Functions

In a C programming language a function is a group of statements having a name that together perform a special task and may take some input and produce output. In every C program has at least one function, which is **main()**.

C function is a self-contained block of statements that can be executed repeatedly whenever we need it.

In a C library has a numerous predefined or built-in functions. For example, **strcat()** to concatenate two strings, main() from where program execution starts.

Note, function names are identifiers and should be unique.

In a C programming function has a 3 part:

- Function declaration
- Function definition
- Function call

Function declaration : function **declaration** tells the compiler about a function's name, return type, and parameters.

A syntax of function declaration –

```
return_type   function_name( parameter list );
```

Example of the function declaration is as follows −

```
int max(int num1, int num2);
```

here int is a return type of function which tells function will return integer value and max is a name of the function. " int num1, int num2" is a parameter list which tells max function will receive two integer arguments. Arguments are optional which depends on program requirements.

Function definition : function **definition** is an actual body of the function.

Syntax of function definition:

```
return_type function_name( parameter list )
{
  body of the function
}
```

function **definition** consist a return type, function name, parameter list and body of the function.

Function call : whenever program calls a function, the flow of control is transferred to the called function definition. A called function performs a defined task and when flow of control reached to function-ending closing brace, flow of control returns to back to the main program.

Syntax of function call:

```
function_name( parameter list ) ;
max(num1, num2);
```

There are two types of functions in C

- Built-in(Library) Functions
- User Defined Functions

Built-in(Library) Functions

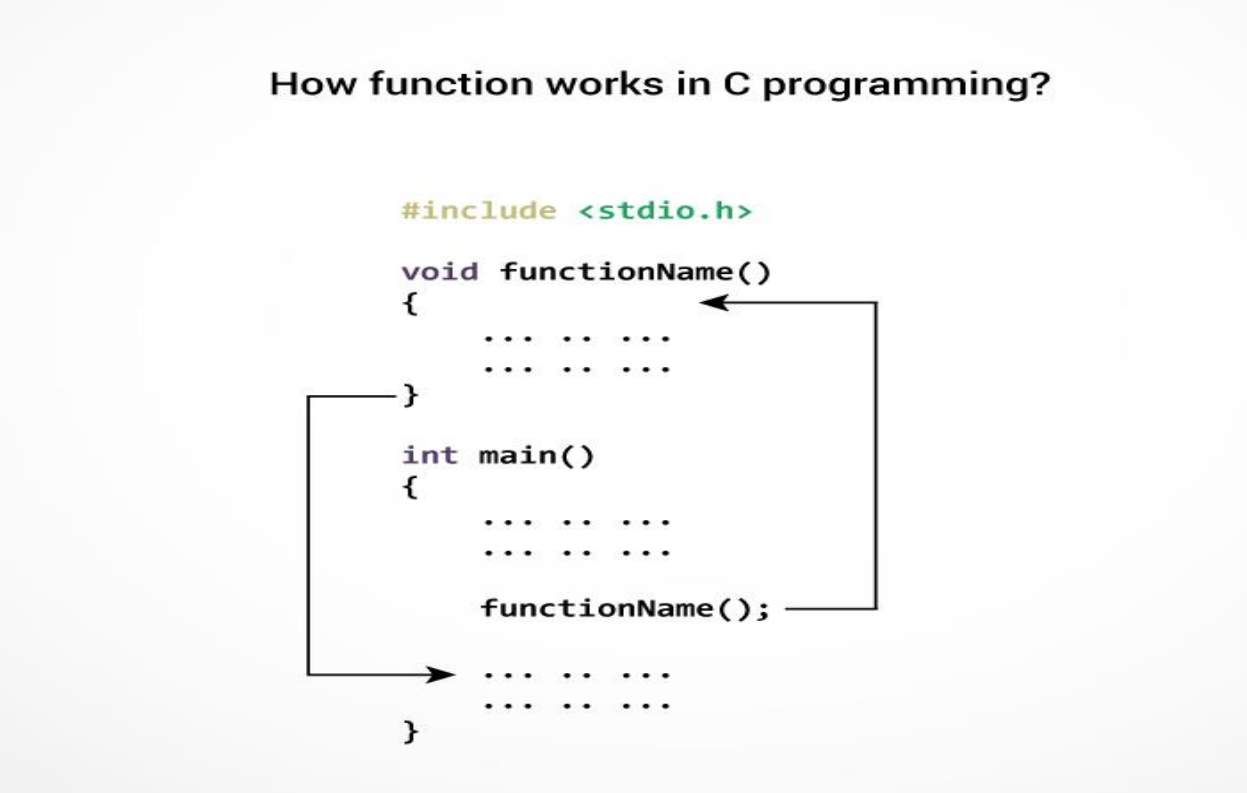
- The system provided these functions and stored in the library. Therefore it is also called *Library Functions*.
  e.g. scanf(), printf(), strcpy(), strcmp(), strlen(), strcat() etc.
- To use these functions, you just need to include the appropriate C header files.
  - printf() shows the output in the user's format.
  - scanf() used to take the user's input which can be a character, numeric value, string,etc.

