Tic Tac Toe Project, Fall 2025

Tong Yi

Contents

1 CopyRight Warning

2. Our project is different from similar products in Internet.
   (a) Ask help only from teaching staff of this course.
   (b) Use solutions from artificial intelligence like ChatGPT or online tutoring websites like, but not limited to, chegg.com, violates academic integrity and is not allowed.

2 Example of Separating Declaration, Definition, and Test Code: FuelTank Class

For robust software engineering practices, especially in large projects, the standard C++ approach is to divide the code into three distinct files:

Header File (.h or .hpp): Contains the declaration of the class (the class blueprint, member variables, and function prototypes).

Source File (.cpp): Contains the definitions (implementation logic) of the class's member functions.

Test File (.cpp): Contains the main() function and the code dedicated to testing the functionality of the class.

This separation improves compile times (only changed implementation files need recompilation), enhances code organization, and facilitates code reuse across different parts of a project.

Code link: `https://onlinegdb.com/MzXcjYT2S-`.

In online compilers like onlinegdb, the primary source file containing the `main` function must be named `main.cpp`. When working on your local computer, you have more flexibility and can name the test file something more descriptive, such as `TestFuelTank.cpp`.

2.1 FuelTank.hpp

```cpp
#ifndef FUEL_TANK_H
#define FUEL_TANK_H
class FuelTank {
public:
    FuelTank();
    FuelTank(int startLevel);
    int getGasLevel() const;
    void fillGas(int numGallons);
    void sendGas(int numGallons);

private:
    int currGasLevel;
};
#endif
```

2.2 FuelTank.cpp

```cpp
#include "FuelTank.hpp"

```

```
FuelTank::FuelTank() {
   //int currGasLevel = 0; //wrong, with int before currGasLevel, the
      currGasLevel is a local variable
   //for default constructor,
   //that statement does not initialize data member currGasLevel
   currGasLevel = 0;
}

FuelTank::FuelTank(int startGasLevel) {
    if (startGasLevel >= 0) {
       currGasLevel = startGasLevel;
    }
    else {
       currGasLevel = 0;
    }
}

int FuelTank::getGasLevel() const {
    return currGasLevel;
}

void FuelTank::sendGas(int gallons) {
    if (currGasLevel >= gallons) {
       currGasLevel -= gallons;
    }
}

void FuelTank::fillGas(int gallons) {
    currGasLevel += gallons;
}
```

2.3 TestFuelTank.cpp

```
//Sample output:
//gas level of an empty fuel tank is 0
//add 5 gallons to tank
//send 3 gallons out from tank
//current gas level is 2

#include "FuelTank.hpp"
#include <iostream>

int main() {
    //TODO: Call default constructor of FuelTank class to instantiate an
       object called tank
    //related: call default constructor of string class to instantiate an empty string object called str.
    FuelTank tank(0); //same as FuelTank tank; //same as FuelTank tank; //same as FuelTank tank; //same as FuelTank tank;
    //similar to
    //string str; //str is an empty string

    //TODO: print out the current gas level
    std::cout << "gas level of an empty fuel tank is "
         << tank.getGasLevel() << std::endl; //fill in ... part

    //TODO: write a statement to add 5 gallons of gas to tank
    //related: call substr method of str object to get a substring
    std::cout << "add 5 gallons to tank" << std::endl;
    tank.fillGas(5);

    std::cout << "send 3 gallons out from tank" << std::endl;

    //TODO: write a statement to send 3 gallons of gas from tank
    tank.sendGas(3);

    std::cout << "current gas level is ";
    //TODO: write a statement to find out and print the current gas level of tank
    //related: find out the length of a string object str
    std::cout << tank.getGasLevel() << std::endl;

    //WRONG: currGasLevel is a private data member and cannot be accessed directly from a function outside the class where currGasLevel is defined
    //std::cout << tank2.currGasLevel << std::endl;
    return 0;
}
```

To run the code,

```
g++ -o run FuelTank.cpp TestFuelTank.cpp
./run
```

3 Overview of TicTacToeGame Project

This project involves two classes: TicTacToeBoard and TicTacToeGame.

- TicTacToeBoard is a square-shape grid with characters 'X', 'O', and space character ' '. There are operations like `clear`, `mark`, `getValue` to operate on the grid.
- TicTacToeGame lets a human and a computer play games using a TicTacToeBoard.

To run the project, we do the following.

