C-ARRAYS

ARRAYS

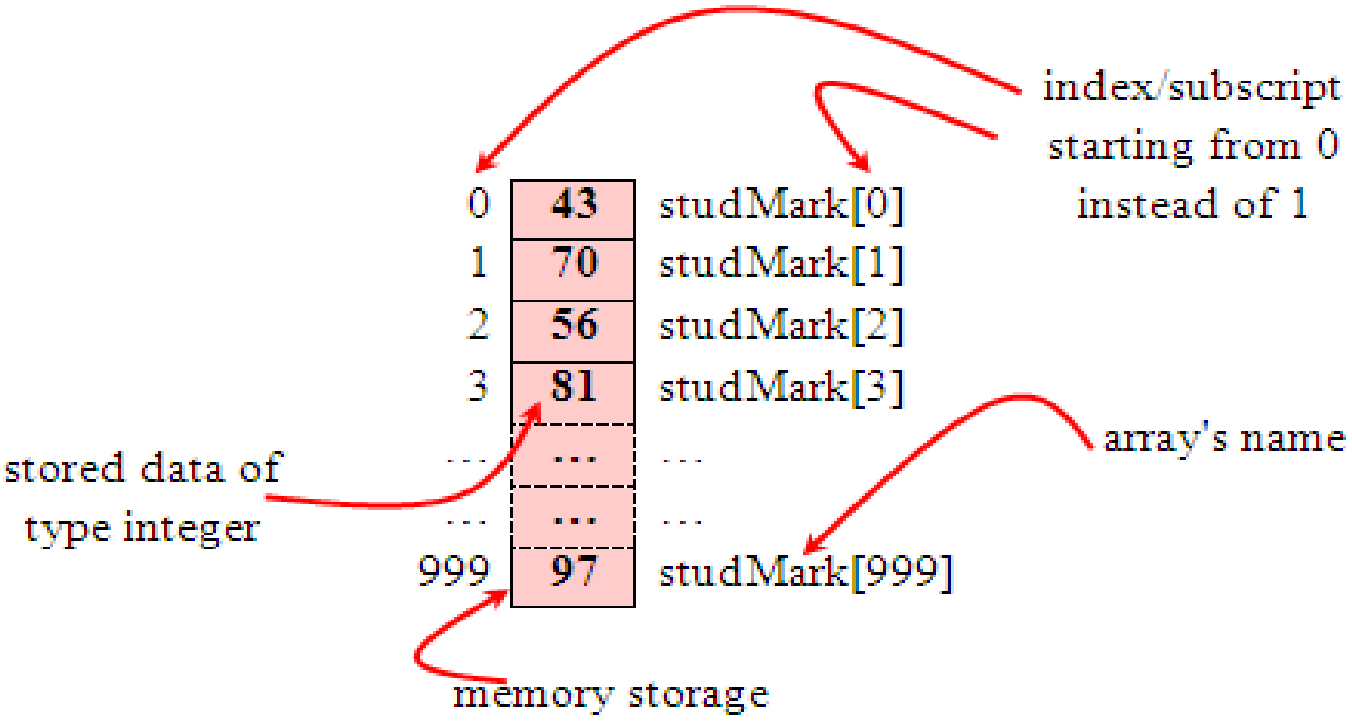
- An array is a collection of elements of the same type that are referenced by a common name.
- Compared to the basic data type (`int`, `float`) it is an aggregate or derived data type.
- All the elements of an array occupy a set of contiguous memory locations.
- Why need to use array type?
- Consider the following issue:

- **"We have a list of 1000 students' marks of an integer type. If using the basic data type (int), we will declare something like the following…"**
  - `int  studMark0, studMark1, ...studMark999`

- Can you imagine how long we have to write the declaration part by using normal variable declaration?

```
int main(void)
{
   int studMark1, studMark2, studMark3,
   studMark4, …, …, studMark998, stuMark999,
   studMark1000;
   …
   …
   return 0;
}
```

