CMSC 12300: Computer Science with Applications III

The University of Chicago, Spring 2021

Quiz 2

Friday, May 21, 2021 – Saturday, May 22, 2021

This quiz has a **90-minute time limit**. This timer starts as soon as you view any material on the quiz beyond this page of instructions, or when you view the starter file in your repository.

By submitting the quiz you are implicitly certifying that you have adhered to the standards of academic honesty. In particular, you are confirming that:

- You wrote your responses in one 90-minute sitting (or more time if by prior special arrangement).
- You did not discuss the quiz content with anyone else.
- You will refrain from communicating with anyone about the quiz content until after the quiz submission deadline, and refrain from any online posts until after it has been graded and returned.
- You did not refer to any materials during the quiz other than your own class notes, the class Ed Discussion site, your own prior assignments, recordings of lectures from the class, and the class web site.

You may choose when you start the quiz, but must complete all work within 90 minutes from starting. You will need to commit your completed work to your git repository. There is no chisubmit step, but please make sure you have performed a `git add` for `quiz2.c`, a `git commit`, and a `git push`. This quiz is due Saturday, May 22 at 11:59pm Chicago time. Your submission will be collected from your git repository at that time.

This quiz consists of 3 pages, including this one. There are 4 problems. In total, all problems are worth 50 points.

There is a file named `quiz2.c` in the `quiz2` directory of your repository. You must edit this file to contain your answers and commit the updated contents to your repository.

You may modify the provided test code to perform additional tests, but your solutions must work correctly if they were to be run with the original tests. The tests we have provided are not intended to be comprehensive, and your code will be evaluated on complete correctness, not merely on limited test cases.

The individual tasks are explained on the following pages.

You may run and test your code, and debug it. Please keep in mind the time limit for the overall quiz, however.

To compile and run:

```
clang -Wall quiz2.c
./a.out
```

You may not ask clarifying questions during the test; read the text of the test, including these instructions, closely and literally, and respond as best you can. It is unfair for certain students to interact with instructors while others don't. If you feel a question is unclear, state your assumptions.

For all problems, strive for the most efficient possible implementation. Your code will be evaluated both for correctness and for optimality. That said, code that works, even if less efficiently, is always better than "efficient" code that does not work.

Introduction and Data Description

For this quiz, you will write multiple functions that work with matrices in C. The file `quiz2.c` provides data type definitions similar to those in PA#2, including for a type, `matrix_t`, that represents a pointer to a `struct` with the matrix elements, and the row and column dimensions.

Here is an example 5x5 matrix:

```
11 12 13 14 15
21 22 23 24 25
31 24 24 24 35
13 23 24 24 53
51 51 53 14 55
```

Although this example happens to be square, the matrix need not be square in general.

You may assume that any indices passed into a function that takes in index parameters are, in fact, within the bounds of the matrix. You do not need to check for violations of this or write code that behaves robustly if given an out-of-bounds input. That said, you do need to be careful never to access the matrix out-of-bounds in situations where you were given proper inputs. You may not assume any particular dimensions, and should examine the values of fields `n` and `m` in the matrix `struct`.

Problem 1 [*10 points*]

Complete the implementation of function `row_matches_col`. This function takes in a matrix and returns `true` if and only if at least one row, reading left to right, has entirely the same elements in the same order as at least one column, reading top to bottom. For instance, counting from index 0, in the example matrix given earlier, row 3 matches column 2.

Problem 2 [*10 points*]

Complete the function `col_palindrome` that returns `true` if and only if the column with the specified index is a palindrome. (A *palindrome* is a sequence that is the same forwards and backwards.) For instance, the column with index 3 is a palindrome in the example matrix, but no others are.

Problem 3 [*10 points*]

Complete the function `alternator`. This function should take in a matrix and while keeping its dimensions the same, overwrite its contents to be a pattern of alternating 1s and 0s, with a 1 a at the top left corner, and alternating between 1s and 0s moving in both the horizontal and vertical directions from any point in the matrix.

For instance, for a 5x5 matrix, the result should be the matrix:

```
1  0  1  0  1
0  1  0  1  0
1  0  1  0  1
0  1  0  1  0
1  0  1  0  1
```

For a 4x4 matrix, the result should be the matrix:

```
1  0  1  0
0  1  0  1
1  0  1  0
0  1  0  1
```

Problem 4 [*20 points*]

Complete the function `horiz_run`, which returns `true` if and only if there are at least `run` many horizontally-adjacent copies of the same number in some row. For instance, in the original example matrix, there are three copies of 24 in row 2, so the property is true of the overall matrix if `run` is 3, but not true if `run` is 4 (there are no longer runs of any number).