1. Build a directory in your computer by entering the following command with return key in a terminal.

```
mkdir TicTacToeGame
```

2. Move to the directory by pressing the following command with return key in a terminal.

```
cd TicTacToeGame
```

3. Download the following files and save in the above directory.

   (a) Header file of TicTacToeBoard: **TicTacToeBoard.hpp**

   (b) Header file of TicTacToeGame from **TicTacToeGame.hpp**

   (c) Source code to test important methods of TicTacToeBoard class: **TestTicTacToeBoard.cpp**

   (d) Source code to test TicTacToeGame class: **TestTicTacToeGame.cpp**

   (e) Makefile automates the process of building executable programs and other files from source code: **makefile**
   A `makefile` is a plain text file that contains instructions used by the make utility to automate software compilation and other tasks. It defines targets, dependencies, and shell commands (recipes) needed to build a project efficiently.

4. You implement TicTacToeBoard and TicTacToeGame classes, that is,

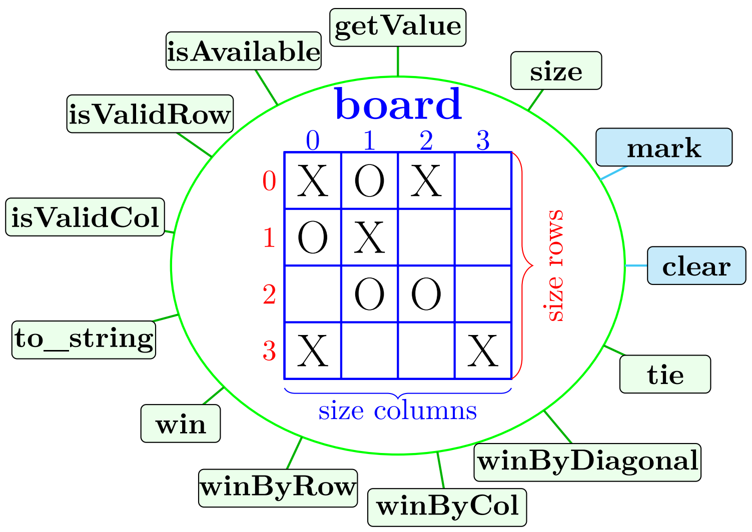
   (a) Define TicTacToeBoard.cpp in Tasks A and B.
   Download **TicTacToeBoard.cpp**. Finish the TODO parts. For more details, see Section 4.3.

   (b) Define TicTacToeGame.cpp in Tasks C and D.

4 Define TicTacToeBoard Class

4.1 Illustration of TicTacToeBoard Class

The following figure shows the data member of a TicTacToeBoard object and its methods.



1. Inside the circle, we have data member called `board`, which is a two-dimensional array of chars with equal number of rows and columns.

   The data member is enclosed in the circle, the only way to access or modify it is through the buttons on the circle. You may think the buttons as little robots as well.

2. A cyan-colored button – clear, mark – repressents a modifier (or a setter), method that changes data members.

3. A green button represents an accessor (or a getter), method that only access data members without changing them.

4. Constructors are object-makers, not methods and are **not** shown in the figure.

   (a)
   ```
   void clear();
   ```

   Set each cell of data member `board` to be a space symbol.

   A game finishes with symbol 'X' wins.

	0	1	2	3
0	X	O	O	
1	O	X		
2			X	
3				X

   Clear the board by setting each symbol to a space so that we can play another game.

	0	1	2	3
0				
1				
2				
3				

   (b)
   ```
   void mark(int row, int col, char symbol);
   ```

   set the cell at (`row`)th row index and (`col`)th column index to be `symbol`.

   A board starts with all space characters.

	0	1	2	3
0				
1				
2				
3				

   Place X at the cell with row index 0 and column index 1.

	0	1	2	3
0		X		
1				
2				
3				

   (c) For requirements of constructors and other methods, please read the comments in **TicTacToe-Board.cpp**.