- getchar() takes character input from the user.
- gets() reads a line.
- Strcpy()Copies string s2 into string s1.
- strcat(s1, s2) Concatenates string s2 onto the end of string s1.
- strlen(s1) Returns the length of string s1.
- strcmp(s1, s2) function compare the string.

- To use these functions, you just need to include the appropriate C header files.

User Defined Functions

- User Defined Functions : These functions are defined by the user at the time of writing the program



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.

The advantages of using functions are:

- Avoid repetition of codes.
- Increases program readability.
- Divide a complex problem into simpler ones.
- Reduces chances of error.
- Modifying a program becomes easier by using function.
  - It reduces the complexity of a program and gives it a *modular structure*.

Function categories

Function can be categories as:

- Function with no argument, no return value
- Function with argument but no return value
- Function with argument and return value
- Function with no argument, but return value

Function with no argument & no return value

Example: Write a program to find the area of circle using function with no argument, no return value.

The formula for the area of the circle is :

```
Area_circle = Π * r * r
```

Program:

```
#include <stdio.h>

void areaOfcircle();              //Function declaration
```

```
 void  main()
{

        areaOfcircle();          // Function Call

}

void areaOfcircle()              // function definition to calculate area of circle
{

  float area, radius;
                printf("Enter the radius of circle : ");
                scanf("%f", &radius);                    // take radius as input

                area = 3.14 * radius * radius;

                printf("\n Area of circle= : %.2f",  area);

}
```

Output:

Enter the radius of circle

5 // enter by user

Area of circle=78.50

Function with argument but no return value

Example: Write a program to find the area of circle using function with argument but no return value.

```
#include <stdio.h>
void areaOfcircle(float r);              //Function declaration with argument

void main()
{       float radius;

                printf("Enter the radius of circle : ");
                scanf("%f", &radius);            // take radius as input

                areaOfcircle(radius);                    // Function Call

   }
```

```
void areaOfcircle(float r)              // function definition to calculate area of circle
{       float area;

                area = 3.14 * r * r;
                printf("\n Area of circle : %.2f",  area);

}
```

Output:

Enter the radius of circle

5 // enter by user

Area of circle=78.50

Function with argument & return value

Example: Write a program to find the area of circle using function with argument but return value.

```
#include <stdio.h>
float areaOfcircle(float r);            //Function declaration with argument

int main()
{       float radius, a;

                printf("Enter the radius of circle : ");
                scanf("%f", &radius);           // take radius as input

                a = areaOfcircle(radius);                       // Function Call
                printf("\n Area of circle : %.2f",  a);

    return 0;

}

float areaOfcircle(float r)             // function definition to calculate area of circle
{       float area;

                area = 3.14 * r * r;
        return area;
}
```

Output:

Enter the radius of circle

5 // enter by user

Area of circle=78.50

Function with no argument but return value

Example: Write a program to find the area of circle using function with no argument but return value.

```
#include <stdio.h>
float areaOfcircle();                    //Function declaration


int main()
{       float a;
        a= areaOfcircle();          // Function Call
        printf("\n Area of circle : %.2f",  a);
return 0;

}

float areaOfcircle(){          // function definition to calculate area of circle

        float area, radius;
        printf("Enter the radius of circle : ");
        scanf("%f", &radius);                    // take radius as input

        area = 3.14 * radius * radius;
return area;                             // return value

 }
```

Output:

Enter the radius of circle

5 // enter by user

Area of circle=78.50

Example: WAP to find the factorial by using function (function with no argument & no return value).

