

1.1 Concepts of Two-Dimensional Numeric Array:

- 1.1.1 Declaring Two-Dimensional numeric array
- 1.1.2 Two-Dimensional numeric Array operations (Addition, Subtraction, Multiplication, Transpose)
- 1.1.3 Element Address in array (Row major and Column major)
- 1.1.4 Two-Dimensional Character Array:
 - 1.1.4.1 Declaring & Initializing Two-Dimensional character array
 - 1.1.4.2 Two-Dimensional character Array operations (Searching elements, copying, merging, finding length of given string)

1.2 Concepts of structure and Union:

- 1.2.1 Defining, declaring and Initializing structure and Union
- 1.2.2 typedef and accessing structure member
- 1.2.3 Difference between structure and union

1.3 User defined functions :

- 1.3.1 Function return type, parameter list, local function variables
- 1.3.2 Passing arguments to function
- 1.3.3 Calling function from main() function or from other function.
- 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.
- 1.3.5 Recursive Function

1.1 Concepts of Two-Dimensional Numeric Array:

- When the list within variable is stored using two subscripts (rows and columns), then it is known as two dimensional array.

- Syntax:

Data_type array_name[rows][columns];

- Here row indicates the maximum numbers of horizontal elements and column indicates the maximum numbers of vertical elements.

- First array element is stored in array_name[0][0].

- Ex: int number[2,3];

- o If numbers are 5, 10, 15, 20, 25, 30 then the values are stored as

	0	1	2
0	0,0	0,1	0,2
1	1,0	1,1	1,2

	0	1	2
0	5	10	15
1	20	25	30

- number[0][0]=5;
 - number[0][1]=10;
 - number[0][2]=15;

- number[1][0]=20;
 - number[1][1]=25;
 - number[1][2]=30;

- Initialization:

- o Compile time:

- Ex: int a[2][2]={1,2},{3,4}};

- o Run time:

- Ex:

```
int a[2][2],i,j;
for(i=0;i<2;i++)
{
    for(j=0;j<2;j++)
    {
        a[i][j]=0;
    }
}
```

1.1.2 Two-Dimensional numeric Array operations

1) Taking Input and Display Output of Two-Dimensional Array

```
#include<stdio.h> //STANDARD INPUT OUTPUT
#include<conio.h> // CONSOLE INPUT OUTPUT
void main()
{
    int a[3][3],i,j;
    clrscr();
    printf("\n Enter matrix elements:");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n a[%d][%d]=",i,j);
            scanf("%d",&a[i][j]);
        }
    }
}
```

```
    }

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

2) Addition and Subtraction of Two-Dimensional Array

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3],i,j,b[3][3], add[3][3],sub[3][3];
    clrscr();
    printf("\n Enter first matrix elements:");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n a[%d][%d]=",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    printf("\n Enter second matrix elements:");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n b[%d][%d]=",i,j);
            scanf("%d",&b[i][j]);
        }
    }
    printf("\n----Matrix A-----");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf(" %3d",a[i][j]);
        }
        printf("\n");
    }
    printf("\n----Matrix B-----");

    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf(" %3d",b[i][j]);
        }
        printf("\n");
    }
    printf("\n Addition and Subtraction of two matrices:");
    for(i=0;i<3;i++)
```

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

3) Multiplication of Two-Dimensional Array:

Let us understand the multiplication of matrixes by the below figure:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$

$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

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

void main()
{
    int a[3][3],b[3][3],mul[3][3],i,j,k;
    clrscr();
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d",&a[i][j]);
```

```
    }
}
printf("enter the second matrix element=\n");

for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        scanf("%d",&b[i][j]);
    }
}

printf("multiply of the matrix=\n");

for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        mul[i][j]=0;

        for(k=0;k<3;k++)
        {
            mul[i][j]+=a[i][k]*b[k][j];
        }
    }
}

