WEEK 4 - ARRAYS

You are required to do the following:

1. Lab Questions – please do the lab questions during the lab session. When doing your lab questions, please follow exactly the question requirements on program input/output as our automated assessment system is based on test cases using exact string matching on program input/output.
2. Lab Assignment Questions – please do the assignment questions and submit your code to the online Automated Programming Assessment System (APAS) for grading.

Lab Questions

1. **(reverseAr1D)** Write a C function `printReverse()` that prints an array of integers in reverse order. For example, if `ar`[5] = `{1,2,3,4,5}`, then the output `5,4,3,2,1` will be printed after applying the function `printReverse()`. The function prototype is given as follows:

   ```
   void printReverse(int ar[], int size);
   ```

   where `size` indicates the size of the array.

   Write two versions of `printReverse()`. One version `printReverse1()` uses the index notation and the other version `printReverse2()` uses the pointer notation for accessing the element of each index location.

   In addition, Write a C function `reverseAr1D()` that takes in an array of integers **`ar`** and an integer *`size`* as parameters. The parameter *`size`* indicates the size of the array to be processed. The function converts the content in the array in reverse order and passes the array to the calling function via call by reference.

   ```
   void reverseAr1D(int ar[ ], int size);
   ```

   Write a C program to test the functions.

   Some sample input and output sessions are given below:

   (1) Test Case 1:

   ```
   Enter array size:
   5
   Enter 5 data:
   1 2 3 6 7
   printReverse1(): 7 6 3 2 1
   printReverse2(): 7 6 3 2 1
   reverseAr1D(): 7 6 3 2 1
   ```

   (2) Test Case 2:

   ```
   Enter array size:
   1
   Enter 1 data:
   5
   printReverse1(): 5
   printReverse2(): 5
   reverseAr1D(): 5
   ```

   (3) Test Case 3:

   ```
   Enter array size:
   ```

```
7
Enter 7 data:
1 2 3 4 5 6 7
printReverse1(): 7 6 5 4 3 2 1
printReverse2(): 7 6 5 4 3 2 1
reverseAr1D(): 7 6 5 4 3 2 1
```

(4) Test Case 4:

```
Enter array size:
2
Enter 2 data:
2 4
printReverse1(): 4 2
printReverse2(): 4 2
reverseAr1D(): 4 2
```

A sample template for the program is given below:

```
#include <stdio.h>
void printReverse1(int ar[], int size);
void printReverse2(int ar[], int size);
void reverseAr1D(int ar[], int size);
int main()
{
   int ar[10];
   int size, i;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i <= size-1; i++)
      scanf("%d", &ar[i]);
   printReverse1(ar, size);
   printReverse2(ar, size);
   reverseAr1D(ar, size);
   printf("reverseAr1D(): ");
   if (size > 0) {
      for (i=0; i<size; i++)
         printf("%d ", ar[i]);
   }
   return 0;
}
void printReverse1(int ar[], int size)
{
   /* using index - Write your program code here */
}
void printReverse2(int ar[], int size)
{
   /* using pointer - Write your program code here */
}
void reverseAr1D(int ar[ ], int size)
{
   /* Write your program code here */
}
```

2. (**findAr1D**) Write a function **findAr1D()** that returns the subscript of the **first appearance** of a target number in an array. For example, if ar = **{ 3,6,9,4,7,8 }**, then **findAr1D(6,ar,3)** will return 0 where 6 is the size of the array and 3 is the number to be found, and **findAr1D(6,ar,9)** will return 2. If the required number is not in the array, the function will return -1. The function prototype is given as follows:

```
int findAr1D(int size, int ar[ ], int target);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter array size:
5
Enter 5 data:
1 2 3 4 5
Enter the target number:
3
findAr1D(): 2
```

(2) Test Case 2:

```
Enter array size:
1
Enter 1 data:
5
Enter the target number:
5
findAr1D(): 0
```

