APS 105 — Computer Fundamentals

Lab #7: Reversi Game — Board Configuration and Move Legality Checking

Winter 2022

You must use `examify.ca` to electronically submit your program by 11:59 pm on **Saturday, March 19, 2022**.

This is the first part of a series of two labs (Lab 7 and Lab 8) that will complete an implementation for a board-type game called *Reversi* (also called *Othello*). The goal of this lab is to write code that sets up the input and checks the legality of moves in this game. These two labs make use of two-dimensional arrays, as well as all of the previous material covered before two-dimensional arrays in this course. They require some careful thinking about how to convert human-thinking into working software. **Please read through this lab handout carefully**.

Objective

The goal of this lab is to write a program that will be used (in Lab 8) as part of a Reversi game, as well as a little bit of the 'thinking' code that will be used in that lab to have the computer play against a human opponent.

Here is a brief description of the full Reversi game. Reversi is played on a board (like a chess board or a checkers board) that has dimensions $n \times n$, where n is even. In the picture below $n = 4$. The game uses tiles that are white on one side, and black on the other side (they can be "flipped" over to change their colour). One player plays white; the other player plays black. The picture below shows the initial board configuration, which has two white and two black tiles placed in advance at the centre. Observe that rows and columns are labelled with letters.

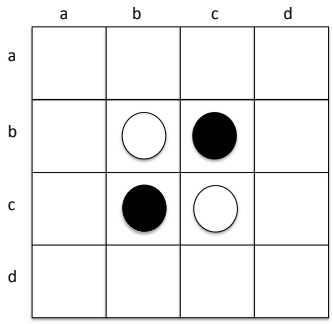

Figure 1: Starting positions on the Reversi game board.

A "turn" consists of a player laying a tile of his/her own colour on a candidate empty board position, subject to the following two rules:

1. There must be a continuous straight line of tile(s) of the opponent's colour in at least one of the eight directions from the candidate empty position (North, South, East, West, and diagonals).
2. In the position immediately following the continuous straight line mentioned in #1 above, a tile of the player's colour must already be placed.

After playing a tile at a position that meets the above critera, all of the lines of the opponent's tiles that meet the criteria above are flipped to the player's colour.

In the picture below, all of the candidate positions for White's next move are shown shaded.

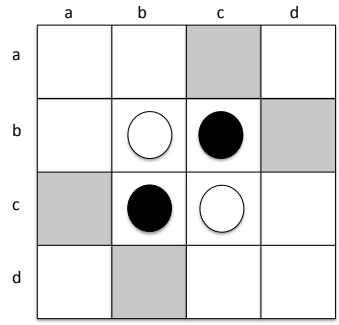

Figure 2: All of the candidate positions for White's next move.

If the White player decides to play at row c, column a, the Black tile at row c, column b is flipped and the board looks like this:

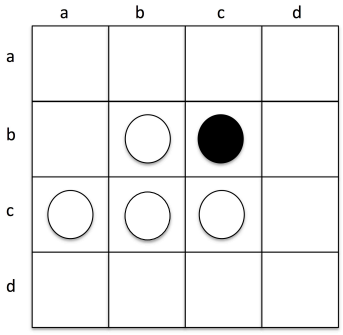

Figure 3: White plays at row c, column a.

The picture below shows the possible move positions for the Black player:

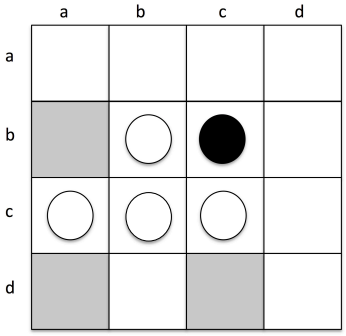

Figure 4: All of the candidate positions for Black's next move after White plays at (c, a).

If the Black player lays a tile at (b, a), the board appears like this:

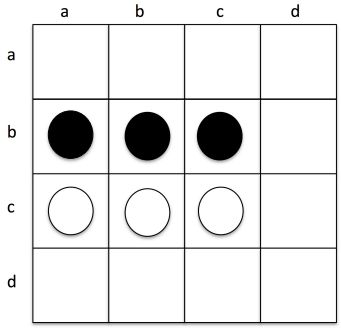

Figure 5: Black plays at (b, a).

