

Tic Tac Toe Project, Fall 2025

Tong Yi

Contents

1	CopyRight Warning	1
2	Example of Separating Declaration, Definition, and Test Code: FuelTank Class	2
2.1	FuelTank.hpp	2
2.2	FuelTank.cpp	2
2.3	TestFuelTank.cpp	3
3	Overview of TicTacToeGame Project	4
4	Define TicTacToeBoard Class	5
4.1	Illustration of TicTacToeBoard Class	5
4.2	Header File of TicTacToeBoard class: TicTacToeBoard.hpp	6
4.3	Task A: Define constructors, clear, and to_string Methods of TicTacToeBoard.cpp	9
4.3.1	Hint: help function to Draw Separate Line in to_string method	11
4.3.2	Code to Test TicTacToeBoard Class: TestTicTacToeBoard.cpp	12
4.4	Task B: Define the Rest Methods in TicTacToeBoard.cpp	18
5	Define TicTacToeGame class	18
5.1	Header File of TicTacToeGame Class: TicTacToeGame.hpp	18
6	Run the Project after Defining TicTacToeBoard.cpp and TicTacToeGame.cpp	20
6.1	Code to Test TicTacToeGame: TestTicTacToeGame.cpp	20
6.2	makefile	20
6.3	Sample Run	22
7	Implement TicTacToeGame Class	27
7.1	Data Members of TicTacToeGame Class	27
7.2	Static Data Members	28
7.3	Task C: Implement Constructors and isGameOver, humanPlay, and computerPlay Methods of TicTacToeGame	28
7.4	Task D: Implement the Remaining Methods of TicTacToeGame	28

1 CopyRight Warning

1. This is copyrighted materials; you are not allowed to upload to the Internet.

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

```
1 #ifndef FUEL_TANK_H
2 #define FUEL_TANK_H
3 class FuelTank {
4 public:
5     FuelTank();
6     FuelTank(int startLevel);
7     int getGasLevel() const;
8     void fillGas(int numGallons);
9     void sendGas(int numGallons);
10
11 private:
12     int currGasLevel;
13 };
14 #endif
```

2.2 FuelTank.cpp

```
1 #include "FuelTank.hpp"
2
```

```

3 FuelTank::FuelTank() {
4     //int currGasLevel = 0; //wrong, with int before currGasLevel, the currGasLevel is a
      local variable
5     //for default constructor,
6     //that statement does not initialize data member currGasLevel
7     currGasLevel = 0;
8 }
9
10 FuelTank::FuelTank(int startGasLevel) {
11     if (startGasLevel >= 0) {
12         currGasLevel = startGasLevel;
13     }
14     else {
15         currGasLevel = 0;
16     }
17 }
18
19 int FuelTank::getGasLevel() const {
20     return currGasLevel;
21 }
22
23 void FuelTank::sendGas(int gallons) {
24     if (currGasLevel >= gallons) {
25         currGasLevel -= gallons;
26     }
27 }
28
29 void FuelTank::fillGas(int gallons) {
30     currGasLevel += gallons;
31 }

```

2.3 TestFuelTank.cpp

```

1 //Sample output:
2 //gas level of an empty fuel tank is 0
3 //add 5 gallons to tank
4 //send 3 gallons out from tank
5 //current gas level is 2
6
7 #include "FuelTank.hpp"
8 #include <iostream>
9
10 int main() {
11     //TODO: Call default constructor of FuelTank class to instantiate an object called
      tank

```

```

12 //related: call default constructor of string class to instantiate an empty string
   object called str.
13 FuelTank tank(0); //same as FuelTank tank; //same as FuelTank tank; //same as
   FuelTank tank; //same as FuelTank tank;
14 //similar to
15 //string str; //str is an empty string
16
17 //TODO: print out the current gas level
18 std::cout << "gas level of an empty fuel tank is "
19     << tank.getGasLevel() << std::endl; //fill in ... part
20
21 //TODO: write a statement to add 5 gallons of gas to tank
22 //related: call substr method of str object to get a substring
23 std::cout << "add 5 gallons to tank" << std::endl;
24 tank.fillGas(5);
25
26 std::cout << "send 3 gallons out from tank" << std::endl;
27
28 //TODO: write a statement to send 3 gallons of gas from tank
29 tank.sendGas(3);
30
31 std::cout << "current gas level is ";
32 //TODO: write a statement to find out and print the current gas level of tank
33 //related: find out the length of a string object str
34 std::cout << tank.getGasLevel() << std::endl;
35
36 //WRONG: currGasLevel is a private data member and cannot be accessed directly from
   a function outside the class where currGasLevel is defined
37 //std::cout << tank2.currGasLevel << std::endl;
38 return 0;
39 }

