# Tutorial 3 (Week 4)
# 1D Arrays and 2D Arrays

# Q1 (histogram)

Write a program which will draw the histogram for **n integers** from **0 to 99**. n is input by the user. Each of the n numbers will be generated by calling **rand() % 100**. The program will consist of two functions:

(i) to collect the frequency distribution of the numbers

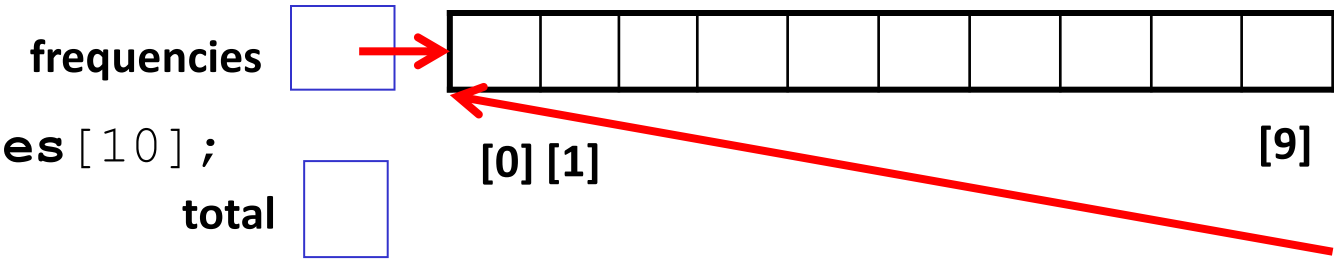
(ii) to print the histogram.

An example histogram is shown here.

```
0 – 9      |*********************
10 – 19    |************
20 – 29    |*************
30 – 39    |**
………
90 – 99    |***************
```

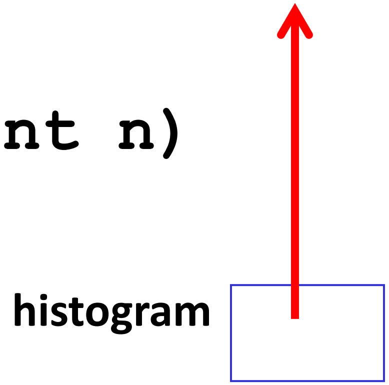
# Q1 (histogram)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void getFrequency(int histogram[10], int n);
void printFrequency(int histogram[10]);
int main()
{
   int frequencies[10];
   int total;

   printf("Please input the number of random numbers: ");
   scanf("%d", &total);
   srand(time(NULL));    // generate a seed number

   getFrequency(frequencies, total);
   printFrequency(frequencies);

   return 0;
}
```



# Q1 (histogram)

```
void getFrequency(int histogram[10],int n)
{
    int count;
    // int category;

    for (count = 0; count < 10; count++)
        histogram[count] = 0;

    for (count = 0; count < n; count++)
    {
        histogram[(rand() % 100)/10]++;
        /* OR category = (rand() % 100)/10;
              histogram[category]++; */
    }
}
```



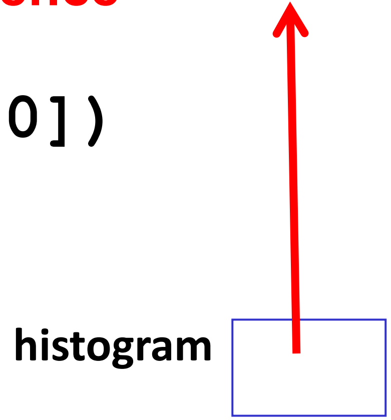
histogram

**passing array between functions using call by reference**

Note that the **'/' operator** will divide the data (0-99) into **10 categories/groups** [i.e. 0 to 9]. Each category will form an index for the array.

# Q1 (histogram)

passing array between functions using call by reference



```
void printFrequency(int histogram[10])
{
    int count, index;

    for (count = 0; count < 10; count++)
    {
       printf("%2d--%2d  |", count*10, count*10+9);

       for (index = 0; index < histogram[count]; index++ )
          putchar('*');

       putchar('\n');
    }
}
```

```
0 – 9     |*********************
10 – 19   |************
20 – 29   |*************
30 – 39   |**
.........
90 – 99   |***************
```

# Q2 (transpose)

Write a function that takes a **square matrix** **ar**, and the array **size**s 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}. That is, for the 4-by-4 matrix:

*1 2 3 4*
*5 1 2 2*
*6 3 4 4*
*7 5 6 7*

the resultant array after performing the transpose2D function is:

1 5 6 7
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);
```

**SIZE** is a constant defined at the beginning of the program.
For example, #define SIZE 10.

The parameters **rowSize** and **colSize** are used to **specify the dimensions** of the 2-dimensional array (e.g. 4x4) that the function should process.

Write a program to test the function.

