Algorithms & Data Structures

REPORT

Fletcher Thomas Moore | SET08122 | 30/03/2019

Contents

Introduction

The task at hand for the Algorithms and Data Structures coursework was to create a program using the knowledge and understanding of the theory and practical work gained throughout the module. Ultimately, the program will be a fully functional game of tic-tac-toe that allows two players to compete against one another. The coding language is to be C and the game will run on the *Developer Command Prompt for VS 2017*.

The program should consist of at *least* four features. First off, a game board laid out in a clear and clean display with nine available placements for the user's 'X' and 'O' to be placed. Second, the players must be defined in someway so that the player's whose turn it is can be displayed along with whoever has won the game. Next is the pieces that the user's will be placing on the board such as the 'X' and 'O' depending on the player. Lastly, the positions on the board need to be made clear somehow so each player knows and can execute their move.

Other features could be added that I feel are necessary or would benefit the program as well as additional features for a more respectable final mark if this can be done within the time constraints given.

Design

Instead of having the program running completely from the *Main* method, which can be messy and bad practice, the program was divided into methods that could be called when the main method was running. I found it best to break the program down to deal with each of the features individually. This also helped when it came to finding error's in my code each time I ran it.

To begin with, I created the *Board* method which dealt with the layout of the board for the game play. When this is called in the main method, the board places (1-9) are inserted into the board so that the users can simply pick a number displayed for where they would like their 'O' or 'X' to be placed into.

The method *PlayerTurn* determines which player will be using the 'O' pieces and which player will be using the 'X' pieces. The player will then be prompted to select a place on the board where they would like their piece to be placed. This input is validated so that it is between 1-9 and if it isn't an error message is displayed with the chance of another input from the user. The place selected can't already have a game piece within it, otherwise another error is displayed, and the user is prompted for another input. After the player input is valid, it is then stored and displayed on the board.

PossibleWinner, is the last method outside of the *Main* method and determines what player has won. The method covers all the possibilities of winning the game; horizontally, vertically, and diagonally. Each winning possibility returns '1' and if none of the outcomes are met then the method returns '0'. This is then used in the *Main* method.

Lastly, there is the *Main* method that contains to counters. One counter, *player*, keeps track of the player number and the other counter *turns*, keeps track of the amount of turns that game has played through. Everything is contained in a while loop which continues to run whilst *PossibleWinner* is returning '0' as no-one has one unless the *turn* counter reaches 9. It is then stalemate and the user is given the option to restart the game or finish. When there is a winner, the program declares which player has won and then asks if the game is to be reset to play again or finish.

All the code is commented for easy editing if the program was to be enhanced or needed maintenance carried out. This makes code clear and readable.

Enhancements

The additional tasks would have been a bonus, but in my case and circumstances at the time it wasn't manageable. The three extra tasks in the coursework would have been a nice feature, such as the undo and redo facility as well as recording the history of the game. This would have allowed the user to *save* a specific gameplay to be played out later.

Personalizing the game would have been a pleasant extra by asking for each user's name and replacing "player 1" and "player 2" with the name's input at the beginning. It would have also helped when *saving* a game as it could be saved under the winner's name or a versus title, for example: *John vs Smith*.

Another, more complicated and intriguing enhancement, would be changing up the board shape and size as well as having multiple players. The beginning of the program could ask for the number of players and possibly the player's names and at the start of each turn generate a random number (or a name from the list of players) to determine whose turn it is, randomizing the order of the players turns.

Critical Evaluation

One of the features that worked well was the option for the user to restart the game whether there was an overall winner of the game or the game was left at a stalemate. This allows the players to simply input 'y' or 'n' for the board to clear and counters to be reset rather than loading the full game from scratch once again.

There is a minor issue with the validation in the input for the board placement. If one of the user's input anything other than a number whether it be a character or string of characters, the program repeatedly outputs the error message and does not allow the user to input anything else. *CTRL + C* stops the program and it must restart. I did try and resolve the issue, but unfortunately ran out of time to do anything further before focusing on the report.