```

To run the code,

```

1 g++ -o run FuelTank.cpp TestFuelTank.cpp
2 ./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.

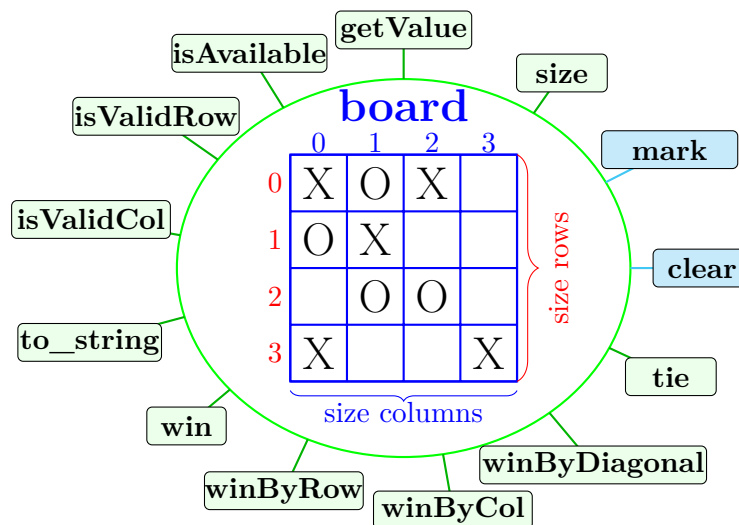
Download [TicTacToeBoard.cpp](#). Finish the TODO parts. For more details, see Section 4.3.

- (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 – represents 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 [TicTacToeBoard.cpp](#).

4.2 Header File of TicTacToeBoard class: TicTacToeBoard.hpp

The contents of [TicTacToeBoard.hpp](#) are as follows.

```
1 #ifndef TicTacToeBoard_H
2 #define TicTacToeBoard_H
3 #include <iostream> //std
4 #include <vector>
5
6 class TicTacToeBoard {
7 public:
```

```

8 //default constructor creates a 3x3 board
9 TicTacToeBoard();
10
11 //If givenSize is less than 3, create a 3x3 board.
12 //Otherwise, initialize the board with the given size for both rows and columns.
13 TicTacToeBoard(int givenSize);
14
15 //Set each cell of the board to be a space character.
16 //Warning: our clear method of TicTacToeBoard is different from clear method of a
vector.
17 //(1) clear method of a vector removes all elements from the vector (which are
destroyed), leaving the container with a size of 0.
18 //(2) That is, board.clear() removes all elements of board and make board an empty
vector.
19 //(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.
20 void clear();
21
22 //if row and col are valid indices,
23 //return the value of the cell of the board at (row, col)
24 //otherwise, return 'W' (means wrong)
25 //Warning: a function with non-void return type
26 //needs to return a value of that return type in every execution path.
27 char getValue(int row, int col) const;
28
29 //Checks if the cell at (row, col) is available or not.
30 //A cell is available if the following conditions satisfy.
31 //(1) row and col indices are valid
32 //(2) the value at that cell is a space character (' ').
33 //Otherwise, the cell is not available.
34 bool isAvailable(int row, int col) const;
35
36 //Check whether the given parameter row is valid row index or not.
37 //that is, whether row is in [0, board.size()-1]
38 bool isValidRow(int row) const;
39
40 //Check whether the given parameter col is a valid column index or not
41 //that is, whether col is in [0, board[0].size()-1]
42 bool isValidCol(int col) const;
43
44 //Return the number of rows of board
45 int size() const;
46
47 //IF row and col are valid indices,
48 //place symbol at board[row][col],

```