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

4) Transpose of Two-Dimensional Array

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[3][3],i,j,b[3][3];
    clrscr();
    printf("\n Enter matrix elements:");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n a[%d][%d]=",i,j);
```

```

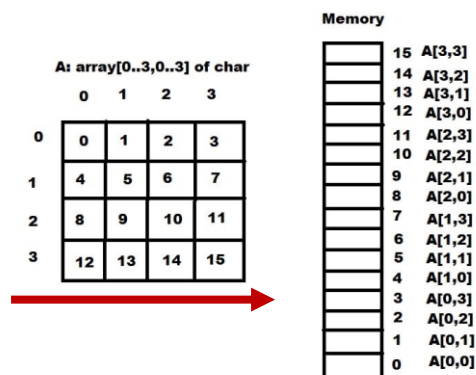
        scanf("%d",&a[i][j]);
    }
}
printf("\n----Matrix-----");
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        printf(" %3d",a[i][j]);
    }
    printf("\n");
}
printf("\n Transpose of matrix:");
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        b[j][i]=a[i][j];
    }
}
printf("\n----Matrix-----");
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        printf(" %3d",b[i][j]);
    }
    printf("\n");
}
getch();
}

```

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

Row Major Ordering in an array:

- Row major ordering assigns successive elements, moving across the rows and then down the columns, to successive memory locations.



- The formula to compute the Address (offset) for a two-dimension row-major ordered array as:

Address of A[i][j] = Base Address + W * (C * I + j)

Where,

- Base Address is the address of the first element in an array.

- W= is the weight (size) of a data type.
- C= is Total No of Columns.
- I= is the Row Number of an element whose address is to find out.
- J= is Column Number of an element whose address is to find out.

Example:

A matrix P[15][10] is stored with each element requiring 8 bytes of storage. If the base address at P[0][0] is 1400. Determine the address at P[10][7] when the matrix is stored in Row Major Wise.

Address of A[I][J] = Base Address + W * (C * I + j)

= 1400+8*(10*10+7)

= 1400+856

=2256

Program to store the array elements Row Major Order

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

void main()
{
    int i,j,arr[3][3];
    clrscr();

    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n Enter Element : ");
            scanf("%d",&arr[i][j]); //store the elements row major
        }
    }

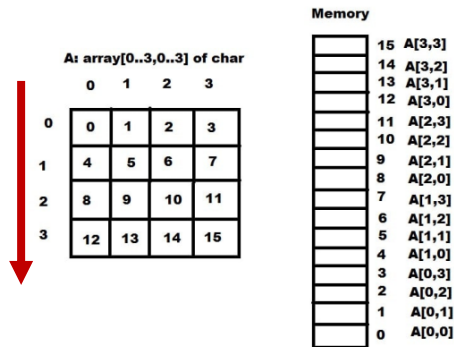
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf(" %d ",arr[i][j]);

        }
        printf("\n");
    }

    getch();
}
```

Column Major Ordering:

- If the element of an array is being stored in Column Wise fashion then it is called column-major ordering in an array.
- Pictorially, a column-major ordered array is organized as shown below:



The formula is as follows:

$$\text{Address of } A[I][J] = \text{Base Address} + W * (R * J + I)$$

Where ,

- Base Address is the address of the first element in an array.
- W= is the weight (size) of a data type.
- R= is Total No of Rows.
- I= is the Row Number of an element whose address is to find out.
- J= is Column Number of an element whose address is to find out.

