p2A: N in a Row

Due Sep 28 by 11:59pm **Points** 60 **Submitting** a file upload **File Types** c
Available until Sep 29 at 12:33am

This assignment was locked Sep 29 at 12:33am.

Learning GOALS

The purpose of this assignment is to practice writing C programs and gain experience working in this low-level, non-object oriented language. After completing both parts A and B of this project, you should be comfortable with pointers, arrays, address arithmetic, structures, command-line arguments, and file I/O in C. For this part, your focus will be on pointers, arrays and address arithmetic.

OVERVIEW

Tic-Tac-Toe is a game where two players alternate putting their mark (either X or O) on a game board until one player wins or there are no spaces available to mark (see Tic-Tac-Toe Wikipedia (https://en.wikipedia.org/wiki/Tic-tac-toe)). In a typical game of tic-tac-toe, the game board is a 3x3 grid and the winner is the first to get 3 of their marks "in a row" either horizontally, vertically, or diagonally. For this assignment, you'll be completing the program **n_in_a_row.c**, which processes a file containing the current game state, represented as a 2D grid of Xs and Os. **Your task is verify if the current state of the game board is valid or not.**

The game board size will be generalized to use a grid of **n** rows and **n** columns. This will require you to work with a dynamically allocated 2D array (heap allocation). The first value in the input file will be the value of n, the board size. Note the total number of Xs plus Os on the board will be in the range of 0 to n*n.

A key objective of this assignment is for you to practice using pointers. To achieve this, you are **not allowed** to use indexing to access the array that represents the tic-tac-toe board. Instead, you are required to use address arithmetic and dereferencing to access it. Submitting a solution using only indexing to access the tic-tac-toe board will result in a **50% reduction of your score**. You may use indexing to access any other arrays that might be used by your program.

You're welcome to develop the solution in phases. You could first code a solution that uses indexing. Once you have that solution working, you can replace indexing with pointer arithmetic before final testing and submission. If you do this approach, make sure to replace all accesses to your board to use address arithmetic and dereferencing to avoid a penalty.

You're strongly encouraged to use incremental development to code your solution rather than coding the entire solution followed by debugging that entire code. Incremental development adds code in small increments. After each increment is added, you test it to work as desired before adding the next increment of code. Bugs in your code are easier to find since they're more likely to be in the new code increment rather than the code you've already tested.

SPECIFICATIONS

You are to develop your solution using the skeleton code in the file **n_in_a_row.c** found on the CS Linux computers at:

```
/p/course/cs354-deppeler/public/code/p2A/n_in_a_row.c
```

The skeleton has several functions some of which have been completed or partially completed. You may also add your own functions if you wish.

The program **n_in_a_row.c** is run as follows:

```
./n_in_a_row <input_filename>
```

Where <input_filename> is the name of the file that contains the data representing the tic-tac-toe board. The format of the file is as follows:

- The first line contains one positive integer *n* for the number of rows and columns of the board. So the dimensions of the board are *n* x *n*. You may assume *n* is in the range of 3 to 99 inclusive.
- Every line after that represents a row in the board, starting with the first row. You may assume there will be *n* such lines where each line has *n* numbers (columns) separated by commas.
- Each number in the board is either 0, 1, or 2. If it is 1, then it denotes the presence of an **X** at that position. If it is a 2, then it denotes the presence of an **O** at that position. 0 corresponds with an unmarked space. You may assume only 0, 1, or 2 will be the values in the game board.

For instance, the following example shows the file format that represents a 3 x 3 game board:

```
3
2,1,2
2,0,1
2,1,1
```

that corresponds to the following board with X and O marks:

```
O|X|O
O| |X
O|X|X
```

Several sample input files, named **board1.txt** through **board4.txt**, are provided in the directory as shown in the example below:

```
/p/course/cs354-deppeler/public/code/p2A/board1.txt
```

These test files are meant to help you start testing your program. You'll need to create your own board files to test your program fully. We'll use secret test files to evaluate your program's correctness.

