Assignment 4: Reversi

Due: 23:59, Thu 3 Nov 2022 File names: `reversi.h` `reversi.cpp` `reversi-game.cpp` Full marks: 100

Introduction

The objective of this assignment is to let you practice the use of 2-D arrays (and optionally pointers) in C++. You are to write a program to simulate a classic board game known as *Reversi* (a.k.a. *Othello*).

Reversi is a two-player game which is played on a square uncheckered board with some game pieces. A piece is a *disc* with two sides in black and white colors respectively. At the beginning of a game, there are four pieces placed on the board as shown in Figure 1. We use a dot '.' to denote an empty cell in the board, the character 'X' to denote a black piece, and the character 'O' to denote a white piece. To denote a position in a game board, we use a coordinate system similar to a spreadsheet program, e.g. "A2" refers to the cell in the first column and the second row. Suppose that a C++ array `arr` is used to represent the game board, cell address "A2" will be mapped to `arr[1][0]`.

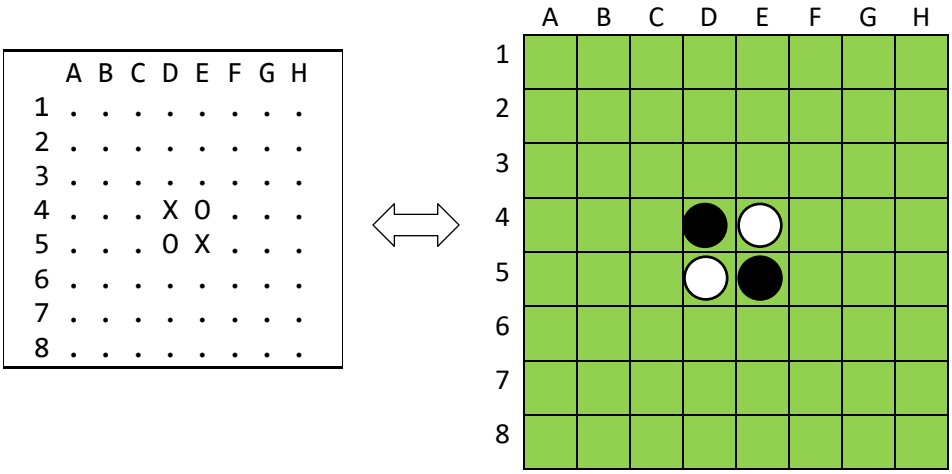

Figure 1: Initial configuration of the game board

The two players, namely "Black" and "White" make moves in turns by putting a disc of their color on an empty cell of the board. Black plays the first move. A move cannot be made arbitrarily to any of the empty cells. As a *valid move*, the newly placed disc must be in a position such that it forms *at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same color, with one or more contiguous pieces of the opposite color between them*. After the move, those contiguous pieces will all be *flipped* to become the color of the new piece. Figure 2 shows an example move made by White. Note that a move can simultaneously form more than one straight line. In such a case, the sandwiched pieces on *all* the lines formed will be flipped. Figure 3 shows such an example.

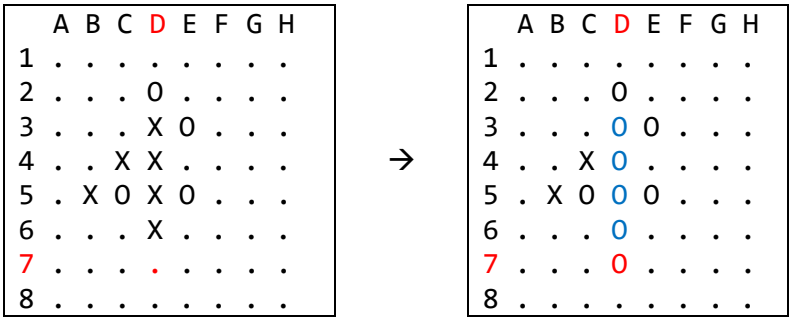

Figure 2: An example move by White ('O') at position D7, forming a vertical occupied line from D7 to D2. Four black ('X') pieces are "clipped" by two white pieces at both ends and are flipped.

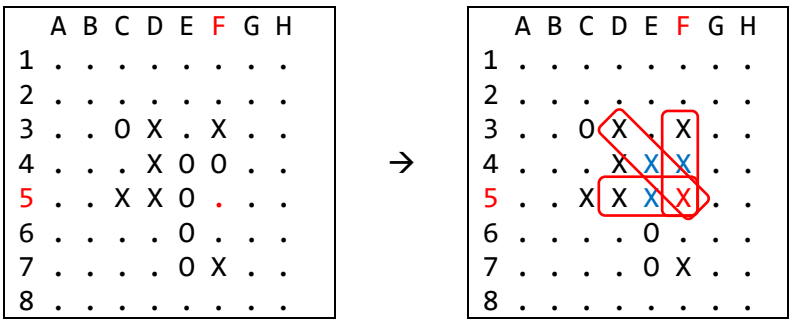

Figure 3: An example move by Black at F5, forming three occupied lines (i) F5 to F3, (ii) F5 to D5, and (iii) F5 to D3. All sandwiched pieces of white are flipped to black.