Personal Evaluation

In all honesty, I believe I could have achieved a lot more during this coursework and aimed a lot higher than what I have. Pushing my boundaries more is something that I should have done, but instead stuck to my comfort zone. Looking at the program now, I realize this a little too late.

I could have experimented with the board layout, personalized the game a little more and attempted the additional tasks even if they did not work. Instead, I continued to do my best to perfect what I had rather than using the little time I had left pushing and using more creativity to achieve much more than I have.

This program is something I hope to come back to in my spare time and expand on without time constraints to develop my skill set further.

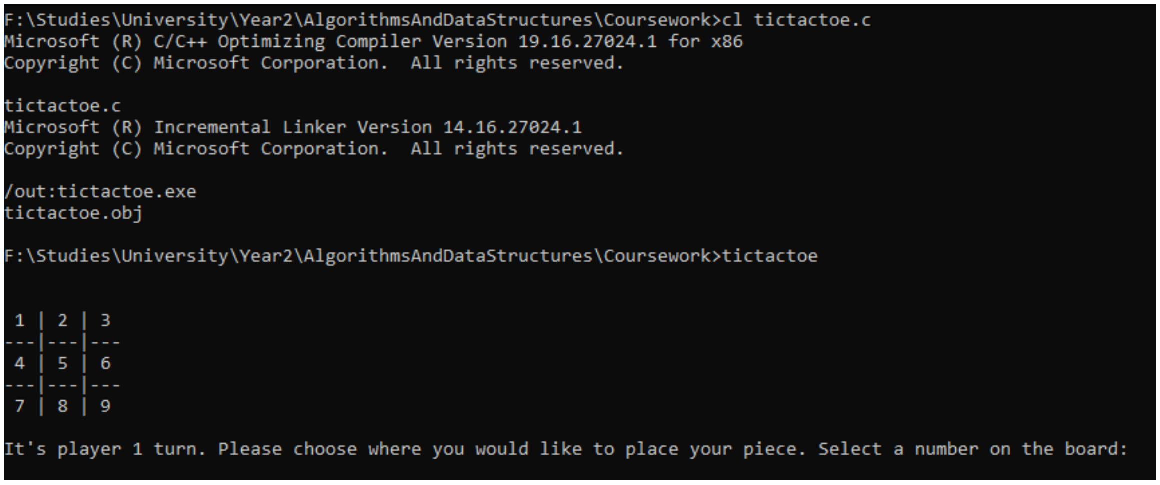
References

For *scanf*: https://en.wikibooks.org/wiki/C_Programming/Simple_input_and_output

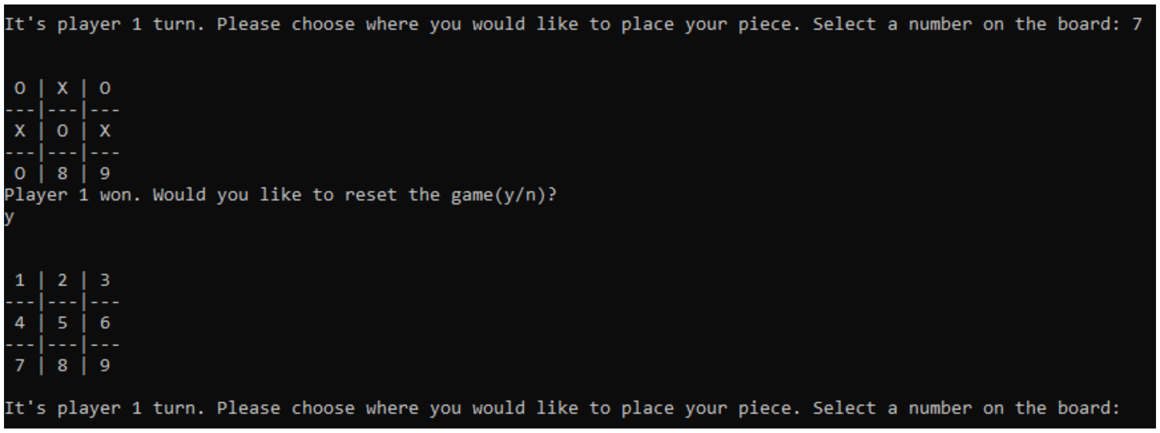
https://github.com/klasp100/tic-tac-toe/blob/master/tic%20tac%20toe.c