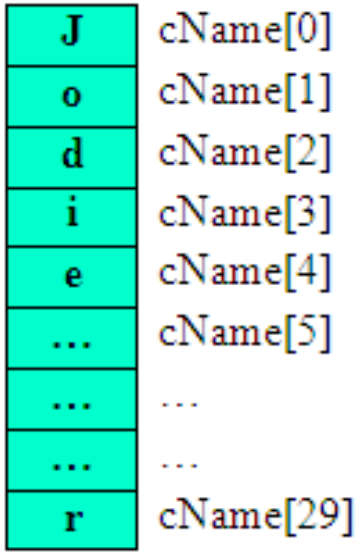
- By using an array, we just declare like this,
  - `int studMark[1000];`
- This will reserve 1000 contiguous memory locations for storing the students' marks.
- Graphically, this can be depicted as in the following figure.



- This absolutely has simplified our declaration of the variables.
- We can use index or subscript to identify each element or location in the memory.
- Hence, if we have an index of `j`, `studMark[j]` would refer to the *`jth`* element in the array of `studMark`.
- For example, `studMark[0]` will refer to the first element of the array.
- Thus by changing the value of `j`, we could refer to any element in the array.
- So, array has simplified our declaration and of course, manipulation of the data.

- A single or one dimensional array declaration has the following form,
  - `array_element_data_type array_name[array_size];`
- Here, *array_element_data_type* defines the base type of the array, which is the type of each element in the array.
- *array_name* is any valid C identifier name that obeys the same rule for the identifier naming.
- *array_size* defines how many elements the array will hold.

- For example, to declare an array of 30 characters, that construct a people name, we could declare,
  - `char cName[30];`
- Which can be depicted as follows,



- In this statement, the array character can store up to 30 characters with the first character occupying location `cName[0]` and the last character occupying `cName[29]`.
- Note that the index runs from `0` to `29`. In C, an index always starts from `0` and ends with array's (size-1).
- So, take note the difference between the array size and subscript/index terms.

- Examples of the one-dimensional array declarations,

```
int     xNum[20], yNum[50];
float   fPrice[10], fYield;
char    chLetter[70];
```

- The first example declares two arrays named `xNum` and `yNum` of type `int`. Array `xNum` can store up to 20 integer numbers while `yNum` can store up to 50 numbers.

- The second line declares the array `fPrice` of type `float`. It can store up to 10 floating-point values.

- `fYield` is basic variable which shows array type can be declared together with basic type provided the type is similar.

- The third line declares the array `chLetter` of type char. It can store a string up to 69 characters.

- Why 69 instead of 70? Remember, a string has a null terminating character (`\0`) at the end, so we must reserve for it.

Array Initialization

- An array may be initialized at the time of declaration.

- Initialization of an array may take the following form,
  - `type array_name[size] = {a_list_of_value};`
- For example:
  - `int idNum[7] = {1, 2, 3, 4, 5, 6, 7};`
  - `float fFloatNum[5] = {5.6, 5.7, 5.8, 5.9, 6.1};`
  - `char chVowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};`
- The first line declares an integer array `idNum` and it immediately assigns the values 1, 2, 3, ..., 7 to `idNum[0]`, `idNum[1]`, `idNum[2]`,. . ., `idNum[6]` respectively.

- The second line assigns the values 5.6 to `fFloatNum[0]`, 5.7 to `fFloatNum[1]`, and so on.

- Similarly the third line assigns the characters 'a' to `chVowel[0]`, 'e' to `chVowel[1]`, and so on. Note again, for characters we must use the single apostrophe/quote (') to enclose them.
- Also, the last character in `chVowel` is `NULL` character ('\0').

- Initialization of an array of type char for holding strings may take the following form,
  - `char array_name[size] = "string_lateral_constant";`
- For example, the array `chVowel` in the previous example could have been written more compactly as follows,
  - `char chVowel[6] = "aeiou";`
- When the value assigned to a character array is a string (which must be enclosed in double quotes), the compiler automatically supplies the `NULL` character but we still have to reserve one extra place for the `NULL`.
- For unsized array (variable sized), we can declare as follow,
  - `char chName[ ] = "Mr. Dracula";`
- C compiler automatically creates an array which is big enough to hold all the initializer.