```

49 //otherwise, do nothing.
50 //Note that before we can use board[row][col],
51 //need to make sure that row and col indices are valid.
52 void mark(int row, int col, char symbol);
53
54 //Return a string representation of the contents of board.
55 //(1) The returned string typically formats the board in a grid layout
56 //for easy display and debugging.
57 //(2) 'X', 'O', and ' ' characters in the string are
58 //    separated by ONLY ONE vertical bar pipe character '|'.
59 //    For example, one row with values 'X', ' ', and 'O' looks like
60 //    | X |   | O |
61 std::string to_string() const;
62
63 //If every single row, column, main diagonal, AND anti-diagonal
64 //each contain both 'X' and 'O' symbols,
65 //then the board is in a tie state (return true).
66 //Otherwise, if any of those lines (row/col/main diagonal/anti-digonal)
67 //is missing an 'X' or an 'O', a win is still possible (return false for tie status
68 ).
69 bool tie() const;
70
71 //Check whether the player at row and col wins.
72 //If there is any win in horizontal, vertical,
73 //diagonal, or anti-diagonal direction, return true,
74 //otherwise, return false.
75 bool win(int row, int col) const;
76
77 //Check Horizontal Win:
78 //Evaluate the row containing the cell (row, col).
79 //If the current player's symbol forms a continuous,
80 //unbroken sequence of the required length within this row,
81 //the function returns true (win found); otherwise, it returns false.
82 //Check whether the player at (row, col) can win that row or not.
83 bool winByRow(int row, int col) const;
84
85 //Check Vertical Win:
86 //Evaluate the column containing the cell (row, col).
87 //If the current player's symbol forms a continuous,
88 //unbroken sequence of the required length within this column,
89 //the function returns true (win found); otherwise, it returns false.
90 bool winByCol(int row, int col) const;
91
92 //Check Digonal (including both diagonal and anti-diagonal) Win:
93 //If the cell (row, col) is not in diagonal or anti-diagonal, return false.
94 //If the cell (row, col) is in the diagonal,

```



```

94 //evaluate the diagonal containing the cell (row, col).
95 //If the current player's symbol forms a continuous,
96 //unbroken sequence of the required length within this diagonal,
97 //the function returns true (win found);
98 //otherwise, evaluate the anti-diagonal containing the cell (row, col).
99 //If the current player's symbol forms a continuous,
100 //unbroken sequence of the required length within this anti-diagonal,
101 //the function returns true (win found); otherwise, return false.
102 bool winByDiagonal(int row, int col) const;
103
104 private:
105     std::vector<std::vector<char>> board; //board is a 2d array of chars
106 };
107 #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.

```
1 Call default constructor TicTacToeBoard board;
2 contents of board is
3   ,   ,   ,
4   ,   ,   ,
5   ,   ,   ,
```

- (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: ”.

```
1 Enter size of the board: 5
2 Call non-default constructor TicTacToeBoard board(5);
3 board size: 5
4 contents of board is
5   ,   ,   ,   ,   ,
6   ,   ,   ,   ,   ,
7   ,   ,   ,   ,   ,
8   ,   ,   ,   ,   ,
9   ,   ,   ,   ,   ,
```

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.

```
1 Enter size of the board: 2
2 Call non-default constructor TicTacToeBoard board(2);
3 board size: 3
4 contents of board is
5   ,   ,   ,
6   ,   ,   ,
7   ,   ,   ,
```

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 | 0 | X |   |
+---+---+---+---+
1 | 0 | X |   |   |
+---+---+---+---+
2 |   | 0 | 0 |   |
+---+---+---+---+
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

```
1 //separate lines of data in board
2 //This function is NOT a member function,
3 //since clients of TicTacToeBoard class do not need to use it.
4 //Only method to_string needs to use it.
5 //Since this is not a method of TicTacToeBoard class,
6 //we need to pass parameter size.
7 std::string separateLine(int size) {
8     std::string str = "   +";
9     for (int i = 0; i < size; i++) {
10         str += "---+";
11     }
12     str += "\n";
13     return str;
14 }
```