Appendices

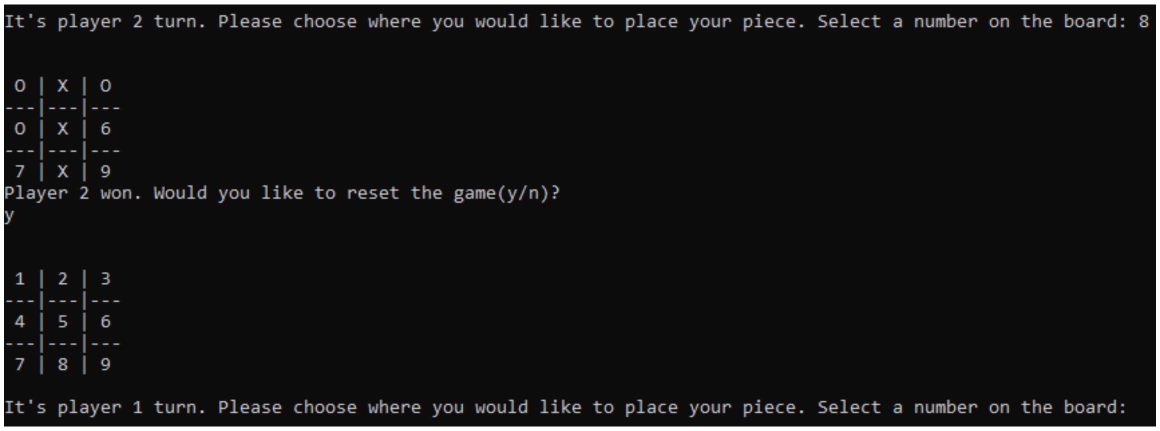
SCREEN SHOTS



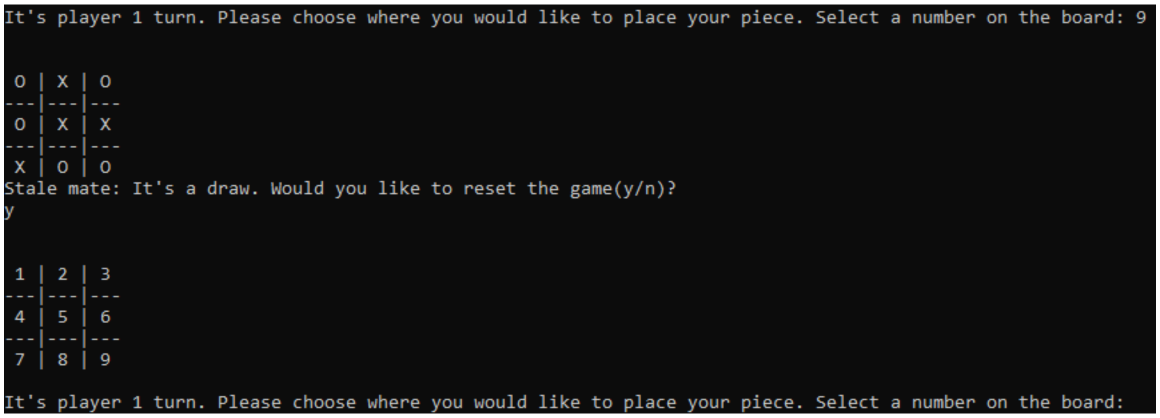
```
F:\Studies\University\Year2\AlgorithmsAndDataStructures\Coursework>cl tictactoe.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.16.27024.1 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

tictactoe.c
Microsoft (R) Incremental Linker Version 14.16.27024.1
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:tictactoe.exe
tictactoe.obj

F:\Studies\University\Year2\AlgorithmsAndDataStructures\Coursework>tictactoe


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

It's player 1 turn. Please choose where you would like to place your piece. Select a number on the board:
```



```
It's player 1 turn. Please choose where you would like to place your piece. Select a number on the board: 7


 O | X | O
---|---|---
 X | O | X
---|---|---
 O | 8 | 9
Player 1 won. Would you like to reset the game(y/n)?
y


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

It's player 1 turn. Please choose where you would like to place your piece. Select a number on the board:
```

