

# Assignment 10: Tic Tac Toe Game Analyzer

EC602 Design by Software

Fall 2021

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Assignment Goals . . . . .	1
1.2	Group Size . . . . .	2
1.3	Due Date . . . . .	2
1.4	Assignment Value . . . . .	2
1.5	Submission Link . . . . .	2
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	Rules of Tic Tac Toe . . . . .	2
<b>3</b>	<b>Tic Tac Toe Analyser</b>	<b>2</b>
3.1	String-representation <code>tttresult</code> . . . . .	2
3.2	Vector-representation <code>tttresult</code> . . . . .	3
3.3	Function <code>get_all_boards</code> . . . . .	5
3.4	Function <code>ttt_tally</code> . . . . .	5
3.5	All the boards . . . . .	6
<b>4</b>	<b>Downloads</b>	<b>7</b>
4.1	Move class . . . . .	7
4.2	Checker . . . . .	7
<b>5</b>	<b>Grading Scheme</b>	<b>7</b>

## 1 Introduction

### 1.1 Assignment Goals

The assignment goals are to help you learn about

- game logic
- data structures and vectors
- error handling

- software reuse

## 1.2 Group Size

The maximum group size is 3 students.

## 1.3 Due Date

This assignment is due Dec 3 at 11:59 ET.

## 1.4 Assignment Value

This assignment is worth 7 points.

Style counts for 20% of the grade.

## 1.5 Submission Link

You can submit here: [ttt hw10 submit link](#)

# 2 Background

## 2.1 Rules of Tic Tac Toe

Player **x** and player **o** alternately place their symbols on a 3x3 board, *starting with x*. The first player to get three symbols in a straight line (vertical, diagonal, or horizontal) wins, and the game is over. If all nine squares are full and no player has three symbols in a row, the game ends in a tie.

# 3 Tic Tac Toe Analyser

Summary: write functions for analysing the game of tic-tac-toe inside a C++ program called `tttalyzer.cpp`, and make a tally of all possible games.

## 3.1 String-representation `tttresult`

Write a function with the following signature:

```
char tttresult(string tttboard);
```

The string `tttboard` has nine characters representing the current situation in a game of tic tac toe:

```
x represents x
o represents o
# represents an unplayed space
```

The first three characters are the top row, the next three the middle row, and the last three are the bottom row.

So the line:

`xox#x#xox`

represents the board

```

  x |  o  |  x
    |    |
  ---+---+---
    |  x  |
  ---+---+---
    |    |
  x |  o  |  x

```

The program should classify the board as one of the following:

- t: tie game and no spaces left
- x: valid, x has won
- o: valid, o has won
- c: valid, game continues
- i: invalid
- e: error condition (bad shape of board, bad chars)

An invalid game board is one in which there was a rule violation. A game can be invalid for many reasons, such as too many winners, unbalanced number of x's and o's, etc.

Do not “anticipate” tie games: a game is considered a tie only if all the spaces are filled.

### 3.1.1 Examples

`tttresult("xox#x#xox")` returns `i`

It is invalid because x played 5 times and o only played 2 times.

`tttresult("xoxoxoxox")` returns `x`

## 3.2 Vector-representation `tttresult`

Write a function with the following signature:

```
char tttresult(vector<Move> board);
```

Each move is for either player `x` or player `o`, and records the row and column of the moves.

The row and column numbers are from 0 to 2.

The order of the Move objects inside the vector represents the order of the moves in a game.

The program should classify the board as one of the following:

- t: tie game and no spaces left
- x: valid, x has won
- o: valid, o has won
- c: valid, game continues
- i: invalid

### 3.2.1 About The Move class

The new class Move is defined as follows:

```
struct Move {
    int r;
    int c;
    char player;
};
```

The structure Move is defined in the file `movedef.h` which must be included in your code using

```
#include "movedef.h"
```

### 3.2.2 Example code using Move:

Here is some example code using the new data structure Move:

```
#include "movedef.h"

int main() {
    vector<Move> moves;
    bool error;
    char result;

    Move m; // make a move
    m.r = 0; // fill the data
    m.c = 1;
    m.player = 'x';

    moves.push_back(m); // put the move on the vector representing the board.

    result = ttresult(moves); // returns 'c'
    result = ttresult("###xxxoo#"); // returns 'x'
    result = ttresult("xxxoooHI!"); // returns 'e'
}
```

### 3.3 Function `get_all_boards`

Write a function `get_all_boards` which generates all  $3^9$  possible tic-tac-toe boards.

The function signature is

```
vector<string> get_all_boards();
```

The vector should contain all possible tic-tac-toe boards. The order is not important.

### 3.4 Function `ttt_tally`

Write a function `ttt_tally` which prints a table of the five possible outcomes `toxic` and the frequency of that result for all  $3^9$  possible tic-tac-toe boards.

```
void ttt_tally();
```

The output should look like

```
x 2213
o 1234
t 1234
i 4524
c 12313
```

except that the numbers should be the real numbers calculated by analyzing all the boards.

The order is irrelevant: you can print out the table in any order.

You should use `get_all_boards` and `tttresult` to generate the tally.

just before your declaration of ``main()`` so that I can properly check it.

#### ## Restrictions

You may include the following libraries but no others:

```
- <iostream>
- <string>
- <array>
- <vector>
- <map>
- "movedef.h"
```

#### ## Program structure

Your program will be tested as follows:

- cpplint and astyle will be run on it.
- it will be compiled and run, to check for the proper tallies
- the following line and everything after it

// MAIN

will be replaced by my own ``main()``, which will check the proper operation of the functions defined above.

## Example starting file

Here is an example starting file: `[tttalyzer_original.cpp](tttalyzer_original.cpp)`

# Hints and suggestions

## Using ``map``

You can use ``map`` to make the tally. Here is some partial code to illustrate:

```
```cpp
string keys ="xotic";
map <char,int> tally;

for each board: // this is pseudo-code, not C++
    result = tttresult(board);
    tally[result] += 1;
```

### 3.5 All the boards

What are all the ttt boards?

Here are the first few:

```
#####
#####o
#####x
#####o#
#####oo
#####ox
...
xxxxxxxx
```

Compare this pattern to the 3-digit binary numbers:

```
000
001
010
011
```

```
100
101
110
111
```

## 4 Downloads

### 4.1 Move class

Download `movedef.h` which defines the `Move` class

### 4.2 Checker

The checker is now available: `hw10_ttt_check.py`

Run it in the devbox using:

```
python hw10_ttt_check.py
```

in the directory that has your program in it.

The current version number is 2.4

## 5 Grading Scheme

Out of 100 total points, the grade is determined as follows:

- 40 points for passing the specifications of `tttresult(string)`
- 20 points for passing the specifications of `tttresult(vector)`
- 15 points for passing the specifications of `get_all_boards()`
- 5 points for passing the specifications of `ttt_tally()`
- 10 points for program brevity (lines + words, not counting comments)
- 5 points for `astyle` (% file unchanged by `astyle`)
- 5 points for `cpplint`, -1 deduction for each problem