4.3.2 Code to Test TicTacToeBoard Class: TestTicTacToeBoard.cpp

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

```
1 #include <iostream>
2 #include <vector>
3 #include "TicTacToeBoard.hpp"
4 //g++ -o tBoard TicTacToeBoard.cpp TestTicTacToeBoard.cpp
5
6 //test default constructor using
7 //./tBoard A
8
9 //test non-default constructor using
10 //./tBoard B
11
12 //test clear method using
13 //./tBoard C
14 //and so on.
15
16 const int NUM_COLUMNS = 4;
17 TicTacToeBoard* assignData(char data[][NUM_COLUMNS]);
18
19 int main(int argc, const char *argv[]) {
20     if (argc != 2) {
21         std::cout << "Need 'A'-'J' in command line parameter" << std::endl;
22         return -1;
23     }
24
25     //unit-testing for constructors and the destructor
26     char type = *argv[1];
27     std::string prompt;
28     TicTacToeBoard *tttBoard;
29
30     if (type == 'A') {
31         prompt = "default constructor TicTacToeBoard board;";
32         tttBoard = new TicTacToeBoard;
33
34         //expected output:
35         //Call default constructor TicTacToeBoard board;
36         //contents of board is
37         // , , ,
38         // , , ,
39         // , , ,
40         //
41     }
42     else if (type == 'B') {
43         std::cout << "Enter size of the board: ";
```

```

44     int size;
45     std::cin >> size;
46     prompt = "non-default constructor TicTacToeBoard board(" + std::to_string(size)
+ ");";
47     tttBoard = new TicTacToeBoard(size);
48
49 //sample input/output:
50 //Enter size of the board: 5
51 //Call non-default constructor TicTacToeBoard board(5);
52 //contents of board is
53 // , , , , ,
54 // , , , , ,
55 // , , , , ,
56 // , , , , ,
57 // , , , , ,
58 //
59
60 //do not take the parameter as it is,
61 //need to make sure that parameter size is >= 3
62 //another sample input/output:
63 //Enter size of the board: 2
64 //Call non-default constructor TicTacToeBoard board(2);
65 //contents of board is
66 // , , ,
67 // , , ,
68 // , , ,
69 //
70 }
71 else if (type == 'C' || type == 'D') {
72     //test clear method,
73     char data[][NUM_COLUMNS] = {
74         {'X', 'O', 'X', ' '},
75         {'O', 'X', ' ', ' '},
76         {' ', 'O', 'O', ' '},
77         {'X', ' ', ' ', 'X'},
78     };
79
80     tttBoard = assignData(data);
81
82     if (type == 'C') {
83         //test clear method
84         tttBoard->clear();
85
86         //after calling clearing method, each element of board should be ' '
87         bool isWrong = false;
88         for (int row = 0; row < tttBoard->size() && !isWrong; row++) {

```

```

89         for (int col = 0; col < tttBoard->size() && !isWrong; col++) {
90             if (tttBoard->getValue(row, col) != ' ') {
91                 std::cout << "clear method is not correct. Each element should be
a space character\n";
92                 isWrong = true; //set a tag
93                 //break; //break only can break the inner loop, the outer loop
still runs
94             }
95         }
96     }
97
98     if (!isWrong) {
99         std::cout << "clear method is correct\n";
100     }
101     //expected output:
102     //clear method is correct
103 }
104 else if (type == 'D') {
105     //test to_string method
106     std::cout << tttBoard->to_string();
107     //expected output:
108 //      0   1   2   3
109 //  +---+---+---+---+
110 // 0 | X | O | X |   |
111 //  +---+---+---+---+
112 // 1 | O | X |   |   |
113 //  +---+---+---+---+
114 // 2 |   | O | O |   |
115 //  +---+---+---+---+
116 // 3 | X |   |   | X |
117 //  +---+---+---+---+
118 }
119 }
120 else if (type == 'E') {
121     //test winByRow(int row, int col)
122     char data[NUM_COLUMNS] = {
123         {'X', 'O', 'X', ' '},
124         {'X', 'O', 'O', ' '},
125         {'O', 'O', 'O', 'O'},
126         {'X', 'X', 'X', ' '},
127     };
128
129     tttBoard = assignData(data);
130
131     bool result;
132     for (int row = 0; row < tttBoard->size(); row++) {

```

```

133         for (int col = 0; col < tttBoard->size(); col++) {
134             result = tttBoard->winByRow(row, col);
135             if (result) { //result == true
136                 std::cout << std::boolalpha << result;
137             }
138             std::cout << ',';
139             //std::cout << std::boolalpha << tttBoard->winByRow(row, col) << ',';
140         }
141         std::cout << std::endl;
142     }
143     //expected output:
144     //,,,
145     //,,,
146     //true,true,true,true,
147     //,,,
148     }
149     else if (type == 'F') {
150         //test winByCol(int row, int col)
151         char data[][NUM_COLUMNS] = {
152             {'O', 'X', 'O', ' '},
153             {'O', 'X', 'X', 'O'},
154             {'O', 'X', 'O', ' '},
155             {'X', 'X', ' ', 'X'}
156         };
157
158         tttBoard = assignData(data);
159
160         bool result;
161         for (int row = 0; row < tttBoard->size(); row++) {
162             for (int col = 0; col < tttBoard->size(); col++) {
163                 result = tttBoard->winByCol(row, col);
164                 if (result) { //result == true
165                     std::cout << std::boolalpha << result;
166                 }
167                 std::cout << ',';
168
169                 //std::cout << std::boolalpha << tttBoard->winByCol(row, col) << ',';
170             }
171             std::cout << std::endl;
172         }
173         //expected output:
174         //,true,,,
175         //,true,,,
176         //,true,,,
177         //,true,,,
178     }