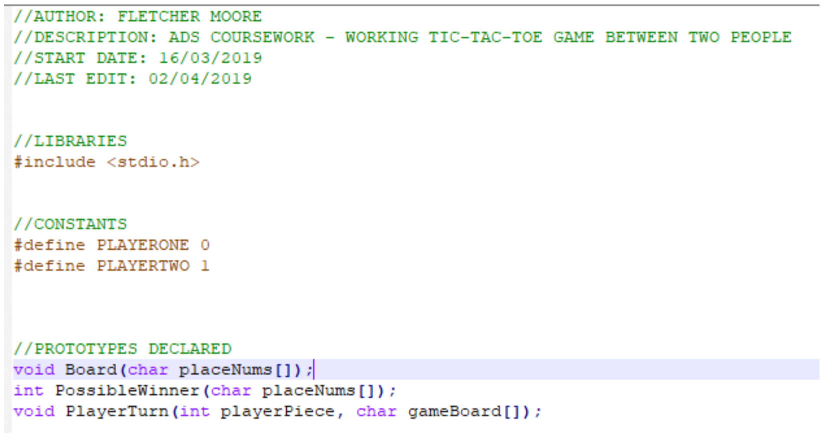


```
It's player 2 turn. Please choose where you would like to place your piece. Select a number on the board: 8


 O | X | O
---|---|---
 O | X | 6
---|---|---
 7 | X | 9
Player 2 won. Would you like to reset the game(y/n)?
y


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

It's player 1 turn. Please choose where you would like to place your piece. Select a number on the board:
```

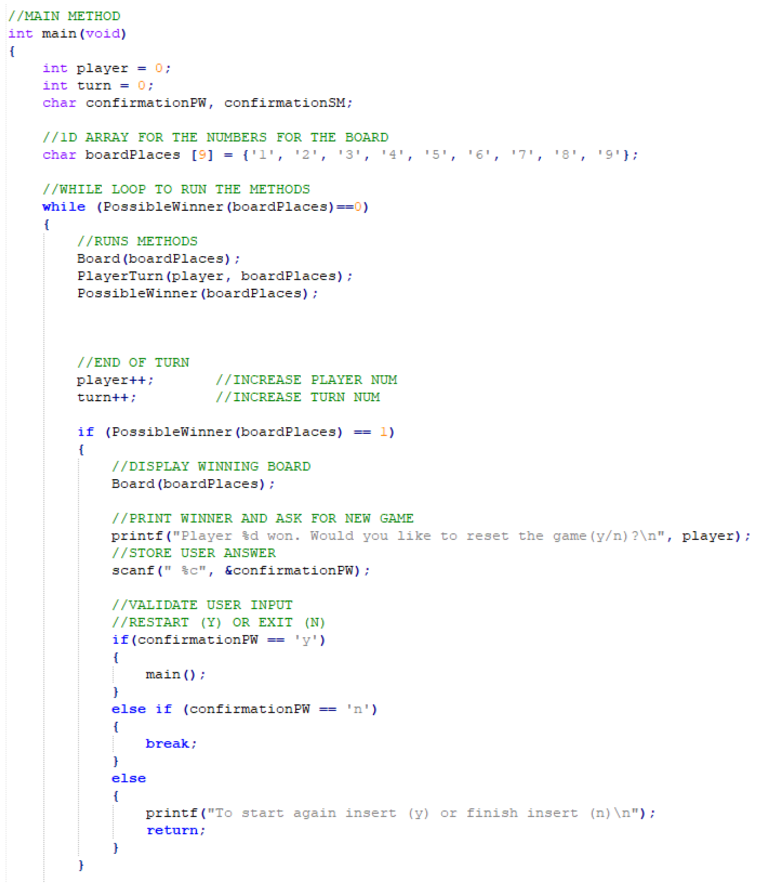


```
It's player 1 turn. Please choose where you would like to place your piece. Select a number on the board: 9


 O | X | O
---|---|---
 O | X | X
---|---|---
 X | O | O
Stale mate: It's a draw. Would you like to reset the game(y/n)?
y


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

It's player 1 turn. Please choose where you would like to place your piece. Select a number on the board:
```