4.2 Header File of TicTacToeBoard class: TicTacToeBoard.hpp

The contents of **TicTacToeBoard.hpp** are as follows.

```
#ifndef TicTacToeBoard_H
#define TicTacToeBoard_H
#include <iostream> //std
#include <vector>

class TicTacToeBoard {
public:
```

```
    //default constructor creates a 3x3 board
    TicTacToeBoard();

    //If givenSize is less than 3, create a 3x3 board.
    //Otherwise, initialize the board with the given size for both rows and
        columns.
    TicTacToeBoard(int givenSize);

    //Set each cell of the board to be a space character.
    //Warning: our clear method of TicTacToeBoard is different from clear
       method of a vector.
    //(1) clear method of a vector removes all elements from the vector (
       which are destroyed), leaving the container with a size of 0.
    //(2) That is, board.clear() removes all elements of board and make
       board an empty vector.
    //(3) Our clear method does not change the structure of board -- its
       number of rows and number of columns will still keep the same --
       just set each element to be a space.
    void clear();

    //if row and col are valid indices,
    //return the value of the cell of the board at (row, col)
    //otherwise, return 'W' (means wrong)
    //Warning: a function with non-void return type
    //needs to return a value of that return type in every execution path.
    char getValue(int row, int col) const;

    //Checks if the cell at (row, col) is available or not.
    //A cell is available if the following conditions satisify.
    //(1) row and col indices are valid
    //(2) the value at that cell is a space character (' ').
    //Otherwise, the cell is not available.
    bool isAvailable(int row, int col) const;

    //Check whether the given parameter row is valid row index or not.
    //that is, whether row is in [0, board.size()-1]
    bool isValidRow(int row) const;

    //Check whether the given parameter col is a valid column index or not
    //that is, whether col is in [0, board[0].size()-1]
    bool isValidCol(int col) const;

    //Return the number of rows of board
    int size() const;

    //IF row and col are valid indices,
    //place symbol at board[row][col],
    //otherwise, do nothing.
    //Note that before we can use board[row][col],
    //need to make sure that row and col indices are valid.
    void mark(int row, int col, char symbol);

    //Return a string representation of the contents of board.
    //(1) The returned string typically formats the board in a grid layout
    //for easy display and debugging.
    //(2) 'X', 'O', and ' ' characters in the string are
    //    separated by ONLY ONE vertical bar pipe character '|'.
    //    For example, one row with values 'X', ' ', and 'O' looks like
    //    | X |   | O |
    std::string to_string() const;

    //If every single row, column, main diagonal, AND anti-diagonal
    //each contain both 'X' and 'O' symbols,
    //then the board is in a tie state (return true).
    //Otherwise, if any of those lines (row/col/main diagonal/anti-digonal)
    //is missing an 'X' or an 'O', a win is still possible (return false
       for tie status).
    bool tie() const;

    //Check whether the player at row and col wins.
    //If there is any win in horizonal, vertical,
    //diagonal, or anti-diagonal direction, return true,
    //otherwise, return false.
    bool win(int row, int col) const;

    //Check Horizontal Win:
    //Evaluate the row containing the cell (row, col).
    //If the current player's symbol forms a continuous,
    //unbroken sequence of the required length within this row,
    //the function returns true (win found); otherwise, it returns false.
    //Check whether the player at (row, col) can win that row or not.
    bool winByRow(int row, int col) const;

    //Check Vertical Win:
    //Evaluate the column containing the cell (row, col).
    //If the current player's symbol forms a continuous,
    //unbroken sequence of the required length within this column,
    //the function returns true (win found); otherwise, it returns false.
    bool winByCol(int row, int col) const;

    //Check Digonal (including both diagonal and anti-diagonal) Win:
    //If the cell (row, col) is not in diagonal or anti-diagonal, return
```

```
        false.
    //If the cell (row, col) is in the diagonal,
    //evaluate the diagonal containing the cell (row, col).
    //If the current player's symbol forms a continuous,
    //unbroken sequence of the required length within this diagonal,
    //the function returns true (win found);
    //otherwise, evaluate the anti-diagonal containing the cell (row, col).
    //If the current player's symbol forms a continuous,
    //unbroken sequence of the required length within this anti-diagonal,
    //the function returns true (win found); otherwise, return false.
    bool winByDiagonal(int row, int col) const;

private:
    std::vector<std::vector<char>> board; //board is a 2d array of chars
};
#endif
```

4.3 Task A: Define constructors, clear, and to_string Methods of TicTacToeBoard.cpp

1. Download **TicTacToeBoard.cpp**, move it to the `TicTacToeGame` directory built in Section 3.

2. In `TicTacToeBoard.cpp`, implement methods declared in `TicTacToeBoard.hpp` in the following **order**. That is, finish the TODO part of `TicTacToeBoard.cpp`.

   In `assignData` function of `TestTicToeBoard.cpp`, three methods – `size`, `getValue`, `mark` – of TicTacToeBoard class are called to assigns values for `board` data member of a TicTacToeBoard object, created by non-default constructor. As a result, we need to define these methods first.

   (a) `size` method

   (b) `getValue` method

   (c) `mark` method

   (d) default constructor

   (e) non-default constructor

   (f) `clear` method

   Goal: set each cell of the board to be a space character.

   Warning: our clear method of TicTacToeBoard is different from clear method of a vector.

   i. clear method of a vector removes all elements from the vector (which are destroyed), leaving the container with a size of 0.

   ii. That is, board.clear() removes all elements of board and make board an empty vector.

   iii. Our clear method does not change the structure of board – its number of rows and number of columns still keeps the same – just set each element to be a space.

   (g) `to_string` method

3. Test `TicTacToeBoard.cpp` locally using provided **TestTicTacToeBoard.cpp**.

   (a) Compile and link TicTacToeBoard.cpp and TestTicTacToeBoard.cpp using the following command. The generated runnable file is called `tBoard`.