A game is over when either the board is full or both players have no valid moves. The player having more pieces of his/her color than the opponent wins the game. If both players have the same number of pieces, the game draws.

To add more challenges to the game, this assignment will extend the original Reversi game with an additional feature: some *"blocks"* can be added to the game board, preoccupying some cells so that both players cannot place a disc onto such positions. Blocks are added before the game begins. For example, Figure 4 shows a board with cells A1, A2, B1, B2, G5, G6, G7 and F7 initialized as blocks. Let's call them "block cells". Then players cannot make a move to such positions. Apart from fewer choices for making a move and greater likelihood that the players run out of valid moves before the board becomes full, the overall rules of the game stay the same as in the case without block cells.

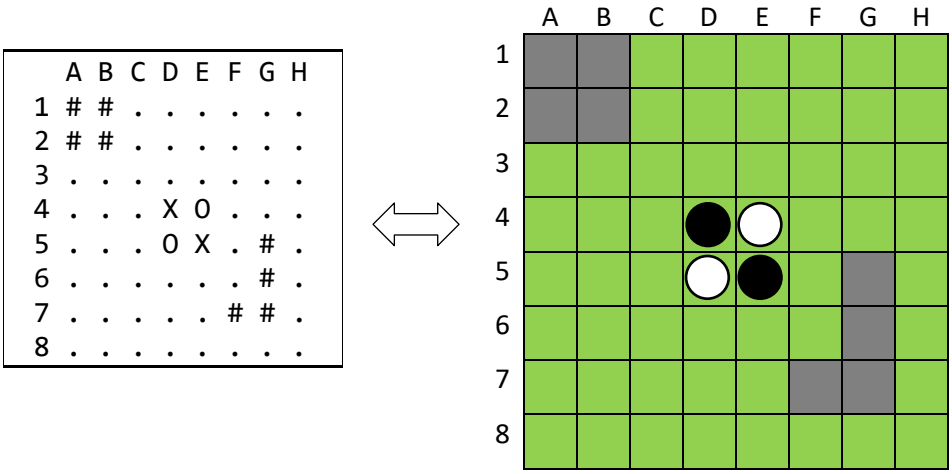

Figure 4: Initial configuration of the game board with blocks (grey areas) added

Program Specification

Game board

You must use a two-dimensional array of `char` type to represent the game board. For example, you may define the following in your `main()` function:

```
char board[N][N];
```

Note that defining the board array as a global variable is forbidden and will invite mark deduction if you do so. You should also avoid declaring it in a hard coded manner like `char board[8][8];`

Make sure your program can support array sizes other than 8. We demand that the array size should be defined as a **global constant** in a header file to make it easily scalable.

```
const int N = 8;          // board size
```

When grading, we may change `N` to a different value (basically between 4 and 12) for more thorough testing of your code. Please follow the above constant definition **exactly**, i.e., declaring a constant of `int` type and naming it with a single capital letter 'N' to facilitate our marking process.

Game Flow

- The program needs to set up the game board first. Before a game begins, the program will prompt the user to enter the number of block cells to add to the board. It can be zero or up to half of the board's area at most. For example, if N (board size) = 8, the maximum number of block cells allowed would be (8 * 8) / 2 = 32. This ensures that the board still has enough empty cells for making moves. It is illegal to set any of the initial four pieces central in the board as a block. Entering a cell address that refers to a block already defined or a cell out of the board's size range is illegal too. If the user enters an exceedingly value for N or any illegal cell addresses, the program repeats prompting until valid inputs are received.
- Then the game starts with round 1 on the board initialized with the first four discs as shown in Figure 1 and optionally with block cells. Black moves first.
- Black and White make moves alternately. For each move, the program should prompt the player to enter the cell address of the intended move, e.g. "D3", and it will convert the characters into two integer values (x, y) that represent the column and row indexes, respectively, to access the corresponding element of the array `board[y][x]`.
- You should check whether a player's input refers to a valid move. If the input position is not valid (either because the input coordinate is out of bounds of the board or because the input position cannot flip any opponent's pieces), the player should be prompted again for a valid move.
    - In fact, it can happen that the board with empty cells has really run out of positions for a player to make a valid move. Thus, your program needs to check whether there still exists at least one empty cell that allows the current player to make a valid move before prompting him/her for entering a cell address. This involves scanning the whole game board for any cells for a valid move per round before prompting.
    - In case of no possible valid moves, the player's round will be "passed" to the opponent.
- If a valid move is made, all relevant pieces should be flipped to the opposite color.

