Section E – Arrays [Ans 2 Specified Qns from this Section]

Part A [Ans 1 specified Qn from this part]

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.

```c
#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. (**find2Max1D**) Write a C function `find2Max1D()` that takes an one-dimensional array of integer numbers *`ar`* and *`size`* (>1) as parameters. The function returns the largest and second largest numbers of the array to the calling function through the two pointer parameters *`max1`* and *`max2`* respectively. For example, if the array *`a`*`[]={1,7,8,6,9,4,5,2,3}`, then *`max1`* is 9, and *`max2`* is 8. The function prototype is given as follows:

```c
void find2Max1D(int ar[], int size, int *max1, int *max2);
```

Write a C program to test the function.

A sample input and output session is given below:

(1) Test Case 1:

```
Enter array size:
5
Enter 5 data:
1 2 3 5 6
Max1: 6
Max2: 5
```

(2) Test Case 2:

```
Enter array size:
6
Enter 6 data:
-4 0 -7 3 2 1
Max1: 3
Max2: 2
```

A sample program to test the function is given below.

```
#include <stdio.h>
void find2Max1D(int ar[], int size, int *max1, int *max2);
int main()
{
   int max1,max2;
   int ar[10],size,i;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i<size; i++)
      scanf("%d", &ar[i]);
   find2Max1D(ar,size,&max1,&max2);
   printf("Max1: %d\nMax2: %d\n",max1,max2);
   return 0;
}
void find2Max1D(int ar[], int size, int *max1, int *max2)
{
   /* 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. **(symmetry2D)** Write the C function `symmetry()` that takes in a two-dimensional array of integer numbers *M* and the array sizes for rows and columns as parameters, and returns 1 if *M* is symmetric or 0 otherwise. A two-dimensional matrix is symmetric iff it is equal to its transpose. It means that `M[i][j]` is equal to `M[j][i]` for `0<=i<=rowSize` and `0<=j<=colSize`. For example, if `rowSize` and `colSize` are 4, and *M* is `{{1,2,3,4},{2,2,5,6},{3,5,3,7},{4,6,7,4}}`, then *M* will be symmetric. The function prototype is given as follows:

```
int symmetry2D(int M[][SIZE], int rowSize, int colSize);
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the array size (rowSize, colSize):
4 4
Enter the matrix (4x4):
1 2 3 4
2 2 5 6
3 5 3 7
4 6 7 4
symmetry2D(): Yes
```

(2) Test Case 2:

```
Enter the array size (rowSize, colSize):
4 4
Enter the matrix (4x4):
1 2 3 4
2 2 5 6
3 5 3 7
```

```
5 6 7 4
symmetry2D(): No
```

(3) Test Case 3:

```
Enter the array size (rowSize, colSize):
3 3
Enter the matrix (3x3):
1 2 3
2 6 7
3 7 3
symmetry2D(): Yes
```

A sample program to test the function is given below:

```
#include <stdio.h>
#define SIZE 10
#define INIT_VALUE -1
int symmetry2D(int M[][SIZE], int rowSize, int colSize);
int main()
{
   int M[SIZE][SIZE],i,j, result = INIT_VALUE;
   int rowSize, colSize;

   printf("Enter the array size (rowSize, colSize): \n");
   scanf("%d %d", &rowSize, &colSize);
   printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
   for (i=0; i<rowSize; i++)
      for (j=0; j<colSize; j++)
         scanf("%d", &M[i][j]);
   result=symmetry2D(M, rowSize, colSize);
   if (result == 1)
      printf("symmetry2D(): Yes\n");
   else if (result == 0)
      printf("symmetry2D(): No\n");
   else
      printf("Error\n");
   return 0;
}
int symmetry2D(int M[][SIZE], int rowSize, int colSize)
{
   /* Write your code here */
}
```

Part B [Ans 1 specified Qn from this part]