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

void fact();              //Function declaration without argument

void main()
{

        fact();           // Function Call

}

void fact()               // function definition to calculate factorial
{
         int f=1, num, i;
        printf("Enter number to find factorial \n");
        scanf("%d", & num);          // number enter by user to find factorial

        for(  i= 1; i<=num;  i++)
            f = f  * i ;

         printf(" factorial is = %d",  f);
}
```

OUTPUT: Enter number to find factorial
5 // enter by user
factorial is = 120

Example: WAP to find the factorial by using function (function with argument & no return value).

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

void fact(int num);              //Function declaration without argument

void main()
{       int n;

        printf("Enter number to find factorial \n");
        scanf("%d", & n);              // number enter by user to find factorial
        fact( n );                     // Function Call

}

void fact(int num)               // function definition to calculate factorial
{
         int f, i;
         f=1;
        for(  i= 1; i<=num;  i++)
           f = f  * i ;

        printf(" factorial is = %d",  f);
}
```

OUTPUT: Enter number to find factorial
5 // enter by user
factorial is = 120

Example: WAP to find the factorial by using function (function with no argument and return value).

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

int fact();                  //Function declaration without argument

void main()
{ int f;

       f =  fact();                    // Function Call
```

```
        printf(" factorial is = %d",  f);

}

void fact()                 // function definition to calculate factorial
{
          int f, num, i;
          f=1;
        printf("Enter number to find factorial \n");
        scanf("%d", & num);             // number enter by user to find factorial

        for(  i= 1; i<=num;  i++)
            f = f  * i ;

     return f;
}
```

OUTPUT: Enter number to find factorial
5 // enter by user
factorial is = 120

Example: WAP to find the factorial by using function (function with argument and return value).

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

int fact(int num);                  //Function declaration without argument

void main()
{ int f, n;

        printf("Enter number to find factorial \n");
        scanf("%d", & n);               // number enter by user to find factorial
        f =  fact( n );                 // Function Call

        printf(" factorial is = %d",  f);

}

int fact(int  num)                  // function definition to calculate factorial
{
          int f=1,  i;
        for(  i= 1; i<=num;  i++)
            f = f  * i ;
```

```
    return f;
}
```

OUTPUT: Enter number to find factorial
5 // enter by user
factorial is = 120

Example: WAP to find the cube of any number using function.

```
#include <stdio.h>

int cube(int num);    /* Function declaration */


void main()
{
    int num,  c;

    /* Input number to find cube from user */
    printf("Enter any number: \n");
    scanf("%d", &num);

    c = cube(num);

    printf("Cube of entered number is= %d ", c);

}

int cube(int num)        // Function to find cube of any number
{
    return (num * num * num);
}
```

OUTPUT

```
Enter any number:
5
Cube of entered number is=   125
```

Example: C program to find maximum and minimum using functions.

```
#include <stdio.h>
void max_min(int num1, int num2);          // Function declarations

void main()
{
    int num1, num2;
```

```
    /* Input two numbers from user */
    printf("Enter any two numbers: ");
    scanf("%d%d", &num1, &num2);

    max_min(num1, num2);                    // Call max&min function
    getch();
}



void max_min(int num1, int num2)
{
    if( num1 > num2)
    {    printf("\nMaximum value is= %d" , num1);
         printf("\n Minimum value is= %d" , num2);

     }
     else
     {    printf("\nMaximum value is= %d" , num2);
          printf("\n Minimum value is= %d" , num1);
      }
}
```

OUTPUT

```
Enter any two number:
5
10
Maximum value is= 10
Minimum value is = 5
```

Example: C program to check even or odd using functions

```
#include <stdio.h>
void odd_even(int num);        // Function declarations

void main()
{
    int num1;

    /* Input a number from user to check number is even or odd */
    printf("Enter any number: \n ");
    scanf("%d", &num1);

    odd_even(num1);                    // Call max&min function
    getch();
}
```

```
void odd_even(int n)
{
    if(n%2==0)
       printf("%d is even number",n);
    else
      printf("%d is odd number",n);
}
OUTPUT
Enter any number:
5
5 is odd number
```

Example: Prime number using function in C.

```
#include<stdio.h>
int check_prime(int a);

void main()
{
   int n, result;

   printf("Enter an integer to check whether it is prime or not.\n");
   scanf("%d",&n);

   result = check_prime(n);

   if ( result == 1 )
      printf("%d is prime.\n", n);
   else
      printf("%d is not prime.\n", n);

   getch();
}

int check_prime(int a)
{
   int c;

   for ( c = 2 ; c <= a - 1 ; c++ )
   {
      if ( a%c == 0 )
     return 0;
   }
   return 1;
```

```
}
```

```
OUTPUT: Enter an integer to check whether it is prime or not
        7
        7 is prime .
```

