Computer Science I

Exam 2 - Fall 2025

School of Computing College of Engineering University of Nebraska-Lincoln

- Absolutely **no collaboration is allowed with any individual**
- The use of any AI tools is strictly prohibited, all work must be your own
- Regular help hours will be held, but LAs will not give the same level of help as they would on a hack/lab
- For those in the *HONORS* section: you must do both the C and Java versions (see `FloodUtils.java`).
- 9 points are awarded for the usual style, documentation, design, the remaining 66 points for correctness
- Good luck!

Problem Statement

Civil engineers studying flood control have hired you to develop several utility functions for their flood monitoring computer system.

The function prototypes have been provided in the header file, `flood_utils.h`. We have also provided a basic informal testing suite, `floodTester.c` that you should use to test your code and add your own test cases. You must provide your own source file, `flood_utils.c` with the function implementations as specified below.

All of your functions must: - Have proper documentation - Have proper error handling and return appropriate error codes or values - Run with no segmentation faults

Flood Utility Functions

Waterway Flow Conversion

Consider a waterway (such as a canal) with monitoring stations at various intervals. At each interval the monitoring station reports a cross-sectional area of water in square meters (m^2). This data is stored in a 1-D array; for example:

```
{350, 352.5, 351.2, 355.2, 354.0}
```

Write the following function that converts this array to square feet (sqft) but in a *new* array that is to be returned from the function.

```
double * convertFlow(const double *flow, int numPoints)
```

Note that 1 square meter = 10.76391 square feet.

Elevation Tools

In order to predict flood conditions, the engineers have produced a grid of elevations in an area which is represented as an `n x m` two dimensional array of `double`s. For example, the following is a `3 x 5` grid:

```
{ 9.50, 4.75, 7.25, 8.25, 8.25 }
{ 8.50, 9.35, 6.45, 6.50, 7.25 }
{ 7.50, 8.60, 4.50, 5.50, 5.75 }
```

You have been tasked with writing the following functions.

1. Write a function to find and report the location (grid points) of the minimum elevation (the highest risk of flood). Since a location is identified by two coordinates (a row and a column), you'll need to "return" those coordinates via the two pass-by-reference variables:

```
int minElevation(double **elevations, int n, int m, int *minRowIndex,
int *minColIndex)
```

For the example above, the minimum elevation is 4.5 at row/column `(2, 2)`.

2. Write a function to compute the *potential flood capacity* of the area given a potential water level, `waterLevel`. For example, if the potential water level were `6.0` then any location that was *less* than `6.0` would be flooded. This function will sum up the *differences* in all such locations.

For the grid example above: the first location has an elevation of `9.5` which is *above* the `6.0` water level and so is not flooded. The second (in the first row) is `4.75` which is *below* the water level, so there are `6.0 - 4.75 = 1.25` meters of flood water. When we sum up all such differences in the example above we get total of 3.5 meters of flood water over the entire grid.

```
double floodCapacity(double **elevations, int n, int m, double
waterLevel)
```

Instructions & Grading

- Submit your `flood_util.h` header file and `flood_util.c` source files through codepost and make sure all tests pass.
- Grading will be based on style, documentation, design, and correctness just like a Hack. However, correctness will count for more than usual as detailed on codepost.