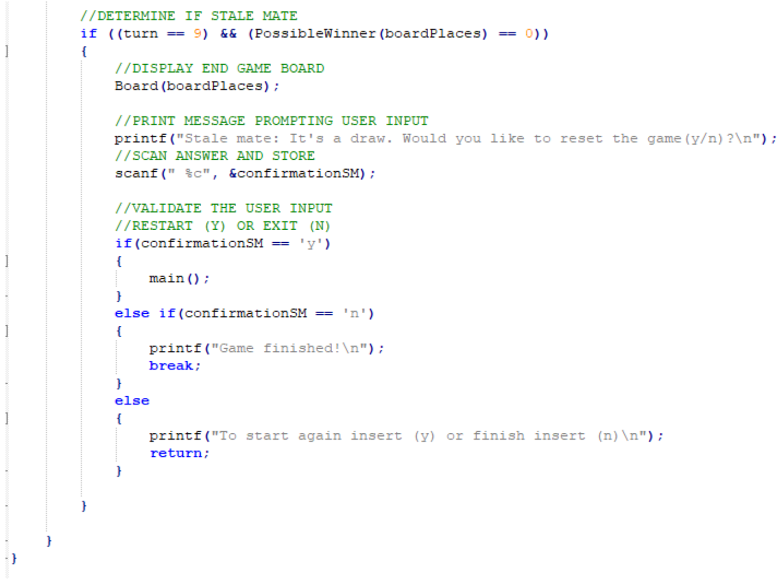
CODE



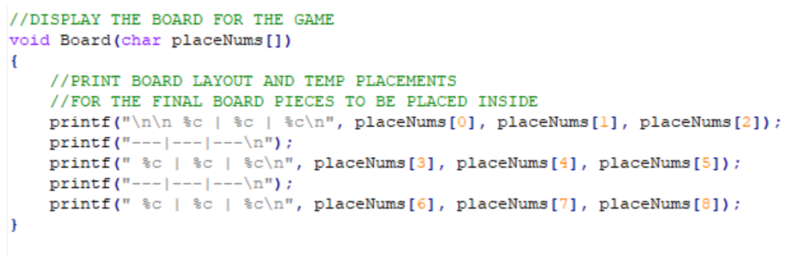
```c
//AUTHOR: FLETCHER MOORE
//DESCRIPTION: ADS COURSEWORK - WORKING TIC-TAC-TOE GAME BETWEEN TWO PEOPLE
//START DATE: 16/03/2019
//LAST EDIT: 02/04/2019


//LIBRARIES
#include <stdio.h>


//CONSTANTS
#define PLAYERONE 0
#define PLAYERTWO 1



//PROTOTYPES DECLARED
void Board(char placeNums[]);
int PossibleWinner(char placeNums[]);
void PlayerTurn(int playerPiece, char gameBoard[]);
```