Example:

A matrix P[15][10] is stored with each element requiring 8 bytes of storage. If the base address at P[0][0] is 1500. Determine the address at P[10][7] when the matrix is stored in Column Major Wise.

$$\begin{aligned} &= \text{Base Address} + W * (R * J + I) \\ &= 1500 + 8 * (15 * 7 + 10) \\ &= 1500 + 920 \\ &= 2420 \end{aligned}$$

Program to store the array elements column major order

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

void main()
{
    int i,j,arr[3][3];
    clrscr();

    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("\n Enter Element : ");
            scanf("%d",&arr[j][i]); //store the elements column major
        }
    }
}
```



```
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        printf(" %d ",arr[i][j]);

    }
    printf("\n");
}

getch();
}
```

1.1.4 Two-Dimensional Character Array:

- 2D character arrays are very similar to the 2D integer arrays.
- A 2D character array is more like a String array. It allows us to store multiple strings under the same name.
- (String is nothing but an array of characters which ends with a '\0'.)

Declaration and Initialization of a 2D character array

A 2D character array is **declared as follows**:

Syntax:

```
char variable_name[subscript1][subscript2];
```

Example:

```
char name[5][10];
```

Initialization of the character array occurs in this manner:

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

In the below diagram the memory allocation is explained for each of elements for 2-D character Array:

Memory location(base address)	Array elements									
25860	t	r	e	e	\0					
25870	b	o	w	l	\0					
25880	h	a	t	\0						
25890	m	i	c	e	\0					
25900	t	o	o	n	\0					

[5] names stored in 5 different memory locations

length of each String is [10]

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

1) Taking Data input and printing output

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

void main()
{
    char name[5][10];
    clrscr();
    for(i=0 ;i<5 ;i++ )
    {
        scanf("%s",&name[i]); // gets(name[i]);
    }

    for(i=0 ;i<5 ;i++)
    {
        printf("%s\n",name[i]); //puts(names[i]);
    }
}
```

2) Copying the elements from one array to another array to 2D Character Array

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

void main()
{
    int i,j;
    char arr1[5][10],arr2[5][10];
    clrscr();

    printf("\n Enter strings in new line: ");

    for(i=0;i<5;i++)
    {
        gets(arr1[i]);
    }

    //Loop to Copy elements from one array to another array

    for(i=0;i<5;i++)
    {
        strcpy(arr2[i],arr1[i]);
    }
}
```

```
printf("\n Elements copied in second array: \n");

for(i=0;i<5;i++)
{
    puts(arr2[i]);
}

getch();
}
```

3) Merging the elements from one array to another array to 2D Character Array

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

void main()
{
    int i,j,k,size1,size2,total;
    char arr1[10][10],arr2[10][10],merg[20][10];
    clrscr();

    printf("\n Enter size for 1st array: ");
    scanf("%d",&size1);

    printf("\n Enter size for 2nd array: ");
    scanf("%d",&size2);

    printf("\n Enter Array 1 strings in new line: ");

    for(i=0;i<size1;i++)
    {
        gets(arr1[i]);
        strcpy(merg[i],arr1[i]);
    }

    printf("\n Enter Array 2 strings in new line: ");
    k=i;

    for(j=0;j<size2;j++)
    {
        gets(arr2[j]);
        strcpy(merg[k],arr2[j]);
        k++;
    }

    total=size1+size2;
```

```
//loop to print the merg array elements

printf("\n Elements from Merg Array: \n");

for(i=0;i<total;i++)
{
    puts(merg[i]);
}
getch();
}
```

4) Finding length of each given strings from 2D Character Array

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

void main()
{
    int i,length=0;
    char arr1[5][10];
    clrscr();

    printf("\n Enter strings in new line: ");

    for(i=0;i<5;i++)
    {
        gets(arr1[i]);
    }

    //loop to print the array elements

    for(i=0;i<5;i++)
    {
        puts(arr1[i]);
    }
    //Loop to calculate length of each string in array

    for(i=0;i<5;i++)
    {
        length=strlen(arr1[i]);
        printf("\n Length of %s is %d",arr1[i],length);
        length=0;
    }
    getch();
}
```

5) Searching of Element in 2-Dimensional Character Array

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

void main()
{
    char arr1[5][10];
    char search[10];
    int i,j;
    int flag=0; //not found
    clrscr();

    printf("\n Enter Array Elements : \n ");

    for(i=0;i<5;i++)
    {
        gets(arr1[i]);
    }

    printf("\n Enter Elements to be searched : ");
    gets(search);

    for(i=0;i<5;i++)
    {
        if(strcmp(arr1[i],search)==0)
        {
            flag=1;
            break;
        }
    }
    if(flag == 1)
    {
        printf("\n Searched Element found");
    }
    else
    {
        printf("\n Searched Element not found");
    }
    getch();
}
```

1.2 Concepts of structure and Union:

Structure:

- A structure is a user defined data type in C.
- A structure is a collection of variables (can be of different types) under a single name.
- 'struct' keyword is used to create a structure.