- The game should *end when either (1) the board is full, or (2) two consecutive passes are made (i.e. Black passes to White but White immediately passes back to Black since both of them have no valid moves to make)*. If Black wins, the message "Player X wins!" should be printed; if White wins, the message "Player O wins!" gets printed; and if the game is a draw, "Draw game!" should be printed.
- Your program should be *reasonably decomposed into different functions* for various tasks such as printing the game board, initializing the board, adding blocks, checking if a move is valid, and updating the board to realize a move, etc. The game board array and other data like the input coordinate values can be passed between the functions.

Provided and Required Functions (`reversi.cpp` and `reversi.h`)

Your program must contain the following functions. *Some of them are written for you already (Provided) and you shall not modify their contents.* The others will be written by you (Required). These functions shall be implemented in a source file `reversi.cpp` with the function prototypes in a header file `reversi.h`. These functions shall be called *somewhere* in your program. You must *not* modify the prototypes of all these functions. You can design extra functions if you find necessary.

Besides, *all the required functions below will be graded individually.* That is, we shall connect your `reversi.cpp` with another source file with our testing code to call the functions one by one for grading. So your code for each function shall implement the description of that function only. You shall *not* write any code in a function that is beyond that function's description.

(Provided) `void print_board(char board[][N]);`
Print the game board to the screen using the format in Figure 1.

(Provided) `bool cin_failed();`
Return true if console input fails (e.g., the user enters alphabets while an integer is expected.) or false otherwise. This function bypasses any weird input by skipping to the next newline and clearing the error flag on the cin object. Upon receiving true, the caller of this function may execute a `continue` statement, say, to skip to the next iteration of the enclosing loop, if any.

(Required)
`bool valid_move(char board[][N], char p, int y, int x, bool flip = false);`
Return true if the move to be made by player p at cell (y, x) is valid, or false otherwise.

This function has two usages: (1) check if a move is valid; (2) carry out the move on the board. It has an optional Boolean parameter `flip` (default to false). If the `flip` argument is passed as true, it will carry out the move on the board too, i.e. flipping the opponent's discs in all possible directions.

For better code readability, you may optionally create a *wrapper* function named `flip_discs()` or `make_move()`, etc., which calls this function with `flip` passed as true. Then your main program can call `valid_move()` to check move validity and call the wrapper function to make a move on the board respectively.

(Required) `bool has_valid_moves(char board[][N], char p);`
Return true if player p still has at least one valid move on the board, or false otherwise.

(Required) `bool has_empty_cells(char board[][N]);`
Return true if the game board still has at least one empty cell, or false otherwise.

Assumptions

- The board is always a square. So only a single global constant N is needed to define its size.
- The minimum and maximum board sizes are 4 and 26 respectively. So, the largest column is up to Z only. You need not spend too much time on testing big game boards (up to 12 is enough).
- N is always even, so we won't have 5x5, 7x7, 9x9, … boards but 4x4, 6x6, 8x8, … boards only.
- The row index markers on the left of the printed board are always right adjusted at a fixed width of 2 characters, i.e. for single-digit row indexes, there is a single space ahead of them. For example, Figure 5 shows a board with N = 10. Cell addresses being entered will still be like "A1", "B8", "C10", etc. No zero padding is used, so the expected inputs are not like "A01", "B08", etc.
- We assume that cell address inputs always follow the format of one letter (A-Z or a-z) plus one integer (1-26). Lowercase inputs like "a1", "h10", etc. will be accepted as normal. You need NOT handle weird inputs like "AA1", "A01", "abc", "A3.14", "#a2", … for cell addresses and inputs like "-1", "6.28", "#$@", … for the number of blocks. The behavior upon receiving these is unspecified, probably running into forever loop or program crash. We won't test your program with these inputs. (Having that said, we have provided a function called `cin_failed()` in the starter code; you are encouraged to use it to handle cin failures and avoid forever loop around user input prompt. But this is optional, and you won't get any mark deduction for not doing so.)

```
   A B C D E F G H I J
 1 . . . . . . . . . .
 2 . . . . . . . . . .
 3 . . . . . . . . . .
 4 . . . . . . . . . .
 5 . . . . X O . . . .
 6 . . . . O X . . . .
 7 . . . . . . . . . .
 8 . . . . . . . . . .
 9 . . . . . . . . . .
10 . . . . . . . . . .
```

Figure 5: Initial configuration of a game board with 10 rows and 10 columns

Restrictions

- You *cannot declare any global variables* (i.e. variables declared outside any functions). But global constants or arrays of constants are allowed.
- Please use a character array to represent the board. *Avoid using other container classes* like `std::array`, `vector`, `map`, etc. available in the standard template library (STL).
- *Avoid defining your own classes* in this assignment. We will have other assignments on classes and objects to come. But using "struct" in a manner like a C structure (without any member functions) is still allowed if you find it fit for this assignment.