   ```
   g++ -o tBoard TicTacToeBoard.cpp TestTicTacToeBoard.cpp
   ```

   (b) To test default constructor, you can use

   ```
   ./tBoard A
   ```

   You should be able to see the following output if the default constructor is defined correctly in `TicTacToeBoard.cpp`.

   ```
   Call default constructor TicTacToeBoard board;
   contents of board is
    , , ,
    , , ,
    , , ,
   ```

   (c) To test non-default constructor, you should run the following command.

   ```
   ./tBoard B
   ```

   You should see the following output when you input 5 after seeing prompt “Enter size of the board: ”.

   ```
   Enter size of the board: 5
   Call non-default constructor TicTacToeBoard board(5);
   board size: 5
   contents of board is
    , , , , ,
    , , , , ,
    , , , , ,
    , , , , ,
    , , , , ,
   ```

   You should see the following output when you input 2 after seeing prompt “Enter size of the board: ”. Note that when the parameter of non-default constructor in TicTacToeBoard class is smaller than 3, a 3x3 TicTacToeBoard object is created.

   ```
   Enter size of the board: 2
   Call non-default constructor TicTacToeBoard board(2);
   board size: 3
   contents of board is
    , , ,
    , , ,
    , , ,
   ```

   Watch **define constructors of TicTacToeBoard class** video in brightspace.

(d) To test `clear` method, do the following.

```
./tBoard C
```

The expected result is as follows.

```
clear method is correct
```

If clear method is not correctly defined, you would see the following prompt.

```
clear method is not correct. Each element should be a space
   character
```

(e) To test `to_string` method, do the following.

```
./tBoard D
```

The expected result is as follows.

```
     0   1   2   3
   +---+---+---+---+
 0 | X | O | X |   |
   +---+---+---+---+
 1 | O | X |   |   |
   +---+---+---+---+
 2 |   | O | O |   |
   +---+---+---+---+
 3 | X |   |   | X |
   +---+---+---+---+
```

Note that `to_string` method just returns a string representation of the contents of TicTacToeBoard object in tabular format. The above illustration is the **print out** of the return of `to_string` method of a TicTacToeBoard object.

4. Submit `TicTacToeBoard.cpp` to gradescope for grading.

   Warning: do not put `main` function in `TicTacToeBoard.cpp`.

4.3.1 Hint: help function to Draw Separate Line in to_string method

```
//separate lines of data in board
//This function is NOT a member function,
//since clients of TicTacToeBoard class do not need to use it.
//Only method to_string needs to use it.
//Since this is not a method of TicTacToeBoard class,
//we need to pass parameter size.
std::string separateLine(int size) {
    std::string str = "   +";
    for (int i = 0; i < size; i++) {
        str += "---+";
    }
    str += "\n";
    return str;
}
```

