

# ECE 477 Lab 2

Christian Knight, Nikko Noble

September 16, 2020

## Introduction

The purpose of this lab is to develop a C program that will output comma separated values to a file or `stdout`. Specifically, the program identifies all legal Tic Tac Toe boards, classifies them, and prints them either to a `.csv` file that can be imported into a spreadsheet or to the command line. The program takes the name of the desired output file as a command line argument or prints the results to the command line if no argument is given.

For part B, another C program was developed which does the opposite of part A; it takes as an argument a `.csv` file which contains possible Tic Tac Toe scenarios and identifies if the board is legal or not, classifying the board if it is legal. This program can output a `.csv` file with the results in a similar fashion to part A if the file name is given as a second argument, but will print the results to the command line otherwise. This part makes use of file input, which part A did not.

## Part A

This section describes the development of the program that identifies all legal Tic Tac Toe board positions, and checks what state the board is in. The output of the program will put be stored in a `.csv` file if given an argument, otherwise the output is printed to the command line. The assumption is made is that 'X' has first move every time.

To start, the program must set itself up to output the results to a file if necessary. If the program is given the name of a `.csv` file to output to as an argument, the `fopen()` function in the `stdio` library is used with the name of the output file as the first argument and the file permissions as the second argument. The return value is assigned to a file pointer, which is used later on. If no output file is given, the file pointer is assigned `stdout` to print results to the command line instead.

Next, a way to represent the game board within the C code was needed. Since each space on the board has three possibilities ('X', 'O', or empty) and there are nine spaces on the board, it makes sense to represent the board as nine trinary (base 3) numbers. Therefore, the game board is represented as the string `char board[9]`, where each character can be

an 'X', 'O' or ' '. There are 19,683 unique combinations to test, so in order to test each one, every number between 0 and 19682 is converted from a base 10 number to a trinary number with its corresponding character in a `for` loop. For each conversion, the function `dec2tri()` is called, its first argument being the integer to convert and the second argument being the string to output the converted trinary number to. This function, along with others mentioned later, is defined in the header file `ttt_func.h`.

After converting the integer corresponding to the board to the trinary representation, the program needs to analyze the board to determine if it's legal, and if so, the state of the board. The function `checkstate()` is used to determine what state the board is in. The function takes `board` as an argument and returns a value assigned to the variable `gamestate`. `gamestate` has 5 different possibilities, which are shown below.

- `gamestate=0; // The board is illegal.`
- `gamestate=1; // The game is in progress.`
- `gamestate=2; // 'X' wins.`
- `gamestate=3; // 'O' wins.`
- `gamestate=4; // There is no winner, or a tie.`

The first check in `checkstate()` is to see if the number of 'X's and 'O's on the board constitute a legal board. Assuming 'X' always goes first, a legal board should have either an equal number of 'X's and 'O's or one more 'X' than 'O's. To do this check, a `for` loop counts the 'X's and 'O's stores the results in the variables `Xcount` and `Ocount`. The variables are compared, and if they aren't equal, or value of `Xcount` isn't one more than `Ocount`, the board is illegal, and `gamestate = 0`.

Assuming that there are a legal number of 'X's and 'O's on the board, the next step is to determine if the game is still in progress or has come to an end. To check if there has been a win by 'X' or 'O', the function `checkwin()` is used. Figure 1 shows the winning board combinations that `checkwin()` checks for.

8	7	6	Winning Combinations:
5	4	3	
2	1	0	
			012
			345
			678
			258
			147
			036
			840
			246

Figure 1: All possible winning combinations on Tic Tac Toe board

In `checkwin()`, the character to check (either 'X' or 'O') is given as an argument, and an `if` statement checks if any of the winning combinations shown above are present. The function returns 1 if three in a row are found and returns zero if not. In `checkstate()`, `checkwin()` is used to check if either 'X' or 'O' has three in a row, and makes sure that the opponent doesn't also have 3 in a row if so. These conditions are checked in an `if` statement, along with the proper number of 'X's and 'O's. Depending on if 'X' or 'O' has won, `gamestate` will either be set to 2 or 3.

If the board is legal and neither 'X' nor 'O' has won, the function checks for a tie (no empty spaces, no winner). If this is the case, `checkstate()` returns a 4. If the board is legal, neither 'X' nor 'O' has won, and there are empty spaces left, the game is still in progress, and `gamestate` is set equal to 1 by `checkstate()`.