Example: C Program to find prime numbers in a given range

```
#include<stdio.h>
void check_prime(int n1, int n2);

void main()
{
   int n1,n2;

   printf("Enter two range(input integer numbers only)");
   scanf("%d%d",&n1,&n2);

   check_prime(n1,n2);
   getch();
}

void check_prime(int num1, int num2)
{
   int i,flag,j;
    printf("Prime numbers from %d and %d are:  ", num1, num2);
    for(i = num1 ; i <= num2 ; i++)
    {
       flag = 0;
       for(j=2; j<=i/2; ++j)
       {
          if(i%j==0)
          {
             flag=1;
             break;
          }
       }
       if(flag==0)
          printf("\n %d",i);
   }
}
```

```
OUTPUT
Enter two range(input integer numbers only)
1
20
Prime numbers from 1 and 20 are:
2
3
5
```

```
7
11
13
17
19
```

Example: A C Program to Find Armstrong Number

Armstrong Number - ***An Armstrong Number is a Number which is equal to it's sum of digit's cube***. For example - **153 is an Armstrong number: here 153 = (1*1*1) + (5*5*5) + (3*3*3)**.

This program will **read an integer number and check whether it is Armstrong Number or Not**, to check Armstrong number, we have to calculate sum of each digit's cube and then compare number is equal to Sum or not. If Number and Sum of digit's cube then Number will be an Armstrong Number otherwise not.

```
#include<stdio.h>
int armstrong(int );

void main()
{
   int n, result;

  printf("Enter an integer to check whether it is Armstrong or not.\n");
  scanf("%d",&n);

  result = armstrong(n);

   if ( result == n )
      printf("%d is armstrong .\n", n);
   else
      printf("%d is not Armstrong .\n", n);

   getch();
}

int armstrong(int num)
{
   int rem, sum=0;
          while(num!=0)
          {
          rem=num%10;
          sum=sum + (rem*rem*rem);
          num= num/10;
```

```
            }

    return sum;
}
OUTPUT:
Enter an integer to check whether it is Armstrong or not
153
153 is Armstrong
```

Example: C program to check perfect number or not using function.

Perfect number will be equal to the sum of all its positive divisior ***excluding the number itself*** . For example: 6 is a Perfect number (that is, 1 + 2 + 3 = 6) because it's proper divisors are 1, 2, 3 and it's sum is equals to 6. 24 is a Perfect number (that is, 1 + 2 + 4 + 7 + 14 = 28). The b

```
#include<stdio.h>
int  perfect(int);                    // function declaration
void main()
{
       int  num, s;
      printf("Give an integer number: ");
       scanf("%d", &num);

       s = perfect(num);              // perfect function call

       if(s == num)
       {       printf("\nThe given number  is a perfect number");   }
       else
       {    printf("\nThe given number not a perfect number");    }
       getch();
 }

  int perfect(int number)                    // function definition
{
       int a=1, sum=0;
       while(a< number)
       {
                    if(number % a == 0)
                    {          sum=sum+a;              }
       a++;
       }
  return(sum);
  }
OUTPUT:
Give an integer number:
6
The given number  is a perfect number
```

Call by value & Call by reference

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. These two ways are generally differentiated by the type of values passed to them as parameters.

The parameters passed to function are called *actual parameters* whereas the parameters received by function are called *formal parameters*.

Call by value : Whenever we call a function and passes the value of variable to the called function is called "call by value". In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

```
#include <stdio.h>
void swap(int x, int y);                    /* function declaration */

 int main ()
 {
   int a = 10;
   int b = 20;

   printf(" Value of 'a' before swap=  %d\n", a );
   printf("Value of 'b' before swap= : %d\n", b );

   swap(a, b);              // calling a function  swap & passes the value of variable

   printf("Value of 'a' after swap= %d\n", a );
   printf("Value of 'b' after swap= %d\n", b );

   return 0;
}

void swap(int x, int y)                     // function definition to swap the values
{
   int temp;

   temp = x;
   x = y;
   y = temp;
```

```
  return;
}
```

Output:

```
Value of 'a' before swap=  10
Value of 'b' before swap=  20
Value of 'a' after swap= 10
Value of 'b' after swap= 20
```