4.3.2 Code to Test TicTacToeBoard Class: TestTicTacToeBoard.cpp

Use the following `TestTicTacToeBoard.cpp` to test your TicTacToeBoard.cpp.

```
#include <iostream>
#include <vector>
#include "TicTacToeBoard.hpp"
//g++ -o tBoard TicTacToeBoard.cpp TestTicTacToeBoard.cpp

//test default constructor using
//./tBoard A

//test non-default constructor using
//./tBoard B

//test clear method using
//./tBoard C
//and so on.

const int NUM_COLUMNS = 4;
TicTacToeBoard* assignData(char data[][NUM_COLUMNS]);

int main(int argc, const char *argv[]) {
    if (argc != 2) {
       std::cout << "Need 'A'-'J' in command line parameter" << std::endl;
       return -1;
    }

    //unit-testing for constructors and the destructor
    char type = *argv[1];
    std::string prompt;
    TicTacToeBoard *tttBoard;

    if (type == 'A') {
       prompt = "default constructor TicTacToeBoard board;";
       tttBoard = new TicTacToeBoard;

//expected output:
//Call default constructor TicTacToeBoard board;
//contents of board is
// , , ,
// , , ,
// , , ,
//
    }
    else if (type == 'B') {
        std::cout << "Enter size of the board: ";
        int size;
        std::cin >> size;
        prompt = "non-default constructor TicTacToeBoard board(" + std::
           to_string(size) + ");";
        tttBoard = new TicTacToeBoard(size);

//sample input/output:
//Enter size of the board: 5
//Call non-default constructor TicTacToeBoard board(5);
//contents of board is
// , , , , ,
// , , , , ,
// , , , , ,
// , , , , ,
// , , , , ,
//

//do not take the parameter as it is,
//need to make sure that parameter size is >= 3
//another sample input/output:
//Enter size of the board: 2
//Call non-default constructor TicTacToeBoard board(2);
//contents of board is
// , , ,
// , , ,
// , , ,
//
    }
    else if (type == 'C' || type == 'D') {
        //test clear method,
        char data[][NUM_COLUMNS] = {
           {'X', 'O', 'X', ' '},
           {'O', 'X', ' ', ' '},
           {' ', 'O', 'O', ' '},
           {'X', ' ', ' ', 'X'},
        };

        tttBoard = assignData(data);

        if (type == 'C') {
           //test clear method
           tttBoard->clear();
```

```

            //after calling clearing method, each element of board should be
                ' '
            bool isWrong = false;
            for (int row = 0; row < tttBoard->size() && !isWrong; row++) {
                for (int col = 0; col < tttBoard->size() && !isWrong; col++)
                    {
                    if (tttBoard->getValue(row, col) != ' ') {
                       std::cout << "clear method is not correct. Each
                          element should be a space character\n";
                       isWrong = true; //set a tag
                       //break; //break only can break the inner loop, the
                          outer loop still runs
                    }
                }
            }

            if (!isWrong) {
               std::cout << "clear method is correct\n";
            }
        //expected output:
        //clear method is correct
        }
        else if (type == 'D') {
                //test to_string method
                std::cout << tttBoard->to_string();
        //expected output:
//      0   1   2   3
//    +---+---+---+---+
//  0 | X | O | X |   |
//    +---+---+---+---+
//  1 | O | X |   |   |
//    +---+---+---+---+
//  2 |   | O | O |   |
//    +---+---+---+---+
//  3 | X |   |   | X |
//    +---+---+---+---+
        }
    }
    else if (type == 'E') {
        //test winByRow(int row, int col)
        char data[][NUM_COLUMNS] = {
           {'X', 'O', 'X', ' '},
           {'X', 'O', 'O', ' '},
           {'O', 'O', 'O', 'O'},
           {'X', 'X', 'X', ' '},
        };

        tttBoard = assignData(data);

        bool result;
        for (int row = 0; row < tttBoard->size(); row++) {
            for (int col = 0; col < tttBoard->size(); col++) {
                result = tttBoard->winByRow(row, col);
                if (result) { //result == true
                    std::cout << std::boolalpha << result;
                }
                std::cout << ',';
                //std::cout << std::boolalpha << tttBoard->winByRow(row,
                    col) << ',';
            }
            std::cout << std::endl;
        }
//expected output:
//,,,,
//,,,,
//true,true,true,true,
//,,,,
    }
    else if (type == 'F') {
        //test winByCol(int row, int col)
        char data[][NUM_COLUMNS] = {
            {'O', 'X', 'O', ' '},
            {'O', 'X', 'X', 'O'},
            {'O', 'X', 'O', ' '},
            {'X', 'X', ' ', 'X'},
        };

        tttBoard = assignData(data);

        bool result;
        for (int row = 0; row < tttBoard->size(); row++) {
            for (int col = 0; col < tttBoard->size(); col++) {
                result = tttBoard->winByCol(row, col);
                if (result) { //result == true
                    std::cout << std::boolalpha << result;
                }
                std::cout << ',';

                //std::cout << std::boolalpha << tttBoard->winByCol(row,
                    col) << ',';
            }
            std::cout << std::endl;
        }
//expected output:
//,true,,,
//,true,,,
//,true,,,
//,true,,,
    }
    else if (type == 'G') {
        //test winByDiagonal(int row, int col)
        char data[][NUM_COLUMNS] = {
           {'X', 'X', 'O', ' '},
           {'O', 'X', 'O', 'O'},
           {'X', 'O', 'X', ' '},
           {'O', 'X', ' ', 'X'},
        };

        tttBoard = assignData(data);

        bool result;
        for (int row = 0; row < tttBoard->size(); row++) {
            for (int col = 0; col < tttBoard->size(); col++) {
                result = tttBoard->winByDiagonal(row, col);
                if (result) { //result == true
                   std::cout << std::boolalpha << result;
                }
                std::cout << ',';

                //std::cout << std::boolalpha << tttBoard->winByDiagonal(
                   row, col) << ',';
            }
            std::cout << std::endl;
        }
//expected output:
//true,,,,
//,true,,,
//,,true,,
//,,,true,
    }
    else if (type == 'H') {
        //test winByDiagonal(int row, int col)
        char data[][NUM_COLUMNS] = {
           {'X', 'X', 'O', 'O'},
           {'O', 'X', 'O', 'O'},
           {'X', 'O', ' ', 'X'},
           {'O', 'X', ' ', 'X'},
        };

        tttBoard = assignData(data);

        bool result;
        for (int row = 0; row < tttBoard->size(); row++) {
            for (int col = 0; col < tttBoard->size(); col++) {
                result = tttBoard->winByDiagonal(row, col);
                if (result) { //result == true
                    std::cout << std::boolalpha << result;
                }
                std::cout << ',';
            }
            std::cout << std::endl;
        }
//expected output:
//,,,true,
//,,true,,
//,true,,,
//true,,,,
    }
    else if (type == 'I') {
        //test tie()
        char data[][NUM_COLUMNS] = {
            {'X', 'X', 'O', ' '},
            {'O', 'X', 'O', 'O'},
            {'X', 'O', 'X', ' '},
            {'O', 'X', ' ', 'X'},
        };

        tttBoard = assignData(data);

        std::cout << std::boolalpha << tttBoard->tie() << '\n';

//expected output:
//false
    }
    else if (type == 'J') {
        //test tie()
        char data[][NUM_COLUMNS] = {
            {'X', 'X', 'O', ' '},
            {'O', 'X', 'X', 'O'},
            {'X', 'O', 'O', ' '},
            {'O', 'X', ' ', 'X'},
        };

        tttBoard = assignData(data);

        std::cout << std::boolalpha << tttBoard->tie() << '\n';

//expected output:
//true
    }

    //'A' for default constructor and
    //'B' for non-default constructor
    if (type == 'A' || type == 'B') {
       std::cout << "Call " << prompt << '\n';
       std::cout << "contents of board is\n";
       for (int row = 0; row < tttBoard->size(); row++) {
           for (int col = 0; col < tttBoard->size(); col++) {
               std::cout << tttBoard->getValue(row, col) << ',';
           }
           std::cout << '\n';
       }
    }

    delete tttBoard; //release dynamic allocated memory
    tttBoard = nullptr; //handle dangling pointer problem

    return 0;
}

//TODO: need to define mark method of TicTacToeBoard.cpp before using
   assignData function
TicTacToeBoard* assignData(char data[][NUM_COLUMNS]) {
   TicTacToeBoard *tttBoard = new TicTacToeBoard(NUM_COLUMNS);

   for (int row = 0; row < tttBoard->size(); row++) {
       for (int col = 0; col < tttBoard->size(); col++) {
           tttBoard->mark(row, col, data[row][col]); //set board[row][col]
              of *tttBoard -- a TicTacToeBoard object -- to be data[row][
              col]
       }
   }

   return tttBoard;
}
```