After the board has been analyzed by `checkstate()`, the board combination and outcome is saved to the `.csv` as long as `gamestate` isn't 0, indicating an illegal board.

After all possible board layouts have been checked, the last operation the program executes before returning 0 is `fclose()`, with the output file pointer as an argument. This ends the writing to the `.csv` file. Figure 2 shows a sample of the program being run from the command line.

```
Terminal - student@christian-yoga: ~/Desktop/ECE477/lab2
File Edit View Terminal Tabs Help
student@christian-yoga:~/Desktop/ECE477/lab2$
student@christian-yoga:~/Desktop/ECE477/lab2$ gcc ttt.c -o ttt
student@christian-yoga:~/Desktop/ECE477/lab2$ ./ttt part_a.csv
student@christian-yoga:~/Desktop/ECE477/lab2$ ./ttt
, , , , , , , , , , 1,Game in Progress
, , , , , , , , , , X,1,Game in Progress
, , , , , , , , , , X, , 1,Game in Progress
, , , , , , , , , , X,0,1,Game in Progress
, , , , , , , , , , 0,X,1,Game in Progress
, , , , , , , , , , X, , , 1,Game in Progress
, , , , , , , , , , X, , 0,1,Game in Progress
, , , , , , , , , , X,X,0,1,Game in Progress
, , , , , , , , , , X,0, , 1,Game in Progress
, , , , , , , , , , X,0,X,1,Game in Progress
, , , , , , , , , , 0, , X,1,Game in Progress
, , , , , , , , , , 0,X, , 1,Game in Progress
, , , , , , , , , , 0,X,X,1,Game in Progress
, , , , , , , , , , X, , , , 1,Game in Progress
```

Figure 2: Sample of output for part A

Figure 3 shows a sample of the output `.csv` for part A.

	A	B	C	D	E	F	G	H	I	J	K	
1											1 Game in Progress	
2								X			1 Game in Progress	
3							X				1 Game in Progress	
4							X	O			1 Game in Progress	
5							O	X			1 Game in Progress	
6						X					1 Game in Progress	
7						X		O			1 Game in Progress	
8						X	X	O			1 Game in Progress	
9						X	O				1 Game in Progress	
10						X	O	X			1 Game in Progress	
11						O		X			1 Game in Progress	
12						O	X				1 Game in Progress	
13						O	X	X			1 Game in Progress	
14											1 Game in Progress	
...												
2880	X	X		O	X		O		O		1 Game in Progress	
2881	X	X		O	X		O	X	O		2 X wins	
2882	X	X		O	X		O	O			1 Game in Progress	
2883	X	X		O	X		O	O	X		2 X wins	
2884	X	X		O	X	X		O	O		1 Game in Progress	
2885	X	X		O	X	X	O		O		1 Game in Progress	
2886	X	X		O	X	X	O	O			1 Game in Progress	
2887	X	X		O	X	X	O	O	O		3 O wins	
2888	X	X		O	X	O					1 Game in Progress	
2889	X	X		O	X	O			O		1 Game in Progress	
2890	X	X		O	X	O		X	O		2 X wins	
2891	X	X		O	X	O		O			1 Game in Progress	
2892	X	X		O	X	O		O	X		2 X wins	
2893	X	X		O	X	O	X		O		1 Game in Progress	
2894	X	X		O	X	O	X	O			1 Game in Progress	
2895	X	X		O	X	O	X	O	O		1 Game in Progress	
2896	X	X		O	X	O	O				1 Game in Progress	
2897	X	X		O	X	O	O		X		2 X wins	
2898	X	X		O	X	O	O	X			2 X wins	
2899	X	X		O	O						1 Game in Progress	
2900	X	X		O	O				X		1 Game in Progress	
...												
5463	O	O	O	X			X	O	X	X	3 O wins	
5464	O	O	O	X	X				X		3 O wins	
5465	O	O	O	X	X			X			3 O wins	
5466	O	O	O	X	X		X				3 O wins	
5467	O	O	O	X	X		X	X	O		3 O wins	
5468	O	O	O	X	X		X	O	X		3 O wins	
5469	O	O	O	X	X		O	X	X		3 O wins	
5470	O	O	O	X	X	O		X	X		3 O wins	
5471	O	O	O	X	X	O	X		X		3 O wins	
5472	O	O	O	X	X	O	X	X			3 O wins	
5473	O	O	O	X	O	X		X	X		3 O wins	
5474	O	O	O	X	O	X	X		X		3 O wins	
5475	O	O	O	X	O	X	X	X			3 O wins	
5476	O	O	O	O	X	X		X	X		3 O wins	
5477	O	O	O	O	X	X	X		X		3 O wins	
5478	O	O	O	O	X	X	X	X			3 O wins	
5479												
5480												

Figure 3: Sample of output .csv for part A

As shown in Figure 3, there are 5478 legal boards output by the program.

