FYBCA-SEM2-204 - Programming Skills

UNIT-1: Arrays, Structure & Union and User defined function in C programming Language:

1.1Concepts of Two-Dimensional Numeric Array:

1.1.1 Declaring Two-Dimensional numeric array:

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C:

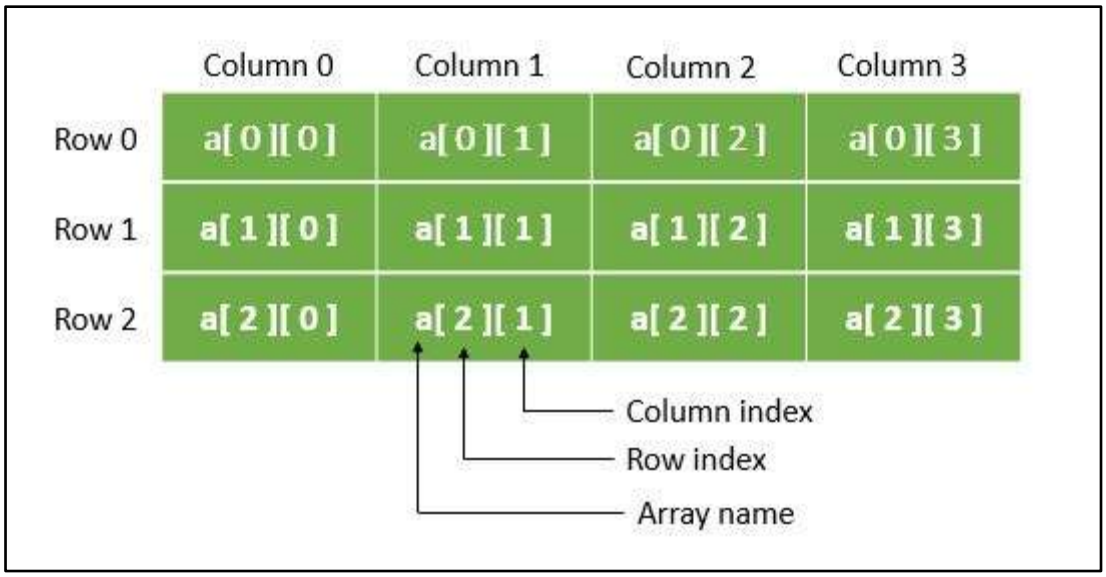
The syntax to declare the 2D array is given below.

```
data_type array_name[rows][columns];
```

Consider the following example.

```
int a[3][4];
```

Here, **a** is a two-dimensional (2D) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.



Initialization of a 2D array

There are many different ways to initialize two-dimensional array

```
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};

int c[][3] = {{1, 3, 0}, {-1, 5, 9}};

int c[2][3] = {1, 3, 0, -1, 5, 9};

int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

Two-dimensional array example in C

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i=0,j=0;
    int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};    clrscr();
```

```
//traversing 2D array
    for(i=0;i<4;i++)    // rows
    {
       for(j=0;j<3;j++)    // cols
       {
          printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
        }//end of j
    }//end of i
getch(); return 0;
}
```

OUTPUT:

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 2
arr[1][1] = 3
arr[1][2] = 4
arr[2][0] = 3
arr[2][1] = 4
arr[2][2] = 5
arr[3][0] = 4
arr[3][1] = 5
arr[3][2] = 6
```

Example 1: Two-dimensional array to store and print values

```
// C program to store temperature of two cities of a week and display it.
#include <stdio.h>
const int CITY = 2;
const int WEEK = 7;

int main()
{
  int temperature[CITY][WEEK];

  // Using nested loop to store values in a 2d array
  for (int i = 0; i < CITY; ++i)
  {
    for (int j = 0; j < WEEK; ++j)
    {
      printf("City %d, Day %d: ", i + 1, j + 1);
      scanf("%d", &temperature[i][j]);
    }
  }
  printf("\nDisplaying values: \n\n");
  // Using nested loop to display vlues of a 2d array
  for (int i = 0; i < CITY; ++i)
  {
    for (int j = 0; j < WEEK; ++j)
    {
       printf("City %d, Day %d = %d\n", i + 1, j + 1, temperature[i][j]);
    }
  }
    return 0;
}
```

Output:

```
City 1, Day 1: 33
City 1, Day 2: 34
City 1, Day 3: 35
City 1, Day 4: 33
City 1, Day 5: 32
City 1, Day 6: 31
City 1, Day 7: 30
City 2, Day 1: 23
City 2, Day 2: 22
City 2, Day 3: 21
City 2, Day 4: 24
City 2, Day 5: 22
City 2, Day 6: 25
City 2, Day 7: 26

Displaying values:

City 1, Day 1 = 33
City 1, Day 2 = 34
City 1, Day 3 = 35
City 1, Day 4 = 33
City 1, Day 5 = 32
City 1, Day 6 = 31
City 1, Day 7 = 30
City 2, Day 1 = 23
City 2, Day 2 = 22
City 2, Day 3 = 21
City 2, Day 4 = 24
City 2, Day 5 = 22
City 2, Day 6 = 25
City 2, Day 7 = 26
```

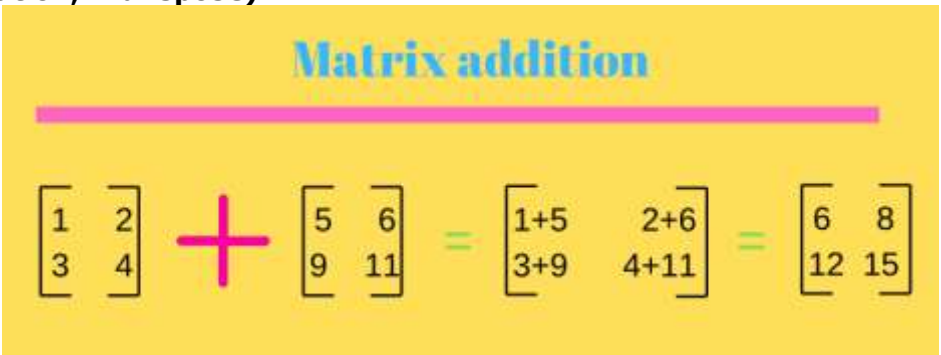
Example 2: 2D array example:- Storing elements in a matrix and printing it.

```
#include<stdio.h>
#include<conio.h>
void main ()
{
   int arr[3][3],i,j;    clrscr();
   for (i=0;i<3;i++)
   {
      for (j=0;j<3;j++)
      {
         printf("Enter a[%d][%d]: ",i,j);
         scanf("%d",&arr[i][j]);
      }
   }
   printf("\n Printing the elements ....\n");
   for(i=0;i<3;i++)
   {
      printf("\n");
      for (j=0;j<3;j++)
      {
         printf("%d\t",arr[i][j]);
      }
   }     getch();
}
```

Output:

```
Enter a[0][0]: 56
Enter a[0][1]: 10
Enter a[0][2]: 30
Enter a[1][0]: 34
Enter a[1][1]: 21
Enter a[1][2]: 34
Enter a[2][0]: 45
Enter a[2][1]: 56
Enter a[2][2]: 78

Printing the elements ....