1.2.1 How to define structures?

To define a struct, the **struct** keyword is used.

Syntax of struct

```
struct structureName
{
    dataType member1;
    dataType member2;
    ...
};
```

Example:

```
struct Person
{
    char name[50];
    int pincode;
    float salary;
};
```

1.2.1 How to Initialize Structure?

When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Example:

```
struct Person
{
    char name[50];
    int pincode;
    float salary;
};

void main()
{
    struct Person person1, person2, p[20];
}
```

Another way to create structure variable:

```
struct Person
{
    char name[50];
    int pincode;
    float salary;
} person1, person2, p[20];
```

Note: In both cases, two variables person1, person2, and an array variable p having 20 elements of type struct Person is called array of structure variable.

1) Addition of Two Structure Variable.

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

struct distance
{
    int feet;
    int inch;
}d1,d2,d3; // creating structure variable.

void main()
{
    clrscr();

    printf("\n Enter Feet: and inch: ");
    scanf("%d %d",&d1.feet,&d1.inch);

    printf("\n Enter Feet: and inch: ");
    scanf("%d %d",&d2.feet,&d2.inch);

    d3.feet=d1.feet+d2.feet;
    d3.inch=d1.inch+d2.inch;

    printf("\n Sum of two distance is %d feet %d inch",d3.feet,d3.inch);

    getch();
}
```

2) Array of Structure Variable.

```
//Program to prepare the marksheet of student using student structure.

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

struct stud
```

```
{
    int sid;
    char name[25];
    int m1,m2;
};

void main()
{
    struct stud s[3];
    int i,total,percent;
    clrscr();

    for(i=0;i<3;i++)
    {
        printf("\n Enter sid ");
        scanf("%d",&s[i].sid);

        printf("\n Enter sname: ");
        scanf("%s",s[i].name);

        printf("\n Enter Marks1 and Marks2: ");
        scanf("%d %d",&s[i].m1,&s[i].m2);
    }

    clrscr(); //to clear the input
    printf("\n SID:   Name:   M1:   M2:   Total   Percent");

    for(i=0;i<3;i++)
    {
        total=s[i].m1+s[i].m2;
        percent=(total*100)/200;
        printf("\n %-7d %-10s %-7d %-7d %-7d %-7d",s[i].sid,s[i].name,s[i].m1,s[i].m2,total,percent);
        total=0;
        percent=0;
    }
    getch();
}
```

1.2.1 Access members of a structure

There are two types of operators used for accessing members of a structure.

- 1) **.(dot)** Member operator
- 2) **-> (arrow)** Structure pointer operator

To access the salary of person2, use **.(dot)** operator as follows:

```
person2.salary
```


1.2.2 Keyword typedef

- We use the typedef keyword to create an alias name for data types.
- It is commonly used with structures to simplify the syntax of declaring variables.

```
struct Distance{
    int feet;
    float inch;
};

void main() {
    struct Distance d1, d2;
}
```

Is equivalent to

```
typedef struct Distance{
    int feet;
    float inch;
} distances;

void main() {
    distances d1, d2;
}
```

Union

- A union is a user-defined type similar to **structs** in C except for one key difference.
- Structs allocate enough space to store all its members where as unions allocate the space to store only the largest member.

Create union variables.

- When a union is defined, it creates a user-defined type.
- However, no memory is allocated when union are declared.
- To allocate memory for a given union type and work with it, we need to create variables.

How to create union?