We've already provided code in the skeleton that reads and parses the input file, which you'll use to construct a dynamically allocated 2D array representing the board.

The program should print either valid or invalid followed by a newline (only these two outputs in lowercase will be accepted). Print valid only if the input file contains a valid board configuration, otherwise print invalid. To determine if the input board configuration is valid, you may also need to determine if there is a winner. You'll need to iterate over the board to check for winning lines, that is, **n** of the same marks in a row, column, or diagonal. **A valid board has:**

- an odd size; even size boards are invalid
- either the same number Xs as Os, or 1 more X than O since we're assuming X always moves first
- either no winner or one winner; X and O cannot both be winners
- either one winning line (i.e., row, column, or diagonal), or two winning lines that intersect on one mark; two parallel winning lines are invalid

The sample runs below shows the expected behavior of the program:

```
[deppeler@liederkranz] (33)$ ./n_in_a_row
Usage: ./n_in_a_row <input_filename>
[deppeler@liederkranz] (34)$ cat board1.txt
3
2,1,2
2,0,1
2,1,1
[deppeler@liederkranz]] (35)$ ./n_in_a_row board1.txt
valid
[deppeler@liederkranz] (36)$ cat board4.txt
5
1,2,2,2,2
1,1,1,2,2
1,2,1,1,2
1,2,2,1,2
1,0,0,1,2
[deppeler@liederkranz]] (37)$ ./n_in_a_row board4.txt
invalid
```

HINTS

Using library functions is something you will do a lot when writing programs in C. Each library function is fully specified in a manual page. The **man** command is very useful for learning the parameters a library

function takes, its return value, detailed description, etc. For example, to view the manual page for **fopen**, you would issue the command **man fopen**. If you are having trouble using man, the same manual pages are also available online. You will need some of these library functions to write this program and will see that some of them are already used in our code. You do not need to use all of these functions since a couple of them are just different ways to do the same thing.

- **fopen()** to open the file.
- **malloc()** to allocate memory on the heap
- **free()** to free up any dynamically allocated memory
- **fgets()** to read each input from a file. fgets can be used to read input from the console as well, in which case the file is stdin, which does not need to be opened or closed. An issue you need to consider is the size of the buffer. Choose a buffer that is reasonably large enough for the input.
- **fscanf()/scanf():** Instead of fgets() you can also use the fscanf()/scanf() to read input from a file or stdin. Since this allows you to read formatted input you might not need to use strtok() to parse the input.
- **fclose()** to close the file when done.
- **printf()** to display results to the screen.
- **fprintf()** to write output to a file.
- **atoi()** to convert the input which is read in as a C string into an integer
- **strtok()** to tokenize a string on some delimiter character. In this program the input file for a square has every row represented as columns delimited by a comma. See **here (http://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm)** for an example on how to use strtok to tokenize a string.

REQUIREMENTS

- Your program must dynamically allocate (i.e., on the heap) the tic-tac-toe board.
- Your program must use address arithmetic and dereferencing to access the array representing the board.
- Your program must operate exactly as the sample runs above.
- Your program must print an error message, as shown in the sample runs above, and then call exit(1) if the user invokes the program incorrectly (for example, without any arguments, or with two or more arguments).
- Your program must check the return values for errors of the library functions, malloc(), fopen(), and fclose(). Handle errors by displaying an appropriate error message and then calling exit(1).
- Your program must properly free up all dynamically allocated memory at the end of the program.
- Your program must follow style guidelines as given in the **Style Guide**.
- Your program must follow commenting guidelines as given in the **Commenting Guide**.
- We will compile your programs with **gcc -Wall -m32 -std=gnu99** on the Linux lab machines. So, your programs must compile there, and without warnings or errors.

SUBMITTING & VERIFYING

SUBMISSION FOR p2A HAS BEEN ENABLED.

Leave plenty of time before the deadline to complete the two steps for submission found below. There is a 33 minute grace period after the deadline for last minute emergencies. Submitting during this grace period results in your submission being marked late but it will be accepted for grading without penalty. No submissions or updates to submissions are accepted after this grace period.