Finally, if the White player lays a tile at (a, c) the board appears like this:

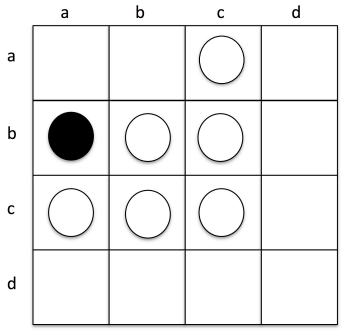

Figure 6: White responds to Black by playing at (a, c).

Note that in White's move, two lines of Black tiles were flipped: the line directly to the South, and the line to the South West.

The turns alternate between the players, unless one player has no available move, in which case the only player with an available move is allowed to continue to make moves until a move becomes available for the opponent. At this point, the opponent is allowed to take a turn and the alternating turns between the players resumes. The game ends when either: 1) the entire board is full, or 2) neither player has an available move.

For this lab, you will implement part of the game-playing functionality. You will complete the game in Lab 8, using the functionality you have built in this lab, so please be mindful to build clean and re-usable code for this lab.

You will write a C program that will do the following: (Note that the specific details of input and output will be given in the example runs below this section.)

1. The first input to the program will be n, giving the size of the $n \times n$ board. You may assume that n will be an even number and will never be larger than 26, and should declare a static 2-dimensional array. There is no need to allocate memory dynamically or to declare variable-length arrays. Your program should initialize the board as shown above and print it.

2. The next sequence of inputs will describe a board configuration, representing a situation part-way through a game of Reversi. Each line of input will consist of three characters with no spaces in between. The first character will be a colour: `B` or `W`; the second character will be the row (`a` — `z`); the third character will be the column (`a` — `z`). The three characters represent

a tile of the specified colour placed at the specified row and column. The three-character sequence `!!!` ends the board configuration entry phase. Character arithmetic can be used to translate the rows/columns into array indices, e.g. `'b' - 'a'` equals 1. **Note: your program should not check for move legality during this phase. This phase is simply to input an intermediate board configuration.**

3. Then, your program should print a list of the available moves for the White player, followed by a list of the available moves for the Black player, given the board configuration input in the previous step. The available moves for each player should be printed in the order of increasing rows, then in the order of increasing columns (for available moves in the same row).

4. Next, your program should ask the user to input a move, represented in the same three-character format. Your program should check if the move is valid, and if so, make the move, flipping the tiles correspondingly. If the move is invalid, your program should indicate so.

5. Your program should print the final board configuration and terminate.

Your program must use the following characters to represent the state of each board position:

```
U - for unoccupied
B - occupied by black
W - occupied by white
```

For example, after the entire board above is entered, it would be printed as follows:

```
  abcd
a UUWU
b BWWU
c WWWU
d UUUU
```

To print the board, your program should contain a function with the following prototype:

```
void printBoard(char board[][26], int n);
```

where `board` is the 2D array representing the current board state, and n is the board dimensions.

Here is an example execution of the program:

```
Enter the board dimension: 4
  abcd
a UUUU
b UWBU
c UBWU
d UUUU
Enter board configuration:
Bba
Wca
Bac
!!!
  abcd
a UUBU
b BWBU
```

```
c WBWU
d UUUU
Available moves for W:
aa
bd
db
Available moves for B:
ab
cd
da
dc
Enter a move:
Wdb
Valid move.
  abcd
a UUBU
b BWBU
c WWWU
d UWUU
```

Here is another example execution of the program where the final move is invalid:

```
Enter the board dimension: 6
  abcdef
a UUUUUU
b UUUUUU
c UUWBUU
d UUBWUU
e UUUUUU
f UUUUUU
Enter board configuration:
Bbd
Bad
Wde
Wcb
!!!
  abcdef
a UUUBUU
b UUUBUU
c UWWBUU
d UUBWWU
e UUUUUU
f UUUUUU
Available moves for W:
ae
bc
ce
db
ec
ed
Available moves for B:
```

```
ba
bc
ca
db
df
ed
ef
Enter a move:
Bbe
Invalid move.
  abcdef
a UUUBUU
b UUUBUU
c UWWBUU
d UUBWWU
e UUUUUU
f UUUUUU
```