```

```

179     else if (type == 'G') {
180         //test winByDiagonal(int row, int col)
181         char data[][NUM_COLUMNS] = {
182             {'X', 'X', 'O', ' '},
183             {'O', 'X', 'O', 'O'},
184             {'X', 'O', 'X', ' '},
185             {'O', 'X', ' ', 'X'},
186         };
187
188         tttBoard = assignData(data);
189
190         bool result;
191         for (int row = 0; row < tttBoard->size(); row++) {
192             for (int col = 0; col < tttBoard->size(); col++) {
193                 result = tttBoard->winByDiagonal(row, col);
194                 if (result) { //result == true
195                     std::cout << std::boolalpha << result;
196                 }
197                 std::cout << ' ';
198
199                 //std::cout << std::boolalpha << tttBoard->winByDiagonal(row, col) <<
200                 ' ';
201                 std::cout << std::endl;
202             }
203             //expected output:
204             //true,,,
205             //,true,,,
206             //,,true,,
207             //,,,true,
208         }
209         else if (type == 'H') {
210             //test winByDiagonal(int row, int col)
211             char data[][NUM_COLUMNS] = {
212                 {'X', 'X', 'O', 'O'},
213                 {'O', 'X', 'O', 'O'},
214                 {'X', 'O', ' ', 'X'},
215                 {'O', 'X', ' ', 'X'},
216             };
217
218             tttBoard = assignData(data);
219
220             bool result;
221             for (int row = 0; row < tttBoard->size(); row++) {
222                 for (int col = 0; col < tttBoard->size(); col++) {
223                     result = tttBoard->winByDiagonal(row, col);

```



```

224         if (result) { //result == true
225             std::cout << std::boolalpha << result;
226         }
227         std::cout << ',';
228     }
229     std::cout << std::endl;
230 }
231 //expected output:
232 //,,,true,
233 //,,,true,,
234 //,true,,,
235 //true,,,
236 }
237 else if (type == 'I') {
238     //test tie()
239     char data[][NUM_COLUMNS] = {
240         {'X', 'X', 'O', ' '},
241         {'O', 'X', 'O', 'O'},
242         {'X', 'O', 'X', ' '},
243         {'O', 'X', ' ', 'X'},
244     };
245
246     tttBoard = assignData(data);
247
248     std::cout << std::boolalpha << tttBoard->tie() << '\n';
249
250 //expected output:
251 //false
252 }
253 else if (type == 'J') {
254     //test tie()
255     char data[][NUM_COLUMNS] = {
256         {'X', 'X', 'O', ' '},
257         {'O', 'X', 'X', 'O'},
258         {'X', 'O', 'O', ' '},
259         {'O', 'X', ' ', 'X'},
260     };
261
262     tttBoard = assignData(data);
263
264     std::cout << std::boolalpha << tttBoard->tie() << '\n';
265
266 //expected output:
267 //true
268 }
269

```

```

270 // 'A' for default constructor and
271 // 'B' for non-default constructor
272 if (type == 'A' || type == 'B') {
273     std::cout << "Call " << prompt << '\n';
274     std::cout << "contents of board is\n";
275     for (int row = 0; row < tttBoard->size(); row++) {
276         for (int col = 0; col < tttBoard->size(); col++) {
277             std::cout << tttBoard->getValue(row, col) << ',';
278         }
279         std::cout << '\n';
280     }
281 }
282
283 delete tttBoard; //release dynamic allocated memory
284 tttBoard = nullptr; //handle dangling pointer problem
285
286 return 0;
287 }
288
289 //TODO: need to define mark method of TicTacToeBoard.cpp before using assignData
290 //function
291 TicTacToeBoard* assignData(char data[][NUM_COLUMNS]) {
292     TicTacToeBoard *tttBoard = new TicTacToeBoard(NUM_COLUMNS);
293
294     for (int row = 0; row < tttBoard->size(); row++) {
295         for (int col = 0; col < tttBoard->size(); col++) {
296             tttBoard->mark(row, col, data[row][col]); //set board[row][col] of *tttBoard
297             -- a TicTacToeBoard object -- to be data[row][col]
298         }
299     }
300
301     return tttBoard;
302 }