4.4 Task B: Define the Rest Methods in TicTacToeBoard.cpp

Continue from the code of Task A, define the rest methods of `TicTacToeBoard.cpp`.

To define winByCol and winByDiagonal methods, watch **winByCol and winByDiagonal methods** video in brightspace.

5 Define TicTacToeGame class

5.1 Header File of TicTacToeGame Class: TicTacToeGame.hpp

```cpp
#ifndef TicTacToeGame_H
#define TicTacToeGame_H
#include "TicTacToeBoard.hpp"
class TicTacToeGame {
public:
    TicTacToeGame();
    TicTacToeGame(int size);
    void runRepeat();
    void start(); //start to play the game
    bool isGameOver() const;

    //This is how the user plays:
    //Enter row and col such that
    //1. row is in [0, size-1]
    //2. col is in [0, size-1]
    //3. the corresponding cell in tttBoard is available
    //   (hint: call isAvailable method of tttBoard).
    //As long as the input row or col is not valid or is not available

    //begin
    //   prompt what error(s) are, for example,
    //   (1) row is not in [0, size-1] or
    //   (2) col is not in [0, size-1] or
    //   (3) the corresponding cell in tttBoard is not available
    //   prompt user to re-enter.
    //end
    //
    //Once we exit the above repetition loop,
    //row and col are valid,
    //mark the corresponding cell in tttBoard by HUMAN_ID.
    void humanPlay();

    //computer play
    //Computer checks tttBoard from the first row to the last row.
    //In each row, the computer checks from the first column to the last
       column.
    //A more sophisticated approach is to use "mark first; if unfit, then
       remove mark"
    //1. Try to win first.
    //   Mark an available cell by computerId,
    //   if this leads to win by computer,
    //   take this cell and return,
    //   otherwise, do not take this cell (that is, set this cell to be
        available).
    //2. Try to block the opponent from winning.
    //   This approach is adopted after the try-to-win approach fails.
    //   Mark an available cell by userId (that is, suppose this cell is
        taken by userId),
    //   if this leads to win by user,
    //   mark this cell by computerId, and return.
    //   otherwise, do not take this cell (that is, set this cell to be
        available).
    //3. If neither one of the above two approaches works
    //   (that is, the computer does not take a cell yet),
    //   then mark the first available cell.
    void computerPlay();

private:
    TicTacToeBoard tttBoard;
    int currRow;
    int currCol; //the row and col the current player chooses.
    static const char HUMAN_ID = 'X';
    static const char COMPUTER_ID = 'O';
};
#endif
```