Retrieving Array Elements

- If you want to retrieve specific element then then you have to specify not only the array or variable name but also the index number of interest.

- For example:
- int Arr[]={1,3,5,6,8};
- printf("%d\t%d\n",Arr[1],Arr[2]);
- Output: 3 5

Array example

- Take 10 integer input from user and store then in an array and find the sum of all numbers stored in array.

```
#include<stdio.h>
int main(){
int i,sum=0,arr[10];
for(i=0;i<10;i++)
    scanf("%d",&arr[i]);
for(i=0;i<10;i++)
    sum+=arr[i];
printf("Sum of input integers is %d\n",sum);
return 0;
}
```

Summarize the response of a survey.

Input: Response of the survey, can be in the range between 0 and 10. Assume the population size to be 40.

Output: Frequency of each response.

```
#include<stdio.h>
#define SIZE 40
#define ANS 11

int main(void) {
        int response[SIZE];
        int freq[ANS] = {0};
        int i;
        for(i=0; i< SIZE; i++){
           scanf(“%d”,&response[i]);
           ++freq[response[i]];
       }

        for(i=0;i<ANS;i++)
           printf("Frequency of %d is %d\n",i,freq[i]);
}
```

Assignment

- Read from user ages of all students in class and save them in an array which can store floating point and find average, minimum and maximum age.
- A six faced die is rolled 600 times. Find the frequency of the occurrence of each face?

- **int rand(void):** returns a pseudo-random number in the range of 0 to RAND_MAX.
- **RAND_MAX:** is a constant whose default value may vary between implementations but it is granted to be at least 32767.
- **Issue:** If we generate a sequence of random number with rand() function, it will create the same sequence again and again every time program runs.

- The srand() function sets the starting point for producing a series of pseudo-random integers. If srand() is not called, the rand() seed is set as if srand(1) were called at program start.
- The pseudo-random number generator should only be seeded once, before any calls to rand(), and the start of the program.
- Standard practice is to use the result of a call to **srand(time(0))** as the seed. However, time() returns a time_t value which vary everytime and hence the pseudo-random number vary for every program call.

```
#include<stdio.h>

int main(){
int i,j,arr[7]={0};
srand(time(0));
for (i=0;i<600;i++){
        j=rand()%6;
        j=j+1;
        arr[j]++;
}
for(i=1;i<=6;i++)
        printf("%d came out for %d times\n",i,arr[i]);
return 0;
}
```

- [sourav@gaya]$ ./a.out

1 came out for 113 times

2 came out for 114 times

3 came out for 102 times

4 came out for 86 times

5 came out for 99 times

6 came out for 86 times

Assignment

- Store marks obtained by students in an array. Find if there is more than one student who scored same marks. Assume minimum marks obtained is 30 and maximum marks obtained is 85.

```
#include<stdio.h>

int main(){
int i,j,x,arr[10];

printf("Enter 10 integer number\n");

for(i=0;i<10;i++){
            scanf("%d",&arr[i]);
}

for(i=0;i<9;i++){
       x=arr[i];
       for(j=i+1;j<10;j++){
             if(x==arr[j])
                   printf("%d number appeared more than once\n",x);
       }
}
return 0;
}
```

