COMP 1130 Principles of Programming II
Project 3 – Tic Tac Toe
Maximum Possible Points: 20

Due Date:

- The In-Lab part of the program needs to be demonstrated to the instructor by 6:00 PM on September 6.
- The source code of the completed program and the program execution screenshot need to be submitted by 3:00 PM on September 13.

Objectives:

- To gain experience with 2D arrays.
- To gain experience with random number generation.

Overview:

Writing computer games is a very interesting area of Computer Science. You have to write a program that will allow two players to play the well-known Tic-Tac-Toe game. The game starts with an empty board as shown below. A move by Player1 is represented by an 'X' and a move by Player2 is represented by an 'O'. The player who gets three marks in a row (horizontal, vertical, or diagonal) wins the game. If all the nine positions are exhausted without a result, then it's a tie.

You will need to use a 2-dimensional 3x3 array of characters for representing the game board. You will use the NULL character ('\0' or ASCII value 0) to represent a blank position. You will be using several functions in the program in addition to main.

The *initializeBoard* function will generate the initial game board. It should accept a 2D array of characters representing the game board and will initialize all the 2D elements to the NULL character ('\0' or 0). This function will not have any return type.

The *printBoard* function will be responsible for displaying the game board at any point. This function should accept a 2D array of characters representing the game board and display the game board along with the grid (as shown in the sample executions).

The *getStatus* function will return the current status of the game. The function should accept a 2D array of characters representing the game board and return an integer (0-3) representing the game status. It should return 0 if there is a tie, 1 if it's a win situation for player1, 2 if it's a win situation for player2, and 3 if there isn't any result yet.

The *makeMove* function will allow a player to make a move. This function should accept a 2D array of characters representing the game board, an integer representing which player's turn it is, and a string representing the player's name. It should prompt the player to specify his move in terms of the row and column index. Then a change has to be made in the game board, i.e., the specified cell of the 2D array should be set to an appropriate character depending on which player it is. If it is player1, then the corresponding position of the 2D array should be set to 'X'. If it is player2, then the corresponding position of the 2D array should be set to 'O'. Note that a player will be entering row and column numbers

between 1 and 3, but the actual row and column indices in the 2D array should be between 0 and 2. For example, if a player enters 3 2, that will be row index 2 and column index 1 in the 2D array. This function should also make sure that a player makes its move to a blank and valid (inside the array bounds) position. If a player attempts to make a move to an invalid or already occupied position, it should keep on re-prompting the player (as shown in the sample executions) to make another move as long as the player is not making a valid move.

Inside your *main*, you will need to declare the 2D array representing the game board and initialize the board by calling the *initializeBoard* function. Then you will have to prompt the players to enter their names and implement a toss to select one of them as the first player. You will need to generate a random number (1 or 2) to implement the toss (refer to the lecture slides on functions). Suppose the two players enter their names as follows:

```
First player, enter your name: John
Second player, enter your name: David
```

You may use two string variables (*player1* and *player2*) to save these names. If the random number is 1, then John wins the toss and remains Player1 (and David remains Player2). If the random number is 2, then David wins the toss and becomes Player1 (and John becomes Player2). In this case, you will need to swap the values of *player1* and *player2*.

After selecting the order of the players, you will need to start a loop that should continue until a result is reached. You will need to use the *getStatus* function to determine when the loop should terminate. The loop should continue as long as the *getStatus* function is returning 3 (no results yet). Inside the loop, you should alternate between player1 and player2, calling the *printBoard* and *makeMove* functions for each player. Your program trace should have exactly the same format as shown in the sample executions. Note that the actual trace may be different because of the random toss.

You will need to write the program in the following steps:

1) Implement the *initializeBoard* function.
2) Implement the *printBoard* function. To test this function, create a 2D array of characters in main, call the *initializeBoard* function, and then the *printBoard* function. This should display an empty board in the grid. Then assign 'X' to [0,0] and 'O' to [2,2] and call the *printBoard* function. If these functions are correct, the game should have an 'X' at the top left corner and a 'Y' at the bottom right corner.
3) Implement the *getStatus* function. To test this function, cout the return value of the *getStatus* function after you have called the *initializeBoard* function. This should display 3. Then assign 'X' and 'O' to the different cells and check the return value of the *getStatus* function. Repeat this process once for each possible outcome, i.e., a win for 'X', a win for 'O', and a tie.
4) Implement the *makeMove* function and test it from main.