6 Run the Project after Defining TicTacToeBoard.cpp and TicTacToeGame.cpp

1. In `TicTacToeGame` directory created in 4.2, besides `TicTacToeBoard.hpp` (provided) and `TicTacToeBoard.c` (your implementation), `TicTacToeGame.hpp` (provided) and `TicTacToeGame.cpp` (your implementation),

6.1 Code to Test TicTacToeGame: TestTicTacToeGame.cpp

```
#include "TicTacToeGame.hpp"

int main() {
   TicTacToeGame game; //3 x 3 tic tac toe board
   //TicTacToeGame game(4); //4 x 4 tic tac toe board
   game.runRepeat();
}
```

6.2 makefile

When working with several source files for this project, it is highly recommended to use a **makefile** and the `make` utility. This automates the build process, eliminating the need to manually remember and type complex compilation commands for every file. A key advantage of using a Makefile is its efficient dependency tracking: if a change is made to a single source file, the `make` utility intelligently recompiles only that specific file and then re-links the final executable, significantly reducing build times by avoiding unnecessary recompilation of unmodified files.

```
# This is an example Makefile for tic tac toe project. This
# program uses TicTacToeBoard, TicTacToeGame, and TestTicTacToeGame modules
    .
# Typing 'make' or 'make run' will create the executable file.
#

# define some Makefile variables for the compiler and compiler flags
# to use Makefile variables later in the Makefile: $()
#
#  -g    adds debugging information to the executable file
#  -Wall turns on most, but not all, compiler warnings
#
# for C++ define  CC = g++
CC = g++ -std=c++11
#CFLAGS  = -g -Wall

# typing 'make' will invoke the first target entry in the file
# (in this case the default target entry)
# you can name this target entry anything, but "default" or "all"
# are the most commonly used names by convention
#
all: run

# To create the executable file run we need the object files
# TestTicTacToeGame.o, TicTacToeBoard.o
run:  TicTacToeBoard.o TestTicTacToeGame.o TicTacToeGame.o
    $(CC) -o run TestTicTacToeGame.o TicTacToeGame.o TicTacToeBoard.o

# To create the object file TestTicTacToeGame.o, we need the source
# files TestTicTacToeGame.cpp and TicTacToeBoard.hpp
#
TestTicTacToeGame.o:  TestTicTacToeGame.cpp
    $(CC) -c TestTicTacToeGame.cpp

TicTacToeGame.o:  TicTacToeGame.cpp
    $(CC) -c TicTacToeGame.cpp

# To create the object file TicTacToeBoard.o, we need the source
# files TicTacToeBoard.cpp, TicTacToeBoard.hpp
TicTacToeBoard.o:  TicTacToeBoard.cpp
	$(CC) -c TicTacToeBoard.cpp

# To start over from scratch, type 'make clean'.  This
# removes the executable file, as well as old .o object
# files and *~ backup files:
#
clean:
	$(RM) run *.o *~
```

To use makefile, just type

```
make
```

From `-o run` of Line 26 of the above make file, we notice that the runnable file is called `run`. Then run the project using

```
./run
```

When you want to delete all the temporary files generated during the build process (such as object files .o, intermediate data files, and the final executable itself). Run the following command.

```
make clean
```