```

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

```

1 #ifndef TicTacToeGame_H

```

```

2  #define TicTacToeGame_H
3  #include "TicTacToeBoard.hpp"
4  class TicTacToeGame {
5  public:
6      TicTacToeGame();
7      TicTacToeGame(int size);
8      void runRepeat();
9      void start(); //start to play the game
10     bool isGameOver() const;
11
12     //This is how the user plays:
13     //Enter row and col such that
14     //1. row is in [0, size-1]
15     //2. col is in [0, size-1]
16     //3. the corresponding cell in tttBoard is available
17     //    (hint: call isAvailable method of tttBoard).
18     //As long as the input row or col is not valid or is not available
19
20     //begin
21     //    prompt what error(s) are, for example,
22     //    (1) row is not in [0, size-1] or
23     //    (2) col is not in [0, size-1] or
24     //    (3) the corresponding cell in tttBoard is not available
25     //    prompt user to re-enter.
26     //end
27     //
28     //Once we exit the above repetition loop,
29     //row and col are valid,
30     //mark the corresponding cell in tttBoard by HUMAN_ID.
31     void humanPlay();
32
33     //computer play
34     //Computer checks tttBoard from the first row to the last row.
35     //In each row, the computer checks from the first column to the last column.
36     //A more sophisticated approach is to use "mark first; if unfit, then remove mark"
37     //1. Try to win first.
38     //    Mark an available cell by computerId,
39     //    if this leads to win by computer,
40     //    take this cell and return,
41     //    otherwise, do not take this cell (that is, set this cell to be available).
42     //2. Try to block the opponent from winning.
43     //    This approach is adopted after the try-to-win approach fails.
44     //    Mark an available cell by userId (that is, suppose this cell is taken by
45     //    userId),
46     //    if this leads to win by user,
47     //    mark this cell by computerId, and return.

```

```

47 // otherwise, do not take this cell (that is, set this cell to be available).
48 //3. If neither one of the above two approaches works
49 // (that is, the computer does not take a cell yet),
50 // then mark the first available cell.
51 void computerPlay();
52
53 private:
54     TicTacToeBoard tttBoard;
55     int currRow;
56     int currCol; //the row and col the current player chooses.
57     static const char HUMAN_ID = 'X';
58     static const char COMPUTER_ID = 'O';
59 };
60 #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.cpp (your implementation), TicTacToeGame.hpp (provided) and TicTacToeGame.cpp (your implementation),

6.1 Code to Test TicTacToeGame: TestTicTacToeGame.cpp

```

1 #include "TicTacToeGame.hpp"
2
3 int main() {
4     TicTacToeGame game; //3 x 3 tic tac toe board
5     //TicTacToeGame game(4); //4 x 4 tic tac toe board
6     game.runRepeat();
7 }

```

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.

```

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

```

```

4 #
5
6 # define some Makefile variables for the compiler and compiler flags
7 # to use Makefile variables later in the Makefile: $()
8 #
9 # -g adds debugging information to the executable file
10 # -Wall turns on most, but not all, compiler warnings
11 #
12 # for C++ define CC = g++
13 CC = g++ -std=c++11
14 #CFLAGS = -g -Wall
15
16 # typing 'make' will invoke the first target entry in the file
17 # (in this case the default target entry)
18 # you can name this target entry anything, but "default" or "all"
19 # are the most commonly used names by convention
20 #
21 all: run
22
23 # To create the executable file run we need the object files
24 # TestTicTacToeGame.o, TicTacToeBoard.o
25 run: TicTacToeBoard.o TestTicTacToeGame.o TicTacToeGame.o
26     $(CC) -o run TestTicTacToeGame.o TicTacToeGame.o TicTacToeBoard.o
27
28 # To create the object file TestTicTacToeGame.o, we need the source
29 # files TestTicTacToeGame.cpp and TicTacToeBoard.hpp
30 #
31 TestTicTacToeGame.o: TestTicTacToeGame.cpp
32     $(CC) -c TestTicTacToeGame.cpp
33
34 TicTacToeGame.o: TicTacToeGame.cpp
35     $(CC) -c TicTacToeGame.cpp
36
37 # To create the object file TicTacToeBoard.o, we need the source
38 # files TicTacToeBoard.cpp, TicTacToeBoard.hpp
39 TicTacToeBoard.o: TicTacToeBoard.cpp
40     $(CC) -c TicTacToeBoard.cpp
41
42 # To start over from scratch, type 'make clean'. This
43 # removes the executable file, as well as old .o object
44 # files and *~ backup files:
45 #
46 clean:
47     $(RM) run *.o *~

```

To use makefile, just type