Call by reference : Whenever we call a function and passes the reference/address of variable to the called function is called call by reference. In the call by reference, to pass the reference of the variable we use pointer. Both the actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

```
#include <stdio.h>
 void swap(int *x, int *y);                /* function declaration */
int main ()
{
   int a = 10;
   int b = 20;

   printf(" Value of 'a' before swap=  %d\n", a );
   printf("Value of 'b' before swap= : %d\n", b );

   swap(&a, &b);         //call function swap and passes the reference of variable

    printf("Value of 'a' after swap= %d\n", a );
   printf("Value of 'b' after swap= %d\n", b );

   return 0;
}

void swap(int *x, int *y)                  // function definition to swap the values
{          int temp;

           temp = *x;
           *x = *y;
           *y = temp;
     return;
```

}

Output:

Value of 'a' before swap= 10
Value of 'b' before swap= : 20
Value of 'a' after swap= 20
Value of 'b' after swap= 10

Difference between call by value and call by reference.

CALL BY VALUE	CALL BY REFERENCE
While calling a function, we pass values of variables to it. Such functions are known as “Call By Values”.	While calling a function, instead of passing the value of variables, we pass address of variables(location o variables) to the function known as “Call B References.
In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in th calling function are copied into the dummy variables o the called function.
With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.	With this method, using addresses we would have a access to the actual variables and hence we would b able to manipulate them.

```
// C program to illustrate
// call by value

#include <stdio.h>

// Function Prototype
void swapx(int x, int y);

// Main function
int main()
{
   int a = 10, b = 20;

   // Pass by Values
```

```
// C program to illustrate
// Call by Reference

#include <stdio.h>

// Function Prototype
void swapx(int*, int*);

// Main function
int main()
{
   int a = 10, b = 20;

   // Pass reference
   swapx(&a, &b);
```

CALL BY VALUE	CALL BY REFERENCE

swapx(a, b);
 	
printf("a=%d b=%d\n", a, b);
 	
return 0;
}
 	
// Swap functions that swaps
// two values
void swapx(int x, int y)
{
int t;
 	
t = x;
x = y;
y = t;
 	
printf("x=%d y=%d\n", x, y);
}
Output:
x=20 y=10
a=10 b=20
	printf("a=%d b=%d\n", a, b);
	
	return 0;
	}
	
	// Function to swap two variables
	// by references
	void swapx(int* x, int* y)
	{
	int t;
	
	t = *x;
	*x = *y;
	*y = t;
	
	printf("x=%d y=%d\n", *x, *y);
	}
	Output:
	x=20 y=10
	a=20 b=10

Thus actual values of a and b remain unchanged even after exchanging the values of x and y.	Thus actual values of a and b get changed aft exchanging values of x and y.
In call by values we cannot alter the values of actual variables through function calls.	In call by reference we can alter the values of variable through function calls.
Values of variables are passes by Simple technique.	Pointer variables are necessary to define to store th address values of variables.

Passing array to function

Example 1: Passing single element of an array to function using call by value method

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

```
void display();

void main()
{
          int arr[] = {2, 3, 4};
          display(arr[2]);                    //Passing array element ageArray[2]
         getch();
}

void display(int a)
{

   printf("%d", a);

}
```

Output: 4

Passing array to function using call by reference

When we pass the address of an array while calling a function then this is called function call by reference. When we pass an address as an argument, the function declaration should have a pointeras a parameter to receive the passed address.

```
#include <stdio.h>
void print1();
void  main()
{   int i;
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    for (i=0; i<10; i++)
    {
       print1(&arr[i]);                    /* Passing addresses of array elements*/

    }
getch();
}

void print1( int *num)
{
   printf("%d ", *num);
}
```

Output: 1 2 3 4 5 6 7 8 9 10

Example 2: Passing an entire array to a function

```
#include <stdio.h>
void print(int a[]);
void main()
{
        int age[] = {1, 2, 3, 4, 5};
        print(age); // calling function print & pass entire array as an argument
        getch();
}
void print(int a[])
{
        int i;
                for (i = 0; i < 5; ++i) {
            printf("%d\t", a[i]);
        }
}
```

Output: 1 2 3 4 5

Recursion

In a C programming language, a function that calls itself is known as a recursive function. And, this technique is called as recursion. Recursive functions are very useful to solve many problems such as Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

```
void recurse()
{
   ... .. ...
  recurse();
  ... .. ...
}

int main()
{
 ... .. ...
 recurse();
 ... .. ...
}
```

base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.

