Initialising Two Dimensional array:

A two dimensional array can be initialised at the time of declaration with a list of values enclosed in braces.

Eg: int m[2][3]={1,2,3,4,5,6};

The initalisation is done row by row. So the above example create the 2D array like:

1	2	3
4	5	6

The above declaration can also be done by specifying rows separately as: int m[2][3]={{1,2,3},{4,5,6}};

Accessing elements in a Two dimensional array:

Each element is accessed using two subscripts – row number, and column number. Thus the index of first element is [0][0] and next element is [0][1] and so on. A nested for loop is commonly used to access two dimensional array elements. For example, to read values to the 2D array m, we can use the following nested for loop:

```
for(i=0;i<2;i++)
    for(j=0;j<3;j++)
        scanf("%d",&m[i][j]);
```

Sample programming questions using 2D arrays:

1. Read a 4x3 matrix and print it in matrix format
2. Read an MxN matrix and print its transpose
3. Find sum of all elements of a matrix
4. Find sum of two matrices
5. Find product of two matrices
6. Find row sum of a matrix
7. Find column sum of a matrix
8. Find sum of diagonal elements
9. Find sum of even numbers in a matrix
10. Check whether the matrix is Symmetric or not

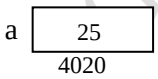
MODULE III : POINTERS AND STRINGS

Pointer is a variable used to store address. It is one of the most powerful features of C language. Pointers are closely related with array and strings. Pointer has the following advantages:

1. Pointers are used to manipulate dynamic data structures
2. Pointers increase program execution speed
3. Pointers reduce the length and complexity of program
4. Pointers enable to return more than one value from a function
5. Pointers can be used to pass arrays and strings from one function to another.

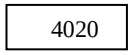
When we declare a variable, a memory location is allocated to that variable. Each memory location has a unique address.

For example, consider the declaration: **int a=25;** it means that the variable 'a' is integer type with an initial value 25. To store this value, a memory location is allocated by the compiler, let this location has an address 4020 as depicted below:



Now, it is possible to access the value 25 either by using the variable 'a' or by using the address 4020. This address can be stored in variables and such variables are called pointers. The memory address assigned by the compiler can be obtained by using the operator &, which is called address operator. The address of 'a' is available by using &a.

So we can use: p=&a; Now the address (ie. 4020) is stored in p, where p is a pointer as given below:



Pointer Declaration: Pointers, like any other variables, must be declared before they can be used. When declaring, a * (asterisk) is used before the name of the pointer. The general form of pointer declaration is:

data type *ptr_name;

where *data type* is the base type of the pointer. It specifies the type of the data to which the pointer points. In other words, we can say that, the pointer ptr_name holds the address of a value of the specified *data type*.

Example: int *x,y;

In the above example, x is a pointer that holds the address of an integer type data, and y is an ordinary integer variable.

Similarly, float *p; declares p as a floating point pointer.

Initializing Pointers:

Example:
```
int a,*b;
b=&a;
```

A pointer can be initialized to 0 or NULL or an address (using & operator). A pointer with the value 0 or NULL means, it points to nothing. The value 0 is the only integer that can be assigned directly to a pointer.

Pointer Operators: There are two operators in C to work with pointers: & and *. The & is called "address operator", which is unary operator, that returns the address of the variable. The * is called "value operator" (also called indirection operator or dereferencing operator) which gives the value stored at the address. For example,

```
int a=20,*b,c;
b=&a;
c=*b;
```

in this example, the variable 'c' will contain the value '20'

POINTER ARITHMETIC

Pointer arithmetic is different from ordinary arithmetic. In pointer arithmetic, the operation is performed relative to the size of the base type.(integer- 2, float -4, char -1 and double - 8).

Let ptr1 and ptr2 are two integer pointers (ie int *ptr1,*ptr2), then following are the valid arithmetic operations on pointers:

1. A pointer may be incremented. (Eg: ptr1++;)
2. A pointer may be decremented (Eg: ptr1--;)
3. An integer constant may added to a pointer (Eg: pt1r=ptr1+3;)
4. An integer constant may be subtracted from a pointer (Eg: ptr1=ptr1-3;)
5. A pointer can be assigned to another pointer, provided both pointers point to same type. (Eg: ptr1=ptr2;)

From the above list (1 to 4) , it can be understood that, addition or subtraction of integer constant is only permitted with pointers. Multiplication, division or modulus operations with pointers are not allowed.

Take the example given below:

```
int *a,b;      float *x,y;
a=&b;          x=&y;
```

Let the addresses stored in 'a' is 2030 and in 'x' is 4030.

Then the expression a++ ; will increment the address by 2 (since the size of int type is 2)
a=a+3; will increment the address by 6 (,ie 3x2=6)

This means that the increment or decrement will be based on the size of the data type

Similarly, x=x+3 will increment the address by 12 (ie. 3x4=12)

POINTERS AND ARRAYS: Pointers and arrays are closely related in C. When an array is declared, the array name contains the address of the first location of the array. It means that the array name itself is a pointer. For example, consider the following array:

```
int a[5]={10,20,30,40,50};
```

The above array can be represented as

Value	10	20	30	40	50
Index	0	1	2	3	4
Address	1000	1002	1004	1006	1008

Here, &a[0] is 1000. Similarly a is also 1000, because the array name holds the address of first location. By using pointer notation, we can access all elements of the array. For example, *a is 10, *(a+1) is 20, *(a+3) is 40. thus we can say that :

*a is same as a[0], *(a+1) is same as a[1], *(a+3) is same as a[3]. In general, *(a+i) is same as a[i] and (a+i) is same as &a[i].

Also note that *(a+1) and *a+1 are different. Here *a+1 will be 11 (10+1).

The following program will read and print numbers in an array using pointers.

```
#include <stdio.h>
main()
{   int num[10];    // integer array 'num' with size 10
    int i;
    for(i=0;i<10;i++)
        scanf("%d",(num+i));   // reading numbers,
    for (i=0;i<10;i++)
        printf("\n %d",*(num+i));   // printing nummbers
}
```

STRINGS

A string is a sequence of characters. Sequence of characters enclosed in double quote is string constant. Strings are used to represent alphabetic or alpha numeric data such as names of persons, objects etc or messages.(Eg: "John", "College", "Welcome", "KL 55 8818", "Silence Please" etc.). Thus in C, anything enclosed in double quote is treated as string. So 100 is integer, but "100" is string.

In C, there is no data type to represent string. To represent it, character arrays are used. So the declaration of string variable has the following general format: ***char str_name[size];***

The *size* determines the number of characters in the *str_name*. Example: char name[15];

When a character array is declared, the compiler automatically assigns a *null* character ('\0') at the end to determine the end of the string. To accommodate this null character, the size should be atleast the actual string size plus one.

For example: To declare a character array city to store a 10 character city names: *char city[11];*

Initializing Strings: Character arrays can be initialized at the time of declaration. This can be done in two methods.

Method 1: char city[10]="DELHI";

Method 2: char city[10]={'D','E','L','H','I','\0'};

When using the second method, we have to explicitly write the null character as the last character of the string.