(3) Test Case 3:

```
Enter array size:
7
Enter 7 data:
1 3 5 7 9 11 15
Enter the target number:
15
findAr1D(): 6
```

(4) Test Case 4:

```
Enter array size:
7
Enter 7 data:
1 3 5 7 9 11 15
Enter the target number:
2
findAr1D(): Not found
```

A sample program to test the functions is given below.

```
#include <stdio.h>
#define INIT_VALUE -1000
int findAr1D(int size, int ar[], int target);
int main()
{
   int ar[20];
   int size, i, target, result = INIT_VALUE;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i<=size-1; i++)
      scanf("%d", &ar[i]);
   printf("Enter the target number: \n");
   scanf("%d", &target);
   result = findAr1D(size, ar, target);
   if (result == -1)
      printf("findAr1D(): Not found\n");
   else
      printf("findAr1D(): %d", result);
   return 0;
```

```
}
int findAr1D(int size, int ar[], int target)
{
   /* Write your program code here */
}
```

3. **(swap2RowsCols2D)** Write the code for the following matrix functions:

```
void swap2Rows(int ar[][SIZE], int r1, int r2);
/* the function swaps the row r1 with the row r2 of a 2-dimensional array ar */

void swap2Cols(int ar[][SIZE], int c1, int c2);
/* the function swaps the column c1 with the column c2 of a 2-dimensional array ar */
```

Write a C program to test the above functions. In addition, your program should print the resultant matrix after each operation. You may assume that the input matrix is a 3x3 matrix when testing the functions.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Select one of the following options:
1: getInput()
2: swap2Rows()
3: swap2Cols()
4: display()
5: exit()
Enter your choice:
1
Enter the matrix (3x3):
5 10 15
15 20 25
25 30 35
Enter your choice:
2
Enter two rows for swapping:
1 2
The new array is:
5 10 15
25 30 35
15 20 25
Enter your choice:
5
```

(2) Test Case 2:

```
Select one of the following options:
1: getInput()
2: swap2Rows()
3: swap2Cols()
4: display()
5: exit()
Enter your choice:
1
Enter the matrix (3x3):
5 10 15
15 20 25
25 30 35
Enter your choice:
3
Enter two columns for swapping:
1 2
The new array is:
5 15 10
```

```
15 25 20
25 35 30
Enter your choice:
5
```

(3) Test Case 3:

```
Select one of the following options:
1: getInput()
2: swap2Rows()
3: swap2Cols()
4: display()
5: exit()
Enter your choice:
1
Enter the matrix (3x3):
1 2 3
4 5 6
7 8 9
Enter your choice:
2
Enter two rows for swapping:
0 2
The new array is:
7 8 9
4 5 6
1 2 3
Enter your choice:
3
Enter two columns for swapping:
0 2
The new array is:
9 8 7
6 5 4
3 2 1
Enter your choice:
5
```

(4) Test Case 4:

```
Select one of the following options:
1: getInput()
2: swap2Rows()
3: swap2Cols()
4: display()
5: exit()
Enter your choice:
1
Enter the matrix (3x3):
1 2 3
4 5 6
7 8 9
Enter your choice:
2
Enter two rows for swapping:
1 2
The new array is:
1 2 3
7 8 9
4 5 6
Enter your choice:
3
Enter two columns for swapping:
1 2
The new array is:
1 3 2
```

```
7 9 8
4 6 5
Enter your choice:
5
```

A sample program to test the functions is given below:

```
#include <stdio.h>
#define SIZE 3
void swap2Rows(int ar[][SIZE], int r1, int r2);
void swap2Cols(int ar[][SIZE], int c1, int c2);
void display(int ar[][SIZE]);
int main()
{
   int array[SIZE][SIZE];
   int row1, row2, col1, col2;
   int i,j;
   int choice;