```
union car
{
    char name[50];
    int price;
};
void main()
{
    union car car1;
}
```

Access members of a Union

There are two types of operators used for accessing members of union.

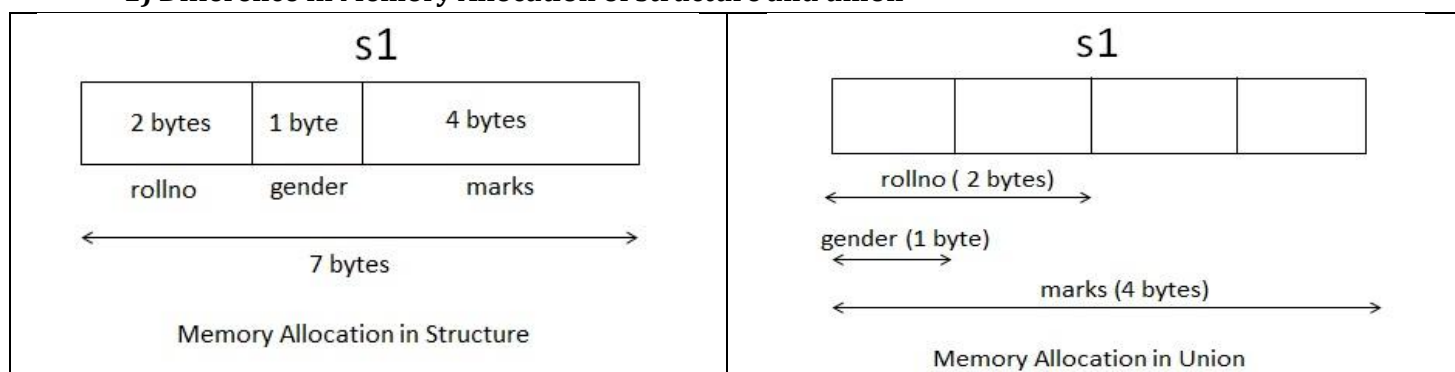
- 1) . (dot) Member operator
- 2) -> (arrow) union pointer operator

To access the price of car, use .(dot operator) as follows:

```
car.price;
```

1.2.3 Difference between Structure and Union:

1) Difference in Memory Allocation of 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.

1.2.3 Program for Difference between Structure and Union:

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

struct book
{
    int bid;
    char name[25];
}b1;

union product
{
    int pid;
    char pname[25];
}p1;

void main()
{
    clrscr();
    printf("\n struct size: %d",sizeof(b1));
    printf("\n union size: %d",sizeof(p1));
    getch();
}
```

Output:

```
Struct Size: 27
Union Size: 25
```

1.3 User defined functions:

- A function is a block of code that performs a specific task.
- C allows user to define functions.
- These functions are known as user-defined functions.

Elements of User Defined Functions

There are 3 elements for User Defined Functions as follows:

1. Function Declaration
2. Function Definition.
3. Function Call.

Each elements are explained as below:

1. User Defined Function Declaration:

```
returnType functionName(type1 parameter1,type2 parameter2, ...);
```

- Function declaration informs the compiler about the function name, parameters it accepts, and its return type.
- The actual body of the function can be defined separately.
- It's also called as Function Prototyping.
- Function declaration consists of **4 parts**.
 - 1) return type
 - 2) function name
 - 3) parameter list
 - 4) terminating semicolon

1) return Type

- Return type specifies the type of value (int, float, char, double) that function is expected to return to the program which called the function.

Note: In case, if function doesn't return any value, the return type would be **void**.

2) functionName

- Function name is an identifier and it specifies the name of the function.
- The function name is any valid C identifier and therefore must follow the same naming rules like other variables in C language

3) parameter list

- The parameter list declares the type and number of arguments that the function expects when it is called.

4) Terminating Semicolon

- The terminating semicolon is important as it instructs the compiler that declaration is completed here.