56      10      30
34      21      34
45      56      78
```

1.1.2 Two-Dimensional numeric Array operations (Addition,Subtraction, Multiplication, Transpose)



Matrix addition in C language to add two matrices, i.e., compute their sum and print it. A user inputs their orders (number of rows and columns) and the matrices. For example, if the order is 2, 2, i.e., two rows and two columns and the matrices are:

First matrix:
1 2
3 4
Second matrix:
4 5
-1 5
The output is:
5 7
2 9

1. Two-Dimensional numeric Array operations: Addition of two Matrix

```
/*Addition of two matrix in C  */
#include<stdio.h>
#include<conio.h>
int main()
{
     int a[3][3],b[3][3],sum[3][3],i,j;
     clrscr();

     printf("Enter Matrix A Elements: ");
     for(i=0;i<3;i++)// rows
     {
          for(j=0;j<3;j++)    //cols
          {
            printf("\n Enter Array element A[%d][%d] :",i,j);
```

```
            scanf("%d",&a[i][j]);
          }
     }
     printf("Enter Matrix B Elements: ");
     for(i=0;i<3;i++)    // rows
     {
          for(j=0;j<3;j++)    //cols
          {
            printf("\n Enter Array element B[%d][%d] :",i,j);
            scanf("%d",&b[i][j]);
          }
     }
      printf("\n Matrix A : \n");
      printf("~~~~~~~~~~~~~~~~~~~~\n");
      for(i=0;i<3;i++)
      {
          for(j=0;j<3;j++)
          {
            printf(" %d ",a[i][j]);
          }
          printf("\n");
      }

      printf("\n Matrix B : \n");
      printf("~~~~~~~~~~~~~~~~~~~~\n");
      for(i=0;i<3;i++)
      {
          for(j=0;j<3;j++)
          {
            printf(" %d ",b[i][j]);
          }
          printf("\n");
      }
      printf("\n Sum of two matrices : \n");
      printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
      for(i=0;i<3;i++)
      {
          for(j=0;j<3;j++)
          {
        sum[i][j]=a[i][j]+b[i][j];
            printf("%d\t",sum[i][j]);
          }
          printf("\n");
      }
getch();     return 0;
}
```

2. Two-Dimensional numeric Array operations: Subtraction of two Matrix.

```
/* Subtraction of two matrix in C  */
#include<stdio.h>
#include<conio.h>

int main()
{
     int a[3][3],b[3][3],sub[3][3],i,j;
     clrscr();
```

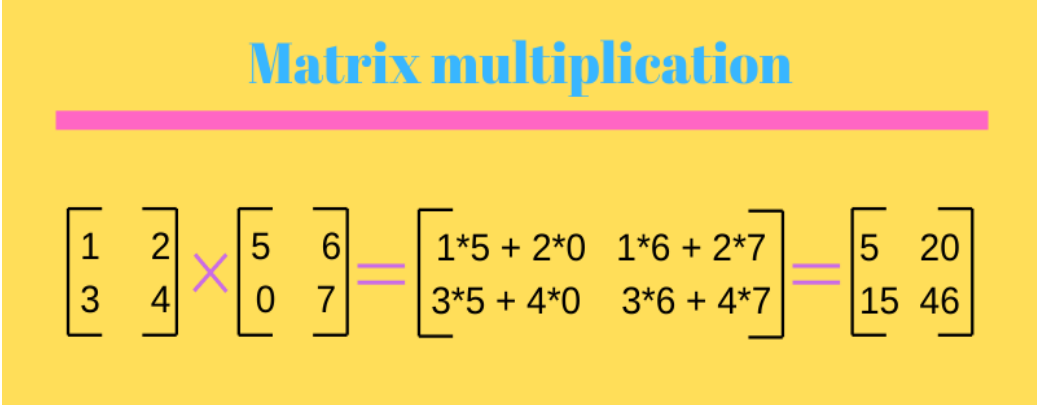
```
    printf("Enter Matrix A Elements: ");
    for(i=0;i<3;i++)// rows
    {
        for(j=0;j<3;j++)    //cols
        {
          printf("\n Enter Array element A[%d][%d] :",i,j);
          scanf("%d",&a[i][j]);
        }
    }
    printf("Enter Matrix B Elements: ");
    for(i=0;i<3;i++)    // rows
    {
        for(j=0;j<3;j++)    //cols
        {
          printf("\n Enter Array element B[%d][%d] :",i,j);
          scanf("%d",&b[i][j]);
        }
    }
     printf("\n Matrix A : \n");
     printf("~~~~~~~~~~~~~~~~~~~~\n");
     for(i=0;i<3;i++)
     {
        for(j=0;j<3;j++)
        {
          printf(" %d ",a[i][j]);
        }
        printf("\n");
     }
     printf("\n Matrix B : \n");
     printf("~~~~~~~~~~~~~~~~~~~~\n");
     for(i=0;i<3;i++)
     {
        for(j=0;j<3;j++)
        {
          printf(" %d ",b[i][j]);
        }
        printf("\n");
     }

printf("\n Subtraction of two matrices : \n");
     printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
     for(i=0;i<3;i++)
     {
        for(j=0;j<3;j++)
        {
          sub[i][j]=a[i][j]-b[i][j];
          printf("%d\t",sub[i][j]);
        }
        printf("\n");
     }
      getch();      return 0;
}
```

3. Two-Dimensional numeric Array operations: Multiplication of two Matrix.

Matrix multiplication in C language to calculate the product of two matrices (two-dimensional arrays). A user inputs the orders and elements of the matrices. If the multiplication isn't possible, an error message is displayed.

Some points important points for matrix multiplication:

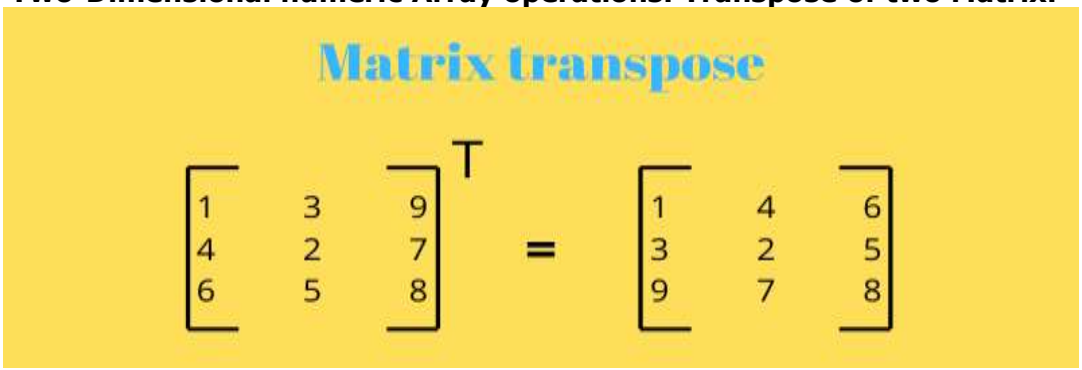
- This program asks the user to enter the size (rows and columns) of two matrices.
- To multiply two matrices, the number of columns of the first matrix should be equal to the number of rows of the second matrix.
- The program below asks for the number of rows and columns of two matrices until the above condition is satisfied.
- For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix.
- The result matrix has the number of rows of the first and the number of columns of the second matrix.




