Practice Questions – Arrays

1. absoluteSum1D
2. findMinMax1D
3. specialNumbers1D
4. platform1D
5. findAverage2D
6. findMinMax2D
7. computeTotal2D
8. diagonals2D
9. symmetry2D
10. compress2D

Questions

1. (**absoluteSum1D**) Write a C function 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.

A sample program template is given below to test the function:

```
#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 */
}
```

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
```

2. (**findMinMax1D**) Write a C function 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);
```

A sample program template is given below to test the function:

```
#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 */
}
```

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
```

3. (**specialNumbers1D**) A special number is a 3 digit positive integer, in which the sum of the cubes of each digit is equal to the number. For example, the number 407 is a special number as 407 = 4×4×4 + 0×0×0 + 7×7×7. Write a C function that computes special numbers from 100 up to the a specified number. The function takes an array of integers `ar`, `num` and `size` as parameters. The parameter `num` is the specified integer number. The parameter `size` is declared as a pointer which will return the size of the array storing the special numbers to the caller. The prototype of the function is given as follows:

```
void specialNumbers1D(int ar[], int num, int *size);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
void specialNumbers1D(int ar[], int num, int *size);
int main()
{
   int a[20],i,size=0,num;

   printf("Enter a number (between 100 and 999): \n");
   scanf("%d", &num);
   specialNumbers1D(a, num, &size);
   printf("specialNumbers1D(): ");
   for (i=0; i<size; i++)
      printf("%d ",a[i]);
   return 0;
}
void specialNumbers1D(int ar[], int num, int *size)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
```
Enter a number (between 100 and 999):
300
specialNumbers1D(): 153
```

(2) Test Case 2:
```
Enter a number (between 100 and 999):
400
specialNumbers1D(): 153 370 371
```

(3) Test Case 3:
```
Enter a number (between 100 and 999):
500
specialNumbers1D(): 153 370 371 407
```

(4) Test Case 4:
```
Enter a number (between 100 and 999):
999
specialNumbers1D(): 153 370 371 407
```

4. (**platform1D**) The number of consecutive array elements in an array that contains the same integer value forms a 'platform'. Write a C function 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 platform1D(int ar[], int size);
```

A sample program template is given below to test the function:

```
#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 */
}
```

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
```

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

5. (**findAverage2D**) Write a C function 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]);
```

A sample program template is given below to test the function:

```
#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 */
}
```

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
```

6. (**findMinMax2D**) Write a C function 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);
```

A sample program template is given below to test the function:

```
#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 */
}
```

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 27
1 2 3 4 5
min = -3
max = 27
```

7. (**computeTotal2D**) Write a C function that takes a 4x4 two-dimensional array matrix of integer numbers as a parameter. The function computes the total 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 computeTotal2D(int matrix[SIZE][SIZE]);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
#define SIZE 4
void computeTotal2D(int matrix[SIZE][SIZE]);
int main()
{
   int a[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", &a[i][j]);
   printf("computeTotal2D(): \n");
   computeTotal2D(a);
   for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++)
         printf("%d ", a[i][j]);
      printf("\n");
   }
   return 0;
}
void computeTotal2D(int matrix[SIZE][SIZE])
{
   /* write your code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1

```
Enter the matrix data (4x4):
1 2 3 4
2 3 4 5
3 4 5 6
6 7 8 9
computeTotal2D():
1 2 3 6
2 3 4 9
3 4 5 12
6 7 8 21
```

(2) Test Case 2

```
Enter the matrix data (4x4):
1 2 3 -4
2 3 -4 5
3 -4 5 6
-6 7 8 9
```

```
computeTotal2D():
1 2 3 6
2 3 -4 1
3 -4 5 4
-6 7 8 9
```

(3) Test Case 3

```
Enter the matrix data (4x4):
1 -2 3 4
2 -3 4 5
3 -4 5 6
6 -7 8 9
computeTotal2D():
1 -2 3 2
2 -3 4 3
3 -4 5 4
6 -7 8 7
```

8. (**transpose2D**) Write a function 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);
```

A sample program template is given below to test the function:

```
#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 */
}
```

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
```

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

9. **(symmetry2D)** Write the C function that takes in a square 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 square 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);
```

A sample program template is gven below to test the function:

```
#include <stdio.h>
#define SIZE 10
```

```
#define INIT_VALUE 999
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 */
}
```

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
```

(4) Test Case 4:

```
Enter the array size (rowSize, colSize):
5 5
Enter the matrix (5x5):
1 2 3 4 5
2 2 5 6 7
```

```
3 5 3 7 8
4 6 7 4 5
5 7 8 5 5
symmetry2D(): Yes
```

10. (**compress2D**) Write a function that takes as input a square 2-dimensional array 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], int rowSize, int colSize);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
#define SIZE 100
void compress2D(int data[SIZE][SIZE], int rowSize, int colSize);
int main()
{
   int data[SIZE][SIZE];
   int i,j;
   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", &data[i][j]);
   printf("compress2D(): \n");
   compress2D(data, rowSize, colSize);
   return 0;
}
void compress2D(int data[SIZE][SIZE], int rowSize, int colSize)
{
   /* Write your program code here */
}
```

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 1 1 0
0 0 1 1
1 1 1 1
0 0 0 0
compress2D():
1 3 0 1
0 2 1 2
1 4
0 4
```

(2) Test Case 2:
```
Enter the array size (rowSize, colSize):
5 5
Enter the matrix (5x5):
1 1 1 0 0
0 0 1 1 1
1 1 1 1 1
0 0 0 0 0
1 1 1 1 1
```

```
compress2D():
1 3 0 2
0 2 1 3
1 5
0 5
1 5
```

(3) Test Case 3:

```
Enter the array size (rowSize, colSize):
5 5
Enter the matrix (5x5):
0 0 0 0 0
1 1 1 1 1
1 1 1 1 1
0 0 0 0 0
1 1 1 1 1
compress2D():
0 5
1 5
1 5
0 5
1 5
```

(4) Test Case 4:

```
Enter the array size (rowSize, colSize):
10 10
Enter the matrix (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
```