## Part B

For part B, a program `checkboard.c` was developed that operates similarly to the program from part A, but instead of generating all possible board outcomes, it reads in a `.csv` file that contains one or more board layouts and determines the outcome. Unlike part A, this part makes use of file input in addition to output. Figure 4 shows a sample input `.csv` with one of each possible board outcome as a layout (In progress, X wins, O wins, no winner, or illegal board).

	A	B	C	D	E	F	G	H	I	
1	O	O	X	X		X	O			
2	X	O	X	X	O	O	O	X	X	
3	X		X	O	O	O	X	O	X	
4		O	O	X	X	X	X	O		
5	X	X	X	X	X	O	O	O	O	
6										

Figure 4: Example input `.csv` data for `checkboard.c`

To read in the `.csv` file to `checkboard.c`, a file pointer is declared the same way as in part A, and the name of the `.csv` to be read is given to `fopen()` as an argument, and the return value is given to the file pointer. The program uses `fgets()` as an argument to a `while` loop to run the code as long as there is a row to read in from the `.csv`. Within this loop, `sscanf()` is used inside of an `if` statement to save the board to a buffer if it is 9 characters long. Then, the `checkstate()` function is used in the same way as part A to analyze the board and determine its outcome. The analyzed result is then given to an output file pointer to either save the results to another `.csv` or `stdout`. Figure 5 shows the output of the program run with and without an output `.csv` argument.

```
Terminal - student@christian-yoga: ~/Desktop/ECE477/lab2
File Edit View Terminal Tabs Help
student@christian-yoga:~/Desktop/ECE477/lab2$ gcc checkboard.c -o checkboard
student@christian-yoga:~/Desktop/ECE477/lab2$ ./checkboard part_b_test.csv
0,0,X,X, ,X,0, , , Game in progress
X,0,X,X,0,0,0,X,X, No winner
X, ,X,0,0,0,X,0,X, 0 wins
,0,0,X,X,X,X,0, , X wins
X,X,X,X,X,0,0,0,0, Board not legal
student@christian-yoga:~/Desktop/ECE477/lab2$ ./checkboard part_b_test.csv part_b_result.csv
student@christian-yoga:~/Desktop/ECE477/lab2$
```

Figure 5: Output for example `.csv` input to `checkboard.c`

Figure 6 shows the resulting `.csv` from `checkboard.c`.

	A	B	C	D	E	F	G	H	I	J	
1	O	O	X	X		X	O			Game in progress	
2	X	O	X	X	O	O	O	X	X	No winner	
3	X		X	O	O	O	X	O	X	O wins	
4		O	O	X	X	X	X	O		X wins	
5	X	X	X	X	X	O	O	O	O	Board not legal	
6											

Figure 6: Output `.csv` for example `.csv` input to `checkboard.c`

As shown in Figure 6, the program successfully identifies the states of the boards in the sample `.csv`.

## Conclusion

The development of a C program that can output comma separated values to a file or `stdout` has been described. The program identifies all legal Tic Tac Toe boards, classifies them, and prints them either to a `.csv` file that can be imported into a spreadsheet or to the command line. The program takes the name of the desired output file as a command line argument or prints the results to the command line if no argument is given. The program identified 5478 legal boards.

For part B, another C program was developed which does the opposite of part A; it takes as an argument a `.csv` file which contains possible Tic Tac Toe scenarios and identifies if the board is legal or not, classifying the board if it is legal. This program can output a `.csv` file with the results in a similar fashion to part A if the file name is given as a second argument, but will print the results to the command line otherwise. This part makes use of file input, which part A did not.

## A Part A Source Code

Attached: `ttt.c`, `ttt_func.h`

## B Part B Source Code

Attached: `checkboard.c`, `ttt_func.h`