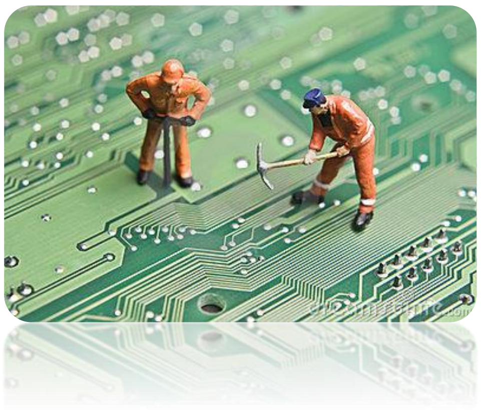
CS 100 Lab 12

# Computational Problem Solving (CS100)

*Shafay Shamail, Arif Zaman and*

*Safee Ullah Chaudhary*



# Lab 12 Objectives:

- 2D- Arrays

## Lab Guidelines

1. Make sure you get your work graded before the lab time ends.
2. You put all your work onto the LMS folder designated for the lab (i.e. “Lab12”) before the time the lab ends.
3. Talking to each other is NOT permitted. If you have a question, ask the lab assistants.
4. If you are a hot-shot C++ expert, you are still not allowed to use any feature of C++ that has not been covered in class or in the lab.
5. The object is not simply to get the job done, but to get it done in the way that is asked for in the lab.

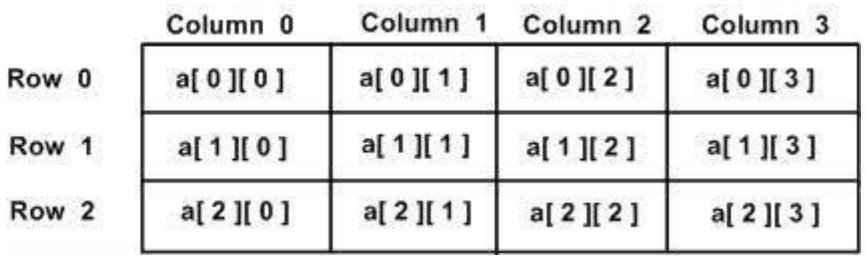
## Two-Dimensional Arrays

A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows:

```
Type array_name[x][y];
```

Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.

A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array **a**, which contains three rows and four columns can be shown as below:



| | Column 0 | Column 1 | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | a[ 0 ][ 0 ] | a[ 0 ][ 1 ] | a[ 0 ][ 2 ] | a[ 0 ][ 3 ] |
| Row 1 | a[ 1 ][ 0 ] | a[ 1 ][ 1 ] | a[ 1 ][ 2 ] | a[ 1 ][ 3 ] |
| Row 2 | a[ 2 ][ 0 ] | a[ 2 ][ 1 ] | a[ 2 ][ 2 ] | a[ 2 ][ 3 ] |

Thus, every element in array a is identified by an element name of the form **a[ i ][ j ],** where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

## Initializing Two-Dimensional Arrays

Multidimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] = {

   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */

   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */

   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */

};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

## Accessing Two-Dimensional Arrays Elements

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

```
int val = a[2][3];
```

The above statement will take 4th element from the 3rd row of the array. You can verify it in the above digram.

## Passing Two-Dimensional Arrays to Functions

One important thing for passing multidimensional arrays is, first array dimension does not have to be specified. The second (and any subsequent) dimensions must be given.

```
void print(int arr[][10], int m)      //where m is the size of the 1st dimension

{

}
```

https://www.tutorialspoint.com/cplusplus/cpp_multi_dimensional_arrays.htm

# CS 100 Lab 12

## Task 1

Two dimensional arrays can be very helpful in representing matrices. In this task we will be making a 4 by 4 matrix and will be applying some matrix functions on it.

- Define a 2D-Array with values of your choice.
- Make a function that takes a 2D array and a char as input and returns the sum of left or right diagonal. If the character is L, the function should return the sum of the left diagonal. If the character is R, the function should return the sum of the right diagonal. The function definition is provided below.

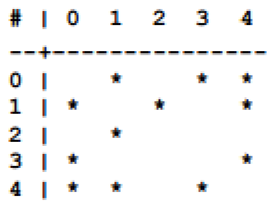
```
//m = number of rows.
int sum_D(int 2d_arr[][4], int m, char L_R)
{

 }
```

**<u>STOP AND SHOW YOUR WORK TO THE TA</u>**

## Task 2

Create a 2D array of 5-by-5 values of type integer.