Example: Write a program in a C language to find the factorial using recursion.

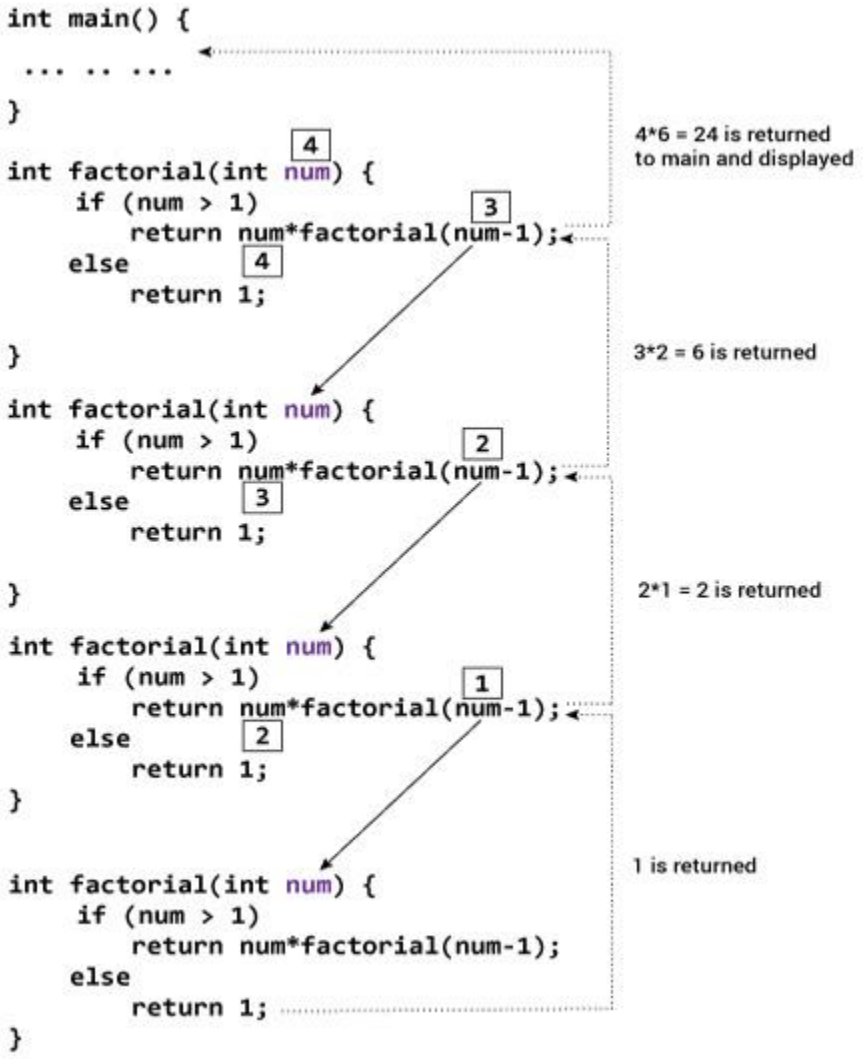
```
#include <stdio.h>
long int factorial(int n);

int main()
{
   int n;
   printf("Enter a number to find factorial: ");
   scanf("%d", &n);
   printf("Factorial of %d = %ld", n, factorial(n));
   getch();
   return 0;
}
long int factorial(int n)
{
   if (n > 1)
      return n*factorial(n-1);
   else
      return 1;
}
```

Output:

```
Enter a number to find factorial:
4                                   // enter by user
Factorial of 4 =  24
```

Working of factorial function



Example: Write a program in a C language to print the Fibonacci series using recursion.

Fibonacci series are 0, 1, 1, 2, 3, 5, 8, ...,.

In fibonacci series, except for the first two terms of the series, every other term is the sum of the previous two terms, for example, 5 = 2 + 3.

```
#include<stdio.h>
 int fibonacci(int);

int main()
{
```

```
 int n, c;
 printf("Enter the limit in fibonacci series  ");
 scanf("%d", &n);

        printf("\n Fibonacci series terms are:\n");

        for (c = 0; c <n; c++)
        {
                printf("%d\n", fibonacci(c));

        }
 getch();
  return 0;
}

int fibonacci(int n)
{
        if (n == 0 || n == 1)
                return n;
          else
                 return (fibonacci(n-1) +fibonacci(n-2));
}
```

Output:

```
Enter the limit in fibonacci series
7                   // enter by user
 Fibonacci series terms are:
0
1
1
2
3
5
8
```

Example: Write a recursive function that calculate sum of first n natural numbers.