2. User Defined Function Definition

The general syntax of function definition is:

```
returntype functionName(type1 parameter1, type2 parameter2,...)
{
    // function body goes here
}
```

Note: While defining a function, there is no semicolon(;) after the parenthesis in the function header, unlike while declaring the function or calling the function.

Function Body

- The function body contains the declarations and the statements (algorithm) necessary for performing the required task.

- The body is enclosed within curly braces { } and consists of **three parts**.
 - 1) **local variable** declaration(if required) variables which are used inside the function body.
 - 2) **function statements** to perform the task inside the function.
 - 3) **a return statement** to return the result evaluated by the function(if return type is void, then no return statement is required).

3.User Defined Function Call

A function can be called by simply using function name followed by the list of actual parameters separated by comma and within the round brackets ()

For Example:

```
void main()
{
    int ans;
    ans=multiply(5,6); // here multiply function is called
    printf("Answer is %d",ans);
}
```

Example for user defined function:

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

int addNumbers(int a, int b);    // function declaration

void main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    sum = addNumbers(n1, n2);    // function call
    printf("sum = %d",sum);
}

int addNumbers(int a, int b)    // function definition
{
    int result;
    result = a+b;
    return result;              // return statement
}
```

Passing arguments to a function

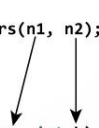
How to pass arguments to a function?

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}
```

A diagram with two arrows pointing from the arguments 'n1' and 'n2' in the function call 'sum = addNumbers(n1, n2);' in the main function to the parameters 'a' and 'b' in the function definition 'int addNumbers(int a, int b)'. The arrows indicate the flow of data from the caller to the callee.

Return Statement

- The return statement terminates the execution of a function and returns a value to the calling function.

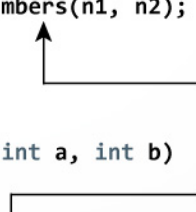
Return statement of a Function

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    return result;
}
```

A diagram showing a box labeled 'sum = result' next to the 'return result;' statement in the 'addNumbers' function. An arrow points from this box to the 'sum = addNumbers(n1, n2);' line in the 'main' function, illustrating how the return value is passed back to the caller.

sum = result

Syntax of return statement

return (expression);

For example,

return a;

```
return (a+b);
```

NOTE:

- **Return statement can contain a value for an expression.**
- The type of value returned from the function and the return type specified in the function prototype and function definition must match.

1.3.4 Function with no arguments and no return type

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

void welcomeMsg(); //function declaration

void main()
{
    clrscr();
    welcomeMsg(); //function call
    getch();
}

void welcomeMsg()
{
    printf("\n Welcome to C Programming");
}
```

1.3.4 Function with no arguments and return type

```
#include <stdio.h>
int area(); //function declaration
int main()
{
    int square_area;
    square_area = area(); //function call
    printf("Area of Square = %d",square_area);
    return 0;
}
int area()
{
    int square_area,square_side;
    printf("Enter the side of square :");
    scanf("%d",&square_side);
    square_area = square_side * square_side;
    return square_area;
}
```

1.3.4 Function with arguments and no return type

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

```
void area( int square_side); //function declaration
void main()
{
    int square_side;
    printf("Enter the side of square :");
    scanf("%d",&square_side);
    area(square_side); //function call
}

void area(int square_side)
{
    int square_area;
    square_area = square_side * square_side;
    printf("Area of Square = %d",square_area);
}
```

1.3.4 Function with arguments and return type

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

int addNumbers(int a, int b);    // function declaration

void main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    sum = addNumbers(n1, n2);    // function call
    printf("sum = %d",sum);
}

int addNumbers(int a, int b)    // function definition
{
    int result;
    result = a+b;
    return result;              // return statement
}
```

Passing Arrays to User Defined Function

Example to pass one dimensional array to call a function

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



```
int largest(int a[],int size);

void main()
{
    int a[5]={1,2,3,4,5};
    int ans;
    clrscr();

    ans=largest(a,5); // while calling function pass only name of array.