Sample Runs

In the following sample runs, the blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (same text, symbols, letter case, spacings, etc.). Note that *there is a space after the ':' in the program printout*.

Sample Run #1: (for N = 6)

```
Enter number of blocks: 19↲
Too many blocks!
Enter number of blocks: 7↲
Enter position for block 1: A1↲
Enter position for block 2: A2↲
Enter position for block 3: B1↲
Enter position for block 4: B2↲
Enter position for block 5: C3↲
Invalid position!
Enter position for block 5: D4↲
Invalid position!
Enter position for block 5: X10↲
Invalid position!
Enter position for block 5: A1↲
Invalid position!
Enter position for block 5: B5↲
Enter position for block 6: E5↲
Enter position for block 7: E2↲
Round 1:
   A B C D E F
 1 # # . . . .
 2 # # . . # .
 3 . . X O . .
 4 . . O X . .
 5 . # . . # .
 6 . . . . . .
Player X's turn: E3↲
Round 2:
   A B C D E F
 1 # # . . . .
 2 # # . . # .
 3 . . X X X .
 4 . . O X . .
 5 . # . . # .
 6 . . . . . .
Player O's turn: C2↲

…
(some rounds skipped due to space constraint)
…

Round 17:
   A B C D E F
 1 # # X O . .
 2 # # O O # X
 3 O O X O O X
 4 O O O O O X
```

```
 5 . # O O # .
 6 . . O . . .
Player X's turn: E1↲
Round 18:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X O O X
 4 O O O O O X
 5 . # O O # .
 6 . . O . . .
Player O has no valid moves! Pass!
Round 19:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X O O X
 4 O O O O O X
 5 . # O O # .
 6 . . O . . .
Player X's turn: A5↲
Round 20:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X O O X
 4 O X O O O X
 5 X # O O # .
 6 . . O . . .
```

…

(some rounds skipped due to space constraint)

…

```
Round 24:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X X O X
 4 O X O X X X
 5 O # X O # X
 6 O . O O O .
Player O's turn: B6↲
Round 25:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X X O X
 4 O X O O X X
```

```
 5 O # O O # X
 6 O O O O O .
Player X has no valid moves! Pass!
Round 26:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X X O X
 4 O X O O X X
 5 O # O O # X
 6 O O O O O .
Player O has no valid moves! Pass!
Game over:
   A B C D E F
 1 # # X X X .
 2 # # O X # X
 3 O O X X O X
 4 O X O O X X
 5 O # O O # X
 6 O O O O O .
Player O wins!
```

Sample Run #2: (for N = 4)

(This example is an extreme case with so many block cells that no valid moves can ever be made since the start of the game.)

```
Enter number of blocks: 9↵
Too many blocks!
Enter number of blocks: 8↵
Enter position for block 1: B1↵
Enter position for block 2: C1↵
Enter position for block 3: A2↵
Enter position for block 4: D2↵
Enter position for block 5: A3↵
Enter position for block 6: D3↵
Enter position for block 7: B4↵
Enter position for block 8: C4↵
Round 1:
   A B C D
 1 . # # .
 2 # X O #
 3 # O X #
 4 . # # .
Player X has no valid moves! Pass!
Round 2:
   A B C D
 1 . # # .
 2 # X O #
 3 # O X #
 4 . # # .
```

```
Player O has no valid moves! Pass!
Game over:
   A B C D
 1 . # # .
 2 # X O #
 3 # O X #
 4 . # # .
Draw game!
```

- There are many combinations of possible inputs. Please check your program correctness against the results produced by our sample program executable posted on Blackboard.

Submission and Marking

- Your program file names should be `reversi.h`, `reversi.cpp`, and `reversi-game.cpp`. Submit the *three files* in Blackboard (https://blackboard.cuhk.edu.hk/). If you do not submit the .h file, we shall assume that it is the same as the provided one.
- **Make sure the .h file that you submit defines the constant N (board size) as 8** (after your testing on other board sizes).
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of all your files.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT plagiarize.*** Sending your work to others is subject to the same penalty for copying work.