- [sourav@gaya]$ ./a.out

Enter 10 integer number

1

2

3

4

5

6

2

4

8

9

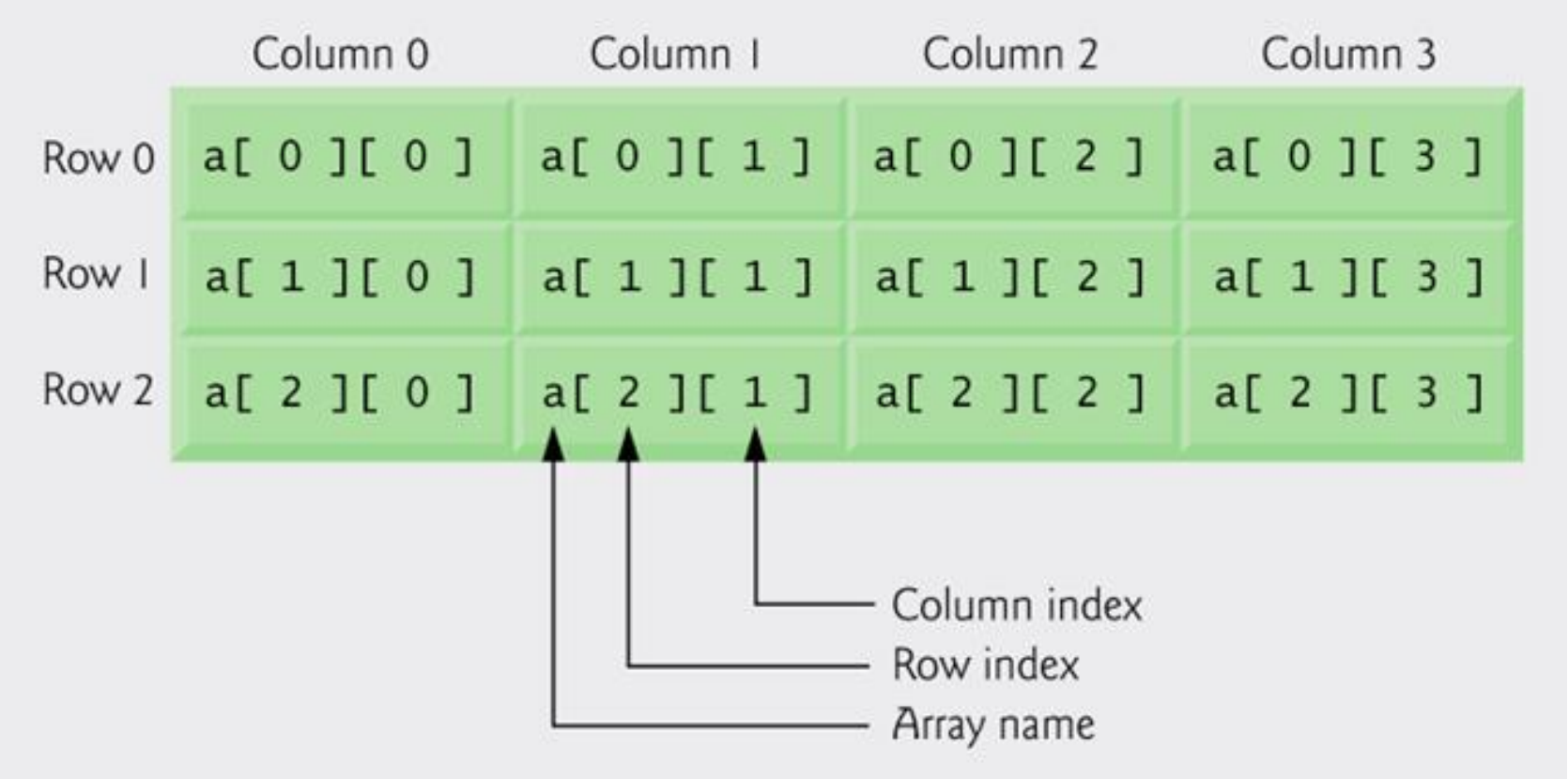
2 number appeared more than once

4 number appeared more than once

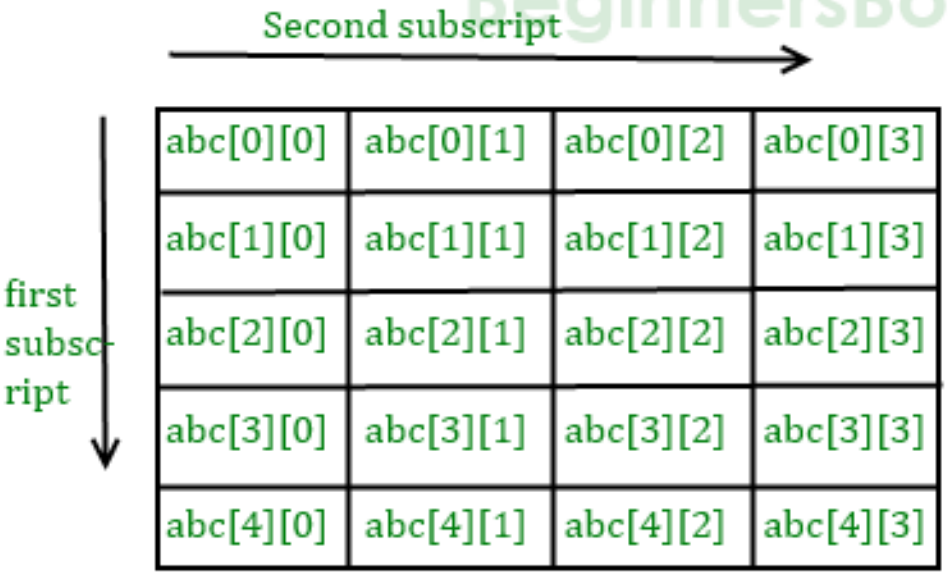
TWO DIMENSIONAL / 2D ARRAYS

- A two dimensional array has two subscripts/indexes.
- The first subscript refers to the row, and the second, to the column.
- Its declaration has the following form,
  - `data_type array_name[1st dimension size][2nd dimension size];`
- For examples,
  - `int xInteger[3][4];`
  - `float matrixNum[20][25];`
- The first line declares `xInteger` as an integer array with 3 rows and 4 columns.
- Second line declares a `matrixNum` as a floating-point array with 20 rows and 25 columns.

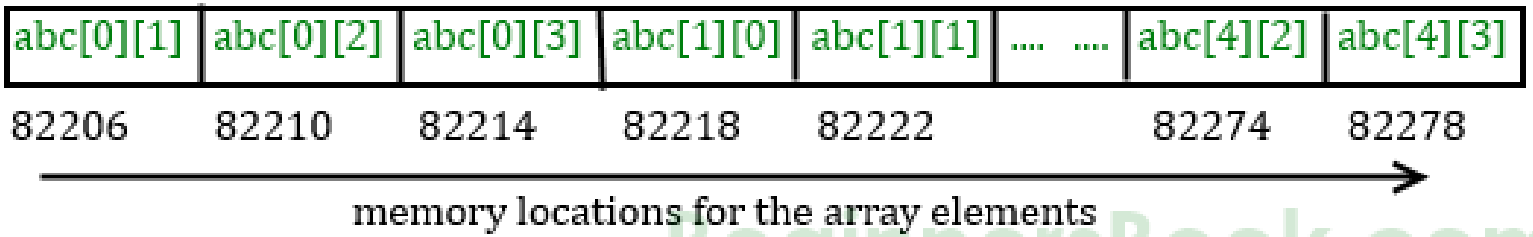
Double scripted array with 3 rows and 4 columns



2D array conceptual memory representation



Here my array is abc[5][4], which can be conceptually viewed as a matrix of 5 rows and 4 columns. Point to note here is that subscript starts with zero, which means abc[0][0] would be the first element of the array.



Array is of integer type so each element would use 4 bytes that's the reason there is a difference of 4 in element's addresses.

