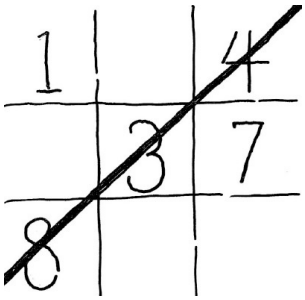
CISC 3110

# Lab 1: Tic-Tac-Math and Multiple Files

Objectives:

- To gain experience writing and editing multiple-source-file programs.
- To practice designing modular programs.
- To practice using a build automation system (in this course, make).

## Part I: Overview




Tic-Tac-Math is a lot like Tic-Tac-Toe. The game is played on a 3x3 grid; players take turns marking one cell at a time. The difference is that instead of placing Xs and Os on the grid, the players place the numbers 1 to 9. And, instead of trying to make a row, column, or diagonal filled with 3 Xs or 3 Os, the players try to make a row, column, or diagonal whose cells sum to 15. One wrinkle: each number can only be placed ONCE. Play a couple games with a classmate.

Let's think about implementing this game. (For now, we won't use classes; everyone has different amounts of experience with them, so for now we'll just use functions.) We have a couple questions to think about before we start coding:

1) How should we represent the board? You might be tempted to say "a two-dimensional array!" But C++'s multi-dimensional arrays cause great confusion even among experienced programmers, so I stay away from them. In this case, a simple way to start is to represent the board as a 9-element array of `int`.

2) And what behaviors (functions) do we need? There's infinite variety here, but generally speaking we try to have each function to one small well-defined task, So (in addition to `main()`, of course), we might have

- A function to `getAMove()` from the user (we'll assume that two players are sharing one keyboard).
- A function to `validate()` the input. (Make sure the input values are in legal ranges.)
- A function to `playAMove()` onto the board (this function will have to do a little more validation)
- A function to `display()` the current state of the board.
- A function to `checkForWin()`.

Notice I haven't given you the complete function signatures, and I certainly haven't described how they should be used (are they all called from `main()`? Or do some of these call the others?)

## Part 2: Design and Implementation

Download the file http://bc-cisc3110-s17.github.io/lab1/lab1.zip to your Linux machine. Note that I've organized the functions above into modules. You should be able to discern the organizing principle I used—how would you describe it?

Think about what would need to be changed if, say, we wanted to get input from a different source (maybe a graphical user interface), or change the way the board is represented. A good design should minimize the impact of those changes (that is, minimize the amount of code that must be changed)—how well does my design do that? Could it be improved?

For this lab:

- complete the header files I've given you by completing the signature for each function (this is the main challenge)
- write a `.cc` file corresponding to each header. Write "stubs" for each function that just identify the function being called. If the function will need to call another function in our design, include that function call. (You may need to provide some "nonsense values" for parameters) So, if the function foo() will need to call bar(), implement it as

```
std::cout << "Running function foo." << std::endl;
bar()
```

- write a file containing a stub of `main()`
- write a complete makefile for the project.
- Verify this compiles and runs

Email me a zip of these files, in an email with the subject "CISC 3110 Lab 1", by 11:59pm, Monday February 27.