6.3 Sample Run

```
      0   1   2
    +---+---+---+
 0  |   |   |   |
    +---+---+---+
 1  |   |   |   |
    +---+---+---+
 2  |   |   |   |
    +---+---+---+

Round 1: User, enter row and col to place X: 0 0
      0   1   2
    +---+---+---+
 0  | X |   |   |
    +---+---+---+
 1  |   |   |   |
    +---+---+---+
 2  |   |   |   |
    +---+---+---+
Round 2: Computer places O at row 0 and col 1.
      0   1   2
    +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 |   |   |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 3: User, enter row and col to place X: 1 1
     0   1   2
   +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 4: Computer places O at row 2 and col 2.
     0   1   2
   +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 |   |   | O |
   +---+---+---+
Round 5: User, enter row and col to place X: 2 0
     0   1   2
   +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 | X |   | O |
   +---+---+---+
Round 6: Computer places O at row 0 and col 2.
     0   1   2
   +---+---+---+
 0 | X | O | O |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 | X |   | O |
   +---+---+---+
Round 7: User, enter row and col to place X: 1 0
     0   1   2
   +---+---+---+
 0 | X | O | O |
   +---+---+---+
 1 | X | X |   |
   +---+---+---+
 2 | X |   | O |
   +---+---+---+
Human wins. Yay!!!
Do you want to continue (yes/no): y
     0   1   2
   +---+---+---+
 0 |   |   |   |
   +---+---+---+
 1 |   |   |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+

Round 1: User, enter row and col to place X: 0 0
     0   1   2
   +---+---+---+
 0 | X |   |   |
   +---+---+---+
 1 |   |   |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 2: Computer places O at row 0 and col 1.
     0   1   2
   +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 |   |   |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 3: User, enter row and col to place X: 1 0
     0   1   2
   +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 | X |   |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 4: Computer places O at row 2 and col 0.
     0   1   2
   +---+---+---+
 0 | X | O |   |
   +---+---+---+
 1 | X |   |   |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 5: User, enter row and col to place X: 0 2
     0   1   2
   +---+---+---+
 0 | X | O | X |
   +---+---+---+
 1 | X |   |   |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 6: Computer places O at row 1 and col 1.
     0   1   2
   +---+---+---+
 0 | X | O | X |
   +---+---+---+
 1 | X | O |   |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 7: User, enter row and col to place X: 1 2
     0   1   2
   +---+---+---+
 0 | X | O | X |
   +---+---+---+
 1 | X | O | X |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 8: Computer places O at row 2 and col 1.
     0   1   2
   +---+---+---+
 0 | X | O | X |
   +---+---+---+
 1 | X | O | X |
   +---+---+---+
 2 | O | O |   |
   +---+---+---+
Computer wins. Yuck.
Do you want to continue (yes/no): y
     0   1   2
   +---+---+---+
 0 |   |   |   |
   +---+---+---+
 1 |   |   |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+

Round 1: User, enter row and col to place X: 1 1
     0   1   2
   +---+---+---+
 0 |   |   |   |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 2: Computer places O at row 0 and col 0.
     0   1   2
   +---+---+---+
 0 | O |   |   |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 3: User, enter row and col to place X: 0 2
     0   1   2
   +---+---+---+
 0 | O |   | X |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 |   |   |   |
   +---+---+---+
Round 4: Computer places O at row 2 and col 0.
     0   1   2
   +---+---+---+
 0 | O |   | X |
   +---+---+---+
 1 |   | X |   |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 5: User, enter row and col to place X: 1 0
     0   1   2
   +---+---+---+
 0 | O |   | X |
   +---+---+---+
 1 | X | X |   |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 6: Computer places O at row 1 and col 2.
     0   1   2
   +---+---+---+
 0 | O |   | X |
   +---+---+---+
 1 | X | X | O |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 7: User, enter row and col to place X: 0 1
     0   1   2
   +---+---+---+
 0 | O | X | X |
   +---+---+---+
 1 | X | X | O |
   +---+---+---+
 2 | O |   |   |
   +---+---+---+
Round 8: Computer places O at row 2 and col 1.
     0   1   2
   +---+---+---+
 0 | O | X | X |
   +---+---+---+
 1 | X | X | O |
   +---+---+---+
 2 | O | O |   |
   +---+---+---+
Round 9: User, enter row and col to place X: 0 1
The square you pick up is not available.User, re-enter row and col to place
    X: 1 0
The square you pick up is not available.User, re-enter row and col to place
    X: 2 1
The square you pick up is not available.User, re-enter row and col to place
    X: 2 2
     0   1   2
   +---+---+---+
 0 | O | X | X |
   +---+---+---+
 1 | X | X | O |
   +---+---+---+
 2 | O | O | X |
   +---+---+---+
It is a tie.
Do you want to continue (yes/no): no
Bye
```

7 Implement TicTacToeGame Class

7.1 Data Members of TicTacToeGame Class

The `TicTacToeGame` class contains a `TicTacToeBoard` object and two integers representing the row and column indices of the cell chosen by the current player, as shown in the following code segment.

```
private:
    TicTacToeBoard tttBoard;
    int currRow;
    int currCol; //the row and col the current player chooses.
    static const char HUMAN_ID = 'X';
    static const char COMPUTER_ID = 'O';
```

7.2 Static Data Members

`HUMAN_ID` and `COMPUTER_ID` are declared as `static because their values remain the same across all TicTacToeGame objects.`

`As another example, in the Person class, the total population of the world is a static (also called class) data member - the class only needs to store a single copy of that value.`

7.3 Task C: Implement Constructors and `isGameOver`, `humanPlay`, and `computerPlay` Methods of `TicTacToeGame`

1. `Complete constructors and the required methods of TicTacToeGame class. That is, implement TicTacToeGame.cpp. For detailed requirements, refer to the comments in TicTacToeGame.hpp`

2. `Submit both TicTacToeBoard.cpp and TicTacToeGame.cpp to gradescope.`

7.4 Task D: Implement the Remaining Methods of `TicTacToeGame`