1.) Submit only the file listed below under Project p2A in Assignments on Canvas. Do not zip, compress, or submit your file in a folder.

- **n_in_a_row.c**

Repeated Submission: You may resubmit your work repeatedly so we strongly encourage you to use Canvas to store a backup of your current work. If you resubmit, Canvas will modify your file names by appending a hyphen and a number (e.g., n_in_a_row-1.c).

2.) Verify your submission to ensure it is complete and correct. If not, resubmit all of your work rather than updating just some of the files.

- **Make sure you have submitted all the files listed above.** Forgetting to submit or not submitting one or more of the listed files will result in you losing credit for the assignment.
- **Make sure the files that you have submitted have the correct contents.** Submitting the wrong version of your files, empty files, skeleton files, executable files, corrupted files, or other wrong files will result in you losing credit for the assignment.
- **Make sure your file names exactly match those listed above.** If you resubmit your work, Canvas will modify your file names as mentioned in **Repeated Submission** above. These Canvas modified names are accepted for grading.

Project p2A

Criteria	Ratings				Pts
1. Compiles without warnings or errors	**6.0 pts** **No warnings or errors**	**0.0 pts** **One or more errors or at least 5 warnings**			6.0 pts
2. Follows commenting and style guidelines	**6.0 to >0.0 pts** **Followed**	**0.0 pts** **Not followed**			6.0 pts
3. Implements CLA checking	**3.0 pts** **Meets specifications**	**2.0 pts** **Minor problem** Error message does not match specification	**1.0 pts** **Major problem** Incorrect argc check, no error message displayed, or does not exit	**0.0 pts** **Does not check CLAs**	3.0 pts
4. Checks return values of malloc() and fopen()	**6.0 pts** **Checks all**	**3.0 pts** **Checks some**	**0.0 pts** **Checks none**		6.0 pts
5. Closes all opened files - fclose()	**3.0 pts** **Closed**	**2.0 pts** **Some closed**	**0.0 pts** **None closed**		3.0 pts
Execution test: frees heap memory	**6.0 pts** **Freed**	**6.0 to >0 pts** **Not all freed**			6.0 pts
Execution test: board1.txt (provided)	**2.0 pts** **Correct result**	**0.0 pts** **Incorrect result**			2.0 pts
Execution test: board2.txt (provided)	**2.0 pts** **Correct result**	**0.0 pts** **Incorrect result**			2.0 pts
Execution test: board3.txt (provided)	**2.0 pts** **Correct result**	**0.0 pts** **Incorrect result**			2.0 pts
Execution test: board4.txt (provided)	**2.0 pts** **Correct result**	**0.0 pts** **Incorrect result**			2.0 pts

Criteria	Ratings		Pts
Execution test: valid1.txt (tie, complete board)	**2.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	2.0 pts
Execution test: valid2.txt (tie, incomplete board)	**2.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	2.0 pts
Execution test: valid3.txt (X wins, row + column)	**2.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	2.0 pts
Execution test: valid4.txt (O wins, column)	**2.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	2.0 pts
Execution test: valid5.txt (X wins, row + diagonal)	**1.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: valid6.txt (O wins, column + diagonal)	**1.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: valid7.txt (O wins, two diagonals)	**1.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: valid8.txt (X wins, row)	**1.0 pts** **Displays valid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: invalid1.txt (even board size)	**1.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: invalid2.txt (|X| > |O| + 1)	**2.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	2.0 pts
Execution test: invalid3.txt (|O| > |X|)	**2.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	2.0 pts

Criteria	Ratings		Pts
Execution test: invalid4.txt (two X winning rows)	**1.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: invalid5.txt (two O winning rows)	**1.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: invalid6.txt (two X winning columns)	**1.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	1.0 pts
Execution test: invalid7.txt (X and O winning columns)	**2.0 pts** **Displays invalid**	**0.0 pts** **Incorrect result**	2.0 pts
			Total Points: 60.0