ECE 220: Computer Systems & Programming

Spring 2020 – Midterm Exam 2

April 9, 2020

1. This is an open book exam; you may use any resources available to you.
2. You cannot ask another person for help, provide help to a fellow student, or post/share your answers online.
3. Your programs will be graded on functionalities only. If your code does not compile, you will receive a zero for that question.

Exam Workflow:

1. Fetch exam files from course GitHub (same as if fetching a new MP)
2. Complete the exam and **commit changes to GitHub** (you should fully test your code for each question)
3. Use the following command in the 'MT2' folder to create a zip file: **zip -r mt2.zip P1 P2 P3**
4. Upload mt2.zip to Gradescope

Disclaimer: you can modify the provided test case in each question to fully test your code. Additional test cases will be used during grading.

Problem 1 (30 points): Find the first non-slope row

Write a C function to find the **index of the first non-slope row** in a 2-D array. We define a row is a slope if the values in the row are either monotonically increasing or decreasing. If adjacent values are the same, the row should NOT be considered slope. For example:

A 2-D array:

{ 1, 2, 3, 4, 5, 6 } => slope (monotonically increasing)
{ 6, 5, 4, 3, 2, 1 } => slope (monotonically decreasing)
{ 1, 2, 4, 5, 6, 9 } => slope (monotonically increasing)
{ 8, 1, 6, 5, 4, 3 } => not slope
{ 1, 2, 2, 3, 4, 5 } => not slope
{ 8, 7, 6, 3, 2, 1 } => slope (monotonically decreasing)

In this example, your function should return 3 (the row index starts from 0).

You need to implement the following functions in **slope.c**:

1. **int find_nonslope_row(int array[6][6])**
   This function returns the index of the first row that is not a slope. Otherwise it returns -1.
2. **int read_array(char* filename, int array[6][6])**
   This function reads the file that contains the values of the 2D array and stores them in *array*. The rows and the columns in the file are separated by a new line and a single space, respectively. The function returns 1, if the file is successfully opened. Otherwise, it returns 0.

Details:

1. The size of the 2-D array is 6 x 6.
2. The values are positive integers.
3. You can modify the main function to test your code, we will only grade your slope.c file.
4. To compile: **gcc -std=c99 -Wall main.c slope.c -o slope**
5. To run your code: **./slope**

Problem 2 (35 points): Build matrix

In this problem, assume there is an **n x n** 2-D array. The 0th row and 0th column of this matrix should be assigned with random integers between 0 and 99. Afterwards, find the maximum number by building up the matrix using the rules defined below.

Step 1:

Complete the *init_0th_row* and *init_0th_col* functions with random number from 0 ~ 99 inclusively. The seed has been set in main. Note: 0 will be assigned at position (0, 0).

Step 2:

Complete the *build_matrix_max* function.

We compute the elements by following the arrows shown in the figure:

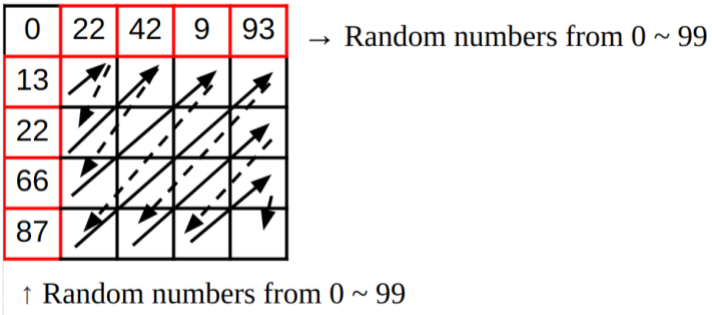
1. Starts from (1,1) and then go from the lower-left to the upper-right in each diagonal (solid arrows)
2. Jump to the next diagonal when we reach the top or right (in dashed arrows).

For each element we go through, we compute the maximum from the element directly to the top and the element directly to the left.

Suggested algorithm of build_matrix_max (you are not required to follow this):

```
initialize i to 1 and j to 1
while( i is smaller than n AND j is smaller than n )
    // update array(i, j)
    get the maximum value of array(i, j-1) and array(i-1, j), assign it to array(i, j)
    // update i and j
    if currently at row 1 AND not the last column
        set i to j+1 and then set j to 1
    else if currently at the last column
        set j to i+1 and then set i to n-1
    else
        decrement i and increment j
```

For each unfilled element in the matrix, we compute its value by computing the maximum between its left and top neighbors.



Below is an example:

	22
13	22 (Max of 22, 13)

The 2-D array should look like the following at the end of the program:

0	22	42	9	93
13	22	42	42	93
22	22	42	42	93
66	66	66	66	93
87	87	87	87	93

Details:

1. The size of the 2-D array is n x n, assume n>=2
2. You ONLY need to implement functions in array.c
3. You can modify the main function to test your code, we will only grade your array.c file
4. To compile: **gcc -g -std=c99 -Wall -Werror main.c array.c -o array**
5. To run your code: **./array**

Problem 3 (35 points): Recursion

For this problem, you are provided a 2D matrix of integers, **`data`**, with ROW rows and COL columns, where the values of the array are in ascending order in the following manner: each column contains values in ascending order, and the value of each **`data[0][j]`** is greater than all **`data[i][j-1]`** for all **`i`**. An example of **`data[4][5]`** is provided below:

3	**10**	**29**	**50**	**59**
6	**14**	**30**	**52**	**63**
7	**18**	**32**	**54**	**64**
9	**27**	**48**	**55**	**66**

Write a function

```
Search(int item, int data[ROW][COL], int start_row,
       int end_row, int start_col, int end_col)
```

to determine whether **`item`** exists within **`data`** between **`start_row`** and **`end_row`**, **`start_col`** and **`end_col`**. If **`item`** exists within **`data`**, return 1; otherwise, return 0;

To accomplish this efficiently, first perform a binary search in Row 0 for the item to locate its column. Then, perform a binary search in the appropriate column for the item.

Suggested algorithm of search function (you are not required to follow this):
Step 1: if start_row is greater than end_row OR start_col is greater than end_col, item is not within data (this is the base case)
Step 2: if start_col is not the same as end_col, perform recursive binary search in Row 0 to locate the appropriate column
Step 3: perform recursive binary search in the appropriate column to find item; return 1 if item is found

Your algorithm must perform recursive binary searches only. Otherwise, you program will receive a score of zero.

Details:

1. You can modify the main function to test your code, we will only grade your Search.c file
2. To compile: **gcc -g -std=c99 -Wall -Werror main.c Search.c -o Search**
3. To run your code: **./Search**
4. We will call your function with arguments **start_row** = 0, **end_row** = ROW-1, **start_col** = 0, **end_col** = COL-1.

End of ECE 220 Midterm Exam 2