We strongly encourage you to break up your program into separate functions, and to carefully test each function separately, before connecting it into the larger program. To help with this, you are **required** to create the following helper functions and use them in your implementation:

```
void printBoard(char board[][26], int n);
```

which prints the game board according to our example output.

```
bool positionInBounds(int n, int row, int col);
```

which checks whether the specified `(row, col)` lies within the board dimensions.
It is very error prone to write separate code to check each of the eight possible line directions that begin at a given tile. To simplify this, you are **required** to write and use the following function:

```
bool checkLegalInDirection(char board[][26], int n, int row, int col,
                           char colour, int deltaRow, int deltaCol);
```

which checks whether `(row, col)` is a legal position for a tile of colour by "looking" in the direction specified by `deltaRow` and `deltaCol`. `deltaRow` and `deltaCol` take on values of -1, 0, and 1, with the restriction that they cannot both be 0. For example, if `deltaRow` = 1 and `deltaCol` = 0, the function searches the South line. If `deltaRow` = -1 and `deltaCol` = 1, the function searches the Northeast line. The idea is that, elsewhere in your program, you will call the helper function 8 times to search the lines in the 8 possible directions. Your lab TA may be checking for the presence and use of these helper functions in your program after you submit it on `examify.ca`.

Starting Point

You are provided with two source files: `reversi.h` and `reversi.c`, as your starting point. You need to put these two files under the same folder for the C/C++ runner extension to compile.

When you submit your program on `examify.ca`, Please comment out the line where the header file is included:

From:

```
...
#include "reversi.h"
...
```

To:

```
...
// #include "reversi.h"
...
```

Marking

There are a total of **10** marks available in this lab, marked as usual on `examify.ca`. The public tests on `examify.ca` do not test all the possible cases of input for your program. If your program works using the public test cases on `examify.ca`, it does not mean that you will get full marks for your program. You are responsible for testing your program using all the cases you can think of, according to the specification given in writing in this lab handout.

Online Interactive Platform

To help you complete your Lab 7 and Lab 8, the APS 105 teaching team developed an online interactive platform located at `http://aps105.ece.utoronto.ca:8090/`. For the purpose of Lab 7, the platform can help you familiarize with the rules in this game. The front-end of this website has been released as open-source, available at `https://github.com/Louis-He/reversi-reactjs`.

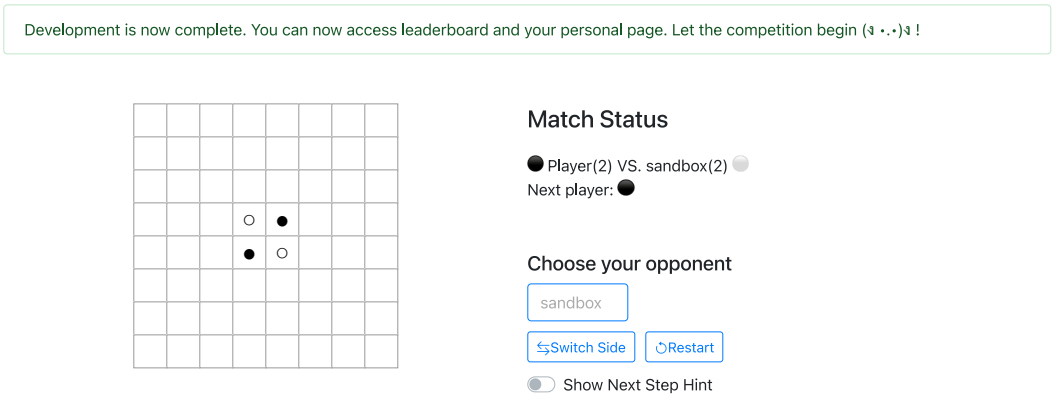

Figure 7: The APS105 Reversi Interactive Platform Website

Stay Safe and Good Luck!