ARRAYS

First, create a sub-directory, *Arrays*, under your home directory. All the programming for this lab experiment will be done in this directory. There is no submission for this lab.

Lab Tutor: For this lab-tutorial session, there are 4 questions. Each question will take about 25 minutes. You may have a discussion with the students after 20 minutes of each question. No need to cover the practice questions in this lab session.

Lab Questions

1. **(printReverse)** 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 index notation and the other version (printReverse2) uses pointer notation for accessing the element of each index location.

   In addition, write another C function **reverseAr()** that takes in an array of integers **ar** and a parameter **size** that 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 reverseAr(int ar[ ], int size);

   Write a C program to test the functions.

   A sample template for the program is given below:

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

   size = readArray(ar);
   printReverse1(ar, size);
   printReverse2(ar, size);
   reverseAr(ar, size);
   printf("reverseAr(): ");
   if (size > 0) {
      for (i=0; i<size; i++)
         printf("%d ", ar[i]);
   }
   return 0;
}
int readArray(int ar[])
{
   int i, size;

   printf("Enter array size: ");
   scanf("%d", &size);
   printf("Enter %d array: ", size);
   for (i=0; i <= size-1; i++)
      scanf("%d", &ar[i]);
```

```
   return size;
}
void printReverse1(int ar[], int size)
{
    /* using index – write your code here */
}
void printReverse2(int ar[], int size)
{
   /* using pointer - write your code here */
}
void reverseAr(int ar[ ], int size)
{
   /* Write your code here */
}
```

Important: Remember to name the source code of this program as **Q1.c**.

Some sample input and output sessions are given below:

(1)

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

(2)

```
Enter array size: 1
Enter 1 array: 1
printReverse1(): 1
printReverse2(): 1
reverseAr(): 1
```

2. **(swap2RowsCols)** Write the code for the following functions:

```
void  swap2Rows(int  M[SIZE][SIZE],  int r1, int r2);
    /* the function swaps the row r1 with the row r2 */

void  swap2Cols(int  M[SIZE][SIZE],  int c1, int c2);
    /* the function swaps the column c1 with the column c2 */
```

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.

A sample program to test the functions is given below:

```
#include <stdio.h>
#define SIZE 3
   void swap2Rows(int ar[SIZE][SIZE], int r1, int r2);
   void swap2Cols(int ar[SIZE][SIZE], int c1, int c2);
   void display(int ar[SIZE][SIZE]);

   int main()
   {
      int array[SIZE][SIZE];
      int row1, row2, col1, col2;
      int i,j;

      printf("Enter the matrix (3x3) row by row: \n");
      for (i=0; i<SIZE; i++)
         for (j=0; j<SIZE; j++)
            scanf("%d", &array[i][j]);

      printf("The array is: \n");
      display(array);

      printf("Enter two rows for swapping: ");
      scanf("%d %d", &row1, &row2);
      swap2Rows(array, row1, row2);
```

```
        printf("The new array is: \n");
        display(array);

        printf("Enter two columns for swapping: ");
        scanf("%d %d", &col1, &col2);
        swap2Cols(array, col1, col2);
        printf("The new array is: \n");
        display(array);
        return 0;
    }
    void display(int M[SIZE][SIZE])
    {
        int l,m;
        for (l = 0; l < 3; l++) {
            for (m = 0; m < 3; m++)
                printf("%5d", M[l][m]);
            printf("\n");
        }
    }
    void  swap2Rows(int M[SIZE][SIZE], int r1, int r2)
    /* swaps row r1 with row r2 */
    {
        /* Write your code here */
    }
    void  swap2Cols(int M[SIZE][SIZE], int c1, int c2)
    /* swaps column c1 with column  c2 */
    {
        /* Write your code here */
    }
```

Important: Remember to name the source code of this program as **Q2.c**.

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (3x3) row by row:
5 10 15
15 20 25
25 30 35
The array is:
    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)

```
Enter the matrix (3x3) row by row:
1 2 3
4 5 6
7 8 9
The array is:
    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
```

3. **(reduceMatrix)** 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 thus be reduced to

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

Write a function to reduce a 4-by-4 matrix. The prototype of the function is:

```
void reduceMatrix(int matrix[4][4]);
```

Write a C program to test the function.

A sample template for the program is given below:

```
#include <stdio.h>
void readMatrix(int M[4][4]);
void reduceMatrix (int matrix[4][4]);
void display(int M[4][4]);
int main()
{
   int A[4][4];
   readMatrix(A);
   reduceMatrix(A);
   printf("reduceMatrix(): \n");
   display(A);
   return 0;
}
void readMatrix(int M[4][4])
{
   int i, j;
   printf("Enter the matrix (4x4) row by row: \n");
   for (i=0; i<4; i++)
      for (j=0; j<4; j++)
         scanf("%d", &M[i][j]);
}
void display(int  M[4][4])
{
   int l,m;

   for (l = 0; l < 4; l++) {
      for (m = 0; m < 4; m++)
            printf("%5d", M[l][m]);
        printf("\n");
   }
}
void reduceMatrix(int matrix[4][4])
{
   /* add your code here */
}
```

Important: Remember to name the source code of this program as **Q3.c**.

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (4x4) row by row:
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)

```
Enter the matrix (4x4) row by row:
1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4
reducMatrix():
    1    0    0    0
    0    2    0    0
    0    0    3    0
    0    0    0    4
```

4. Explain how the addition of 1 to every element of the two dimensional array 'array' is done in the following program. What if the for statement at 'line a' is replaced by this statement:

```
add1(array[0], 3 * 4);
```

```
#include  <stdio.h>

void  add1(int ar[], int size);

int main()
{
        int  array[3][4];
        int h,k;

        for (h = 0; h < 3; h++)
                for (k = 0; k < 4; k++)
                        scanf("%d", &array[h][k]);

        for (h = 0; h < 3; h++)                                  /* line a */
                add1(array[h], 4);

        for (h = 0; h < 3; h++) {
                for (k = 0; k < 4; k++)
                        printf("%10d", array[h][k]);
                putchar('\n');
        }
        return 0;
}
void add1(int ar[], int size)
{
        int k;

        for (k = 0; k < size; k++)
                ar[k]++;
}
```

Practice Questions

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

   float aSum(int size, float vector[ ]);

   where **`size`** is the number of elements in the vector.

   Write a C program to test the function.

   A sample program to test the functions is given below.

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

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

   Some sample input and output sessions are given below:

(1)
```
Enter vector size: 5
Enter 5 data: 1.1 3 5 7 9
absoluteSum(): 25.10
```
(2)
```
Enter vector size: 5
Enter 5 data: 1 -3 5 -7 9
absoluteSum(): 25.00
```

2. (**findAr**) Write a function **`findAr()`** 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 **`findAr(6,ar,3)`** will return 0 where 6 is the size of the array and 3 is the number to be found, and **`findAr(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 findAr(int size, int ar[ ], int target);

   Write a C program to test the function.

   A sample program to test the functions is given below.

```
#include <stdio.h>
int findAr(int size, int ar[], int target);
int main()
{
   int ar[20];
   int size, i, target;
```

```
    printf("Enter array size: ");
    scanf("%d", &size);
    printf("Enter %d data: ", size);
    for (i=0; i<=size-1; i++)
       scanf("%d", &ar[i]);
    printf("Enter the target number: ");
    scanf("%d", &target);
    printf("findAr(): %d",
       findAr(size, ar, target));
    return 0;
}
int findAr(int size, int ar[], int target)
{
    /* write your code here */
}
```

Some sample input and output sessions are given below:

(1)

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

(2)

```
Enter array size: 7
Enter 7 data: 1 3 5 7 9 11 15
Enter the target number: 2
findAr(): -1
```

3. (**findMinMax**) Write a C function findMinMax() that takes an one-dimensional array of integer numbers as a parameter. 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. The function prototype is given as follows:

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

    Write a C program to test the function.

    A sample program to test the functions is given below.

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

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

Some sample input and output sessions are given below:

(1)

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

(2)

```
Enter size of vector: 1
Enter 1 data: 1
min = 1; max = 1
```

4. (**computeDiagonals**) Write a C function **computeDiagonals()** that accepts a two-dimensional array of integers (3x3) as parameter, computes the sum of the elements of the two diagonals, and returns the sums to the calling function through the parameters sum1 and sum2. For example, if the array a[3][3]={1,2,3,4,5,6,7,8,9}, then sum1 is 1+5+9=15, and sum2 is 3+5+7=15. The function prototype is given as follows:

   void computeDiagonals(int a[][3], int *sum1, int *sum2);

   Write a C program to test the function.

   A sample program to test the function is given below.

