Reversi/Othello - the board game

Made by Ankish Raj Prajapati and Debanjan Das

Student numbers - 17202456 and 17202833

Pre-implementation:

We have included 4 library functions - stdio.h, time.h, stdlib.h, string.h - each of which will be used in this project in due time.

Implementation and Logic:

1. We created a structure for each player that has 4 components - a string to store, the name of the player, and integer variables to store the type of piece the player will be playing with, the number of pieces the player has on board and the number of pieces a player can play with. The initial code had just one variable for number of pieces, that kept track of how many were on the board, but we decided to implement two separate ones for the board and in the player's hand because that makes it easier to keep track of what exactly is happening in the game.

2. The main function begins with the declaration of two instances of this structure, one for each player.

3. We assign the player components the values. The game begins with each player having 2 pieces on board and 30 to play with so we initialise that and then get the input for the name of each player.

4. We included a little piece of code to handle the new line character that fgets tends to add at the end of inputs.

5. Next, we use the time.h library to generate a random number. With the help of this number, we randomly assign each player the type of piece (black or white) that they will be playing with.

6. The board initialisation is a little tricky but we got it right by separating the initialisation of the main playing board from the numbered row and column. We first initialise the top row and the first column with the respective number through a single loop.

7. Next, we use a nested loop to access the elements of the board except those in the top row and first column. Since the middle 4 squares, that is, (4,4),(4,5),(5,4) and (5,5), need to be initialised with the pieces, we handle this separately through an if condition. In any other case, we filled it with in the number '9'. We chose 9 because it is unique in this context, that is, it is not being used anywhere else on the board and obviously because the board it is an integer array. Any empty square in the actual board array will be represented by a 9.

8. After that, we display with board. We needed to keep a few things in mind. Firstly, the (0,0) element has never been initialised since it is supposed to be blank. Secondly, the empty squares are 9 in the array but need to be represented by an 'x' when displayed. And three, the bars do not appear between the numbers in the top row but they do after every element in the remaining rows. We used a combination of if, else and else if statements to handle each of these cases and our little program, barring a few typing errors that we fixed when trying to compile, worked like a peach in the very first attempt.

9. We are calling the logic () function inside the main. Then, we print all the possible moves. We do that by scanning all the elements around the piece.

10. After printing the board, we are asking the player to enter the next co-ordinate. (I.e. the next possible move) Next, we check if the co-ordinate entered by the player is valid or not. If it is not valid, then we display a warning otherwise, the co-ordinates entered by the user is stored in the list array.

11. Then, we calculate the difference between those two co-ordinates and using that, we place all 0’s (White) or 1’s (Black) in between depending on whose chance it is.

12. This way, the game keeps on continuing and at the end, the result function is called inside the main. The result function keeps a track of the points and at the end, the player having higher points is declared the winner.

Division of work:

We decided to write a piece of the code one by one and then keep updating the repository online. The other person would go through whatever changes have been made and make sure there are no errors. This made life very simple for the both of us. Debanjan started off by creating the structure and initialising them for each player and taking the input for player names. I made the changes to the structure that I mentioned above and then wrote the code for handling the new line character in fgets and assigning the piece type to each player. Next, we discussed how to handle the initialisation and printing of the board and once we agreed on the algorithm we were going to use, Debanjan coded the initialisation of the board and I coded the printing. The program is fairly well documented and explains the logic that we have used throughout.

END OF README.