Once all these functions are working properly, you may start implementing your game playing loop in main. This program may become difficult if you are not implementing it using a modular approach. Write each individual function, test it, and then write the next function.

Instructions:

- This will be an individual programming project.
- Write your code in a file called **prog2.cpp**.
- Use meaningful variable names, helpful comments, and a consistent coding style.
- Your program file should have the appropriate comment block at the top: `// Name: Your Name`

```
// File Name: prog1.cpp
// Date: Day Month, Year
// Program Description: brief description of the program
```

Deliverables:

You will need to submit the following in Blackboard by 3:00 PM on September 13:

- The source file ***prog1.cpp*** and the screenshot of your program execution in Visual Studio or Eclipse.

Grading:

This project is worth 20 points distributed as follows:

In-Lab Demo (8 pts)
 Correct implementation of initializeBoard function (1 pt)
 Correct implementation of printBoard function (2 pts)
 Correct implementation of getStatus (5 pts)
Program Completion (8 pts)
 Correct implementation of makeMove function (4 pts)
 Correct implementation of main function (4 pts)
Program Style (4 pts)
 Meaningful variable names (1 pt)
 Proper indentation (1 pt)
 Sufficient comments (2 pts)

There will be a 25% penalty if the program does not compile and a 25% penalty if a screenshot is not submitted.

Sample Execution I:

```
First player, enter your name: John
Second player, enter your name: David

David won the toss.

  | |
-----------
  | |
-----------
  | |

David, specify your move: 1 1

 X | |
-----------
  | |
-----------
  | |
```

```
John, specify your move: 2 1
 X | |
-----------
 O | |
-----------
 | |

David, specify your move: 3 3

 X | |
-----------
 O | |
-----------
 | | X

John, specify your move: 3 2

 X | |
-----------
 O | |
-----------
 | O | X

David, specify your move: 1 1
Illegal move, enter again: 2 1
Illegal move, enter again: 3 4
Illegal move, enter again: 2 2

 X | |
-----------
 O | X |
-----------
 | O | X

David has won!
```

Sample Execution II:

```
First player, enter your name: Susie
Second player, enter your name: Jake

Susie won the toss.

 | |
-----------
 | |
-----------
 | |

Susie, specify your move: 3 3

 | |
-----------
 | |
-----------
```

```
 | | X
Jake, specify your move: 2 2

 | |
-----------
 | O |
-----------
 | | X

Susie, specify your move: 1 1

 X | |
-----------
 | O |
-----------
 | | X

Jake, specify your move: 2 3

 X | |
-----------
 | O | O
-----------
 | | X

Susie, specify your move: 1 3

 X | | X
-----------
 | O | O
-----------
 | | X




Jake, specify your move: 1 2
 X | O | X
-----------
 | O | O
-----------
 | | X

Susie, specify your move: 3 2

 X | O | X
-----------
 | O | O
-----------
 | X | X

Jake, specify your move: 2 1

 X | O | X
```

```
-----------
 O | O | O
-----------
  | X | X

Jake has won!
```

Sample Execution III:

```
Welcome to Tic-Tac-Toe
----------------------

First player, enter your name: Robbie
Second player, enter your name: Bill

Bill won the toss.

  | |
-----------
  | |
-----------
  | |

Bill, specify your move: 3 2

  | |
-----------
  | |
-----------
  | X |

Robbie, specify your move: 1 2

  | O |
-----------
  | |
-----------
  | X |

Bill, specify your move: 1 3

  | O | X
-----------
  | |
-----------
  | X |

Robbie, specify your move: 3 3

  | O | X
-----------
  | |
-----------
  | X | O
```

```
Bill, specify your move: 2 3

 | O | X

-----------
 | | X
-----------
 | X | O

Robbie, specify your move: 2 2

| O | X

-----------
 | O | X
-----------
 | X | O

Bill, specify your move: 1 1

 X | O | X
-----------
 | O | X
-----------
 | X | O

Robbie, specify your move: 3 1

 X | O | X
-----------
 | O | X
-----------
 O | X | O

Bill, specify your move: 2 1

 X | O | X
-----------
 X | O | X
-----------
 O | X | O

Game tied!
```

** Anything typed in blue indicates a user input.