```
1 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

```
1 ./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.

```
1 make clean
```

6.3 Sample Run

```
1      0  1  2
2      +---+---+---+
3      0 |  |  |  |
4      +---+---+---+
5      1 |  |  |  |
6      +---+---+---+
7      2 |  |  |  |
8      +---+---+---+
9
10 Round 1: User, enter row and col to place X: 0 0
11      0  1  2
12      +---+---+---+
13      0 | X |  |  |
14      +---+---+---+
15      1 |  |  |  |
16      +---+---+---+
17      2 |  |  |  |
18      +---+---+---+
19 Round 2: Computer places 0 at row 0 and col 1.
20      0  1  2
21      +---+---+---+
22      0 | X | 0 |  |
23      +---+---+---+
24      1 |  |  |  |
25      +---+---+---+
26      2 |  |  |  |
27      +---+---+---+
28 Round 3: User, enter row and col to place X: 1 1
29      0  1  2
30      +---+---+---+
31      0 | X | 0 |  |
32      +---+---+---+
33      1 |  | X |  |
```

```

34      +---+---+---+
35  2 |   |   |   |
36      +---+---+---+
37 Round 4: Computer places 0 at row 2 and col 2.
38      0   1   2
39      +---+---+---+
40  0 | X | 0 |   |
41      +---+---+---+
42  1 |   | X |   |
43      +---+---+---+
44  2 |   |   | 0 |
45      +---+---+---+
46 Round 5: User, enter row and col to place X: 2 0
47      0   1   2
48      +---+---+---+
49  0 | X | 0 |   |
50      +---+---+---+
51  1 |   | X |   |
52      +---+---+---+
53  2 | X |   | 0 |
54      +---+---+---+
55 Round 6: Computer places 0 at row 0 and col 2.
56      0   1   2
57      +---+---+---+
58  0 | X | 0 | 0 |
59      +---+---+---+
60  1 |   | X |   |
61      +---+---+---+
62  2 | X |   | 0 |
63      +---+---+---+
64 Round 7: User, enter row and col to place X: 1 0
65      0   1   2
66      +---+---+---+
67  0 | X | 0 | 0 |
68      +---+---+---+
69  1 | X | X |   |
70      +---+---+---+
71  2 | X |   | 0 |
72      +---+---+---+
73 Human wins. Yay!!!
74 Do you want to continue (yes/no): y
75      0   1   2
76      +---+---+---+
77  0 |   |   |   |
78      +---+---+---+
79  1 |   |   |   |

```

```

80  +---+---+---+
81  2 |   |   |   |
82  +---+---+---+
83
84  Round 1: User, enter row and col to place X: 0 0
85      0   1   2
86  +---+---+---+
87  0 | X |   |   |
88  +---+---+---+
89  1 |   |   |   |
90  +---+---+---+
91  2 |   |   |   |
92  +---+---+---+
93  Round 2: Computer places 0 at row 0 and col 1.
94      0   1   2
95  +---+---+---+
96  0 | X | 0 |   |
97  +---+---+---+
98  1 |   |   |   |
99  +---+---+---+
100 2 |   |   |   |
101 +---+---+---+
102 Round 3: User, enter row and col to place X: 1 0
103      0   1   2
104 +---+---+---+
105 0 | X | 0 |   |
106 +---+---+---+
107 1 | X |   |   |
108 +---+---+---+
109 2 |   |   |   |
110 +---+---+---+
111 Round 4: Computer places 0 at row 2 and col 0.
112      0   1   2
113 +---+---+---+
114 0 | X | 0 |   |
115 +---+---+---+
116 1 | X |   |   |
117 +---+---+---+
118 2 | 0 |   |   |
119 +---+---+---+
120 Round 5: User, enter row and col to place X: 0 2
121      0   1   2
122 +---+---+---+
123 0 | X | 0 | X |
124 +---+---+---+
125 1 | X |   |   |

```



