p2A: Check Queens

Due Feb 15 by 10pm **Points** 60 **Submitting** a file upload **Available** until Feb 15 at 10:33pm

This assignment was locked Feb 15 at 10:33pm.

GOALS OVERVIEW SPECS HINTS REQUIREMENTS SUBMITTING

- This project must be done individually.
  - *It is academic misconduct to share your work with others in any form including posting it on publicly accessible web sites, such as GitHub.*
  - *It is academic misconduct for you to copy or use some or all of a program that has been written by someone else.*
- All work for this project is to be done on the **CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities)** . You're encouraged to remotely login using the ssh command in the terminal app on Macs, or on Windows using an ssh client such as MobaXterm.
- All projects in this course are graded on **CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities)** . To receive credit, make sure your code runs as you expect on these machines.

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

On a chessboard, two queens are said to be in a **non-attacking** position if they do **not** share the same row, column or diagonal. For this assignment you will be writing a program **check_queens.c,** in which you will be given the positions of queens on a chessboard. Your task is to find if there exists **any pair** of queens that can attack each other.

A chessboard has 8 rows and 8 columns and, typically, no more than 2 queens. For this program, this problem will be generalized. The chessboard will vary so that the row and column sizes may be more or less than 8 and the board might not be square. This will require you to work with a dynamically allocated 2D array (heap allocation). The number of queens may also be more or less than 2 (in the range from 0 queens to rows*columns queens).

A key objective of this assignment is for you to practice using pointers. To achieve this, you are **not allowed** to use indexing (e.g., `array[i][j]`) to access the **2D array** that is used to represent the chessboard. Instead, you are required to use address arithmetic and dereferencing to access them. Submitting a solution using indexing to access the 2D arrays will result in a **significant deduction of points**. You may use indexing to access any 1D arrays used by your program.

You're welcome to develop the solution in two phases. 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. 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 **check_queens.c** found on the CS Linux computers at:

```
/p/course/cs354-skrentny/public/code/p2/check_queens.c
```

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

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

```
./check_queens <input_filename>
```

Where <input_filename> is the name of the file that contains the board that has the positions of queens. The format of the file is as follows:

- The first line contains two positive integers *m* and *n* (separated by a comma) for the number of rows and columns of the board respectively. So the dimensions of the board are *m* x *n*. You may assume *m* and *n* are in the range of 1 to 100.
- Every line after that represents a row in the board, starting with the first row. There will be *m* such lines where each line has *n* numbers (columns) separated by commas.
- Each number in the board is either 0 or 1. If it is 1, then it denotes the presence of queen at that position.

So for instance, a file containing a matrix of dimensions 3 x 4 will have this format:

```
3,4
0,1,0,0
1,0,0,0
0,0,1,0
```

Several sample input files, named check1.txt through check4.txt, are provided in the directory as shown in the example below:

```
/p/course/cs354-skrentny/public/code/p2/check3.txt
```

These test files are not meant to cover all possible board configurations. We'll use several 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.

To solve this problem, for **each** queen, you need to check if there exist any other queens that share the same row, column or diagonal. If yes, then you have found a pair of queens that can attack each other. If no such pair exists, then no queens can attack each other. Consider a 5 x 4 chessboard. Two queens are said to be in the same diagonal in the 5 x 4 chessboard if they both lie on cells with the same color as shown in the 2 diagrams below:

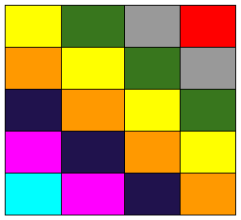
Fig 1: Set of leading diagonals

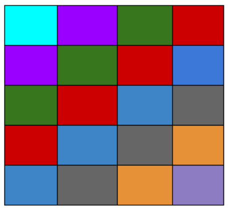
Fig 2: Set of non-leading diagonals

The program should print <u>true</u> if the input file has any pair of queens which can attack each other, and <u>false</u> otherwise (do not use other words like right/wrong, valid/invalid, etc.).