   printf("Select one of the following options: \n");
   printf("1: getInput()\n");
   printf("2: swap2Rows()\n");
   printf("3: swap2Cols()\n");
   printf("4: display()\n");
   printf("5: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("Enter the matrix (3x3): \n");
            for (i=0; i<SIZE; i++)
               for (j=0; j<SIZE; j++)
                  scanf("%d", &array[i][j]);
            break;
         case 2:
            printf("Enter two rows for swapping: \n");
            scanf("%d %d", &row1, &row2);
            swap2Rows(array, row1, row2);
            printf("The new array is: \n");
            display(array);
            break;
         case 3:
            printf("Enter two columns for swapping: \n");
            scanf("%d %d", &col1, &col2);
            swap2Cols(array, col1, col2);
            printf("The new array is: \n");
            display(array);
            break;
         case 4:
            display(array);
            break;
      }
   } while (choice < 5);
   return 0;
}
void display(int ar[][SIZE])
{
   int l,m;
   for (l = 0; l < SIZE; l++) {
      for (m = 0; m < SIZE; m++)
         printf("%d ", ar[l][m]);
      printf("\n");
   }
```

```
}
void swap2Rows(int ar[][SIZE], int r1, int r2)
{
   /* Write your program code here */
}
void swap2Cols(int ar[][SIZE], int c1, int c2)
{
   /* Write your program code here */
}
```

4. **(reduceMatrix2D)** A square matrix (2-dimensional array of equal dimensions) can be reduced to upper-triangular form by setting each diagonal element to the sum of the original elements in that column and setting to 0s all the elements below the diagonal. For example, the 4-by-4 matrix:

```
4  3  8  6
9  0  6  5
5  1  2  4
9  8  3  7
```

would be reduced to

```
27  3  8  6
 0  9  6  5
 0  0  5  4
 0  0  0  7
```

Write a function `reduceMatrix2D()` to reduce a matrix with dimensions of *`rowSize`* and *`colSize`*. The prototype of the function is:

```
void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
reduceMatrix2D():
28 2 3 4
0 30 7 8
0 0 26 12
0 0 0 16
```

(2) Test Case 2:

```
Enter row size of the 2D array:
3
Enter column size of the 2D array:
3
Enter the matrix (3x3):
1 0 0
2 2 0
3 3 3
reduceMatrix2D():
6 0 0
0 5 0
```

```
0 0 3
```

(3) Test Case 3:

```
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
7 8 9 10
5 6 7 8
11 12 13 14
reduceMatrix2D():
24 2 3 4
0 26 9 10
0 0 20 8
0 0 0 14
```

(4) Test Case 4:

```
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
-5 -6 -7 -8
3 4 5 6
-1 -2 -3 -4
6 7 8 9
reduceMatrix2D():
3 -6 -7 -8
0 9 5 6
0 0 5 -4
0 0 0 9
```

A sample template for the program is given below:

```
#include <stdio.h>
#define SIZE 10
void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize);
void display(int ar[][SIZE], int rowSize, int colSize);
int main()
{
   int ar[SIZE][SIZE], rowSize, colSize;
   int i,j;

   printf("Enter row size of the 2D array: \n");
   scanf("%d", &rowSize);
   printf("Enter column size of the 2D array: \n");
   scanf("%d", &colSize);
   printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
   for (i=0; i<rowSize; i++)
      for (j=0; j<colSize; j++)
         scanf("%d", &ar[i][j]);
   reduceMatrix2D(ar, rowSize, colSize);
   printf("reduceMatrix2D(): \n");
   display(ar, rowSize, colSize);
   return 0;
}
void display(int ar[][SIZE], int rowSize, int colSize)
{
   int l,m;
   for (l = 0; l < rowSize; l++) {
      for (m = 0; m < colSize; m++)
         printf("%d ", ar[l][m]);
```

```
            printf("\n");
        }
    }
    void reduceMatrix2D(int ar[][SIZE], int rowSize, int colSize)
    {
        /* Write your program code here */
    }
```

Section B – Arrays

1. (**absoluteSum1D**) Write a C function `absoluteSum1D()` that returns the sum of the absolute values of the elements of a *`vector`* with the following prototype:

```
float absoluteSum1D(int size, float vector[]);
```

where size is the number of elements in the vector.

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter vector size:
5
Enter 5 data:
1.1 3 5 7 9
absoluteSum1D(): 25.10
```

(2) Test Case 2:

```
Enter vector size:
6
Enter 6 data:
1 -3 5 -7 9 -2
absoluteSum1D(): 27.00
```

A sample program to test the function is given below.

```
#include <stdio.h>
#include <math.h>
float absoluteSum1D(int size, float vector[]);
int main()
{
   float vector[10];
   int i, size;

   printf("Enter vector size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i<size; i++)
      scanf("%f", &vector[i]);
   printf("absoluteSum1D(): %.2f", absoluteSum1D(size, vector));
   return 0;
}
float absoluteSum1D(int size, float vector[])
{
   /* write your program code here */
}
```

2. (**findAverage2D**) Write a C function `findAverage2D()` that takes a 4x4 two-dimensional array of floating point numbers *`matrix`* as a parameter. The function computes the average of the first three elements of each row of the array and stores it at the last element of the row. The function prototype is given as follows:

```
void findAverage2D(float matrix[4][4]);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter data:
1 2 3 0
4 5 6 0
7 8 9 0
1 2 3 0
findAverage2D():
1.00 2.00 3.00 2.00
4.00 5.00 6.00 5.00
7.00 8.00 9.00 8.00
1.00 2.00 3.00 2.00
```

(2) Test Case 2:

```
Enter data:
1 2 3 0
4 5 6 0
4 5 6 0
1 2 3 0
findAverage2D():
1.00 2.00 3.00 2.00
4.00 5.00 6.00 5.00
4.00 5.00 6.00 5.00
1.00 2.00 3.00 2.00
```

A sample program to test the function is given below.

```
#include <stdio.h>
void findAverage2D(float matrix[4][4]);
int main()
{
   float ar[4][4];
   int i,j;

   printf("Enter data: \n");
   for (i = 0; i < 4; i++) {
      for (j = 0; j < 4; j++)
         scanf("%f", &ar[i][j]);
   }
   findAverage2D(ar);
   printf("findAverage2D(): :\n");
   for (i = 0; i < 4; i++) {
      for (j = 0; j < 4; j++)
         printf("%.2f ", ar[i][j]);
      printf("\n");
   }
   return 0;
}
void findAverage2D(float matrix[4][4])
{
   /* write your program code here */
}
```

3. (**findMinMax1D**) Write a C function `findMinMax1D()` that takes in an one-dimensional array of integers *ar* and *size* as parameters. The function finds the minimum and maximum numbers of the array. The function returns the minimum and maximum numbers through the pointer parameters *min* and *max* via call by reference. The function prototype is given as follows:

```
void findMinMax1D(int ar[], int size, int *min, int *max);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter array size:
5
Enter 5 data:
1 2 3 5 6
min = 1; max = 6
```

(2) Test Case 2:

```
Enter array size:
1
Enter 1 data:
1
min = 1; max = 1
```

A sample program to test the function is given below.

```
#include <stdio.h>
void findMinMax1D(int ar[], int size, int *min, int *max);
int main()
{
   int ar[40];
   int i, size;
   int min, max;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i<size; i++)
      scanf("%d", &ar[i]);
   findMinMax1D(ar, size, &min, &max);
   printf("min = %d; max = %d\n", min, max);
   return 0;
}
void findMinMax1D(int ar[], int size, int *min, int *max)
{
   /* Write your program code here */
}
```

4. (**findMinMax2D**) Write a C function `findMinMax2D()` that takes a 5x5 two-dimensional array of integers *ar* as a parameter. The function returns the minimum and maximum numbers of the array to the caller through the two pointer parameters *min* and *max* respectively. The function prototype is given as follows:

```
void findMinMax2D(int ar[SIZE][SIZE], int *min, int *max);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the matrix data (5x5):
1 2 3 4 5
2 3 4 5 6
4 5 6 7 8
5 4 23 1 2
1 2 3 4 5
min = 1
max = 23
```