```
/* C program to find multiplication of twomatrices  */
#include<stdio.h>
#include<conio.h>
int main()
{
    int m1[10][10],m2[10][10],mul[10][10],r1,c1,r2,c2,i,j,k;
    clrscr();

    printf("Enter rows and column for the first matrix: ");
    scanf("%d %d", &r1, &c1);
    printf("Enter rows and column for the second matrix: ");
    scanf("%d %d", &r2, &c2);

   // Taking input until 1st matrix columns is not equal to 2nd matrix row
   while(c1 != r2)
   {
        printf("Error!!!! Enter rows and columns again.\n");
        printf("Enter rows and columns for the first matrix: ");
        scanf("%d %d", &r1, &c1);
        printf("Enter rows and columns for the second matrix: ");
        scanf("%d %d", &r2, &c2);
   }
  printf("\nEnter elements for First Matrix: \n");
  for(i=0;i<r1;i++)
  {
    for(j=0;j<c1;j++)
     {
```

```
            printf("Enter m1[%d][%d] : ", i, j);
            scanf("%d", &m1[i][j]);
        }
    }
    printf("\nEnter elements for Second Matrix: \n");
    for(i = 0; i < r2;i++)
    {
        for(j = 0; j < c2; j++)
        {
            printf("Enter m2[%d][%d] : ", i, j);
            scanf("%d", &m2[i][j]);
        }
    }
    for (i = 0; i < r1;i++)      // first matrix row
    {
        for (j = 0; j < c2; j++)   // second matrix column
        {
              mul[i][j] = 0;
            for (k = 0; k < r2; k++)
            {
                  mul[i][j] = mul[i][j] + ( m1[i][k] * m2[k][j] );
            }
        }
    }
    printf("\n First Matrix : \n");
    for(i=0;i < r1;i++)
    {
        for(j=0;j < c1;j++)
        {
            printf("%d\t", m1[i][j]);
        }
        printf("\n");
    }
    printf("\n Second Matrix:\n");
    for(i=0;i < r2;i++)
    {
        for(j=0;j < c2;j++)
        {
            printf("%d\t", m2[i][j]);
        }
        printf("\n");
    }
    printf("\nMultiplication of  Matrices : \n");
    for(i=0;i < r1;i++)
    {
        for(j=0;j < c2;j++)
        {
            printf("%d\t", mul[i][j]);
        }
        printf("\n");
    }
    getch();
    return 0;
}
```

4. Two-Dimensional numeric Array operations: Transpose of two Matrix.



Transpose of a matrix in C language: This C program prints transpose of a matrix. To obtain it, we interchange rows and columns of the matrix. For example, consider the following 3 X 2 matrix:

1 2
3 4
5 6

Transpose of the matrix:

1 3 5
2 4 6

When we transpose a matrix, its order changes, but for a square matrix, it remains the same.

```
/* C program to find transpose of 3 X 3 matrix  */
#include<stdio.h>
#include<conio.h>
int main()
{
int a[3][3],i,j;
clrscr();
printf("Enter Matrix A Elements: ");
 for(i=0;i<3;i++)// rows
{
          for(j=0;j<3;j++)    //cols
          {
            printf("\n Enter Array element A[%d][%d] :",i,j);
            scanf("%d",&a[i][j]);
          }
}
      printf("\n Matrix A : \n");
      printf("~~~~~~~~~~~~~~~~~~~~\n");
      for(i=0;i<3;i++)
      {
          for(j=0;j<3;j++)
          {
        printf("%d\t",a[i][j]);
          }
          printf("\n");
      }
printf("\n Transpose of matrix A : \n");
      printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
      for(i=0;i<3;i++)
      {
```

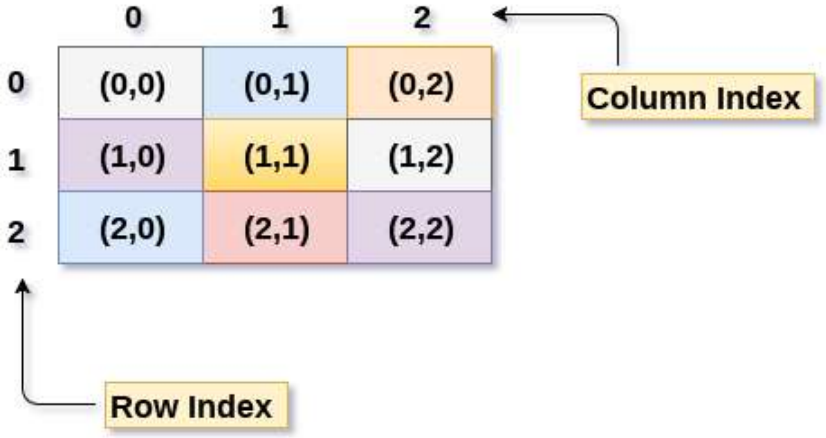
```
        for(j=0;j<3;j++)
        {
              printf("%d\t",a[j][i]);
        }
        printf("\n");
    }
getch();  return 0;
}
```

1.1.3 Element Address in array (Row major and Column major)

2D arrays are created to implement a relational database table lookalike data structure, in computer memory, the storage technique for 2D array is similar to that of an one dimensional array.

The size of a two dimensional array is equal to the multiplication of number of rows and the number of columns present in the array. We do need to map two dimensional array to the one dimensional array in order to store them in the memory.

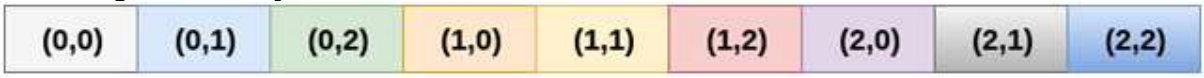
A 3 X 3 two dimensional array is shown in the following image. However, this array needs to be mapped to a one dimensional array in order to store it into the memory.



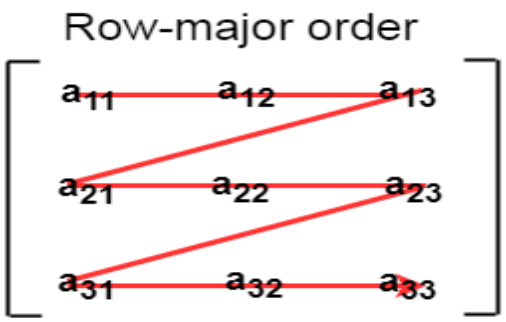
A two dimensional Array A is the collection of 'm X n' elements. Programming language stores the two dimensional array in one dimensional memory in either of two ways - There are two main techniques of storing 2D array elements into memory

1. Row major ordering

In row major ordering, all the rows of the 2D array are stored into the memory contiguously. Considering the array shown in the above image, its memory allocation according to row major order is shown as follows.



first, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.



First row of the array occupies the first set of memory locations reserved for the array; Second row occupies the next set, and so forth.
To determine element address A[i,j]:

Location (A[i,j]) =Base Address + (N x (i - 1)) + (j - 1)

For example :

Given an array [1…5,1…7] of integers. Calculate address of element A[4,6], where BA=900.

Solution:- I = 4 , J = 6 ,M= 5 , N= 7(M=ROW,N=COLUMN)

Location (A [4,6]) = BA + (7 x (4-1)) + (6-1)

= 900+ (7 x 3) +5

= 900+ 21 + 5

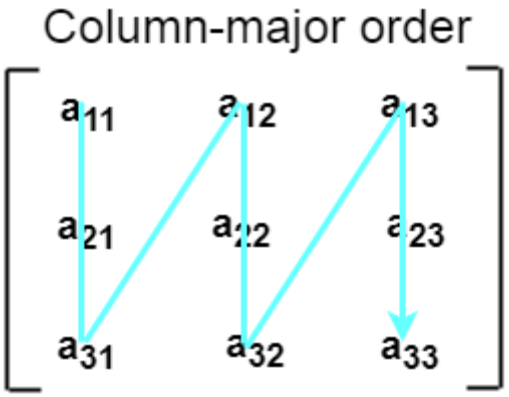
= 926

2. Column major ordering

According to the column major ordering, all the columns of the 2D array are stored into the memory contiguously. The memory allocation of the array which is shown in in the above image is given as follows.

(0,0)	(1,0)	(2,0)	(0,1)	(1,1)	(2,1)	(0,2)	(1,2)	(2,2)

first, the 1st column of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last column of the array.



Order elements of first column stored linearly and then comes elements of next column.
To determine element address A[i,j]:

Location (A[i,j]) =Base Address + (M x (j - 1)) + (i - 1)

For example :

Given an array [1…6,1…8] of integers. Calculate address element A[5,7], where BA=300.

Solution:- I = 5 , J = 7, M= 6 , N= 8

Location (A [5,7])= BA + (6 x (7-1)) + (5-1)

= 300+ (6 x 6) +4

= 300+ 36+4

= 340

1.1.4 Two-Dimensional Character Array:

What is String?

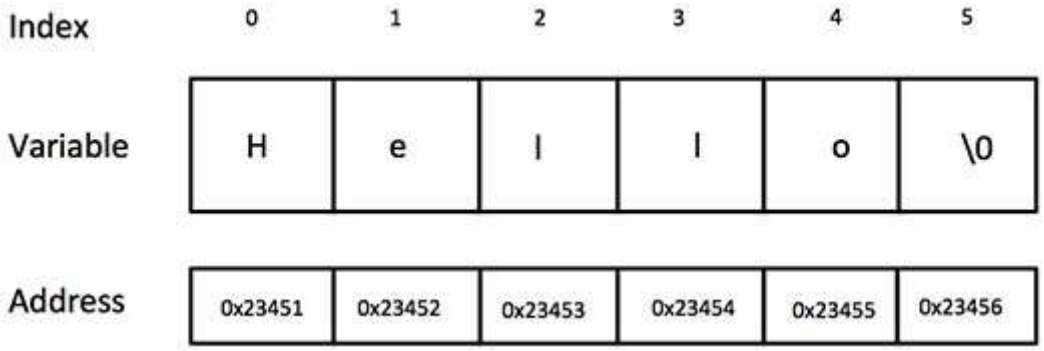
Strings are actually one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
If you follow the rule of array initialization then you can write the above statement as follows −

char greeting[] = "Hello";

Following is the memory presentation of the above defined string in C −



Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string −