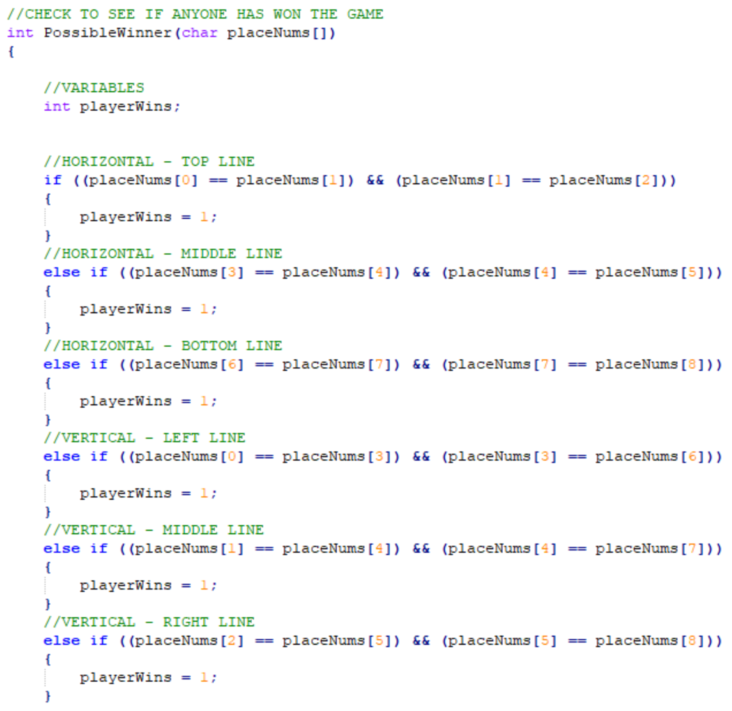
```c
//MAIN METHOD
int main(void)
{
    int player = 0;
    int turn = 0;
    char confirmationPW, confirmationSM;

    //1D ARRAY FOR THE NUMBERS FOR THE BOARD
    char boardPlaces [9] = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};

    //WHILE LOOP TO RUN THE METHODS
    while (PossibleWinner(boardPlaces)==0)
    {
        //RUNS METHODS
        Board(boardPlaces);
        PlayerTurn(player, boardPlaces);
        PossibleWinner(boardPlaces);



        //END OF TURN
        player++;       //INCREASE PLAYER NUM
        turn++;         //INCREASE TURN NUM

        if (PossibleWinner(boardPlaces) == 1)
        {
            //DISPLAY WINNING BOARD
            Board(boardPlaces);

            //PRINT WINNER AND ASK FOR NEW GAME
            printf("Player %d won. Would you like to reset the game(y/n)?\n", player);
            //STORE USER ANSWER
            scanf(" %c", &confirmationPW);

            //VALIDATE USER INPUT
            //RESTART (Y) OR EXIT (N)
            if(confirmationPW == 'y')
            {
                main();
            }
            else if (confirmationPW == 'n')
            {
                break;
            }
            else
            {
                printf("To start again insert (y) or finish insert (n)\n");
                return;
            }
        }
```



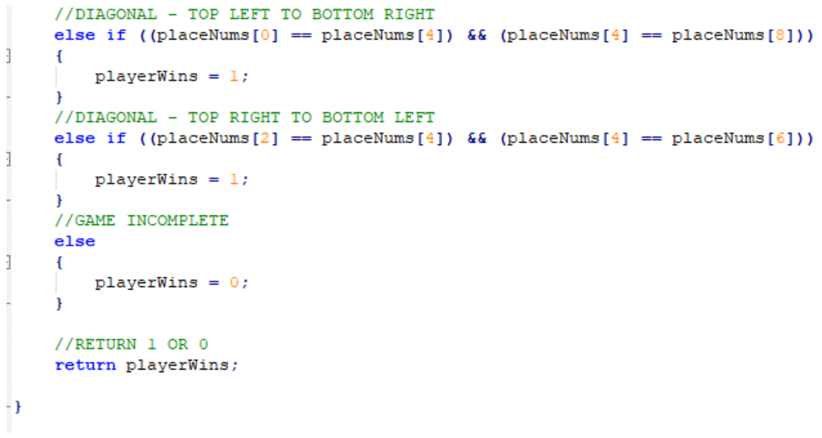
```c
        //DETERMINE IF STALE MATE
        if ((turn == 9) && (PossibleWinner(boardPlaces) == 0))
        {
            //DISPLAY END GAME BOARD
            Board(boardPlaces);

            //PRINT MESSAGE PROMPTING USER INPUT
            printf("Stale mate: It's a draw. Would you like to reset the game(y/n)?\n");
            //SCAN ANSWER AND STORE
            scanf(" %c", &confirmationSM);

            //VALIDATE THE USER INPUT
            //RESTART (Y) OR EXIT (N)
            if(confirmationSM == 'y')
            {
                main();
            }
            else if(confirmationSM == 'n')
            {
                printf("Game finished!\n");
                break;
            }
            else
            {
                printf("To start again insert (y) or finish insert (n)\n");
                return;
            }

        }

    }
}
```



