BİL 102 – Computer Programming
HW 07

Last Submission Date: April 22, 2014 – 09:00

Notes:
1. Because your code may be tested automatically by using test software, **strictly** obey defined I/O format in all inputs (from file or console) and file outputs and also tag declarations. You can (and should) inform user by console output in any reasonable format.

-	-	-	-	-	-	-	-
-	-	2	-	-	2	-	2
-	X	-	2	-	-	-	-
-	X	X	-	-	-	-	X
-	X	-	-	-	-	-	-
-	-	-	-	X	X	-	-
-	-	-	1	-	-	-	-
-	1	-	-	-	1	-	X
-	-	-	-	-	-	-	-
-	-	2	-	1	-	-	1
-	-	1	-	-	-	-	-
-	-	-	-	-	-	-	-

(a)

-	-	-	-	-	-	-	-
-	-	-	-	-	1	-	-
1	-	-	1	-	2	-	-
-	-	-	-	-	-	-	-
X	-	1	-	-	-	1	-
-	-	-	-	1	-	-	-
-	-	X	X	-	-	-	-
-	-	-	-	-	-	X	-
X	-	-	-	-	X	X	-
-	-	-	-	2	-	X	-
2	-	2	-	-	2	-	-
-	-	-	-	-	-	-	-

(b)

Figure 1: A diagram showing a sample GIT checker play: (a) in player1's view, (b) in player 2's view.

This homework will be grade over 200 points (with 100 bonus points).
In this homework, you will implement a GIT checkers game to be played by 2 players, which a modified version of Turkish checkers by the following ways:

- Player 1 starts first.
- The board may have any number of lines bigger than 4 and it has 8 columns.
- There may be some obstacles in the board, where pieces cannot go through.
- There is no obligation to capture the opponent's piece. After a capture the player may capture another piece using the same piece if possible.
- A player wins if:
  - He captures all pieces of his opponent
  - His opponent cannot make any legal move
  - He has a king piece
  - His opponent resign

User Interface

Your program will first take names of the players.
It will obtain the initial game positions from a file "**Initial.txt**".
Your code will work one of the 2 modes:

- Sequential mode: Takes commands from console one by one.
- Batch Mode: Takes all commands from a file named "**Commands.txt**" such that each command exists in a separate line in order.

If “Commands.txt” exists in the same directory with your executable code, your program will work in batch mode, otherwise it will work in sequential mode. (i.e., use the file if you can open it successfully, use sequential mode, otherwise.)
Commands are taken as string expressions as in table 1. Before taking each command current situation of the game is displayed at the corresponding player’s view (player from which the next command is expected) as in figure 1.

Command	Abbr	Operand 1	Operand 2	Operand 3	Example
Move	M	Y coordinate	X Coordinate	Direction: UP, LEFT, RIGHT	For player1: M_5_2_LEFT For player 2: M_10_6_UP
Pass	P	-	-	-	P
Save Game	SG	File name	-	-	SG_checkers.txt
Save Log	SL	File name	-	-	SL_checkers.log
Show Pieces	SP	-	-	-	SP
Resign	R	-	-	-	R
Offer Draw	OD	-	-	-	OD
Accept Draw	AD	-	-	-	AD
Reject Draw	RD	-	-	-	RD

Table 1: Commands

Move command changes the position of a piece of the active player. Unless a capture operation is realized, the active player changes after a move command. Coordinates are given in the corresponding player’s view and starts from 1. If an enemy piece exists in the new position with an empty position next to it, the enemy’s piece is captured according to the normal checkers rules. After a capture, if possible, the same player may capture another piece using the same piece, therefore, a new command is expected from the same user. If such a capture is not possible or the player does not wish to play it, the player gives the Pass command. This is the only situation where the Pass command may be given. According to the rules of the game, the validity of each move is checked strictly. In figure2, the diagram of the game is shown after executing the move commands in table 1 (player 2 wins).

-	-	-	-	-	-	-	-
-	-	2	-	-	2	-	2
-	X	-	2	-	-	-	-
-	X	X	-	-	-	-	X
-	X	-	-	-	-	-	-
-	-	-	-	X	X	-	-
-	-	-	1	-	-	-	-
1	-	-	-	-	1	-	X
-	-	-	-	-	-	-	-
-	-	-	-	1	-	-	1
-	-	-	-	-	-	-	-
-	-	2	-	-	-	-	-

Figure 2: Game’s diagram in the player 1’s view after executing the move commands in table1

“Save Game” command saves the current situation of the game to a text file always in the player 1’s view but does not terminate the game, “Save Log” saves all given commands such that each command exists in a separate line.
After each legal move, name of the active player and a short explanation about move is displayed.
Show Pieces command lists the coordinates of each piece of each player on the console.
After a draw offer, according to the response of the other player the game may be terminated or continued.
If an invalid command is given:

- In sequential mode, the user is warned and a new command is expected from the same player
- In batch mode, a corresponding message is displayed and the game is terminated

Implementation Details

You will represent the game as a 2D integer array in which:

- Each piece of player 1 is represented by 1
- Each piece of player 2 is represented by 2
- Obstacles are represented by -1
- Empty places are represented by 0

Although the row and column numbers in the user interface starts from 1, arrays in user interface starts from index 0, so also all coordinates in function arguments refers indexes starting from 0.
Assume that the table may have maximum 25 rows and maximum 250 commands may be given.
In initial situation file and save game file (created as a response to a Save Game command) game is also represented in this representation such that columns are delaminated by tab characters (a tab character exists between consequent numbers in each row).
Define an enumeration type of “direction_t” to express the direction of a movement having the following values: LEFT, UP, RIGHT.
Implement the following functions with exact given names and arguments. Ordering of the arguments is also important. You can define any other functions as necessary.

- **playGame**: Manages the game play by calling all required functions.
  initFileName: name of the initial state file (Initial.txt should be passed as parameter)
  Returns 0 normally and an error code on abnormal termination of the game
- **getInitState**: gets a 2D array representing the initial state of the game.
  initFile :(input) FILE* showing the file representing the initial state of the game
  table: (output) 2D array representing the game
  rowC: (output) Number of rows in the game
  Returns rowC normally, a negative error code on error
- **makeMove**: realizes a movement of a player if it is legal
  table: (input) 2D array representing the game in the player 1’s view
  rowC: (input) number of rows in the table
  playerNo: number of the active player (1 or 2)
  coorY: (input/output) y coordinate of the piece to be moved – updated in the function call
  coorX: (input/output) x coordinate of the piece to be moved – updated in the function call
  direction: (input) An enumeration indicating the direction of move in the active player’s view
  Returns 0 normally and a negative error code if the move is not legal
- **getPieces:** returns the coordinates of the pieces of each player as 2D arrays with 2

columns such that the first column holds the y coordinates and the second column holds the x coordinates

table: (input) 2D array representing the game in the player 1's view
rowC: (input) number of rows in the table
pieces1: (output) a 2D integer array of player1's pieces
rowCP1: (output) number of rows in pieces1
pieces2: (output) a 2D integer array of player2's pieces
rowCP2: (output) number of rows in pieces2

- **printGame**: prints the diagram of the game to console from the view of any players as in figure 1-a.
  table: (input) 2D array representing the game in any player's view
  rowC: (input) number of rows in the table
- **getOppositeView**: takes a table of the game in a player's view and returns it in the other player's view.
  tableIn: (input) 2D array representing the game in any player's view
  rowC: (input) number of rows in both tables
  tableReversed: (output) 2D array representing the game in the other player's view
- **printPieces**: prints the pieces of a player to the console.
  pieces: (input) a 2D integer array representing the pieces
  rowC: (input) row count of pieces
- **isBlocked**: checks if a piece is blocked by the opponent's pieces and/or obstacles.
  table: (input) 2D array representing the game in the player 1's view
  rowC: (input) number of rows in the table
  playerNo: owner of the piece (1 or 2)
  coorY: (input) y coordinate of the piece
  coorX: (input) x coordinate of the piece
  Returns 1 if it is blocked and 0 otherwise
- **isTerminated**: checks if any player wins the game
  table: (input) 2D array representing the game in the player 1's view
  rowC: (input) number of rows in the table
  Returns 0 if the game continues, 1 if player 1 wins, 2 if player 2 wins
- **saveGame**: saves the positions in the game to a file
  fileName: (input) a string indicating the name of the file
  table: (input) 2D array representing the game in the player 1's view
  rowC: (input) number of rows in the table
- **saveLog**: saves all movement in the game to a given file
  fileName: (input) a string indicating the name of the file
  logs: (input) an array of string holding all movements
  rowC: (input) number of movements

General:

1. Obey honor code principles.
2. **Read your homework <u>carefully</u>** and follow the directives about the I/O format (data file names, file formats, etc.) and submission format **<u>strictly</u>**. Violating any of these directives will be penalized.
3. Obey coding convention.
4. Do not forget to put the required **tags** in the main function.
5. Your submission should include the following file **<u>and NOTHING MORE</u>** (no data files, object files, etc):
   HW07_<StudentName>_<StudentSirname>_<student number>.c
   Do **NOT** compress the files you submit.
6. Do not use non-English characters in any part of your homework (in body, **file name**, etc.).
7. Deliver the printout of your work **until the last submission date**.