```
#include <stdio.h>
int main ()
{
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

OUTPUT:

Greeting message: Hello

❖ 1.1.4.1 Declaring & Initializing Two-Dimensional character array or Multidimensional Character Array

Two-Dimensional character array Syntax:-

char string-array-name[row-size][column-size];

A 2D character array is declared in the following manner:

char name[5][10];

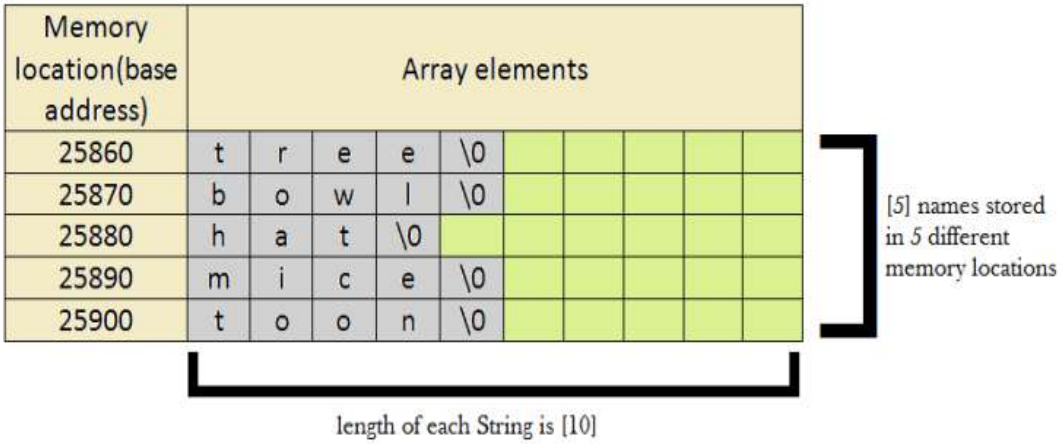
The order of the subscripts is kept in mind during declaration. The first subscript [5] represents the number of Strings that we want our array to contain and the second

subscript [10] represents the length of each String. This is static memory allocation. We are giving 5*10=50 memory locations for the array elements to be stored in the array.

Initialization of the character array occurs in this manner:

```
char name[5][10]={
                "tree",
                "bowl",
                "hat",
                "mice",
                "toon"    };
```

see the diagram below to understand how the elements are stored in the memory location:



The areas marked in green shows the memory locations that are reserved for the array but are not used by the string. Each character occupies 1 byte of storage from the memory.

Example of Multidimensional Character Array

```
#include <stdio.h>
void main()
{       char text[10][80];
        int i;
        clrscr();
        for(i = 0; i < 10; i++)
        {
                printf("\n Enter Some string for index %d: ", i + 1);
                gets(text[i]);
        }
        for(i = 0; i < 10; i++)
        {
                printf("\n Some string for index %d: ", i + 1);
                puts(text[i]);
        }
getch();
}
```

1.1.4.2 Two-Dimensional character Array operations (Searching elements, copying, merging, finding length of given string)

Example1: Program to search for a string in the string array

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int main()
{
    char name[5][10],item[10];
    int i,x;     clrscr();
    for(i=0;i<5;i++ )
    {
         printf("Enter %d strings: ", i+1);
         scanf("%s", &name[i][0]);
    }
        /* entering the item to be found in the string array*/
   printf("Enter the string to be searched: ");
   scanf("%s", &item);
    /* process for finding the item in the string array*/
    for(i=0; i<5 ;i++ )
    {
          x=strcmp(&name[i][0],item);
          if(x == 0)
          {
             printf("\n Searched string %s is at position  %d ",item,i+1);
             break;
          }
    }
    if(i > 4)
    {
        printf("\n The searched string does not match any name in the list...");
    }
    getch();     return 0;
}
```

Output:-

```
Enter 1 strings: one
Enter 2 strings: two
Enter 3 strings: three
Enter 4 strings: four
Enter 5 strings: five
Enter the string to be searched:four
Searched string four is at position : 4
```

Example2: Lexicographic order is the way of ordering words based on the alphabetical order of their component letters. It is also known as lexical order, dictionary order, and alphabetical order. It is similar to the way we search for any word in the dictionary. We start our search by simply searching for the first letter of the word. Then we try to find the second letter and so on. The words in the dictionary are arranged in lexicographic order.

```
/*  Write a program to sort elements in lexicographical order in C language
(dictionary order). */
#include<stdio.h>
```

```
#include<string.h>
#include<conio.h>
int main()
{
     char str[10][50],temp[50];
     int i,j;   clrscr();

     printf("Enter 10 Words:\n");
     for(i=0;i<10;i++)
     {
          printf("\n Enter String %d : ",i+1);
          scanf("%s[^\n]",str[i]);
     }
     for(i=0;i<9;i++)
     {
         for(j=i+1;j<10;j++)
         {
          if(strcmp(str[i],str[j])>0)
          {
              strcpy(temp,str[i]);
              strcpy(str[i],str[j]);
              strcpy(str[j],temp);
          }
         }
      }
printf("\n In Lexicographical order: \n");
      for(i=0;i<10;i++)
           puts(str[i]);
  getch();
  return 0;
}
```

Example3: Write a C program String Copy without using string manipulation library functions.

