- C functions:-

* A function in C must always be declared globally before calling it.

* Syntax:-

```
return_type  function_name (para1_type  para1_name,
                para2_type  para2_name)
{
  //body  of  the function
}
```

* Function call is necessary to bring the program control to the function definition. If not called, the function statements will not be executed.

*
```
#include<stdio.h>
int sum(int a, int b)
{
  return a+b;
}
int  add = sum(10,30);
printf("Sum is : %d", add);
return 0;
}
```

Output:-

Sum is : 40

* Only one value can be returned from a C function. To return multiple values, we have to use pointers or structures.

```
#include<math.h>
#include <stdio.h>
int main()
{
  double Number;
  Number = 49;
  double squareRoot = sqrt(Number);
  printf("The square root of %.2lf = %.2lf,
          Number, squareRoot);
  return 0;
  }
```

Output:-

The square root of 49.00 = 7.00

→ Passing parameters to functions:-

* The data passed when the function is being invoked is known as the Actual parameters.
* Formal parameters are the variable and the data type as mentioned in the function declaration.

→ We can pass arguments to the C function in two ways:-

1. Pass by value:-

→ Parameters passing in this method copies

values from actual parameters into formal function parameters. As a result, any changes made inside the functions do not reflect in the caller's parameters. Actual & formal arguments are created in different memory locations.

```
# include<stdio.h>
void swap(int var1, int var2)
{
 int temp = var1;
 var1 = var2;
 var2 = tem;
}
int main()
{
 int var1 = 3, var2 = 2;
 printf("Before swap value of var1 and var2 is:
    %d, %d \n", var1, var2);
 swap (var1, var2);
 printf("After swap value of var1 and var2 is:
  %d, %d", var1, var2);
  return 0;
  }
```

Output:-

Before swap value of var1 and var2 is: 3,2

After swap value of var1 and var2 is: 3,2

2. Pass by Reference:-

→ The caller's actual parameters and the functions actual parameters refer to the same locations, so any changes made inside the

function are reflected in the caller's actual parameters.

```
#include<stdio.h>
void swap ( int *var1 , int *var2)
{
 int temp = *var1;
 *var1 = *var2;
 *var2 = temp;
}
 int main()
 {
  int var1 =3, var2=2;
  printf("Before swap value of var1 and var2 is:
  %d, %d \n", var1 , var2);
  swap( &var1, &var2);
  printf(" After swap value of var1 and var2 is:
  %d , %d", var1 , var2);
  return 0;
  }
```

Output:-

Before swap value of var1 and var2 is: 3,2
After swap value of var1 and var2 is: 2,3

- User - defined function in C:-

→ C - function prototype:-

* Syntax:-

return_type function_name (type1 arg1 , type2 arg2, ... typeN argN)

→ C function definition:-

*Syntax :-

```
return_type function_name (type1 arg1, ....., typeN
        argN){
    // actual statements to be executed
    // return value if any
}
```

* If the function call is present after the function definition, we can skip the function prototype part and directly define the function.

→ C function call :-

* Syntax :-

```
function_name(arg1, arg2, ...., argN);
```

* // C program to illustrate the use of user-defined function

```
#include <stdio.h>
// Function prototype
int sum(int, int);
// Function definition
int sum(int x, int y)
{
    int sum;
    Sum = x+y;
    return x+y;
}
```

```
int main()
{
    int x=10, y=11;
    // Function call
    int result = sum(x,y);
    printf("Sum of %d and %d = %d", x, y,
                    result);

    return 0;
}
```

Output:-

Sum of 10 and 11 = 21

→ Components of function definition:-

1. function parameters:-

* Also known as arguments.
* We can pass none or any number of function parameters to the function.
* C language provides a method using which we can pass variable number of arguments to the function. Such functions are called variadic function.

2. Function body:-

* statements that are ~~excto~~ enclosed within {} braces.
* They are the statements that are executed when the function is called.

