Section A - Arrays [Answer 1 Specified Qn from this Section]

1. (**findMinMax1D**) Write a C function `findMinMax1D()` that takes in an one-dimensional array of integers *`ar`* and array *`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 functions 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 */
}
```

2. (**reverseAr1D**) 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. For example, if the array *`ar`* is `{1,2,3,4,5}` and *`size`* is 5, then the array will become `{5,4,3,2,1}` after processing. The function prototype is given as follows:

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

where **`size`** indicates the size of the array.

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter array size:
1
Enter 1 data:
3
reverseAr1D(): 3
```

(2) Test Case 2:
```
Enter array size:
7
Enter 7 data:
1 2 3 6 7 9 8
reverseAr1D(): 8 9 7 6 3 2 1
```

A sample program to test the functions is given below.

```
#include <stdio.h>
void reverseAr1D(int ar[], int size);
int main()
{
   int ar[80], 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]);
   reverseAr1D(ar, size);
   printf("reverseAr1D(): ");
   if (size > 0) {
      for (i=0; i<size; i++)
         printf("%d ", ar[i]);
   }
   printf("\n");
   return 0;
}
void reverseAr1D(int ar[ ], int size)
{
   /* 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:
```
Enter the matrix (3x3):
5 10 15
15 20 25
```

```
25 30 35
Enter two rows for swapping:
1 2
The new array is:
5 10 15
25 30 35
15 20 25
Enter two columns for swapping:
1 2
The new array is:
5 15 10
25 35 30
15 25 20
```

(2) Test Case 2:

```
Enter the matrix (3x3):
1 2 3
4 5 6
7 8 9
Enter two rows for swapping:
0 2
The new array is:
7 8 9
4 5 6
1 2 3
Enter two columns for swapping:
0 2
The new array is:
9 8 7
6 5 4
3 2 1
```

A sample program to test the functions is given below:

```c
#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;

   printf("Enter the matrix (3x3): \n");
   for (i=0; i<SIZE; i++)
      for (j=0; j<SIZE; j++)
         scanf("%d", &array[i][j]);
   printf("Enter two rows for swapping: \n");
   scanf("%d %d", &row1, &row2);
   swap2Rows(array, row1, row2);
   printf("The new array is: \n");
   display(array);
   printf("Enter two columns for swapping: \n");
   scanf("%d %d", &col1, &col2);
   swap2Cols(array, col1, col2);
   printf("The new array is: \n");
   display(array);
   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. (**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 */
   }
```

5. (**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 */
}
```

6. (**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 */
}
```

7. **(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
reduceMatrix():
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
reducMatrix():
6 0 0
0 5 0
0 0 3
```

A sample template for the program is given below:

```c
#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 */
}
```