The addresses are generally represented in hex. This diagram shows them in integer just to show you that the elements are stored in contiguos locations, so that you can understand that the address difference between each element is equal to the size of one element(int size 4). For better understanding see the program below.

Actual memory representation of a 2D array

- #include <stdio.h>
- int main() {
  - int abc[5][4] ={ {0,1,2,3}, {4,5,6,7}, {8,9,10,11}, {12,13,14,15}, {16,17,18,19} };
  - for (int i=0; i<=4; i++) {
    - printf("%d ",abc[i]);
  - }
  - return 0;
- }
- Output: 1600101376 1600101392 1600101408 1600101424 1600101440

List of students and their subject marks

Marks →

Students ↓

10	23	31	11
20	43	21	21
12	22	30	13
30	31	26	41
13	03	41	15

Find the average mark scored by each student?

```
#include<stdio.h>
#define ROW  5
#define COL 4

int main(void)
{
        int i, j;
        double total;
        int marks[ROW][COL]= { 10, 23, 31, 11, 20, 43, 21, 21,12,
22, 30, 13, 30, 31, 26, 41,13, 03, 41, 15 };
        for(i = 0; i < ROW ; i++)
        {
                 total = 0.0;
                for (j=0; j<COL; j++)
                        total+=marks[i][j];
              printf("Average  of student %d is %f\n", i, total/4.0);
        }
}
```

Initialization of 2D Array

- int disp[2][4] = { {10, 11, 12, 13}, {14, 15, 16, 17} };
- OR
- int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
- 1st one is recommended.

