Algorithm

1. Welcome the user to the program.
2. Welcome the user to the program. Tell user to option 3 if they do not know how to play.
3. Initialize menu
4. Get user option: Play game, options, how to play, exit.

Choice 1 (Play game):

5. Draw 8x8 grid using 2 for loops (a 2D array is mapped to the grid)
6. Initialize the grid to default starting Reversi position, 2 "x" and 2 "o" in the centre. "X" represents player 1's pieces; "O" represents player 2's pieces, "-"represents nothing (blank space)
7. If playerturn==1, player 1 is going, set piecevalue to "x", if playerturn==2, set piecevalue to "o"
8. Display whether player 1 or player 2 is going, and ask the player to enter the row and column of their next piece or pass
9. Print the score of both players beside the board and indicate which player's turn it is.
10. Display board array in a grid format along with row and column indicators.
11. The program processes the coordinate to see if this is valid move by checking the points around the inputted piece, if this point lies on a blank space "-"; the first stage of the check has been passed. Now the program checks if the move will flip/convert an enemy piece into a friendly piece, if the move does not perform this task, the move is deemed invalid.

Processing

12. Using while loops and if statements, the program goes through every position next to the point specified
    Ex. If the game was on a 3x3 board, and the point is (2,2), the programs checks (1,2) (3,2)(2,3)(2,1)(3,3)(1,1)(3,1)(1,3), as long if the point is in range, if point is (-1,0) program won't check it. All directions will be checked(up, down, left, right, up-left diagonal, down-left diagonal, etc.)
13. If the point checked is the location of an opposition piece, call, increment checkfurther[direction] so it will check the next point in that direction, method and record the point checked, if not then check next point
    a. Continue checking points in the **same direction,** incrementing the counter with each point counted
    b. The function ends when the opposite of your piece is encountered, if an empty space is encountered, then reset the counter to zero
14. Using a for loop, with initial row, column and checkfurther[direction] value, the necessary elements on a 2D array are switched into opposite pieces.
15. Print the undated array, if flag is false, tell the user that the move is not valid
16. Go through the array, count number of "x" and "o" to display as scores in the next loop or end-game display.

17. Go through the array with other various check methods, if there are no empty spaces or both players cannot make a move or if one player has no pieces left, end the game, evaluated the number of black and white pieces, the pieces with the greatest count wins, display the ending message, go to step 19
18. If no end condition has been met, reset all variables to default (without affecting the arrays), switch playerturn, 1 becomes 0, 0 becomes 1 and go back to step 7
19. Display board and final score. Congratulate the winning player. Reset scores to 2 and end-game Boolean variables to default. This is to prepare for another game.
20. Re-initialize menu and return to step 4

Choice 2(Options)

21. Display options.
22. Get user choice of what gameplay aspect to modify.
23. If user chooses to set board size, then get boardlength and boardwidth and display the user's choice of dimensions.
24. Else if the user chooses to assign AI, display 4 choices for the user: (1)Player vs. Player (2)Player vs. Ai (3)AI vs. AI (4)AI vs. Player
25. Re-initialize menu and return to step 4

Choice 3(How to play)

26. Display user guide/tutorial
27. Re-initialize menu and return to step 4

Choice 4(Exit)

28. Display a message of thanks to the user for using the program/playing the game.
29. Terminate program.