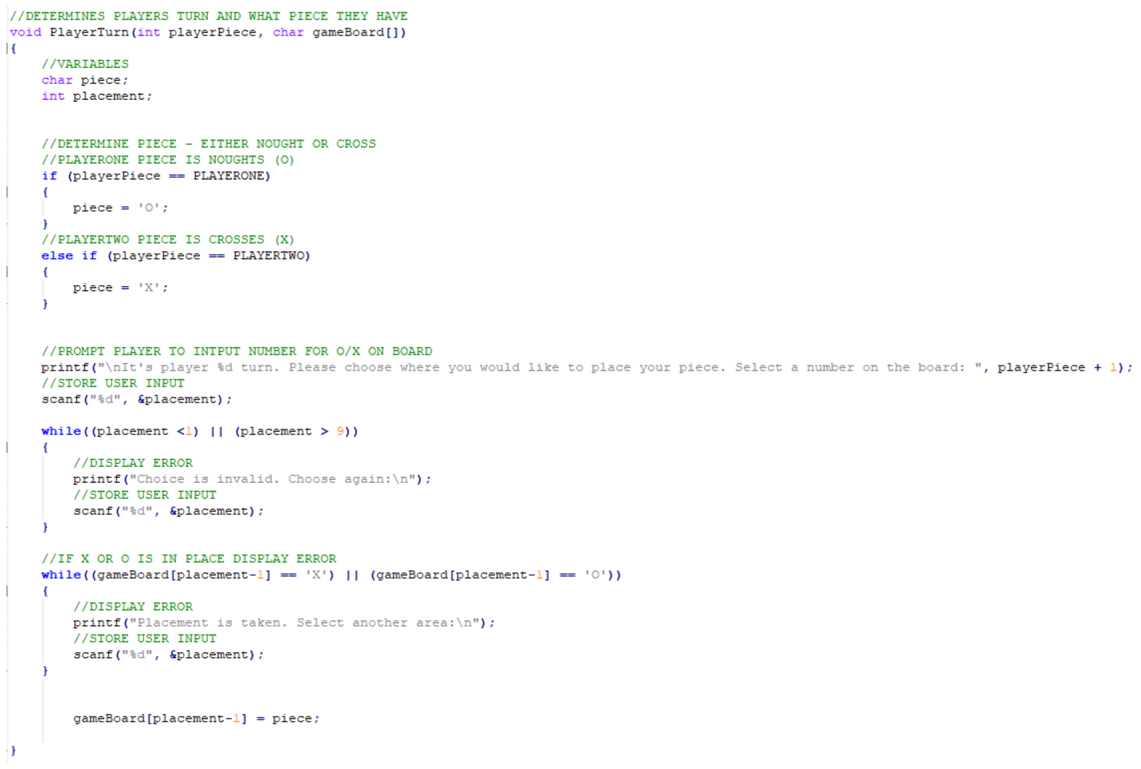
```c
//DISPLAY THE BOARD FOR THE GAME
void Board(char placeNums[])
{
    //PRINT BOARD LAYOUT AND TEMP PLACEMENTS
    //FOR THE FINAL BOARD PIECES TO BE PLACED INSIDE
    printf("\n\n %c | %c | %c\n", placeNums[0], placeNums[1], placeNums[2]);
    printf("---|---|---\n");
    printf(" %c | %c | %c\n", placeNums[3], placeNums[4], placeNums[5]);
    printf("---|---|---\n");
    printf(" %c | %c | %c\n", placeNums[6], placeNums[7], placeNums[8]);
}
```



```c
//CHECK TO SEE IF ANYONE HAS WON THE GAME
int PossibleWinner(char placeNums[])
{

    //VARIABLES
    int playerWins;


    //HORIZONTAL - TOP LINE
    if ((placeNums[0] == placeNums[1]) && (placeNums[1] == placeNums[2]))
    {
        playerWins = 1;
    }
    //HORIZONTAL - MIDDLE LINE
    else if ((placeNums[3] == placeNums[4]) && (placeNums[4] == placeNums[5]))
    {
        playerWins = 1;
    }
    //HORIZONTAL - BOTTOM LINE
    else if ((placeNums[6] == placeNums[7]) && (placeNums[7] == placeNums[8]))
    {
        playerWins = 1;
    }
    //VERTICAL - LEFT LINE
    else if ((placeNums[0] == placeNums[3]) && (placeNums[3] == placeNums[6]))
    {
        playerWins = 1;
    }
    //VERTICAL - MIDDLE LINE
    else if ((placeNums[1] == placeNums[4]) && (placeNums[4] == placeNums[7]))
    {
        playerWins = 1;
    }
    //VERTICAL - RIGHT LINE
    else if ((placeNums[2] == placeNums[5]) && (placeNums[5] == placeNums[8]))
    {
        playerWins = 1;
    }
```



```c
    //DIAGONAL - TOP LEFT TO BOTTOM RIGHT