6. **(swapMinMax1D)** Write the C function `swapMinMax1D()` that takes in an array of integers *`ar`* and *`size`* (>1) as parameters, finds the index positions of the largest number and smallest number in the array, swaps the index positions of these two numbers, and passes the array to the calling function via call by reference. For example, if *`ar`* is `{1,2,3,4,5}`, then the resultant array *`ar`* will be `{5,2,3,4,1}` after executing the function. If there are more than one largest or smallest number in the array, we will swap the first occurrence of the largest and smallest numbers. For example, if *`ar`* is `{5,2,1,1,8,9,9}`, then the resultant array ar will be `{5,2,9,1,8,1,9}` after executing the function. The function prototype is:

```
void swapMinMax1D(int ar[], int size);
```

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
swapMinMax1D(): 5 2 3 4 1
```

(2) Test Case 2:

```
Enter array size:
2
Enter 2 data:
5 2
swapMinMax1D(): 2 5
```

(3) Test Case 3:

```
Enter array size:
7
Enter 7 data:
5 2 1 1 8 9 9
swapMinMax1D(): 5 2 9 1 8 1 9
```

A sample program to test the function is given below:

```
#include <stdio.h>
void swapMinMax1D(int ar[], int size);
int main()
{
   int ar[50],i,size;

   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d data: \n", size);
   for (i=0; i<size; i++)
      scanf("%d",ar+i);
   swapMinMax1D(ar, size);
   printf("swapMinMax1D(): ");
   for (i=0; i<size; i++)
      printf("%d ",*(ar+i));
   return 0;
}
void swapMinMax1D(int ar[], int size)
{
  /* Write your code here */
}
```

7. (**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 */
}
```

8. (**arInsert1D**) Write the C code for the `arInsert()` function that inserts an integer into an array. The function prototype is given below:

```
int arInsert1D(int *size, int ar[], int num);
```

The function takes in three parameters. It inserts the number *num* into the array *ar* where **size* is the number of integers stored in *ar*. Before and after the function call, *ar* is an array of integers in ascending order. The program defines the constant *SIZE* as the maximum number of integers which can be stored in *ar*. This means that the function should issue an error message and no insertion should be done if **size* is equal to SIZE before insertion. The function will return 0 if the insertion is correct, or 1 if otherwise.

Write a C program to test the function. The program will first initialize the array with user input data before the insertion operation.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter size (max is 10):
5
Enter 5 data in ascending order:
1 3 5 7 9
Enter an integer to insert:
2
arInsert1D(): 1 2 3 5 7 9
```

(2) Test Case 2:

```
Enter size (max is 10):
10
Enter 10 data in ascending order:
1 2 3 4 5 6 7 8 9 10
Enter an integer to insert:
2
arInsert1D(): The array is full
```

(3) Test Case 3:

```
Enter size (max is 10):
```

```
0
Enter 0 data in ascending order:
Enter an integer to insert:
2
arInsert1D(): 2
```

A sample program to test the function is given below:

```
#include <stdio.h>
#define SIZE 10
#define INIT_VALUE -1
int arInsert1D(int *size, int ar[], int num);
void display(int size, int ar[]);
int main()
{
   int  ar[SIZE], size, i, num, result = INIT_VALUE;

   printf("Enter size (max is %d): \n", SIZE);
   scanf("%d", &size);
   printf("Enter %d data in ascending order: \n", size);
   for (i=0; i < size; i++)
      scanf("%d", &ar[i]);
   printf("Enter an integer to insert: \n");
   scanf("%d", &num);
   printf("arInsert1D(): ");
   result = arInsert1D(&size, ar, num);
   if (result == 0)
      display(size, ar);
   else if (result == 1)
      printf("The array is full\n");
   else
      printf("An error has occurred\n");
   return 0;
}
void display(int size, int ar[])
{
   int i;

   if (size==0)
      printf("Empty array\n");
   else
   {
      for (i = 0; i < size; i++)
         printf("%d ", ar[i]);
      printf("\n");
   }
}
int arInsert1D(int *size, int ar[], int num)
{
   /* Write your program code here */
}
```

9. (**arRemove1D**) Write the C code for the `arRemove()` function that removes an integer from an array. The function prototype is given below:

```
int arRemove1D(int *size, int ar[], int num);
```

The function takes in three parameters. It removes the **first appearance** of the number *num* from the array *ar* which has **size* numbers in it. Before and after the function call, *ar* is an array of integers in ascending order. If `*size` is equal to zero or *num* does not appear in *ar*, the function should issue an error message. The program defines the constant *SIZE* as the maximum number of integers which can be stored in *ar*. The function will return 0 if the removal operation is successful, 1 if the array is empty or 2 if the number does not exist in *ar*.

Write a C program to test the function. The program will first initialize the array with user input data before the removal operation.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter size (max is 10):
5
Enter 5 data in ascending order:
1 3 5 3 9
Enter the integer to remove:
3
arRemove1D(): 1 5 3 9
```

(2) Test Case 2:

```
Enter size (max is 10):
6
Enter 6 data in ascending order:
1 3 5 7 9 11
Enter the integer to remove:
2
arRemove1D(): Array is empty or number does not exist
```

(3) Test Case 3:

```
Enter size (max is 10):
0
Enter 0 data in ascending order:
Enter the integer to remove:
2
arRemove1D(): Array is empty or number does not exist
```

A sample program to test the function is given below:

```c
#include <stdio.h>
#define SIZE 10
#define INIT_VALUE -1
int arRemove1D(int *size, int ar[], int num);
void display(int size, int ar[]);
int main()
{
   int  ar[SIZE], size, i, num, result=INIT_VALUE;

   printf("Enter size (max is %d): \n", SIZE);
   scanf("%d", &size);
   printf("Enter %d data in ascending order: \n", size);
   for (i=0; i < size; i++)
      scanf("%d", &ar[i]);
   printf("Enter the integer to be removed: \n");
   scanf("%d", &num);
   result = arRemove1D(&size, ar, num);
   printf("arRemove1D(): ");
   if (result == 0)
      display(size, ar);
   else if (result == 1)
      printf("Array is empty\n");
   else if (result == 2)
      printf("The number does not exist\n");
   else
      printf("An error has occurred\n");

   return 0;
}
```

```
void display(int size, int ar[])
{
   int i;

   if (size==0)
      printf("Empty array\n");
   else
   {
      for (i = 0; i < size; i++)
         printf("%d ", ar[i]);
      printf("\n");
   }
}
int arRemove1D(int *size, int ar[], int num)
{
   /* Write your program code here */
}
```

10. (**compress2D**) Write a function `compress2D()` that takes as input a 2-dimensional array (10x10) of binary data, compresses each row of the array by replacing each run of 0s or 1s with a single 0 or 1 and the number of times it occurs, and prints on each line the result of compression. For example, the row with data 0011100011 may be compressed into 02130312. The prototype of the function is given as follows:

    ```
    void compress2D(int data[SIZE][SIZE]);
    ```

    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 (10x10):
    1 1 1 1 1 0 0 0 0 0
    0 0 0 0 0 1 1 1 1 1
    1 1 1 1 1 0 0 0 0 0
    0 0 0 0 0 1 1 1 1 1
    1 1 1 1 1 0 0 0 0 0
    0 0 0 0 0 1 1 1 1 1
    1 1 1 1 1 0 0 0 0 0
    0 0 0 0 0 1 1 1 1 1
    1 1 1 1 1 0 0 0 0 0
    0 0 0 0 0 1 1 1 1 1
    compress2D():
    1 5 0 5
    0 5 1 5
    1 5 0 5
    0 5 1 5
    1 5 0 5
    0 5 1 5
    1 5 0 5
    0 5 1 5
    1 5 0 5
    0 5 1 5
    ```

    (2) Test Case 2:

    ```
    Enter the matrix data (10x10):
    1 0 1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0 1 0
    ```

```
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
compress2D():
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1
```

A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 10
void compress2D(int data[SIZE][SIZE]);
int main()
{
   int data[SIZE][SIZE];
   int i,j;

   printf("Enter the matrix data (%dx%d): \n", SIZE, SIZE);
   for (i=0; i<SIZE; i++)
      for (j=0; j<SIZE; j++)
         scanf("%d", &data[i][j]);
   printf("compress2D(): ");
   compress2D(data);
   return 0;
}
void compress2D(int data[SIZE][SIZE])
{
   /* Write your program code here */
}
```