3) Return value :-

* A function can only return a single value and it is optional.
* If no value is to be returned, the return type is defined as void.
* The return keyword is used to return the value from a function.
* Syntax:-

```
return (expression);
```

* We can use pointers or structures to return multiple values from a function in C.

→ Passing parameters to user-defined functions:-

1) Call by value :-

* values aren't changed in the call by value since they aren't passed by reference.

2) Call by reference :-

* The address of the argument is passed to the function and changes that are made to the function are reflected back to the values.
* We use the pointers of the required type to receive the address in the function.

```
// C program to implement call by reference
#include<stdio.h>
void swap (int *a, int *b)
```

```
{
  int temp = *a;
  *a = *b;
  *b = temp;
}

// Driver code
int main()
{
    int x=10, y=20;
    printf(" values of x and y before swap are :
          %d, %d", x, y);

    swap(&x, &y);
    printf(" values of x and y after swap are:
       %d, %d ", x, y);

    return 0;
}
```

Output:-

Values of x and y before swap are: 10,20

Values of x and y after swap are: 20,10

- Difference between call by value and Call by reference in C:-

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

→ Call by value in C:-

* In this, the value of actual parameters are copied to the function's formal parameters.
- There are two copies of parameters stored in different memory locations.
- One is the original copy and the other is the function copy.
- Any changes made inside functions are not reflected in the actual parameters of the caller.

* //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
  swapx(a,b); //Actual parameters
  printf("In the caller :\na = %d  b = %d\n", a, b);
  return 0;
}
//swap functions that swaps two values
void swapx (int x, int y) //Formal parameters
{
  int t;
  t=x;
  x=y;
  y=t;
```

```
printf("Inside function:\n x = %d  y = %d \n", x, y);
}
```

Output:-

Inside function:
x = 20, y = 10
In the caller:
a = 10 b = 20

* Thus, actual values of a and b remain unchanged even after exchanging the values of x and y in the function.

→ Call by Reference in C:-

* In this, the address of the actual parameters is passed to the function as the formal parameters. In C, we use pointers to achieve call-by-reference.

- Both the actual and formal parameters refer to the same locatiocations.
- Any changes made inside the function are actually reflected in the actual parameters of the caller.

- Terminology:-

- Formal parameter - A variable and its type as it appears in the prototype of the function or method.
- Actual parameter - The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

- Modes :-

* IN : Passes info from caller to the callee.
* OUT : Callee writes values in the caller.
* IN/OUT : The caller tells the callee the value of the variable, which the callee may update.

→ Shortcomings of pass by value :-

- In efficiency in storage allocation.
- For objects and arrays, the copy semantics are costly.

→ Pass by Pointes :-

* In function we pass memory address (pointer) of a variable rather than passing the actual value of variable.

* // C program to demonstrate the pass by pointer

```
#include<stdio.h>
/* function to modify the value passed as pointer
 to an int */
void modifyval(int *myptr)
{
  // Access and modifying the value pointed by myptr
  *myptr = *myptr +5;
}
int main()
{
  int x = 5;
  int **myptr = &x;
```

```
// Passing the pointer ptr to the function
modifyVal(myptr);

// printing the modified value of x
printf("Modified value of x is : %d \n", x);
return 0;
}
```

Output:-

Modified value of x is : 10

- What if the function prototype is not specified:-

→ If we ignore the function prototype, a program may compile with a warning, it may give errors and may work properly. But sometimes, it will give strange output and it is very hard to find out such programming mistakes.

(#). Usually, when a function needs to return several values, we use one pointer in return instead of several pointers as arguments.

- Main function in C:-

→ It is user-defined function that is mandatory for the execution of a program because when a C program is executed, the operating system starts executing the statements in the main() function.

→ Syntax:-

```
return_type main(){
    //statement 1;
    //statement 2;
    return;
}
```

(or) ~~int main()~~ int main(){ }

(or) int main(void){ }

* Int means integer return type, and void return type means that does not return any information, or (void) or () means that does not take any information.

→ Important points about C main function:-

- It is the function where the program's execution starts.
- Every program has exactly one main function.
- The name of this function should be "main" not anything else
- The main function always return an integer value or void.
- The main function, is called by OS, not by the user.

- Types of C main functions:-

1). Main function with No arguments and void Return type:-

* The main function is called by the operating system itself and then executes all statements inside this function.

* Syntax:-

```
void main()
{
  // function body
}
```

(or)

```
void main(void)
{
  // function body
}
```

* Note:- The return type of main function according to C standard should be int only. Even if your compiler is able to run void main(), it is recommended to avoid it.