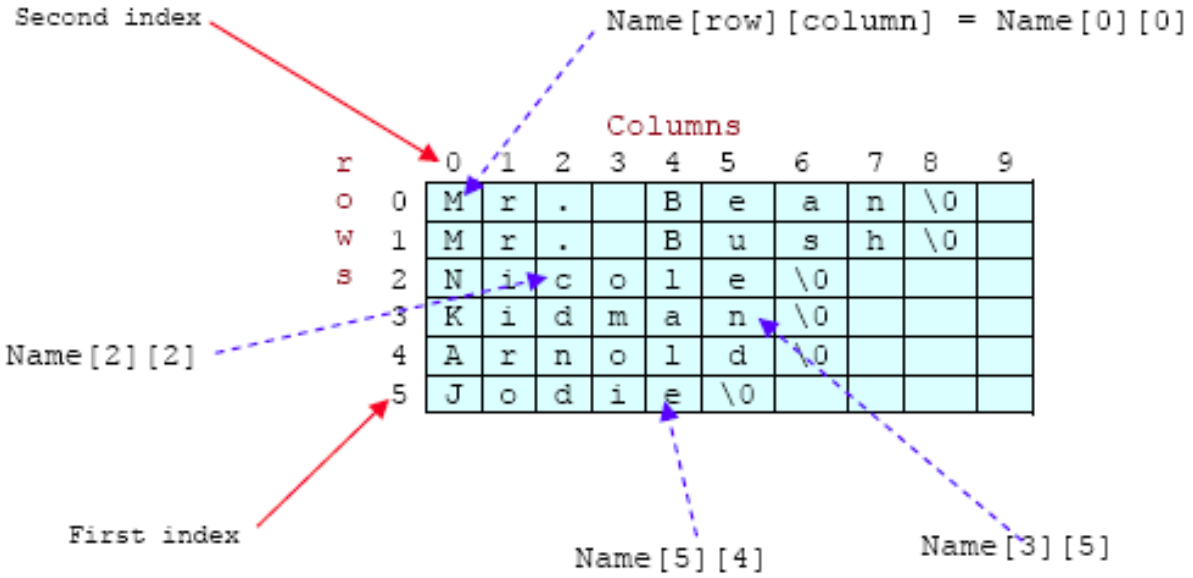
Things that you must consider while initializing a 2D array

- We already know, when we initialize a normal **array** (or you can say one dimensional array) during declaration, we need not to specify the size of it. However that's not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration.

- /* Valid declaration*/
- int abc[2][2] = {1, 2, 3 ,4 }
- /* Valid declaration*/
- int abc[][2] = {1, 2, 3 ,4 }
- /* Invalid declaration – you must specify second dimension*/
- int abc[][] = {1, 2, 3 ,4 }
- /* Invalid because of the same reason mentioned above*/
- int abc[2][] = {1, 2, 3 ,4 }

- For array storing string
  - `char Name[6][10] = {"Mr. Bean", "Mr. Bush", "Nicole", "Kidman", "Arnold", "Jodie"};`

- Here, we can initialize the array with 6 strings, each with maximum 9 characters long.
- If depicted in rows and columns it will look something like the following and can be considered as contiguous arrangement in the memory.