```

126      +---+---+---+
127      2 | 0 |   |   |
128      +---+---+---+
129 Round 6: Computer places 0 at row 1 and col 1.
130      0  1  2
131      +---+---+---+
132      0 | X | 0 | X |
133      +---+---+---+
134      1 | X | 0 |   |
135      +---+---+---+
136      2 | 0 |   |   |
137      +---+---+---+
138 Round 7: User, enter row and col to place X: 1 2
139      0  1  2
140      +---+---+---+
141      0 | X | 0 | X |
142      +---+---+---+
143      1 | X | 0 | X |
144      +---+---+---+
145      2 | 0 |   |   |
146      +---+---+---+
147 Round 8: Computer places 0 at row 2 and col 1.
148      0  1  2
149      +---+---+---+
150      0 | X | 0 | X |
151      +---+---+---+
152      1 | X | 0 | X |
153      +---+---+---+
154      2 | 0 | 0 |   |
155      +---+---+---+
156 Computer wins. Yuck.
157 Do you want to continue (yes/no): y
158      0  1  2
159      +---+---+---+
160      0 |   |   |   |
161      +---+---+---+
162      1 |   |   |   |
163      +---+---+---+
164      2 |   |   |   |
165      +---+---+---+
166
167 Round 1: User, enter row and col to place X: 1 1
168      0  1  2
169      +---+---+---+
170      0 |   |   |   |
171      +---+---+---+

```

```

172  1 |   | X |   |
173    +---+---+---+
174  2 |   |   |   |
175    +---+---+---+
176 Round 2: Computer places 0 at row 0 and col 0.
177      0  1  2
178    +---+---+---+
179  0 | 0 |   |   |
180    +---+---+---+
181  1 |   | X |   |
182    +---+---+---+
183  2 |   |   |   |
184    +---+---+---+
185 Round 3: User, enter row and col to place X: 0 2
186      0  1  2
187    +---+---+---+
188  0 | 0 |   | X |
189    +---+---+---+
190  1 |   | X |   |
191    +---+---+---+
192  2 |   |   |   |
193    +---+---+---+
194 Round 4: Computer places 0 at row 2 and col 0.
195      0  1  2
196    +---+---+---+
197  0 | 0 |   | X |
198    +---+---+---+
199  1 |   | X |   |
200    +---+---+---+
201  2 | 0 |   |   |
202    +---+---+---+
203 Round 5: User, enter row and col to place X: 1 0
204      0  1  2
205    +---+---+---+
206  0 | 0 |   | X |
207    +---+---+---+
208  1 | X | X |   |
209    +---+---+---+
210  2 | 0 |   |   |
211    +---+---+---+
212 Round 6: Computer places 0 at row 1 and col 2.
213      0  1  2
214    +---+---+---+
215  0 | 0 |   | X |
216    +---+---+---+
217  1 | X | X | 0 |

```

```

218 +---+---+---+
219 2 | 0 |   |   |
220 +---+---+---+
221 Round 7: User, enter row and col to place X: 0 1
222   0   1   2
223 +---+---+---+
224 0 | 0 | X | X |
225 +---+---+---+
226 1 | X | X | 0 |
227 +---+---+---+
228 2 | 0 |   |   |
229 +---+---+---+
230 Round 8: Computer places 0 at row 2 and col 1.
231   0   1   2
232 +---+---+---+
233 0 | 0 | X | X |
234 +---+---+---+
235 1 | X | X | 0 |
236 +---+---+---+
237 2 | 0 | 0 |   |
238 +---+---+---+
239 Round 9: User, enter row and col to place X: 0 1
240 The square you pick up is not available.User, re-enter row and col to place X: 1 0
241 The square you pick up is not available.User, re-enter row and col to place X: 2 1
242 The square you pick up is not available.User, re-enter row and col to place X: 2 2
243   0   1   2
244 +---+---+---+
245 0 | 0 | X | X |
246 +---+---+---+
247 1 | X | X | 0 |
248 +---+---+---+
249 2 | 0 | 0 | X |
250 +---+---+---+
251 It is a tie.
252 Do you want to continue (yes/no): no
253 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