Suppose indices represent people and that the value at row i, column j of a 2D array is 1 just in case i and j are friends and 0 otherwise. Initialize your list according to the configuration given below.



| # | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 |  | * |  | * | * |
| 1 | * |  | * |  | * |
| 2 |  | * |  |  |  |
| 3 | * |  |  |  | * |
| 4 | * | * |  | * |  |

(* means "friends")

a) Now, write a function to count how many pairs of friends are represented in the array. Note that each friendship pair appears twice in the array, so in the example above there are 6 pairs of friends).

b) Write a void function to check whether two people have a common friend. For example, in the example above, 0 and 4 are both friends with 3 (so they have a common friend), whereas 1 and 2 have no common friends. The function should take two people as input and display all their common friends.

c) Write a function that displays the ID of the people who have highest amount of common friends. For instance, if 0 and 4 have the highest amount of common friends, the function should display:

```
0 and 4 have highest amount of common friends.
```

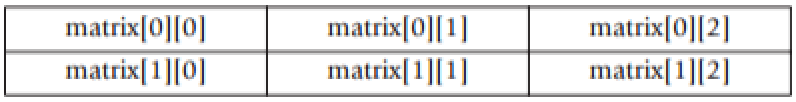
**<u>STOP AND SHOW YOUR WORK TO THE TA</u>**

## Task 3: 1D to 2D

What if C++ had no built-in facility for two-dimensional arrays? It is possible to emulate them yourself with a one dimensional array. The basic idea is shown below. Consider the following two-dimensional array:

```
int matrix[2][3];
```

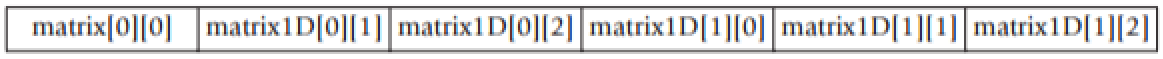
It can be visualized as a table:



| | | |
|---|---|---|
| matrix[0][0] | matrix[0][1] | matrix[0][2] |
| matrix[1][0] | matrix[1][1] | matrix[1][2] |

The two-dimensional array can be mapped to storage in a one-dimensional array where each row is stored in consecutive memory locations (your compiler actually does something very similar to map two-dimensional arrays to memory).

```
int matrix1D[6];
```



| | | | | | |
|---|---|---|---|---|---|
| matrix[0][0] | matrix1D[0][1] | matrix1D[0][2] | matrix1D[1][0] | matrix1D[1][1] | matrix1D[1][2] |

Here, the mapping is as follows:

matrix[0][0] would be stored in matrix1D[0]

matrix[0][1] would be stored in matrix1D[1]

matrix[0][2] would be stored in matrix1D[2]

matrix[1][0] would be stored in matrix1D[3]

matrix[1][1] would be stored in matrix1D[4]

matrix[1][2] would be stored in matrix1D[5]

# CS 100 Lab 12

- For this task you are supposed to get the number of rows and columns as input from the user and create a one-dimensional array to emulate a two dimensional array.

- Based on this idea, complete the definitions for the following functions.

a) `void set(int arr[], int row, int column, int desired_row, int desired_column, int val);`

This stores val into the emulated two-dimensional array at position desired_row, desired_column. The function should print an error message and exit if the desired indices are invalid.

- `arr` is the one-dimensional array used to emulate a two-dimensional array.
- `rows` is the total number of rows in the two-dimensional array.
- `columns` is the total number of columns in the two-dimensional array.
- `desired_row` is the index of the row the caller would like to access.
- `desired_column` is the index of the column the caller would like to access.
- `val` is the value to store at `desired_row` and `desired_column.`

b) `int get(int arr[], int rows, int columns, int desired_row, int desired_column);`

This returns the value in the emulated two-dimensional array at position desired_row, desired_column. The function should print an error message and exit if the desired indices are invalid.

- `arr` is the one-dimensional array used to emulate a two-dimensional array.
- `rows` is the total number of rows in the two-dimensional array.
- `columns` is the total number of columns in the two-dimensional array.
- `desired_row` is the index of the row the caller would like to access.
- `desired_column` is the index of the column the caller would like to access.

**STOP AND SHOW YOUR WORK TO THE TA**