Assignments

- Print Transpose of a Matrix
- Add Two Matrix Using Multi-dimensional Arrays
- Multiply to Matrix Using Multi-dimensional Arrays

```
#include <stdio.h>


void main()
{
    static int array[10][10];
    int i, j, m, n;
    printf("Enter the order of the matrix \n");
    scanf("%d %d", &m, &n);
    printf("Enter the coefiicients of the matrix\n");
    for (i = 0; i < m; ++i){
        for (j = 0; j < n; ++j){
            scanf("%d", &array[i][j]);
        }
    }
```

```
    printf("The given matrix is \n");
    for (i = 0; i < m; ++i){
        for (j = 0; j < n; ++j){
            printf(" %d", array[i][j]);
        }
        printf("\n");
    }
    printf("Transpose of matrix is \n");
    for (j = 0; j < n; ++j){
        for (i = 0; i < m; ++i){
            printf(" %d", array[i][j]);
        }
        printf("\n");
    }
}
```

```
#include <stdio.h>
int main()
{
    int r, c, a[100][100], b[100][100], sum[100][100], i, j;
    printf("Enter number of rows (between 1 and 100): ");
    scanf("%d", &r);
    printf("Enter number of columns (between 1 and 100): ");
    scanf("%d", &c);
    printf("\nEnter elements of 1st matrix:\n");
    for(i=0; i<r; ++i) {
        for(j=0; j<c; ++j) {
            printf("Enter element a%d%d: ",i+1,j+1);
            scanf("%d",&a[i][j]);
        }
    }
```

```
    printf("Enter elements of 2nd matrix:\n");
    for(i=0; i<r; ++i)
        for(j=0; j<c; ++j) {
            printf("Enter element a%d%d: ",i+1, j+1);
            scanf("%d", &b[i][j]);
        }
    // Adding Two matrices
    for(i=0;i<r;++i)
        for(j=0;j<c;++j) {
            sum[i][j]=a[i][j]+b[i][j];
        }
    // Displaying the result
    printf("\nSum of two matrix is: \n\n");
    for(i=0;i<r;++i)
        for(j=0;j<c;++j) {
            printf("%d ",sum[i][j]);
            if(j==c-1) { printf("\n\n"); }
        }
return 0;
}
```

- #include <stdio.h>
- int main() {
  - int a[10][10], b[10][10], result[10][10], r1, c1, r2, c2, i, j, k;
  - printf("Enter rows and column for first matrix: ");
  - scanf("%d %d", &r1, &c1);
  - printf("Enter rows and column for second matrix: ");
  - scanf("%d %d",&r2, &c2);
  - while (c1 != r2) {
    - printf("Error! Not compatible for multiplication\n");
  - }

- printf("\nEnter elements of matrix 1:\n");
- for(i=0; i<r1; ++i)
  - for(j=0; j<c1; ++j) {
    - printf("Enter elements a%d%d: ",i+1, j+1);
    - scanf("%d", &a[i][j]);
  - }

- printf("\nEnter elements of matrix 2:\n");
- for(i=0; i<r2; ++i)
  - for(j=0; j<c2; ++j) {
    - printf("Enter elements b%d%d: ",i+1, j+1);
      scanf("%d",&b[i][j]);
  - }

- // Initializing all elements of result matrix to 0
- for(i=0; i<r1; ++i)
  - for(j=0; j<c2; ++j) {
    - result[i][j] = 0;
  - }
- // Multiplying matrices a and b and // storing result in result matrix
  for(i=0; i<r1; ++i)
  - for(j=0; j<c2; ++j)
    - for(k=0; k<c1; ++k) {
      - result[i][j]+=a[i][k]*b[k][j];
    - }

```
    // Displaying the result
    printf("\nOutput Matrix:\n");
    for(i=0; i<r1; ++i)
        for(j=0; j<c2; ++j) {
            printf("%d ", result[i][j]);
            if(j == c2-1) printf("\n\n");
        }
    return 0;
}
```

3D: A 3D ARRAY IS AN ARRAY OF 2D ARRAYS.

```
#include<stdio.h>
int main(){
int i,j,k,x[][2][3]={{{1,2,3},{4,5,6}},{{7,8,9},{10,11,12}}};
for(i=0;i<2;i++)
        for(j=0;j<2;j++)
                for(k=0;k<3;k++){

printf("x[%d,%d,%d]=%d\n",i,j,k,x[i][j][k]);
                }
return 0;
}
```