(2) Test Case 2:

```
Enter the matrix data (5x5):
1 2 3 4 5
2 3 4 5 6
```

```
4 5 6 7 8
5 4 23 1 2
1 2 3 4 5
min = 1
max = 23
```

A sample program to test the function is given below:

```
#include <stdio.h>
#define SIZE 5
void findMinMax2D(int ar[SIZE][SIZE], int *min, int *max);
int main()
{
   int A[5][5];
   int i,j,min,max;

   printf("Enter the matrix data (%dx%d): \n", SIZE, SIZE);
   for (i=0; i<5; i++)
      for (j=0; j<5; j++)
         scanf("%d", &A[i][j]);
   findMinMax2D(A, &min, &max);
   printf("min = %d\nmax = %d", min, max);
   return 0;
}
void findMinMax2D(int ar[SIZE][SIZE], int *min, int *max)
{
   /* add your code here */
}
```

5. (**platform1D**) The number of consecutive array elements in an array that contains the same integer value forms a 'platform'. Write a C function `platform()` that takes in an array of integers *`ar`* and *`size`* as parameters, and returns the length of the maximum platform in *`ar`* to the calling function. The function prototype is given as follows:

```
int platform(int ar[], int size);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter array size:
5
Enter 5 data:
1 2 2 2 3
platform1D(): 3
```

(2) Test Case 2:

```
Enter array size:
10
Enter 10 data:
1 2 3 4 5 6 7 8 9 0
platform1D(): 1
```

A sample program to test the function is given below:

```
#include <stdio.h>
int platform1D(int ar[], int size);
int main()
{
   int i,b[50],size;
```

```
    printf("Enter array size: \n");
    scanf("%d", &size);
    printf("Enter %d data: \n", size);
    for (i=0; i<size; i++)
       scanf("%d",&b[i]);
    printf("platform1D(): %d\n", platform1D(b,size));
    return 0;
}
int platform1D(int ar[], int size)
{
    /* Write your program code here */
}
```

6. (**diagonals2D**) Write a C function `diagonals2D()` that accepts a two-dimensional array of integers *ar*, and the array sizes for the rows and columns as parameters, computes the sum of the elements of the two diagonals, and returns the sums to the calling function through the pointer parameters *sum1* and *sum2* using call by reference. For example, if the *rowSize* is 3, *colSize* is 3, and the array *ar* is {1,2,3, 1,1,1, 4,3,2}, then *sum1* is computed as 1+1+2=4, and *sum2* is 3+1+4=8. The function prototype is given as follows:

```
void diagonals2D(int ar[][SIZE], int rowSize, int colSize, int *sum1,
int *sum2);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter row size of the 2D array:
3
Enter column size of the 2D array:
3
Enter the matrix (3x3):
1 2 3
1 1 1
4 3 2
sum1=4; sum2=8
```

(2) Test Case 2:

```
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
1 1 2 2
2 2 1 1
5 4 3 2
sum1=5; sum2=13
```

A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 10
void diagonals2D(int ar[][SIZE], int rowSize, int colSize, int
*sum1, int *sum2);
int main()
{
    int ar[SIZE][SIZE], rowSize, colSize;
    int i, j, sum1=0, sum2=0;

    printf("Enter row size of the 2D array: \n");
```

```
    scanf("%d", &rowSize);
    printf("Enter column size of the 2D array: \n");
    scanf("%d", &colSize);
    printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
    for (i=0; i<rowSize; i++)
       for (j=0; j<colSize; j++)
          scanf("%d", &ar[i][j]);
    diagonals2D(ar, rowSize, colSize, &sum1, &sum2);
    printf("sum1=%d; sum2=%d\n",sum1,sum2);
}
void diagonals2D(int ar[][SIZE], int rowSize, int colSize, int
*sum1, int *sum2)
{
    /* Write your program code here */
}
```

7. (**transpose2D**) Write a function `transpose2D()` that takes a square matrix *ar*, and the array sizes for the rows and columns as parameters, and returns the transpose of the array via call by reference. For example, if the *rowSize* is 4, *colSize* is 4, and the array *ar* is {1,2,3,4, 1,1,2,2, 3,3,4,4, 4,5,6,7}, then the resultant array will be {1,1,3,4, 2,1,3,5, 3,2,4,6, 4,2,4,7}. The function prototype is given below:

```
void transpose2D(int ar[][SIZE], int rowSize, int colSize);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
1 1 2 2
3 3 4 4
4 5 6 7
transpose2D():
1 1 3 4
2 1 3 5
3 2 4 6
4 2 4 7
```

(2) Test Case 2:

```
Enter row size of the 2D array:
3
Enter column size of the 2D array:
3
Enter the matrix (3x3):
1 2 3
3 4 5
5 6 7
transpose2D():
1 3 5
2 4 6
3 5 7
```

A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 10
void transpose2D(int ar[][SIZE], int rowSize, int colSize);
void display(int ar[][SIZE], int rowSize, int colSize);
```

```
int main()
{
   int ar[SIZE][SIZE], rowSize, colSize;
   int i,j;

   printf("Enter row size of the 2D array: \n");
   scanf("%d", &rowSize);
   printf("Enter column size of the 2D array: \n");
   scanf("%d", &colSize);
   printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
   for (i=0; i<rowSize; i++)
      for (j=0; j<colSize; j++)
         scanf("%d", &ar[i][j]);
   printf("transpose2D(): \n");
   transpose2D(ar, rowSize, colSize);
   display(ar, rowSize, colSize);
   return 0;
}
void display(int ar[][SIZE], int rowSize, int colSize)
{
   int l,m;
   for (l = 0; l < rowSize; l++) {
      for (m = 0; m < colSize; m++)
         printf("%d ", ar[l][m]);
      printf("\n");
   }
}
void transpose2D(int ar[][SIZE], int rowSize, int colSize)
{
   /* Write your program code here */
}
```

8. (**minOfMax2D**) Write a C function `minOfMax2D()` that takes a two-dimensional array matrix of integers *ar*, and the array sizes for the rows and columns as parameters. The function returns the minimum of the maximum numbers of each row of the 2-dimensional array *ar*. For example, if the *rowSize* is 4, *colSize* is 4, and *ar* is {{1,3,5,2}, {2,4,6,8}, {8,6,4,9}, {7,4,3,2}}, then the maximum numbers will be 5, 8, 9 and 7 for rows 0, 1, 2 and 3 respectively, and the minimum of the maximum numbers will be 5. The prototype of the function is given as follows:

```
int minOfMax2D(int ar[][SIZE], int rowSize, int colSize);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter row size of the 2D array:
4
Enter column size of the 2D array:
4
Enter the matrix (4x4):
1 2 3 4
2 3 4 5
5 6 7 8
8 10 2 4
minOfMax2D(): 4
```

(2) Test Case 2:

```
Enter row size of the 2D array:
3
Enter column size of the 2D array:
3
Enter the matrix (3x3):
```

```
1 -3 3
-3 2 4
3 6 -8
minOfMax2D(): 3
```

A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 10
int minOfMax2D(int ar[][SIZE], int rowSize, int colSize);
int main()
{
   int ar[SIZE][SIZE], rowSize, colSize;
   int i,j,min;

   printf("Enter row size of the 2D array: \n");
   scanf("%d", &rowSize);
   printf("Enter column size of the 2D array: \n");
   scanf("%d", &colSize);
   printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
   for (i=0; i<rowSize; i++)
      for (j=0; j<colSize; j++)
         scanf("%d", &ar[i][j]);
   min=minOfMax2D(ar, rowSize, colSize);
   printf("minOfMax2D(): %d\n", min);
   return 0;
}
int minOfMax2D(int ar[][SIZE], int rowSize, int colSize)
{
   /* Write your program code here */
}
```