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

```
[skrentny@jimbo] (77)$ ./check_queens
Usage: ./check_queens <input_filename>
[skrentny@jimbo] (78)$ cat check4.txt
3,4
1,0,0,0
```

```
0,0,0,1
0,1,0,0
[skrentny@jimbo]] (79)$ ./check_queens check4.txt
false
[skrentny@jimbo] (80)$ cat check3.txt
4,5
1,0,0,0,0
0,1,0,0,0
0,0,0,0,1
1,0,0,0,0
[skrentny@jimbo]] (81)$ ./check_queens check3.txt
true
```

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 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 the integers 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. 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 use address arithmetic and dereferencing to access the 2D array representing the board.
- Your program must follow style guidelines as given in the **Style Guide**.
- Your program must follow commenting guidelines as given in the **Commenting Guide**.
- 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 an argument, or with two or more arguments).
- Your program must check return values of library functions that use return values to indicate errors, e.g., malloc(), fopen(), and fclose(). Handle errors by displaying an appropriate error message and then calling exit(1).
- Your program must free up all dynamically allocated memory at the end of the program. For the 2D array you would need to free up all allocated memory in the reverse order of how you allocated it, i.e., don't just free the pointer to the array of arrays.
- 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

Submit the following **source file** under Project p2A in Assignments on Canvas before the deadline:

1. check_queens.c

It is your responsibility to ensure your submission is complete with the correct file names having the correct contents. The following points will seem obvious to most, but we've found we must explicitly state them otherwise some students will request special treatment for their carelessness:

- **You will only receive credit for the files that you submit.** You will not receive credit for files that you do not submit. Forgetting to submit, not submitting all the listed files, or submitting executable files or other wrong files will result in you losing credit for the assignment.
- **Do not zip, compress, submit your files in a folder, or submit each file individually.** Submit only the text files as listed as a single submission.
- **Make sure your file names exactly match those listed.** If you resubmit your work, Canvas will modify the file names by appending a hyphen and a number (e.g., check_queens-1.c) and these Canvas modified names are accepted for grading.

Repeated Submission: You may resubmit your work repeatedly until the deadline has passed. We strongly encourage you to use Canvas as a place to store a back up copy of your work. **If you resubmit, you must resubmit all of your work rather than updating just some of the files.**

p2A

Criteria	Ratings			Pts
1. Compiles without warnings or errors.	**6.0 pts No warnings or errors**	**0.0 pts error or at least 6 warnings**		6.0 pts
2. Follows the style and commenting guide.	**6.0 pts Full Marks**	**0.0 pts No Marks**		6.0 pts
3. Checks return values of library functions. Must check: malloc(), fopen(), fclose()	**6.0 pts Checks all**	**4.0 pts Checks some**	**0.0 pts No Marks**	6.0 pts
4. Displays the usage error message if argc is incorrect.	**3.0 pts Full Marks**	**0.0 pts No Marks**		3.0 pts
5. Frees all dynamically allocated memory.	**5.0 pts Frees all**	**3.0 pts Frees some**	**0.0 pts No Marks**	5.0 pts
6. Closes all opened files.	**1.0 pts Full Marks**	**0.0 pts No Marks**		1.0 pts
7. Uses address arithmetic instead of indexing a[][] on the 2D array board.	**9.0 pts Full Marks**	**5.0 pts Uses both**	**0.0 pts No Marks**	9.0 pts
8. Properly constructs the board based on input file.	**7.0 pts Full Marks**	**4.0 pts Wrong size - Buffer may overflow**	**0.0 pts No Marks**	7.0 pts
9. test t3	**1.0 pts Full Marks**	**0.0 pts No Marks**		1.0 pts
10. Run test t4	**2.0 pts Full Marks**	**0.0 pts No Marks**		2.0 pts
11. Run test t5	**2.0 pts Full Marks**	**0.0 pts No Marks**		2.0 pts
12. Run test: fcheck1.txt	**1.0 pts Full Marks**	**0.0 pts No Marks**		1.0 pts
13. Run test fcheck2.txt	**1.0 pts Full Marks**	**0.0 pts No Marks**		1.0 pts

Criteria	Ratings		Pts
14. Run test fcheck3.txt	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
15. Run test fcheck4.txt	**1.0 pts** **Full Marks**	**0.0 pts** **No Marks**	1.0 pts
16. Run test fcheck5.txt	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
17. Run test fcheck6.txt	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
18. Run test tcheck2.txt	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
19. Run test tcheck3.txt	**2.0 pts** **Full Marks**	**0.0 pts** **No Marks**	2.0 pts
			Total Points: 60.0