Exercise 07

For this exercise, you get to fill in a missing parts in a program that works with a grid of small integers, stored in a variable-sized array. The program is called hilltop.c. Its job is to find local maxima in the grid, values that are larger than any of their neighboring values. You can download the exercise files from the course webpage, or using the following curl commands:

curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise07/hilltop.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise07/input-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise07/expected-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise07/input-2.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise07/expected-2.txt

The following shows what's in the first sample input file, input-1.txt. The first two values give the size of the grid of numbers (rows then columns). The remaining lines give the contents of the grid, with a row on each input line.

```
4 5
 7 49 73 58 30
72 44 78 23  9
40 65 92 42 87
 3 27 29 40 12
```

We'll say a local maximum is a value that's greater than the value in any of its (up to) eight neighboring elements in the grid. So, for example, the 72 at the start of row 1 is a local maximum. It has five neighbors (7, 49, 44, 65 and 40), and its value is greater than any of them. The 92 in row 2 and the 87 in row 2 are also local maxima.

When you run your program it should read the grid description from standard input. Like in the example above, this will give with the grid size, then all the values in the grid. You can assume the input is valid; you don't need to detect invalid or missing values (but, you can if you want to).

For output, your program should report the row and column locations of all the local maxima in the grid. Report them in row-major order, from the top row of the grid to the bottom and from left to right within each row. Rows and columns are numbered starting from zero. So, for example, if you run your program on the example above, it should produce the following output.

```
$ ./hilltop < input-1.txt
1 0
2 2
2 4
```

Completing the Implementation

On the course homepage and in the following AFS directory, you'll find a partial implementation for this program, along with a couple of sample inputs and expected output files.

/afs/eos.ncsu.edu/courses/csc/csc230/common/www/sturgill/exercise07

To complete the program, you'll need to do the following.

- In main, add code to read the number of rows and columns of the grid from standard input and create a variable-sized, two-dimensional array to store the grid of values. Make your array exactly the right size to hold the values you're about to read.

- Fill in the readGrid() function with code to read input values into your array. Since arrays are automatically passed by reference, this function can change the contents of your array, even though

you created that array in main. (when you're reading an int value with scanf(), don't forget to put the & in front of the parameter, even if that parameter is an array element.)

- Add a function call in main() to call your readGrid() function, right after you've created your array.

- Fill in the parameters and the body of the reportMaxima() function so it goes through all the elements of the array and reports the row and column location of any local maxima. In my solution, I when through all the elements of the array, and, for each one, I compared its value with all its neighbors. This took a quadruply-nested loop the way I did it, but you can solve the problem other ways if you want to.

  Remember, C doesn't do bounds checking on arrays, so, when you're looking at the neighbors of an element, you're probably going to need some code to make sure you don't accidentally check locations that are off the edge of the grid.

- Add a function call in main() to call your reportMaxima() function.

Optional (extra credit)

After reading the size of the array, all values should be initialized (using loops) to zeros.
The input after the array size can consist of *0 or more* rows, up to but not exceeding the number of rows expected. Each row can consist of *0 or more elements (or columns),* up to but not exceeding the number of columns expected. Values not provided in the input should remain initialized to 0. So, for instance, the following would be possible:

```
4 5
7 49 73 58 30
72 44 78 23 9
40 65 92 42 87
3 27 29 40 12
```

i.e. all element values provided, all initialization values overwritten

```
4 5
7 49 73
72 44 78 23
40 65 92 42 87
```

i.e. last 2 elements of the first row have values of 0, the last 1 element of the second row has value 0, and the last row of the array is entirely 0

```
4 5
```

i.e. no values provided for any array elements, so all 0!

An optional input, **optional-input-3.txt**, will be provided, as well as the expected output, **optional-output-3.txt**.

If you elect to do this optional part, do not break your non-optional part!

Submitting your Work

When you're done, submit your completed hilltop.c source file using the exercise_07 assignment on Moodle.