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

```
void main()
{
 int number;
printf("Enter a positive integer :");
scanf("%d", &number);

printf("The sum of first %d numbers is %d.", number, sum(number));
getch();
 }

int sum(int n)
{
	if (n == 0)
		return 0;
	else
		return n + sum(n - 1);
}
```

OUTPUT

```
Enter a positive integer :10
The sum of first 10 numbers is 55.
```

Ackermann function

The Ackermann function is usually defined as follows:

- A(i, j)=j+1 for i= 0
- A(i, j)=A(i-1, 1) for i > 0 and j=0
- A(i, j)=A(i-1, A(i, j-1)) for i, j > 0

```
#include<stdio.h>
int A(int m, int n);

void main()
{
	int m,n;
	printf("Enter two numbers :: \n");
	scanf("%d%d",&m,&n);
	printf("\nOUTPUT :: %d\n",A(m,n));
	getch();
}

int A(int m, int n)
{
	if(m==0)
		return n+1;
	else if(n==0)
		return A(m-1,1);
```

```
        else
                return A(m-1,A(m,n-1));
}
```

OUTPUT
Enter two numbers::
1 1
OUTPUT :: 3

Merge Sort

Merge Sort is base on divide and conquer algorithm.

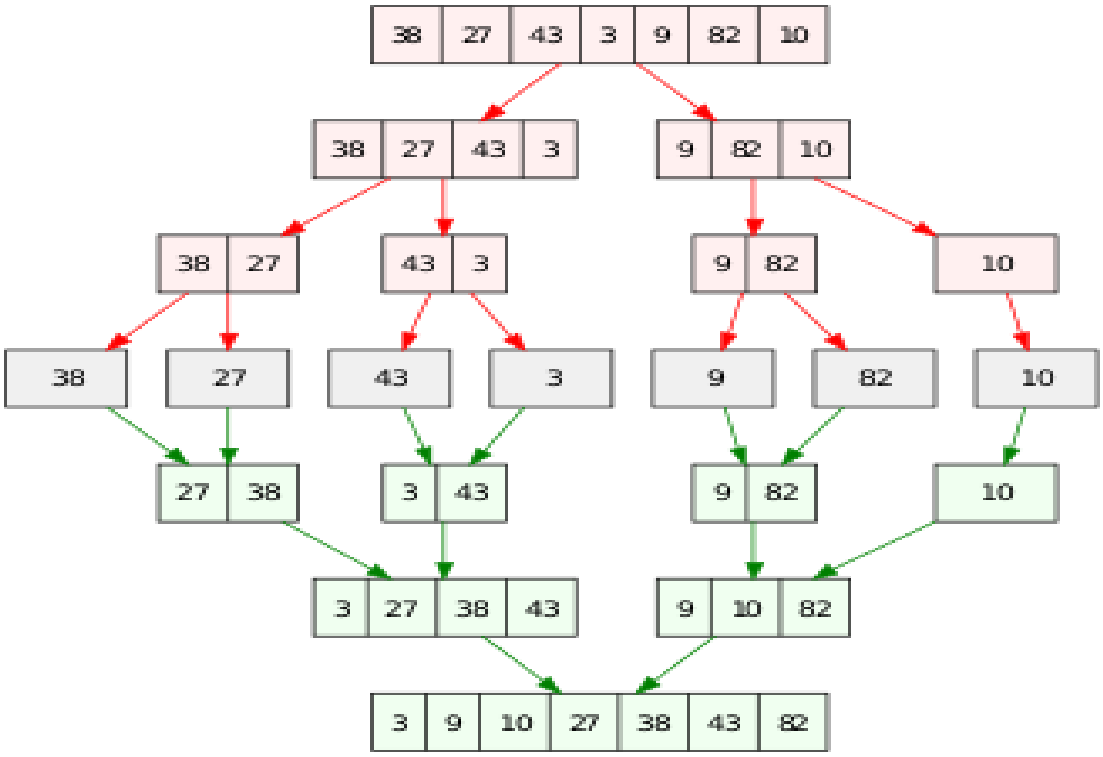
1) DIVIDING

In **Merge Sort**, we take a middle index and break the array into two sub-arrays. These sub-array will go on breaking till the array have only one element.

