Tic-Tac-Toe Lab
COMP 150/170
10 Points

You are to write a C++ program that allows the user to play tic-tac-toe against the computer. A portion of the program has been written for you and is available at \\cs1\Classes\Comp150\tictactoe.cpp.

The tic-tac-toe board is configured so that the different squares are numbered from 1 to 9. The example run shown below illustrates how the game is to proceed (several moves are not shown). The user (X) moves first, and the computer (O) responds. Play continues until either X wins, O wins, or there's a tie.

```
1 | 2 | 3
-----------
4 | 5 | 6
-----------
7 | 8 | 9

Enter your move: 1

X | 2 | 3
-----------
4 | 5 | 6
-----------
7 | 8 | 9

Computer is moving to 4

X | 2 | 3
-----------
O | 5 | 6
-----------
7 | 8 | 9

(Skipping…)

Enter your move: 5

X | 2 | 3
-----------
O | X | O
-----------
7 | 8 | X

X wins!!!
```

The program uses a character array to represent the board:

```
char board[9] = {'1','2','3','4','5','6','7','8','9'};
```

The main and DisplayBoard functions have been written for you, but you need to complete the other 3 functions to get the program to work correctly. Below is a description of the functions:

1. **DisplayBoard** – takes the board its single (read-only) argument and outputs the board to the screen.

2. **CheckForWinner** – takes the board as its single (read-only) argument and returns 0 if there's no winner yet, 1 if it's a tie, 2 if X has won, and 3 if O has won.

3. **GetUserMove** – takes the board as its only argument. It asks the user where the X should be placed. If the user enters an illegal position (it is not 1-9 or there is already an X or O at the position), an error message should be given, and the user should be continually prompted until a valid position is given. The function should set the appropriate slot in the board array to the position. Other than modify the board, this function does not return a value.

4. **GetComputerMove** – takes the board as its only argument. It randomly selects an empty square to place an O. Other than modify the board, this function does not return a value.

The rand function should be used to choose a random number. You must add #include <cstdlib> to your program to use rand. The rand function returns an integer in the range of 0 to RAND_MAX (a really large number). To limit the number to a smaller range, you can use the mod operator:

```
int r = rand() % 100;   // Return a random number between 0 and 99
```

Unfortunately, rand() will always return back the same set of random numbers each time the program is executed unless, you give it a unique seed. This is typically done using the current date and time with the time function (you must include <ctime> to access the time function). This call needs to only appear once in your main function:

```
srand(unsigned int(time(NULL)));  // Initialize random seed
```

You will need to create a loop in GetComputerMove that continues to generate a random number between 0 and 8 (the slot numbers of the board array) until it finds a slot in board that does not already have an X or O. GetComputerMove should then make the randomly selected position of the board an O. Other than modify the board, this function does not return a value.

Bonus 3 points: Instead of having the computer randomly select a location to move each time, make the computer choose smarter moves. This requires adding an artificial intelligence (AI) to GetComputerMove. Your AI should first look for a winning move. If one can't be found, it should look for a blocking move (stopping X from winning). If there aren't any winning or blocking moves available, it should randomly select a move.

Although you could have a long series of if statement checking each location of the board to see if it would produce a winning move, it is easier to make use of the CheckForWinner function. Below is pseudocode using CheckForWinner that you should implement in your GetComputerMove function:

```
// First see if there's a move I can make to win
For each square in the board
        If the square is available
                Set the square temporarily to O
                If CheckForWinner says I would win then
                        Leave the square O and return from the function (return;)
                Else
                        Change the square back to its original value

// Now see if there's a move I can make to block X from winning
For each square in the board
        If the square is available
                Set the square temporarily to X
                If CheckForWinner says X would win then
                        Set the square to O and return from the function (return;)
                Else
                        Change the square back to its original value

// I can't win or block X, so make a random move
Set randomly selected square to O that isn't already used
```

Note that the return statement can be used in void functions to leave the function early. You will find this useful when implementing the algorithm above.

Submit your working program to Easel (http://cs.harding.edu/easel/) before the beginning of the next class period.