```
#include<stdio.h>
#include<conio.h>
int main()
{
  char str1[100], str2[100];
  int i;   clrscr();
  printf("Enter first String: ");
  gets(str1);
  // str2 is empty or having garbage value
  for(i=0;str1[i]!='\0';i++)
  {
    str2[i]=str1[i];
  }
  str2[i]='\0';
  printf("Second string is: ");
  puts(str2);
  getch();   return 0;
}
```

OUTPUT:

Enter first String: Adjust Everywhere

Second string is: Adjust Everywhere

Example 4: Write a C program to Concatenate two strings without using string manipulation library functions.(merging of two string in one array)

```
#include<stdio.h>
#include<conio.h>
int main()
{
   char str1[100], str2[50];
   int i, j;  clrscr();

   printf("Enter first string: ");
   gets(str1);

   printf("Enter second string: ");
   gets(str2);

   // calculate the length of string str1 & store it in the variable i
   for(i=0; str1[i]!='\0'; i++);

   for(j=0; str2[j] != '\0'; j++, i++)
   {
str1[i] = str2[j];
   }
   str1[i] = '\0';
   printf("After Concatenation first string becomes:\n");
   puts(str1);
getch();     return 0;
 }
```

Output:
Enter first string: PROGRAMMING
Enter second string: IS FUN!!!!
After Concatenation first string becomes:PROGRAMMING IS FUN!!!!

Example5: Write a C program to Find the length of the string without using string manipulation library functions.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char str[200];
    int i;   clrscr();
    printf("Enter string: ");
    scanf("%[^\n]",str);
    for(i=0;str[i]!='\0';i++);
    printf("length of entered string is=%d",i);
    getch();       return 0;
}
```

OUTPUT:
Enter string: sdj international college
length of entered string is=25

1.2 Concepts of Structure and Union:

1.2.1 Defining, declaring and Initializing structure and Union

Why we use structure?

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity Student may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

- Construct individual arrays for storing names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types.
- Let's look at the first approach in detail.

```
#include<stdio.h>
void main ()
{
  char names[3][10],dummy;
  int roll_numbers[3],i;
  float marks[3];
  clrscr();
  for (i=0;i<3;i++)
  {
    printf("Enter the name, roll number, and marks of the student %d : ",i+1);
    scanf("%s %d %f",&names[i],&roll_numbers[i],&marks[i]);
    scanf("%c",&dummy); // enter will be stored into dummy character at each iteration
  }
  printf("Printing the Student details...\n");
  for (i=0;i<3;i++)
  {
    printf("%s  %d  %.2f \n",names[i],roll_numbers[i],marks[i]);
  }     getch();
}
```

Output

```
Enter the name, roll number, and marks of the student   1 : Arun 90 91
Enter the name, roll number, and marks of the student   2 : Varun 91 56
Enter the name, roll number, and marks of the student   3 : Sham 89 69

Printing the Student details...
Arun 90 91.000000
Varun 91 56.000000
Sham 89 69.000000
```

The above program may fulfil our requirement of storing the information of an entity student. However, the program is very complex, and the complexity increase with the amount of the input. The elements of each of the array are stored contiguously, but all the arrays may not be stored contiguously in the memory. C provides you with an additional and simpler approach where you can use a special data structure, i.e., structure, in which, you can group all the information of different data type regarding an entity.

❖ What is Structure?

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures can simulate the use of classes and templates as it can store different information. The **struct** keyword is used to define the structure.

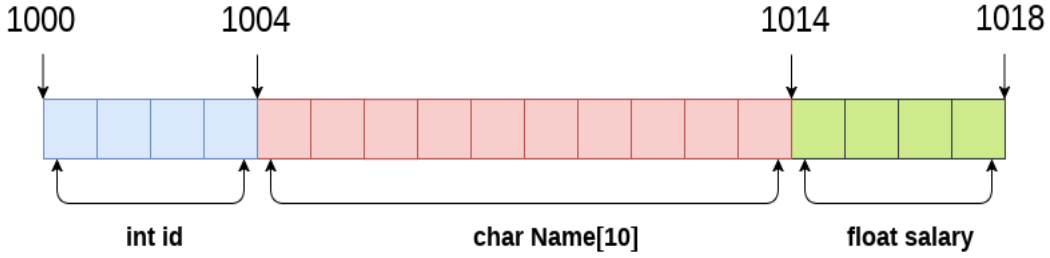
Let's see the syntax to define the structure in c.

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Example to define a structure for an entity employee in c.

```
struct employee
{
 int id;
   char name[10];
   float salary;
};
```

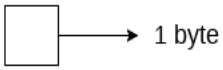
The following image shows the memory allocation of the structure employee that is defined in the above example.




```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

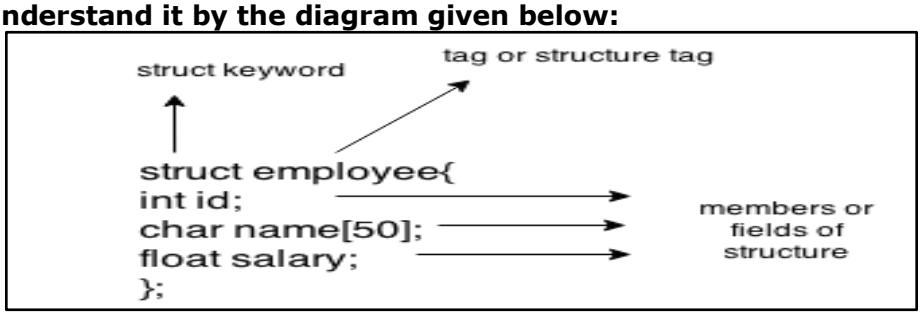
sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte



Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure.

Let's understand it by the diagram given below:



Points to remember while defining a Structure

1. Key Word **struct** is required for defining structure.
2. Structure_name can be any valid identifier.
3. Set of variables will be declared inside curly braces "{ }" is known as structure body.
4. Different types of variable or array or structure can be declared inside structure body.
5. Only declaration is allowed inside structure body, we cannot initialize variable inside structure body.
6. The closing curly brace in the structure type declaration must be followed by a semicolon (;)
7. Defining structure will not declare any variable it acts as a template which means defining structure will not allocate any memory to the variables it is just represents information (set of variables).
8. Structure name will be used for creating instance of a structure (declaring a variable of type structure).
9. Structure Definition is mostly written before starting of main function which can be used globally.(Globally means structure variables can be created anywhere in the program either in main function or User Defined Function).

❖ Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. Structure can be declared by two ways.

1. Tagged Declaration
2. Typedef Declaration

There are two ways to declare Tagged Declaration structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{
 int id;
   char name[50];
   float salary;
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure. Here, e1 and e2 can be treated in the same way as the objects in C++ and Java.

2nd way:

Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{
int id;
   char name[50];
   float salary;
}e1,e2;
```

❖ Which approach is good?

1. If number of variable are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.
2. If number of variable are fixed, use 2nd approach. It saves your code to declare a variable in main() function.

❖ Structure Initialization

Like a variable of any other datatype, structure variable can also be initialized at compile time.

```
struct Patient
{
   float height;
   int weight;
   int age;
};
struct Patient p1 = { 180.75 , 73, 23 };    //initialization
```

OR

```
struct Patient p1;

p1.height = 180.75;     //initialization of each member separately
p1.weight = 73;
p1.age = 23;
```

1.2.2 typedef and accessing structure member

Type Definitions (*typedef*)

The C programming language provides a keyword called **typedef**, which you can use to give a type a new name. Typedef is a keyword that is used to give a new symbolic name for the existing name in a C program. This is same like defining alias for the commands.

```
/* example of typedef */

#include<stdio.h>
#include<conio.h>
void main()
{
        typedef int LENGTH;
        LENGTH l, w, h;
        l=5,w=10,h=45;
        clrscr();
        printf("\n The value of Length is %d",l);
        printf("\n The value of Width is %d",w);
        printf("\n The value of Height is %d",h);
        getch();
}
```

Using typedef with structures

Syntax:

```
typedef struct Structure_name Structure_variable
```

Example:

```
typedef struct student status;
```

- When we use "typedef" keyword before struct <tag_name> like above, after that we can simply use type definition "status" in the C program to declare structure variable.
- Now, structure variable declaration will be, "status record".