```
# include<stdio.h>
void main ()
{
  printf("Hello Geek!");
}
```

Output:-
Hello Geek!

2. Main function with No arguments and int Return type:-

* Indicates the exit status of the program
* The exit status is a way for the program to communicate to the operating system whether the program was executed successfully or not.
* The convention is that a return value of 0 indicates that the program was completed successfully, while any other value indicates that an error occured.

* Syntax:-

```
int main()
{
  //function body
}
```

(or)

```
int main (void)
{
  // Function body
}
```

```
# include<stdio.h>
int main()
{
  printf("Hello Geek!");
  return 0;
}
```

Output:-
Hello Geek!

3). main function with a command line arguments:-

* The first argument argc means argument count which means it stores the number of ~~arg~~ arguments passed in the command line and by default, its value is 1 when no argument is passed.

* The second argument is a char pointer array argv[] which stores all the command line arguments passed. We can also see in the output when we run the program without passing any command line ~~argg~~ argument the value of argc is 1.

* Syntax:-

```
int main (int argc , char * argv[ ]) {
    // Function body
}
```

```
#include<stdio.h>
int main (int argc, char * argv[])
{
    printf("The value of argc is %d \n", argc);
    for (int i =0; i < argc ; i++) {
        printf(" %s \n", argv [i]);
    }
    return 0;
}
```

Output:-

```
geeks
for
geeks
```

- Implicit return type int in C:-

* In C if we do not specify a return type, compiler assumes an implicit return type as int.

```
#include<stdio.h>
fun (int x)
{
  return x*x;
}
int main(void)
{
  printf("%d", fun(10));
  return 0;
}
```

Output:-

100.

- Callbacks in C:-

* A callback is any executable code that is passed as an argument to another code, which is expected to call back (execute) the argument at a given time.
* If a reference of a function is passed to another function as an argument to call it, then it will be called a callback function.
* In C, a callback function is a function that is called through a function pointer.

// A simple C program to demonstrate callback

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

```
void A(){
    printf(" I am function A \n");
}

// callback function
void B(void (*ptr)())
{
   (*ptr)(); // callback to A
}
int main()
{
    void (*ptr)() = &A;
    //calling function B and passing
    //address of the function A as argument
    B (ptr);
    return 0;
}
```

Output:-

I am function A.

-Nested functions in C:-

* Nested function is not supported by C because we cannot define a function within another function in C. We can declare a function inside a function, but it's not a nested function.

* Nested functions have only a limited use. If we try to approach nested function in C, then we will get a compile time error.

```c
/* C program to illustrate the concept of Nested
        function */
#include<stdio.h>
int main(void)
{
  printf("Main");
  int fun()
  {
    printf("Fun");
    // defining view() function inside fun() function.
    int view()
    {
      printf("view");
    }
    return 1;
  }
  view();
}
```

Output:-

Compile time error: undefined reference to 'view'.

* An extension of the GNU C compiler allows the declarations of nested functions. The declarations of nested functions under GCC's extension need to be prefix / start with the auto keyword.

- Variadic functions in C!-

* that can take a variable number of arguments.
* this function can add flexibility to the programs.
  by <stdarg.h>.

- C library math.h functions:-

- C Introduction :-

→ Some of the C Header files :-

- stddef.h - Define several useful types and macros.
- stdint.h - Define exact width integer types.
- stdio.h - Defines core input and output functions
- stdlib.h - Defines numeric conversion functions, pseudo-random number generator, and memory allocation.
- string.h - Defines string handling functions.

- math.h – Defines common mathematical functions.