```
#include <stdio.h>
void computeDiagonals(int a[][3], int *sum1, int *sum2);
int main()
{
   int a[3][3];
   int i, j, sum1=0, sum2=0;
   printf("Enter the matrix (3x3) row by row: \n");
   for (i=0; i<3; i++)
      for (j=0; j<3; j++)
         scanf("%d", &a[i][j]);
   computeDiagonals(a, &sum1, &sum2);
   printf("sum1=%d; sum2=%d\n",sum1,sum2);
}
void computeDiagonals(int a[][3], int *sum1, int *sum2)
{
   /* Write your code here */
}
```

   Some sample input and output sessions are given below:

(1)
```
Enter the matrix (3x3) row by row:
1 2 3
4 5 6
7 8 9
sum1=15; sum2=15
```
(2)
```
Enter the matrix (3x3) row by row:
1 0 0
0 1 0
0 0 1
sum1=3; sum2=1
```

5. (**transpose**) Write a function **`transpose()`** that transposes a square matrix **`M`**. The function prototype is given below:

   void transpose(int M[SIZE][SIZE);

   Write a C program to test the function. 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 function.

   A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 3
void transpose(int M[SIZE][SIZE]);
void display(int M[SIZE][SIZE]);
int main()
{
   int ar[SIZE][SIZE];
```

```
   int i,j;

   printf("Enter the matrix (3x3) row by row: \n");
   for (i=0; i<SIZE; i++)
      for (j=0; j<SIZE; j++)
         scanf("%d", &ar[i][j]);
   printf("transpose():\n");
   transpose(ar);
   display(ar);
   return 0;
}
void display(int M[SIZE][SIZE])
{
   int l,m;
   for (l = 0; l < 3; l++) {
      for (m = 0; m < 3; m++)
         printf("%5d", M[l][m]);
      printf("\n");
   }
}
void transpose(int M[SIZE][SIZE])
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (3x3) row by row:
5 10 15
15 20 25
25 30 35
transpose():
    5   15   25
   10   20   30
   15   25   35
```

(2)

```
Enter the matrix (3x3) row by row:
-5 -6 -7
3 4 5
-1 -2 -3
transpose():
   -5    3   -1
   -6    4   -2
   -7    5   -3
```

6. (**minOfMax**) Write a C function **minOfMax()** that takes a 4x4 two-dimensional array matrix of integer numbers as a parameter. The function returns the minimum of the maximum numbers of each row of a 2-dimensional array `a`. For example, if `a` 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 minOfMax(int ar[4][4]);

   Write a C program to test the function.

   A sample program to test the function is given below.

```
#include <stdio.h>
int minOfMax(int ar[4][4]);
int main() {
   int a[4][4],row,col,min;

   printf("Enter the matrix (4x4) row by row: \n");
   for (row=0; row<4; row++)
      for (col=0; col<4; col++)
         scanf("%d", &a[row][col]);
   min=minOfMax(a);
```

```
    printf("minOfMax(): %d\n", min);
    return 0;
}
int minOfMax(int ar[4][4])
{
    /* write your code here */
}
```

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (4x4) row by row:
1 2 3 4
2 3 4 5
5 6 7 8
8 10 2 4
minOfMax(): 4
```

(2)

```
Enter the matrix (4x4) row by row:
1 -1 3 6
-3 2 4 6
3 6 -8 9
-3 -2 -1 -6
minOfMax(): -1
```

7. (**compress**) Write a function **compress()** 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 compress(int data[10][10]);

    Write a C program to test the function.

    A sample program to test the function is given below.

```
#include <stdio.h>
#define SIZE 10
void compress(int data[SIZE][SIZE]);
int main()
{
    int data[SIZE][SIZE];
    int i,j;
    printf("Enter the matrix (10x10) row by row: \n");
    for (i=0; i<SIZE; i++)
        for (j=0; j<SIZE; j++)
            scanf("%d", &data[i][j]);
    compress(data);
    return 0;
}
void compress(int data[SIZE][SIZE])
{
    /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1)

```
Enter the matrix (10x10) row by row:
1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1
1 1 0 0 1 1 0 0 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
```

```
The compression output:
1 5 0 5
0 5 1 5
1 2 0 2 1 2 0 2 1 2
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
```

(2)

```
Enter the matrix (10x10) row by row:
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0
The compression output:
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1
```