- This is equal to "struct student record". Type definition for "struct student" is status. i.e. status = "struct student"

Structure declaration using typedef in c:

```
typedef struct student
{
        int mark [2];
        char name [10];
        float average;
} status;
```

To declare structure variable, we can use the below statements.

```
status record1;          /* record 1 is structure variable */
status record2;          /* record 2 is structure variable */
```

Example program for CStructure using typedef

```
// Structure using typedef
#include <stdio.h>
#include <string.h>

typedef struct student
{
  int id;
  char name[20];
  float percentage;
}status;
int main()
{
  status record;

  record.id=1;
strcpy(record.name, "Raju");
  record.percentage = 86.5;

printf(" Id is: %d \n", record.id);
printf(" Name is: %s \n", record.name);
  printf(" Percentage is: %f \n", record.percentage);
  getch();return 0;
}
```

OUTPUT:

```
Id is: 1
Name is: Raju
Percentage is: 86.500000
```

❖ Accessing members of the structure

There are two ways to access structure members:

1. By **.** (member or dot operator)
2. By **->** (structure pointer operator)

Let's see the code to access the *id* member of *p1* variable by . (member) operator.

```
p1.id
```

Example of structure in C language.

```
#include<stdio.h>
#include<string.h>
struct employee
```

```
{
int id;
   char name[50];
}e1;   //declaring e1 variable for structure
int main( )
{
   e1.id=101;                                  //store first employee information
  strcpy(e1.name, "Bhumika Patel");          //copying string into char array
  printf( "employee 1 id : %d\n", e1.id);    //printing first employee information
  printf( "employee 1 name : %s\n", e1.name);
  return 0;
}
```

OUTPUT :

```
employee 1 id : 101
employee 1 name : Bhumika Patel
```

Example of the structure in C language to store many employees information.

```
#include<stdio.h>
#include<string.h>
struct employee
{
int id;
   char name[50];
   float salary;
}e1,e2;                          // Declaring e1 and e2 variables for structure
int main( )
{
  e1.id=101;                                 // Initialize first employee information
  strcpy(e1.name, "Tony Stark");    //copying string into char array
  e1.salary=56000;
  e2.id=102;                                  // Initialize second employee information
  strcpy(e2.name, "James Bond");
  e2.salary=126000;

  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
  printf( "employee 1 salary : %f\n", e1.salary);

  //printing second employee information
  printf( "employee 2 id : %d\n", e2.id);
  printf( "employee 2 name : %s\n", e2.name);
  printf( "employee 2 salary : %f\n", e2.salary);
  return 0;
}
```

OUTPUT:

```
employee 1 id : 101
employee 1 name : Tony Stark
employee 1 salary : 56000.000000
employee 2 id : 102
employee 2 name : James Bond
employee 2 salary : 126000.000000
```

Defining Union

- Like structure, **Union in c language** is *a user-defined data type* that is used to store the different type of elements.
- At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.

Structure	Union

struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
	union Employee{
	char x; // size 1 byte
	int y; //size 2 byte
	float z; //size 4 byte
	}e1; //size of e1 = 4 byte

size of e1= 1 + 2 + 4 = 7	**size of e1= 4 (maximum size of 1 element)**

- **Advantage of union over structure**
  1. It occupies less memory because it occupies the size of the largest member only.
- **Disadvantage of union over structure**
  1. Only the last entered data can be stored in the union. It overwrites the data previously stored in the union.

The **union** keyword is used to define the union.

Syntax to define union in c.

```
union union_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

Example to define union for an employee in c language

```
union employee
{
 int id;
   char name[50];
   float salary;
};
```

Example of union in C language.

```
#include <stdio.h>
#include <string.h>
union employee
{
int id;
   char name[50];
```

```
}e1;  //declaring e1 variable for union
int main( )
{
  e1.id=101;//store first employee information
  strcpy(e1.name, "Bhumika Patel");//copying string into char array
  printf( "employee 1 id : %d\n", e1.id);      //printing first employee information
  printf( "employee 1 name : %s\n", e1.name);
  return 0;
}
```

Output:

```
employee 1 id : 1869508435
employee 1 name : Bhumika Patel
```

Note: As you can see, id gets garbage value because name has large memory size. So only name will have actual value.

1.2.3 Difference between structure and union

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Difference: Array v/s Structure

	Array	Structure
1.	An array is a derived data type.	A structure is a user-defined data type.
2.	An array is a collection of similar data type items.	A structure is a collection of different data type items.
3.	Memory for the array will be allocated at the time of its declaration.	Memory is allocated only at the time of structure variable declaration.
4.	Array elements are referred by array index.	Structure elements are referred by its variable name using period sign with

		structure variable.
4.	It is difficult to represent complex related data using array.	It is easy to represent complex and related data using structure.
5.	Array does not have any special keyword to declare it instead it used [] bracket to define array size along with data type.	"struct" keyword is used in structure declaration.

6.	Syntax:
	data type arr_name [size];
		Sytanx :
		struct struct_name
		{
		Type member1 ;
		Type member2 ;
		};

1.3 User defined functions:
What is C function?
A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

A function is a block of statements that performs a specific task. Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. In such case you have two options –
a) Use the same set of statements every time you want to perform the task
b) Create a function to perform that task, and just call it every time you need to perform that task.
Using option (b) is a good practice and a good programmer always uses functions while writing codes in C.

❖ **Benefits of Using the Function in C**
- The function provides modularity.
- The function provides reusable code.
- In large programs, debugging and editing tasks is easy with the use of functions.
- The program can be modularized into smaller parts.
- Separate function independently can be developed according to the needs.

❖ **Uses of C functions:**
1. C functions are used to avoid rewriting same logic/code again and again in a program.
2. There is no limit in calling C functions to make use of same functionality wherever required.
3. We can call functions any number of times in a program and from any place in a program.
4. A large C program can easily be tracked when it is divided into functions.
5. The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understand ability of very large C programs.

❖ **Types of functions**

1. **Built-in (Library) Functions:** The system provided these functions and stored in the library. Therefore it is also called Library Functions. e.g. scanf(), printf(), strcpy(), strlwr(), strcmp(), strlen(), strcat() etc. To use these functions, you just need to include the appropriate C header files.
2. **User Defined Functions:** These functions are defined by the user at the time of writing the program.

1.3.1 Function return type, parameter list, local function variables

1.3.2 Passing arguments to function

❖ **User-defined function(UDF) :** You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

(Note: function names are identifiers and should be unique)

How user-defined function works?