    printf("\n Largest Value is %d",ans);
    getch();
}

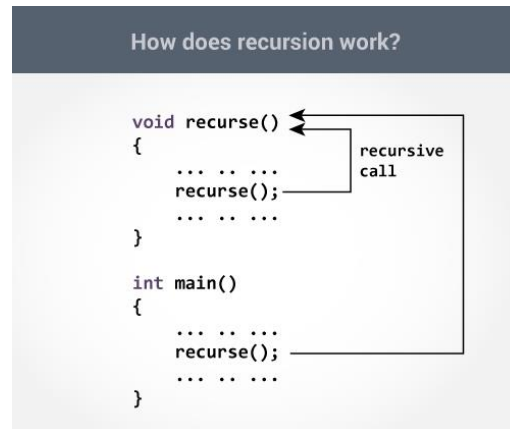
int largest(int a[],int size)
{
    int i;
    int max;
    max=a[0];

    for(i=0;i<size;i++)
    {
        if(max < a[i])
        {
            max=a[i];
        }
    }

    return max;
}
```

1.3.5 Recursive Function

- A function that calls itself is known as a recursive function. And, this technique is known as recursion.
- The C programming language supports recursion, i.e., a function to call itself.
- But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.
- Recursive functions are very useful to solve many mathematical problems, such as calculating the sum of numbers in series , factorial of a number, generating Fibonacci series, etc.



```
#include <stdio.h>
#include <conio.h>
int sum(int n);

void main() {
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
}

int sum(int n) {
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}
```

Output:

```
Enter a positive integer:3
sum = 6
```

Parameters OR Arguments [vimp]

- Parameter is the information that are passed into a function.
- Parameters are of 2 types:
 1. Actual parameters.
 2. Formal Parameters.

Sr No	Actual Parameter	Formal Parameter
1	Actual Parameters are the values that are passed to the function when it is called .	Formal Parameters are the variable listed inside the parentheses in the function definition.
2	Example: <pre>void main() { int n1,n2; printf("Enter n1 and n2 "); scanf("%d %d",&n1,&n2); sum(n1,n2); // here n is actual parameter }</pre>	Example: <pre>void sum(int a,int b) { printf("Sum is %d",(a+b)); } // here a and b are called formal parameters</pre>

Variables: [vimp]

- A variable in C is a named memory location that can store values.
- The values can be of any data type supported by the language, such as integer, string, etc.
- Variables in C are of **2 types**:
 1. Global
 2. Local

Sr NO	Local Variable	Global Variable
1	Local variables in C are those variables that have been declared inside of any scope (i.e. inside {})..	A global variable in C is not bounded by any scope and is usually declared below header file include statements.
2	A scope can be a function, loop, block, structure, or anything that has its own body. Since local variables are declared inside a scope, it is only accessible inside that particular scope only.	It can be accessed anywhere throughout the program.
3	Example: <pre>#include<stdio.h> void main() { int a; // local variable }</pre>	Example: <pre>#include<stdio.h> int number; //global variable int x=5; //global variable void main() { }</pre>

Structure Within Structure

- In C, a structure declaration can be placed inside another structure. This is also known as nesting of structure.
- Structure within structure means nesting of structure.
- The declaration is same as the declaration of data type in structure.
- Structure within structure (or) nesting of structure is used to create complex records.

Example

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

struct student
{
    char name[25];

    struct result
    {
        int m1,m2,m3;
        float percent;
    }res1;
}stud1;

void main()
{
    clrscr();

    printf("\n Enter Name: ");
    scanf("%s",stud1.name);

    printf("\n Enter 3 subject Marks: ");
    scanf("%d %d %d",&stud1.res1.m1,&stud1.res1.m2,&stud1.res1.m3);

    stud1.res1.percent = (stud1.res1.m1+stud1.res1.m2+stud1.res1.m3)/3;

    printf("\n Percentage is %.2f",stud1.res1.percent);

    getch();
}
```