Name: Tien Dung Le (1002024297)

**Problem 1 – 18 pts.**

What does the following code print?

#include <stdio.h>

void foo(int b[3][3], int a[3][2])

{

for (int c = 0; c < 3; c++)

{

for (int r = 0; r < 3; r++)

{

if (b[r][c] < 0)

b[r][c] = 0;

if(c < 2 && a[r][c] < 0)

a[r][c] = 0;

}

}

}

void print3x3Mtx(int mtx[3][3])

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

printf("%3d ", mtx[i][j]);

}

printf("\n");

}

}

int main()

{

int a[3][3] = {{3, 2, -5}, {-8, 2, 3}, {6, 2, -1}};

int b[3][2] = {{1, 2}, {3, -4}, {-4, 2}};

foo(a, b);

print3x3Mtx(a);

return 0;

}

3 2 0  
 0 2 3  
 6 2 0

**Problem 2 – 12 pts**

What does the following code print?

#include <stdio.h>

void foo(int mtx[3][2], int result[2][3])

{

for (int r = 0; r < 3; r++)

{

for (int c = 0; c < 2; c++)

result[c][r] = mtx[r][c];

}

}

void printMtx(int mtx[2][3])

{

for (int r = 0; r < 2; r++)

{

for (int c = 0; c < 3; c++)

printf("%3d", mtx[r][c]);

printf("\n");

}

}

int main()

{

int m[3][2] = {{1, 2}, {3, 4}, {5, 6}};

int t[2][3];

foo(m, t);

printMtx(t);

return 0;

}

1 3 5  
 2 4 6

**Problem 3 – 25 pts**

I have provided a file called problem3.c that contains an incomplete program. Find the smallest and largest values in an integer matrix (2D array). I have used #define statements to set the size of the matrix to have 4 rows and 3 columns. Complete the program by defining these functions:

1. A function called generateMtx that satisfies the following specs:
   1. Takes a 2D integer array as an argument.
   2. Asks the user for each value that should be placed in the array and stores it in the appropriate spot. See the example output for formatting.
2. A function called matrixSmallest that looks through the matrix for the smallest number and returns it.
3. A function called matrixLargest that looks through the matrix for the largest number and returns it.
4. A function called elementPosition that takes a 2D integer array, an integer, and a 1D integer array containing two elements. The function searches the 2D array for the integer and stores the row and column where the element was found in [row, col] format in the 1D array. If it could not find the element, it should store [-1, -1] in the array.

I have written main. It is your job to write the functions. You can add them directly to the source file I provide. You are not allowed to modify main in any way.

Below is the output for a sample run of the program using the following matrix as input.

Enter row 0, column 0: 1

Enter row 0, column 1: 2

Enter row 0, column 2: 3

Enter row 1, column 0: 4

Enter row 1, column 1: 5

Enter row 1, column 2: 6

Enter row 2, column 0: 7

Enter row 2, column 1: 8

Enter row 2, column 2: 9

Enter row 3, column 0: 10

Enter row 3, column 1: 11

Enter row 3, column 2: 12

Largest element: 12

found at row 3, column 2

Smallest element: 1

found at row 0, column 0

**Problem 4 – 45 pts**

For this program, you will write both main and a few other functions. Below, I list a few that you have to write, but you are welcome to write any additional utility functions that might be useful. Write a program called problem4.c that satisfies the following specifications:

1. Has a function called columnAverages that calculates the average of each column in a 2D array of float and somehow gets this information back to main. It is up to you how to do this, but it might prove useful to use an array to store the averages. You can find hints in Problem 3.
2. Has a function called rowAverages that calculates the average of each row in a 2D array of float and somehow gets this information back to main. It is up to you how to do this, but it might prove useful to use an array to store the averages. You can find hints in Problem 3.
3. Has a function called regionAverage that takes a 2D array of floats, an integer top, an integer bottom, an integer left, and an integer right that define a region of the array. The function should return the average of values in all positions (i, j) of the array such that top <= i <= bottom and left <= j <= right.
4. Your program should do the following:
   1. Use #define to specify the number of rows as 5 and the number of columns as 3
   2. Create a 2D float array of the appropriate size.
   3. Ask the user for those values to populate the array.
   4. calls your columnAverages and rowAverages functions to get the averages.
   5. Prints the matrix with headers. You can limit it to 1 decimal place with a field width of 8.
   6. Prints the matrix with headers, the row averages after each row, and the column averages below each column. See the example below for the formatting.
   7. Asks the user for the top, bottom, left, and right edges of a region of the array they want to calculate the average for.
   8. Calls your regionAverage function to calculate the average of that region and prints the region average.
   9. See below for how the output should look.
   10. You can create any extra functions you feel you need.

Given a matrix entered as the following:

Here is the output of the program using that matrix, so you can get the formatting correct:

Enter row 0, column 0: 3.2

Enter row 0, column 1: 2.1

Enter row 0, column 2: 5.3

Enter row 1, column 0: 8.0

Enter row 1, column 1: 4.9

Enter row 1, column 2: 5.7

Enter row 2, column 0: 18

Enter row 2, column 1: 14.9

Enter row 2, column 2: 15.7

Enter row 3, column 0: 28

Enter row 3, column 1: 24.9

Enter row 3, column 2: 25.7

Enter row 4, column 0: 38

Enter row 4, column 1: 34.9

Enter row 4, column 2: 35.7

Col 0 Col 1 Col 2

Row 0 3.2 2.1 5.3

Row 1 8.0 4.9 5.7

Row 2 18.0 14.9 15.7

Row 3 28.0 24.9 25.7

Row 4 38.0 34.9 35.7

Col 0 Col 1 Col 2 Avg

Row 0 3.2 2.1 5.3 3.5

Row 1 8.0 4.9 5.7 6.2

Row 2 18.0 14.9 15.7 16.2

Row 3 28.0 24.9 25.7 26.2

Row 4 38.0 34.9 35.7 36.2

Avg 19.0 16.3 17.6

Enter top region boundary: 0

Enter bottom region boundary: 3

Enter left region boundary: 1

Enter right region boundary: 2

Region average: 12.4

**Submission:**

Place this Word document and the two program source files (problem3.c, problem4.c) in a folder and zip the folder. Submit the zip file in Canvas. Make sure that the zip file contains your source code and Word document before submitting.