TAU Advanced Topics in Programming - 2018B - Exercises

Requirements and Guidelines

The exercises in the course would require you to implement an advanced rock–paper–scissors game.

Note: you will get submission and environment instructions separately. In short: the exercise should be implemented in Linux environment with standard C++ libraries and run on Nova.

Exercise 1

There is a board, N rows * M cols.
There are 2 players.
Each player has K pieces that she or he can position on board.
The pieces include:

- Rock - there are R number of pieces of type Rock
- Paper - there are P number of pieces of type Paper
- Scissors - there are S number of pieces of type Scissors
- Bomb - there are B number of pieces of type Bomb
- Joker - there are J number of pieces of type Joker
- Flag - there are F number of pieces of type Flag

In EX1:
M = N = 10
R = 2
P = 5
S = 1
B = 2
J = 2
F = 1

Rules of the game:
[1]
Positioning
Each player positions her or his pieces on board by using a file in the following format:
<PIECE_CHAR> <X> <Y>
Where PIECE_CHAR = one of: R P S B J F
X = the x coordinate in the range 1-M
Y = the y coordinate in the range 1-N
For a Joker, the line would be:
J <X> <Y> <PIECE_CHAR>
Where PIECE_CHAR for joker can be: R P S B
This sets the initial type of the Joker

Name of the file should be:
player1.rps_board
player2.rps_board

In case a file of one of the players is illegal the player loses the game.
(28/03/2018 - and the other one gets one point - note: game points are not relevant in Ex1).
In case a both files are illegal - both lose and no points are given.
An illegal file includes:

- Bad format
- A PIECE type appears in file more than its number
- Two or more PIECEs (of same player) are positioned on same location
- X coordinate and/or Y coordinate of one or more PIECE is not in range
- Missing Flags - Flags are not positioned according to their number

Note:
It is valid not to position all PIECEs, except for Flags that must be positioned according to their number.

Both players may put their PIECEs anywhere on board.
When the game starts, all PIECEs that share the same position have a fight and only the winning PIECEs stay on board.

Moving and Strength rules of the different PIECEs:
Bomb - cannot move, stronger than all other PIECEs
Rock, Paper, Scissor - can move, strength according to the rules of Rock-Paper-Scissors.
Joker - plays as the PIECE that it represent at this moment
Flag - cannot move, the most weak PIECE

The goal of the game is to capture all the flags of the opponent (one flag in Ex1) or to eat all of its moving PIECEs. A player wins 1 point in case of a victory.

Moves
Player 1 starts.
In each turn the active player can do two things, in this order:

- Move one PIECE (must)
- Change one Joker from current representation to another (optional)

A PIECE cannot move to a square that is already taken by another PIECE of the same player. In the case that a PIECE moves into a square that is occupied by an opponent PIECE, the stronger PIECE wins and take the square (or stays in the square) and the other one is removed from board ("eaten").

Moves Files
The move files are text files in the following format:
Each line represents a move
<FROM_X> <FROM_Y> <TO_X> <TO_Y> [J: <Joker_X> <Joker_Y> <NEW_REP>]
Where:
<FROM_X> <FROM_Y> - represents the PIECE to move by its source square
<TO_X> <TO_Y> - represents the destination
J: hard coded character J followed by a colon and one or more space chars
<Joker_X> <Joker_Y> - Joker, identified by its source square
NEW_REP - represents the PIECE the Joker wants to be, from: R P S B

Note: following paragraph was rephrased, 27/03/2018
The Joker is still a Joker after changing to NEW_REP but for the move and strength rules the change takes effect for the rival move and on (it is done *AFTER* the current move).

In case a move is illegal - the player that tried to do this move loses the game and the other one gets 1 point.
Illegal move = wrong format, bad source, bad destination, moving a PIECE to a position taken by the same player, Joker position that doesn't have a Joker owned by this player, NEW_REP that is not a valid PIECE for a Joker.
Note: it's ok if NEW_REP is the same of the current Joker REP

Output File

The output file should have N + 3 lines:

The first line would be:
Winner: <number>

Where <number> would be: 0 - if no one wins, 1 if player 1 wins, 2 if player 2 wins

The 2nd line would be:
Reason: <reason>

Where <reason> would be one of the following:
All flags of the opponent are captured
All moving PIECEs of the opponent are eaten
A tie - both Moves input files done without a winner
Added 27/03/2018
A tie - all flags are eaten by both players in the position files
Bad Positioning input file for player <player> - line <bad line number>
Bad Positioning input file for both players - player 1: line <X>, player 2: line <Y>
Bad Moves input file for player <player> - line <bad line number>

* Note 1: in case of bad input (in any of the files), additional information regarding the problem - in your own language - should be printed to screen, but not in the output file
Note 2: in case input file is missing or cannot be opened, the program shall not start the game, no output file should be created and the information regarding the problem - in your own language - should be printed to screen
Note 3: in case Moves file is done (all lines were used) and the opponent still have moves: the opponent will still move and the game will continue, the players whose Moves file was done will not move. If both Moves file are done without a winner there is no winner and the appropriate reason above would be selected

The 3rd line would be: **an empty line**

All lines from the 4th line till N+3 would be: **the board state at the end of the game**
Player 1: all chars as capital letters
Player 2: all chars as lower letters

Errors

- Both Positioning files are analyzed at the same "stage" - so if both are bad the result is 0 (no winner).
- Moves files are not analyzed ahead, if there is a problem in a specific line the players whose line currently failed loses
- Any other problem shall be printed to screen in your own language
- In case of bad input (in any of the files), additional information regarding the problem - in your own language - should be printed to screen, but not in the output file
- Program shall not use the exit function - using exit will reduce points
- Program shall not crash
- When you print an error to screen in your own language make sure to use proper english and to give all the required information to follow and understand the error

Additions 27/03/2018

Fight between tools of the same type:

- Both tools are removed from board

A tool attacks a bomb

- Both the attacking tool and the bomb are removed from board (i.e. bomb is removed after exploding)

Move files - name of the file should be:

- player1.rps_moves
- player2.rps_moves

Output file name:

- rps.output

Position of the files (board files, move files, output file):

- Files should be read from the working directory
- If **input** file(s) doesn't exist - print usage
- Output file should be created to the working directory and override any other output file with the same name, if exists there before

Additions 28/03/2018

Output file

All lines from the 4th line till N+3 would be: **the board state at the end of the game**

Each square in the board is a char, space for empty square, the actual representation of a game piece for square that holds a game piece

Player 1: all chars as capital letters

Player 2: all chars as lower letters