2-D ARRAYS

CONSIDER THE FOLLOWING SCENARIO

Suppose, we want to print the consolidated semester result of the whole class with the following details:

- Marks of each subject (total 5 subjects)
- Total number of students in a class = 70

Student_no	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
1					
2					
3					
4					
5					
and so on.....					

HOW TO STORE THE PREVIOUSLY GIVEN DATA?

- In order to do that,
  - Do we need arrays?
    - Yes!!
  - Will a single array be sufficient?
    - No. Because the data is TWO dimensional.
  - We need to store each student's data consecutively after the other, which itself is a 1-D array.
  - One row must immediately follow the other row in the memory
  - Therefore, we need 2-DIMENSIONAL arrays.

2-D ARRAYS

- Declaration of a 2-D array:

  Syntax:

  datatype array_name [ROW_SIZE][COL_SIZE];

- Each row in a 2-D array will have the SAME number of columns.

  Example:

  int a[5][5];

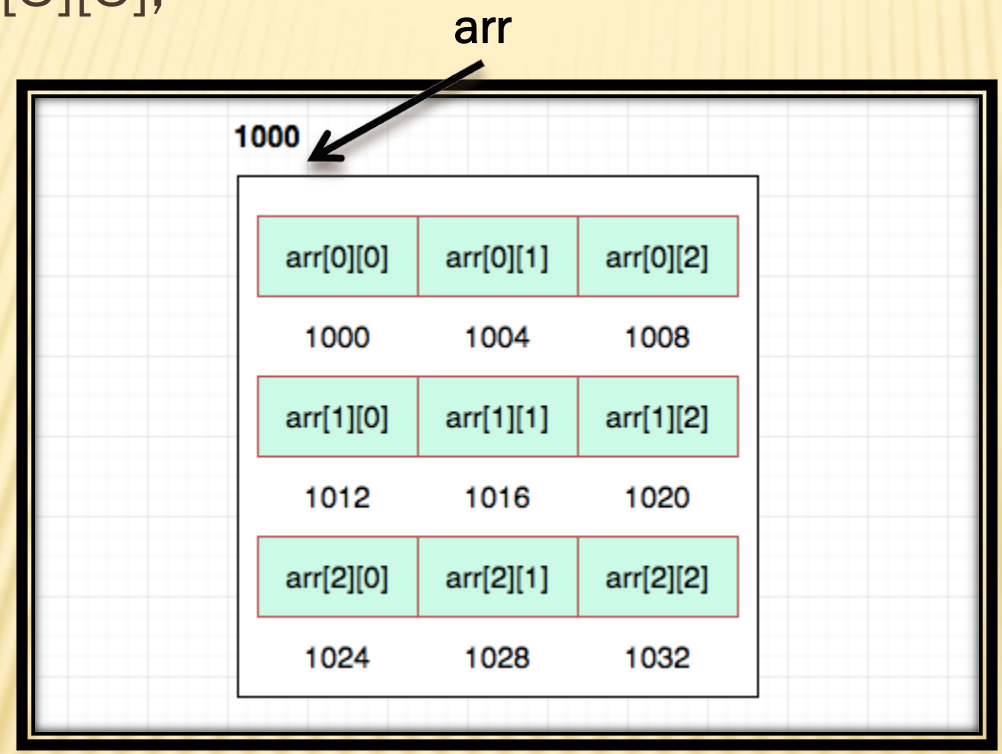
  int b[3][4];

  int c[4][3];

MEMORY REPRESENTATION OF A 2-D ARRAY

- Consider we have the following array (size of int = 4 bytes) :

  int arr[3][3];



Total size of the array =
row_size * col_size * size of each element
i.e 3*3*4= 36 bytes

INITIALIZATION OF A 2-D ARRAY(IN THE LINE OF DECLARATION)

Syntax:

```
datatype array_name[row_size][col_size]= {
                                        { r1c1, r1c2,r1c3...},
                                        { r2c1, r2c2,r2c3...},
                                        ......
                                        };
```

Example:

```
int arr[3][4]= { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

ACCESSING ELEMENTS OF A 2-D ARRAY

- Syntax:

  array_name [row_index][col_index]

- Example:

```
int a[4][3]={ {1,2,3}, {4,5,6}, {7,8,9}, {10,11,12} };
printf("%d", a[1][2]);  ⟶ 6
printf("%d", a[2][1]);  ⟶ 8
printf("%d", a[0][0]);  ⟶ 1
printf("%d", a[3][2]);  ⟶ 12
```

Point to note:
In a 2-D array, both row_index and col_index will begin from 0

QUICK EXERCISE

- Predict the output:

```
int a[10][10]={{1,2,3,4},{5,6,7,8},{9,10,11,12}},i,j;
for(i=0;i<=2;i++)
{
for(j=0;j<=3;j++)
{
printf("%d\t", a[i][j]);
}
printf("\n");
}
```

Output:

1 2 3 4
5 6 7 8
9 10 11 12

QUICK EXERCISE

```
int a[10][10]={{1,2,3,4},{5,6,7,8},{9,10,11,12}},i,j;
for(i=0;i<=2;i++)
{
printf("%d\t", a[i]);
}
```

Output

Compiler Error.

In order to access any element of an array, we MUST have TWO indices after the array name.

QUICK EXERCISE

```
int a[10][10]={{1,2,3,4},{5,6,7,8},{9,10,11,12}},i,j;
for(j=0;j<=3;j++)
{
for(i=0;i<=2;i++)
{
printf("%d  ", a[i][j]);
}
printf("\n");
}
```

Output:

```
1   5   9
2   6   10
3   7   11
4   8   12
```

QUICK EXERCISE

```
int a[3][3]={{1,2,3},{5,6,7},{9,10,11}},i,j;
for(i=0;i<=2;i++)
{
for(j=0;j<=2;j++)
{
if(i==j)
continue;
printf("%d ", a[i][j]);
}
printf("\n");
}
```

Output:

2 3
5 7
9 10

USER INPUT A 2-D ARRAY

1. Declare an array with a CONSTANT row_size and col_size.
2. Ask user to enter the number of rows and cols to be operated with. Let's call that "r" and "c".
3. Take input of array using nested for loops

Example:

```
int a[10][10],r,c;
scanf("%d%d", &r,&c);
for(i=0;i<r;i++)
{
    for(j=0;j<c;j++)
    {
        scanf("%d", &a[i][j]);
    }
}
```

SOME TIPS REGARDING 2-D ARRAY PROGRAMS

1. Always use index “i” for rows
2. Always use index “j” for columns
3. If we want to access a 2-D array row-wise, “i” loop (row loop) will be outside, “j” loop (column loop) will be inside.
4. If we want to access a 2-D array column-wise, “j” loop (column loop) will be outside, “i” loop (row loop) will be inside.
5. In both the above cases, array elements will be accessed as: **a[i][j]** only

PROGRAMS

- Find the sum of all the elements of a 2-D array / matrix
- Program to print the sum of diagonal elements of the matrix.
- Program to print if a given matrix is an identity matrix/not
- Program to input two matrices. Add them and store the result in a third matrix