# Multidimensional Arrays

# Why Multidimensional Arrays?

- Marks of 800 students in 5 subjects each.
- 800X5
- Distance between cities.
- All the above example require 2D Array (Table).

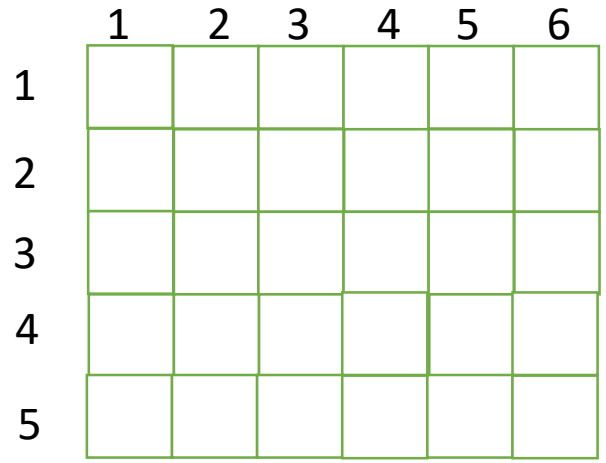
| S1 | S2 | S3 | S4 | S5 |
|---|---|---|---|---|
| 20 | 23 | 25 | 12 | 14 |
| 22 | 23 | 23 | 24 | 10 |
| 12 | 13 | 14 | 16 | 20 |
| … | … | … | … | … |
| 25 | 25 | 25 | 25 | 25 |

| | Seattle | Minneapolis | Boston | Reno | Denver | Raleigh | LA | Austin | Miami |
|---|---|---|---|---|---|---|---|---|---|
| Seattle | 0 | 1377 | 2456 | 557 | 1003 | 2318 | 942 | 1739 | 2685 |
| Minneapolis | 1377 | 0 | 1106 | 1384 | 683 | 966 | 1514 | 1023 | 1483 |
| Boston | 2456 | 1106 | 0 | 2485 | 1739 | 603 | 2572 | 1670 | 1243 |
| Reno | 557 | 1384 | 2485 | 0 | 781 | 2206 | 386 | 1380 | 2427 |
| Denver | 1003 | 683 | 1739 | 781 | 0 | 1425 | 837 | 759 | 1692 |
| Raleigh | 2318 | 966 | 603 | 2206 | 1425 | 0 | 2205 | 1143 | 693 |
| LA | 942 | 1514 | 2572 | 386 | 837 | 2205 | 0 | 1219 | 2307 |
| Austin | 1739 | 1023 | 1670 | 1380 | 759 | 1143 | 1219 | 0 | 1091 |
| Miami | 2685 | 1483 | 1243 | 2427 | 1692 | 693 | 2307 | 1091 | 0 |

# Array Declaration

double mat[5][6];  int mat[5][6];  float mat[5][6];

- This will create a 5x6 matrix of respective data type. It will have 5 rows and 6 columns. Each entry will be of the respective data type.



# Accessing Matrix Element

- (i,j)th member of matrix: mat[i][j].
- In mathematics, we use mat(i,j).
- The row and column, as is the case with 1D array, start with 0.

```
void print_matrix(int mat[5][6]) {
        int i,j;
        for(i=0; i<5; i++) {
                for(j=0; j<6; j++) {
                        printf("%d", mat[i][j]);
                }
                printf("\n");
        }
}
```

# User Input

- The address of i,j th matrix element is &mat[i][j].

```
void read_matrix(int mat[5][6]) {
        int i,j;
        for(i=0; i<5; i++) {
                for(j=0; j<6; j++) {
                        scanf("%d", &mat[i][j]);
                }
        }
}
```

# 2D Array Initialization

- We want a 2D array a[4][3] with following initialization.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 0 | 1 | 2 |

- We initialize it as:

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

# Rules

- Values are given row-wise, first row, then second row, so on.
- Number of columns must be specified.
- Each row is enclosed in braces {}.
- Number of values in row may be less than number of columns. Remaining columns will be set to 0;

# 2D Array Passing into function

```
void print_matrix(int mat[][6], int nRows) {
        int i,j;
        for(i=0; i<5; i++) {
                for(j=0; j<6; j++) {
                        printf("%d", mat[i][j]);
                }
                printf("\n");
        }
}
```

# Why only column is required?

- The memory of computer is a 1D array.
- 2D array are “flattened” into 1D to be stored in memory.
- In C, arrays are flattened using Row-Major order:
  - R1 R2 R3... RN
  - Thus, you need to know the number of columns to find out where the next row starts.
- Generally, in a nD array, you would require last n-1 dimensions.

# Exercise

- Write a program to multiply two matrices.