# Q2 (transpose2D)

```c
#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++)      // read data from user
      for (j=0; j<colSize; j++)
         scanf("%d", &ar[i][j]);

   printf("transpose2D(): \n");
   transpose2D(ar, rowSize, colSize);
   display(ar, rowSize, colSize);
   return 0;
}
```

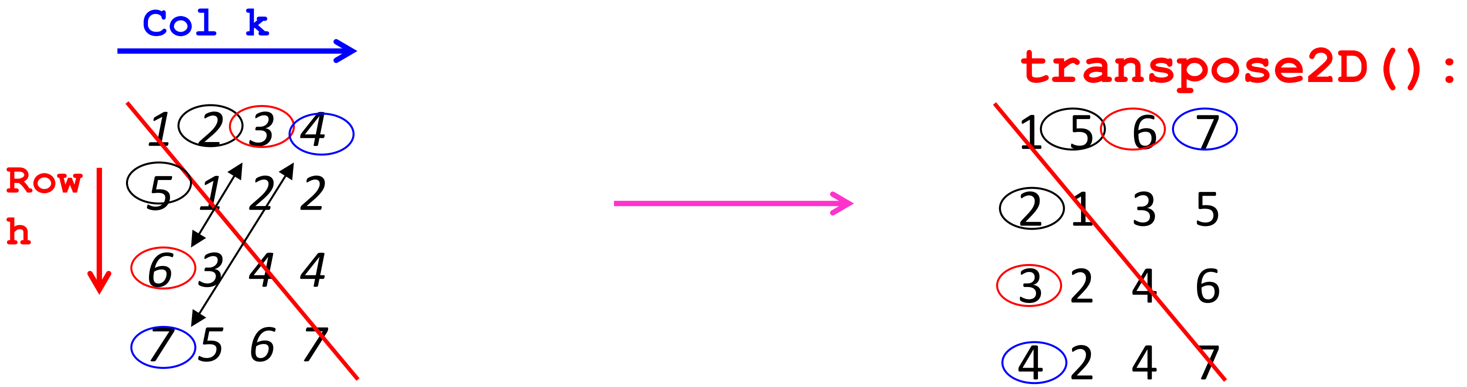
# Q2 (transpose2D)

```
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)
{
    int h, k;
    int temp;
    for (h = 1; h < rowSize; h++){    // traverse row
        for (k = 0; k < h; k++) {     // process column
            temp = ar[h][k];          // swap operation
            ar[h][k] = ar[k][h];
            ar[k][h] = temp;
        }
    }
}
```

Note: For Transpose - **swapping** of ar[row][column] with ar[column][row]

e.g. ar[1][0] will be swapped with ar[0][1]



$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 1 & 2 & 2 \\ 6 & 3 & 4 & 4 \\ 7 & 5 & 6 & 7 \end{array} \longrightarrow \begin{array}{cccc} 1 & 5 & 6 & 7 \\ 2 & 1 & 3 & 5 \\ 3 & 2 & 4 & 6 \\ 4 & 2 & 4 & 7 \end{array}$$

# Q3 (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 **0**s all the elements below the diagonal.

For example, the 4-by-4 matrix:

$$\begin{matrix} 4 & 3 & 8 & 6 \\ 9 & 0 & 6 & 5 \\ 5 & 1 & 2 & 4 \\ 9 & 8 & 3 & 7 \end{matrix}$$

would be reduced to

$$\begin{matrix} 27 & 3 & 8 & 6 \\ 0 & 9 & 6 & 5 \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 0 & 7 \end{matrix}$$

Write a function to reduce a matrix with dimensions of *rowSize* and *colSize*.

The prototype of the function is:

```
void reduceMatrix(int matrix[][SIZE], int rowSize, int colSize);
```

**SIZE** is a constant defined at the beginning of the program. For example, #define SIZE 10. The parameters **rowSize** and **colSize** are used to specify the dimensions of the 2-dimensional array (e.g. 4x4) that the function should process.

Write a program to test the function.