```
#include <stdio.h>
void functionName()      // UDF
{
   ... .. ...
   ... .. ...
}
int main()
{
   ... .. ...
   functionName();
   ... .. ...
}
```

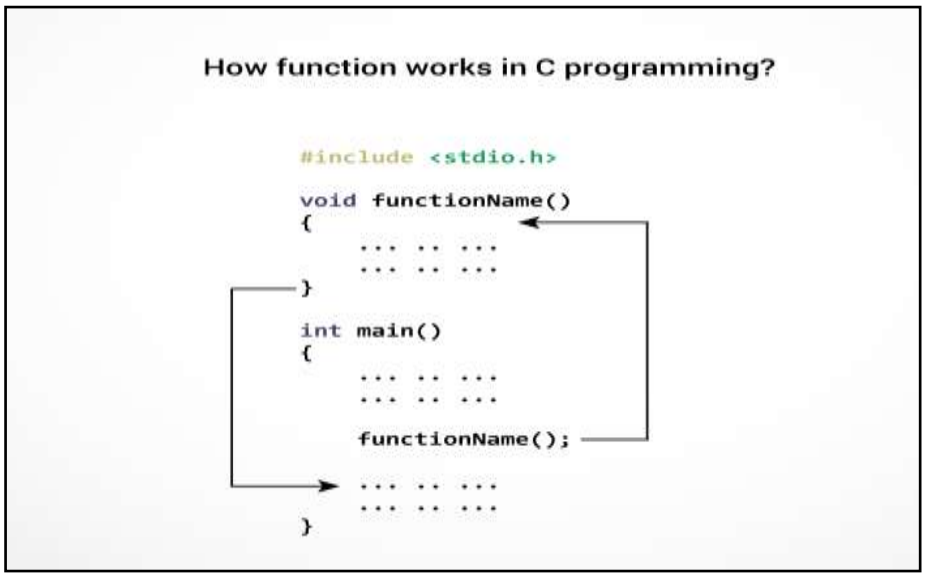
The execution of a C program begins from the main() function.

When the compiler encounters functionName();, control of the program jumps to

```
void functionName()
```

And, the compiler starts executing the codes inside functionName().

The control of the program jumps back to the main() function once code inside the function definition is executed.



❖ **Advantages of user-defined function**

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

❖ **Disadvantage of User Define Function**

1. It will take lots of extra time for program execution.

❖ **Parts of User Define Function**

There are three parts of User Define Function:

1. Function declaration or prototype – This informs compiler about the function name, function parameters and return value's data type.
2. Function call – This calls the actual function
3. Function definition – This contains all the statements to be executed.

1. Function declaration or prototype:

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.
A function prototype gives information to the compiler that the function may later be used in the program.

Syntax of function prototype:

```
returnType functionName(type1 argument1, type2 argument2, ...);
```

In the above example, int addNumbers(int a, int b); is the function prototype which provides the following information to the compiler:

1) name of the function is addNumbers()
2) return type of the function is int
3) two arguments of type int are passed to the function

Note: The function prototype is not needed if the user-defined function is defined before the main() function.

2. Calling a function

Control of the program is transferred to the user-defined function by calling it.

Syntax of function call:

```
functionName(argument1, argument2, ...);
```

In the above example, the function call is made using addNumbers(n1, n2); statement inside the main() function.

3. Function definition

Function definition contains the block of code to perform a specific task. In our example, adding two numbers and returning it.

Syntax of function definition:

```
returnType  functionName(datatype1 argument1, datatype2 argument2, ...)
{
        //body of the function
}
```

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

Example1: Creating a user defined function addnumbers()

```
#include<stdio.h>
int addnumbers(int num1,int num2);     //Function declaration or prototype
int main()
{
int var1, var2,sum;   clrscr();
    printf("Enter number 1: ");
    scanf("%d",&var1);
    printf("Enter number 2: ");
    scanf("%d",&var2);
    sum = addnumbers (var1, var2);    // function calling
    printf("Output: %d", sum);    getch();
return0;
}
int addnumbers (int num1,int num2)    // Function definition
{
int result;
    result = num1+num2;      // Arguments are used here
return result;
}
```

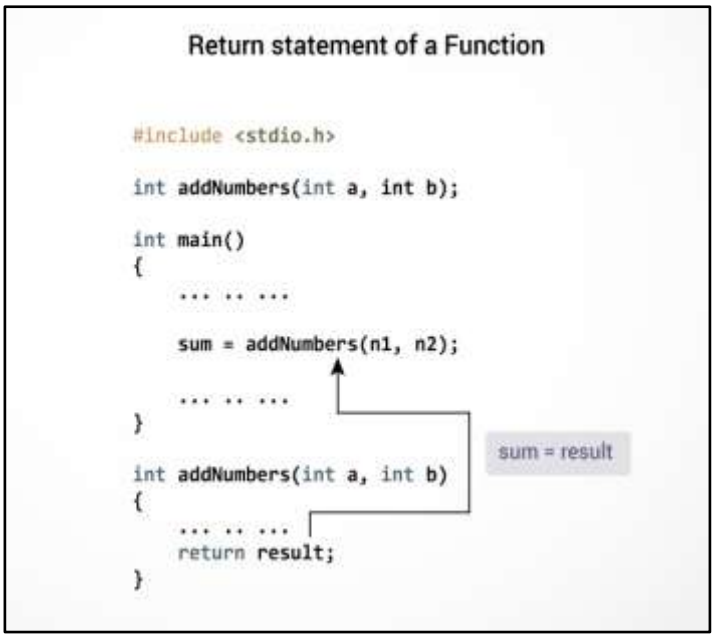
Output:

```
Enter number 1: 100
Enter number 2: 120
Output: 220
```

❖ Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.

In the above example, the value of the **result** variable is returned to the main function. The **sum** variable in the main() function is assigned this value.



Example2: Creating a void user defined function that doesn't return anything

```
#include<stdio.h>
/* function return type is void and it doesn't have parameters*/
void introduction()
{
   printf("HiI am Function \n");
   printf("U can call me any number of time.\n");
   printf("How are you?");
}
int main()
```

```
{
        introduction();/*calling function*/
return0;
}
```

Output:

```
Hi I am Function
U can call me any number of time.
How are you?
```

❖ **Few Points to Note regarding functions in C:**

1) main() in C program is also a function.
2) Each C program must have at least one function, which is main().
3) There is no limit on number of functions; A C program can have any number of functions.
4) return type: Data type of returned value. It can be void also, in such case function doesn't return any value.
5) A function can call itself and it is known as "Recursion".

1.3.3 Calling function from main() function or from other function.

Function call by value is the default way of calling a function in C programming. Before we discuss function call by value, lets understand the terminologies that we will use while explaining this:

Actual parameters: The parameters that appear in function calls.

Formal parameters: The parameters that appear in function declarations.

```
#include<stdio.h>
#include<conio.h>

/* function declaration */
int max(int num1, int num2);          //Formal parameters in function call

int main ()
{
 /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;    clrscr();
/* calling a function to get max value */
   ret = max(a, b);          //Actual parameters in function call
   printf( "Max value is : %d\n", ret);
  getch();     return 0;
}
/* function returning the max between two numbers */
int max(int num1, int num2)
{
 /* local variable declaration */
   int result;

   if (num1 > num2)
result = num1;
   else
result = num2;
```

```
  return result;
}
```

Example : **Calling function from other function.**

```
#include<stdio.h>
#include<conio.h>
/* function declarations */
float pi();
float cal_area(float rad);
int main ()
{
   /* local variable definition */
   float r,area;  clrscr();
   printf("Enter Radius value : ");
   scanf("%f",&r);

   area = cal_area(r);
   printf( "Area of circle is : %f \n", area );
getch();    return 0;
}
float pi()
{
   const float pi=3.14;
   return pi;
}
float cal_area(float rad)
{   /* local variable declaration */
float result;
result = pi() * rad * rad ;// calling function from other function
return result;
}
```

1.3.4 Function with No arguments and no return value, No arguments and a return value, with arguments and no return value, with arguments and a return value.ORTypes of User-defined Functions in C Programming

These 4 programs below check whether the integer entered by the user is an even number or not.

The output of all these programs below is the same, and we have created a user-defined function in each example. However, the approach we have taken in each example is different.

1: No arguments passed and no return value
2: No arguments passed but a return value
3: Argument passed but no return value
4: Argument passed and a return value

Example 1: No arguments passed and no return value

```
#include<stdio.h>
#include<conio.h>

void checkIsEven();//Function declaration or prototype
int main()
{
    clrscr();
    checkIsEven();    // argument is not passed
    getch();return 0;
}
void checkIsEven()    // return type is void meaning doesn't return any value
{
       int n;
       printf("Enter a positive integer: ");
       scanf("%d",&n);
       if(n%2 == 0)
              printf("\n %d is an even number.", n);
       else
             printf("\n %d is not an even number.", n);
}
```

Example 2: No arguments passed but a return value

