Programming Fundamentals C & D
FAST-NU, Lahore, Fall 2019

Homework 5

2-Player Chess with Timed Modes

Due Wed **Dec 4** 11:55 PM on SLATE Marked out of 250 points = 10 absolutes

In this final assignment of the semester, your task is to implement the game of chess.

(1) Chess Logic/Rules/General functionalities of the program

Your program should set up at chess board, allow the two players to make alternate moves, check for invalid moves, and detect special conditions such as check, check-mate, stale-mate, castling, En passant, pawn promotion etc from the state of the board. The game may played in classical, rapid, blitz or lightening modes as described in section 3.The program should have additional capabilities of saving and loading a game (max 3 games at a time). Allowing a player to 'Resign' from the game. Allowing both players to 'Hand Shake'. Replaying the moves of a game while loading it.

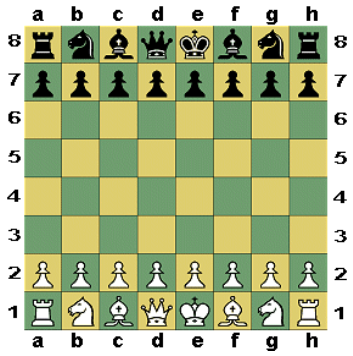
Here is a list of chess rules which **will be our standard** for this assignment:

https://www.chess.com/learn-how-to-play-chess

Consider the contents on the page linked above as part of this assignment statement.

Go through the page carefully, and learn the rules correctly before implementing them. The page provides information about how the pieces move, what are the valid moves, and what are the various game situations etc. Pay attention to details. Rest assured that every feature of the program will be tested during evaluations.

While setting up the board, note that the rows should be labeled 1 to 8 (bottom to top), and columns should be labeled a to h (left to right), such as below. This is the standard initial setup of the board for our purpose.



A 'square' is identified by its column and row number.

If a player inputs[1] e6, for example, she means the fifth column from the left and the third row from the top.

For example, if the black player wants to move their right most pawn by one position, they should enter, "h7 h8"; this will be considered a valid move and the right most pawn will move one step downward. So on each turn, the respective player will enter her move, and a move will comprise of two coordinates: the source square position and the destination square position, such as in "h7 h8". You will need to make sure first that both positions are within the board, that the source contains a piece of the current player not the opponent, that the move is valid for the selected piece, that the move fulfills other criteria of validation such as that if the king is under check the next move must remove this check etc.

To store the board, you should start working with a **char board[8][8];** and use capital characters for white, small for black, and spaces for empty squares.

A method called **printBoard** can display the board, and ask the current user to make a move. Note that white always gets the first move.

Here is a picture of the game-play flow-chart described in class. This is strictly a suggested algorithm and you're by no means restricted to follow the exact same order.

(2) Saving/Loading

You will use the simple sequential text file, as covered in class, to store the moves and while loading you will give the user two options: load instantly, or, load with replay. If during the replay the user presses any key the rest of the replay is skipped and the game loads instantly.

At one time, the user should be able to store a max of 3 different games if they want, and should be able to load any one of these games and continue playing. You should maintain all the files needed for this purpose internally, i.e. the user doesn't need to know what these files are and where they're stored.

(3) Timing the game

When starting a new game, the program should ask the user to specify the game type. Game type could be:

(i) **Classical ('untimed' mode):** unlimited time given to both players. For a player to win in this case, they can either check-mate their opponent, or the opponent resigns. Stale mate is considered draw. A 'Hand Shake' is also a draw. Total

[1] Input can be text-based, if you're making the game for console. It could be graphical, such as a mouse click or a square selection via the keyboard. In any case, though, it will be in the form of the square coordinates as described here.

times taken by the players are maintained (and displayed) nonetheless and the player with less overall time is said to have the 'Advantage' in case of a draw.

(ii) **Rapid:** total time of 10 minutes per player, with a 10 sec increment (extra time) after a move is made. For a player to win in this case, they can either check-mate their opponent, or the opponent resigns, or the opponent runs out of time before they do. Draws occur with stale mate and Hand Shake.

(iii) **Blitz:** similar to Rapid, but total 5 minutes per player and no increments.

(iv) **Lightening:** each move has a fixed time limit of 15 secs max. If a player fails to meet that limit they lose the game then and there. Otherwise, the game continues until check-mate or stale-mate. (No Resign or Hand Shake options in this case.)

The <ctime> library provides all necessary functionalities to accomplish these tasks. Read about ctime here: http://www.cplusplus.com/reference/ctime/ In particular, you'd want to use clock(). Note, however, that if you wish to use other libraries available for the purpose of time measurement, such as <chronos> , we're fine with that too.

(4) Interface

(1) Text Based (Minimum Requirement)

Our minimum requirement is terms of the interface is a text-based board on the console. The display of time and other information etc. is up to you. You can use special characters to make a pretty impressive text based board. One thing you must do is input the coordinates in the standard format mentioned above. Look, again, at the chess board given on page 1.

(2) Graphics Based

Who doesn't want to play a game with some cool graphics?
I will introduce one such library in class. However, there is a number of them available on-line that can be integrated into C++ code. There are no restrictions on you as to which one you decide to use.
I will be introducing the BGI graphics. Here is a nice page at the University of Colorado's CS department to set us going:
https://www.cs.colorado.edu/~main/bgi/visual/

This page allows you to download the necessary library files and provides instructions on how to integrate them with visual studio.

Please note that the graphical interface is above and beyond the minimum requirement. You must only attempt to do this once you have accomplished the gameplay and completed a text-based version.