# Q3 (reduceMatrix2D)

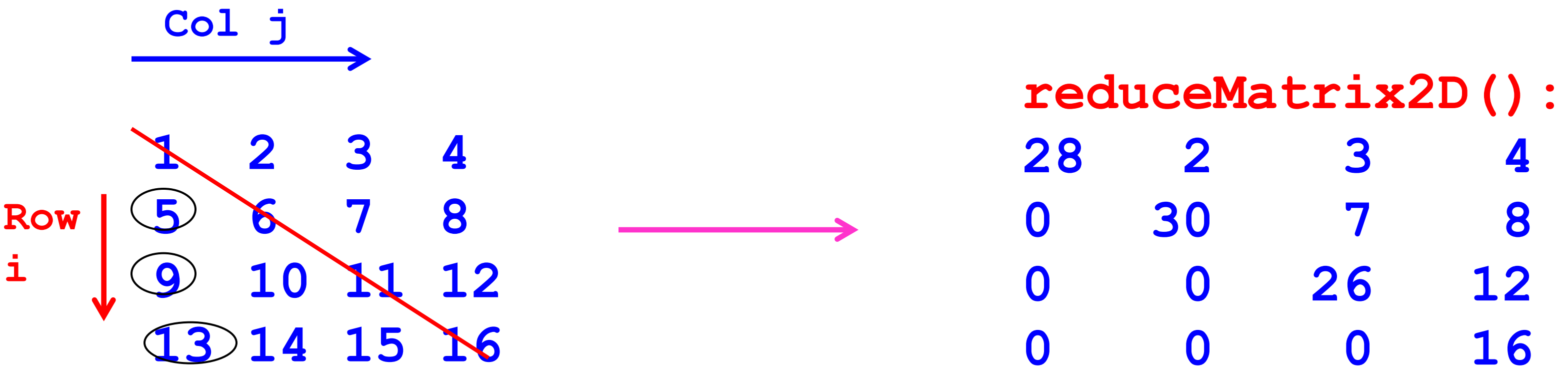
```c
#include <stdio.h>
#define SIZE 10
void reduceMatrix2D(int ar[][SIZE],int rowSize,int colSize)
void display(int ar[][SIZE], int rowSize, int colSize);
int main()
{
   int ar[SIZE][SIZE], rowSize, colSize;
   int i,j;
   printf("Enter row size of the 2D array: \n");
   scanf("%d", &rowSize);
   printf("Enter 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++)      // read data from user
      for (j=0; j<colSize; j++)
         scanf("%d", &ar[i][j]);

   reduceMatrix2D(ar, rowSize, colSize);
   printf("reduceMatrix2D(): \n");
   display(ar, rowSize, colSize);
   return 0;
}
```

# Q3 (reduceMatrix2D)

```
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 matrix[][SIZE], int rowSize, int
colSize)
{
   int i, j, sum;          // i for row, j for column

   for (j = 0; j < colSize; j++){  // traverse column
      sum = 0;
      for (i = j+1; i < rowSize; i++){  // process row
         sum += matrix[i][j];
         matrix[i][j] = 0;
      }
      matrix[j][j] += sum;
   }
}
```

Enter the matrix (4x4):



```
Col j

      1   2   3   4
Row   5   6   7   8
i     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
```

# Q4 (a)

(a) What is the output of the program if the addition of 1 to every element of the two-dimensional array ‘array’ is done in the following program?

```
#include <stdio.h>
void add1(int ar[], int size);
int main()
{
   int array[3][4]={1,2,3,4,  5,6,7,8,  9,10,11,12};
   int 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]++;
}
```

# Q4 (a) – Suggested Answer

```
#include <stdio.h>
void add1(int ar[], int size);
int main()
{
    int array[3][4] = {1,2,3,4, 5,6,7,8, 9,10,11,12};
    int 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]++;
}
```

array[0] array[1] array[2]

ar

**Output:**

```
2      3      4      5
6      7      8      9
10     11     12     13
```

# Q4 (a) – Suggested Answer

(a) What is the output of the program if the addition of 1 to every element of the two-dimensional array ‘array’ is done in the following program?

**Answer:**

- The function **add1()** has **two parameters:**
  - The first one is an **array address**.
  - The second one is the **size** of the array.
  - So the function **adds 1** to every element of the one-dimensional array.
- When the function is called in the ***for*** statement at line a by

  **add1(array[h], 4);**

  - **array[h]** is an one-dimensional array of 4 integers.
  - It is the **(h+1)th row** of the two-dimensional array ‘array’.
  - In fact, **array[h]** is the **address** of the **first element** of the (h+1)th row.
  - **So every function call works on one row of the two-dimensional array**.

# Q4 (b)

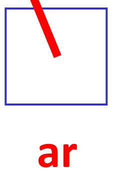
(b) What if the for statement at ‘line a’ is replaced by this statement: **add1(array[0], 3 * 4);**

```
#include <stdio.h>
void add1(int ar[], int size);
int main(){
   int array[3][4]={1,2,3,4,  5,6,7,8,  9,10,11,12};
   int 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]++;
}
```

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

# Q4 (b) – Suggested Answer

```
#include <stdio.h>
void add1(int ar[], int size);
int main()
{
                                   array[0]
    int array[3][4] =  {1,2,3,4,  5,6,7,8,  9,10,11,12};
    int h,k;

     add1( array[0], 3 * 4);  /*line a*/

    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]++;
}
```



**Output:**

```
2       3       4       5
6       7       8       9
10      11      12      13
```

# Q4 (b) – Suggested Answer

(b) What if the for statement at ‘line a’ is replaced by this statement: add1(array[0], 3 * 4);

**Answer:**

- When the **for** statement at **line a** is replaced by **add1(array[0], 3*4)**, it is passing the **address of the first element of the first row** to add1() and telling the function that the **array size is 12**.
- So **add1()** works on a one dimensional array starting at **array[0]** and with **12** elements.