    else if ((placeNums[0] == placeNums[4]) && (placeNums[4] == placeNums[8]))
    {
        playerWins = 1;
    }
    //DIAGONAL - TOP RIGHT TO BOTTOM LEFT
    else if ((placeNums[2] == placeNums[4]) && (placeNums[4] == placeNums[6]))
    {
        playerWins = 1;
    }
    //GAME INCOMPLETE
    else
    {
        playerWins = 0;
    }

    //RETURN 1 OR 0
    return playerWins;

}
```



```
//DETERMINES PLAYERS TURN AND WHAT PIECE THEY HAVE
void PlayerTurn(int playerPiece, char gameBoard[])
{
    //VARIABLES
    char piece;
    int placement;


    //DETERMINE PIECE - EITHER NOUGHT OR CROSS
    //PLAYERONE PIECE IS NOUGHTS (O)
    if (playerPiece == PLAYERONE)
    {
        piece = 'O';
    }
    //PLAYERTWO PIECE IS CROSSES (X)
    else if (playerPiece == PLAYERTWO)
    {
        piece = 'X';
    }


    //PROMPT PLAYER TO INTPUT NUMBER FOR O/X ON BOARD
    printf("\nIt's player %d turn. Please choose where you would like to place your piece. Select a number on the board: ", playerPiece + 1);
    //STORE USER INPUT
    scanf("%d", &placement);

    while((placement <1) || (placement > 9))
    {
        //DISPLAY ERROR
        printf("Choice is invalid. Choose again:\n");
        //STORE USER INPUT
        scanf("%d", &placement);
    }

    //IF X OR O IS IN PLACE DISPLAY ERROR
    while((gameBoard[placement-1] == 'X') || (gameBoard[placement-1] == 'O'))
    {
        //DISPLAY ERROR
        printf("Placement is taken. Select another area:\n");
        //STORE USER INPUT
        scanf("%d", &placement);
    }


        gameBoard[placement-1] = piece;

}
```