0.0
You won!

Board size [1 to 10]: 1
Number of mines [0 to 1]: 1
Random seed: 777
You won!