```
#include<stdio.h>
#include<conio.h>
int GetInteger();//Function declaration or prototype
int main()
{
     int num;
     clrscr();
     num=GetInteger();    // argument is not passed
     if(num%2 == 0)
                printf("\n %d is an even number.", num);
     else
               printf("\n %d is not an even number.", num);
    getch();
    return 0;
}
int GetInteger()    // returns integer entered by the user
{
        int n;
        printf("Enter a positive integer: ");
        scanf("%d",&n);
        return n;
}
```

Example 3: Argument passed but no return value

```
#include<stdio.h>
#include<conio.h>
void GetEven(int n);//Function declaration or prototype
void main()
{
     int num;
     clrscr();
   printf("Enter a positive integer: ");
  scanf("%d",&num);
 GetEven(num);   // num(argument) is passed to the function
getch();
}
void GetEven(int n)  // function return type is void meaning doesn't return any value
{
     if(n%2 == 0)
               printf("\n %d is an even number.", n);
     else
               printf("\n %d is not an even number.", n);
}
```

Example 4: Argument passed and a return value

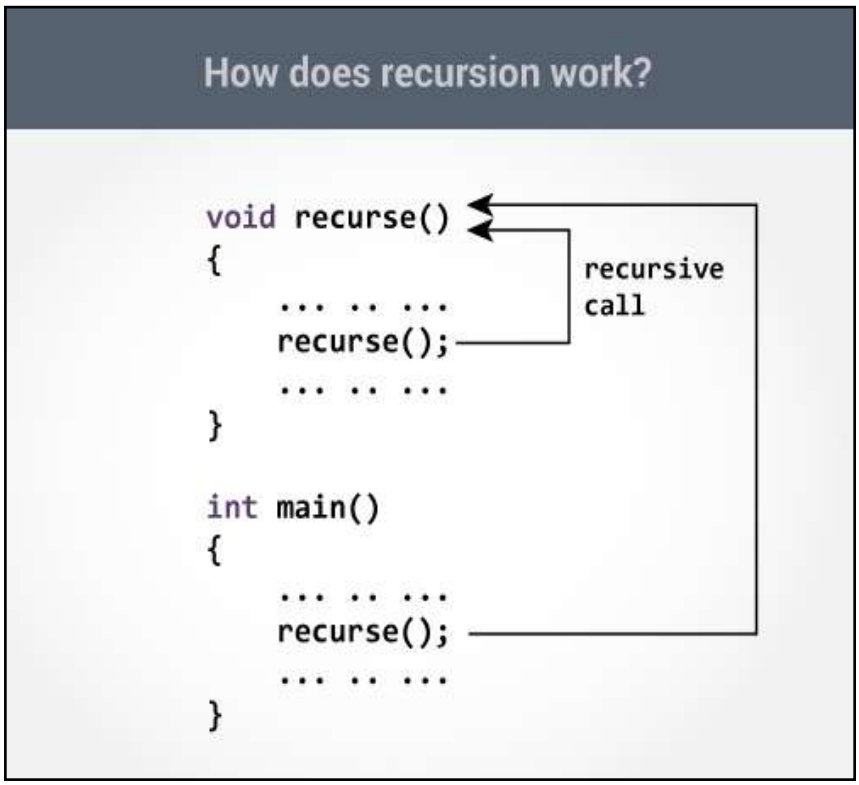
```
#include<stdio.h>
#include<conio.h>
int addnumbers(int num1, int num2);  //Function declaration or prototype
int main()
{
    int var1, var2,sum;
    clrscr();
    printf("Enter number 1: ");
    scanf("%d",&var1);
    printf("Enter number 2: ");
    scanf("%d",&var2);
    sum = addnumbers (var1, var2);    // function calling
    printf("Output: %d", sum);
    getch();
    return 0;
}
int addnumbers (int num1, int num2)  // Function definition
{
    int result;
    result = num1+num2;       // Arguments are used here
    return result;            // function  return int value
}
```

Which approach is better?

Well, it depends on the problem you are trying to solve. In this case, passing argument and returning a value from the function (**example 4**) is better.

1.3.5 Recursive Function

A function that calls itself is known as a recursive function. And, this technique is known as recursion.



- The recursion continues until some condition is met to prevent it.
- To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call, and other doesn't.
- Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.The following example ***calculates the factorial of a given number using a recursive function***

```
#include<stdio.h>
#include<conio.h>

int factorial(int i)
{
  if(i <= 1)
  {
    return 1;
  }
  return i * factorial(i - 1);
}
int  main()
{
int i = 5;   clrscr();
 printf("Factorial of %d is %d\n", i, factorial(i));
getch();   return 0;
}
```

Output:

Factorial of 5 is 120

Example2: Fibonacci Series using Recursive Function

```
#include<stdio.h>
#include<conio.h>

int fibonacci(int i)
{
  if(i == 0)
  {
    return 0;
  }
  if(i == 1)
  {
    return 1;
  }
     return fibonacci(i-1) + fibonacci(i-2);
}
int  main()
{
  int i;  clrscr();
  for (i = 0; i < 10; i++)
  {
    printf("%d \t", fibonacci(i));
  }
  getch();   return 0;
}
```

Output :

0 1 1 2 3 5 8 13 21 34

- **Advantages of Recursion**

1) Recursion is more elegant and requires few variables which make program clean.
2) Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type.

- **Disadvantages of Recursion**

1) In other hand, it is hard to think the logic of a recursive function. It is also difficult to debug the code containing recursion.

Unit-1-Practical Problems

Multidimensional Array

1. Write a Menu base C program with following options.
    1. Addition of two Matrices.
    2. Subtraction of two Matrices.
    3. Multiplication of two Matrices.
    4. Transpose of Single Matrix.
    5. Exit.
2. Write C program to count total duplicate elements in an array.
3. Write C program to check matrix is an identity matrix example.
4. Write C program to find sum of each column in a matrix.
5. Write C program to find sum of each row in a matrix.
6. Write C program to check two matrices are equal or not.
7. Write C program to find lower triangle matrix.
8. Write C program to find upper triangle matrix.
9. Write C program to find sum of each row and column of a matrix.

    10 20 30 60
    40 50 60 150
    70 80 90 240
    120 150 180

User defined Function

1. Write C Program to find area of rectangle using User defined Function.
2. Write C Program to find perimeter of rectangle using User defined Function.
3. Write C program to find area of a triangle using User defined Function.
4. Write C Program to find diameter, circumference and area of circle using User defined Function.
5. Write C program to convert centimeter to meter and kilometer using User defined Function.
6. Write C program to convert temperature from degree Celsius to Fahrenheit using UDF.
7. Write C Program to calculate simple interest using User defined Function.
8. Write C Program to find power of a number using User defined Function.(without pow())
9. Write C Program to find cube using function using User defined Function.
10. Write C program to check the given string is palindrome or not using UDF.
11. Write C Program to convert decimal number to binary number using the UDF.
12. Write C Program to get largest element of an array using the UDF.
13. Write C Program to find maximum and minimum element in array using UDF.
14. Write C program to sort numeric array in ascending or descending order using UDF.
15. Write C program to search element in an array using User define Function.
16. Write C program Menu base Calculator using UDF.
17. Write Menu base C Program with following option using UDF.
    1. To check given number is Prime or not
    2. To check given number is Armstrong or not.
    3. To check given number Perfect or not.
    4. Exit
18. Write C Program to find Sum of Series $1^2+2^2+3^2+.....+n^2$ Using Recursion in C.