2) MERGING

When all we have is single elements we start merging the elements in the same order in which we have divided them. During Merging, we also sort the sub-arrays, because sorting 10 arrays of 2 elements is cheaper than sorting an array of 20 elements.

In the end, we will have an array of elements, which is sorted.

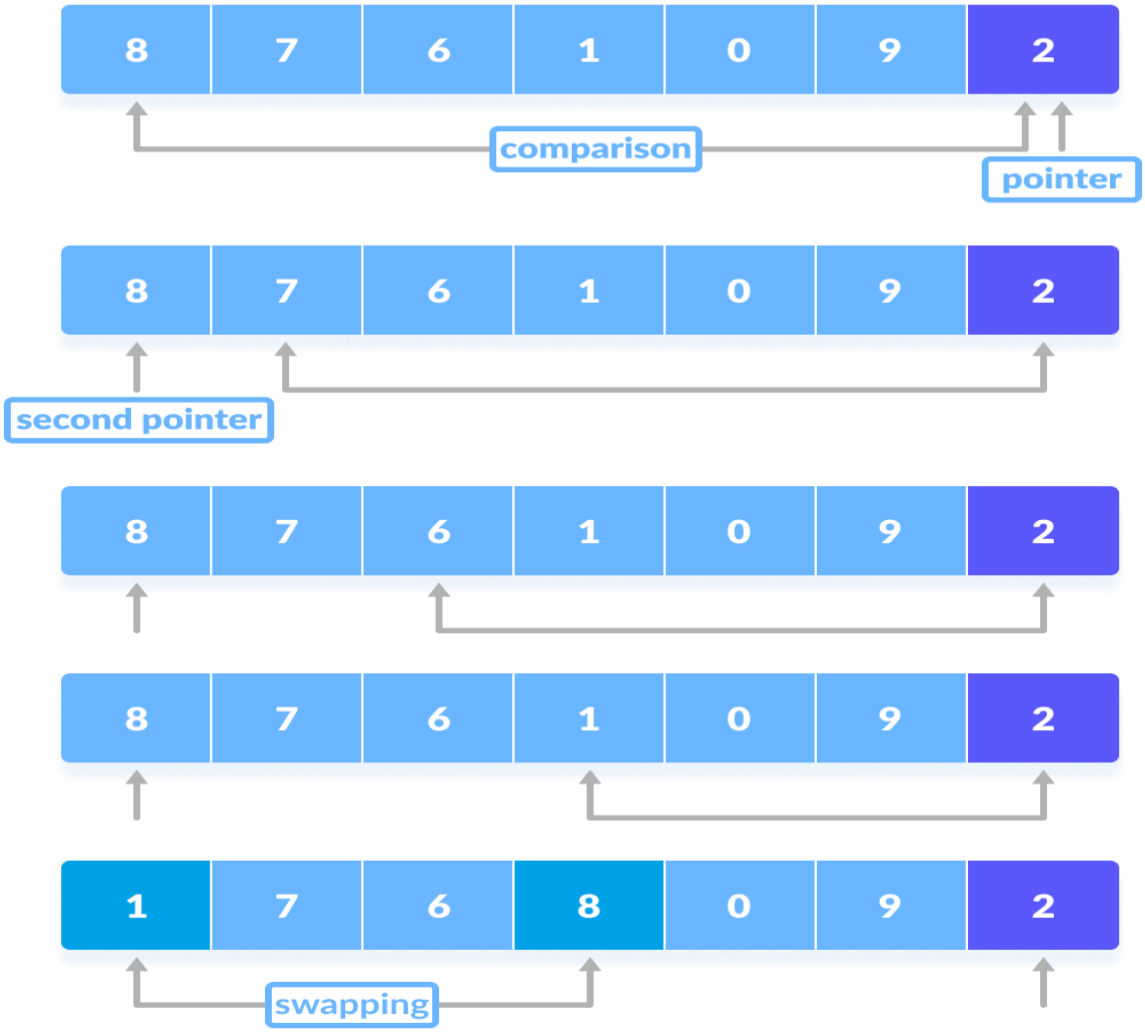


Quick Sort

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.


8 7 6 1 0 9 2
comparison
pointer
8 7 6 1 0 9 2
second pointer
8 7 6 1 0 9 2
8 7 6 1 0 9 2
1 7 6 8 0 9 2
swapping