Function in C Language

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always overhead in a C program.

Function Aspects

- **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

  Syntax => return_type function_name(arguments);

- **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

  Syntax => function_name(arguments);

- **Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

  Syntax => return_type function_name(arguments){function_body;}

Types of Functions

There are two types of functions in C programming:

1. **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

Return Value

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Example without return value:

```
1. void hello(){
2. printf("hello c");
3. }
```

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

Example with return value:

```
int get(){
return 10;
}
```

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

```
float get(){
return 10.2;
}
```

Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- function without arguments and without return value
- function without arguments and with return value
- function with arguments and without return value
- function with arguments and with return value

C Library Functions

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console.

Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by value* and *call by reference*.

Call by value in C

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Call by reference in C

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value

stored at the address of the actual parameters, and the modified value gets stored at the same address.

Recursion in C

- Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called a recursive function, and such function calls are called recursive calls.

Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problems, but it is more useful for the tasks that can be defined in terms of similar subtasks.

Recursive Function

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

Pseudocode for writing any recursive function is given below.

```
1. if (test_for_base)
2. {
3.     return some_value;
4. }
5. else if (test_for_another_base)
6. {
7.     return some_another_value;
8. }
```

```
9.  else
10. {
11.     // Statements;
12.     recursive call;
13. }
```

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic
- External
- Static
- Register

Automatic

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined.The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

Static

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define a static variable is static.

Register

- The variables defined as the register are allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.
- We can not dereference the register variables, i.e., we can not use &operator for the register variable.
- The access time of the register variables is faster than the automatic variables.
- The initial default value of the register local variables is 0.
- The register keyword is used for the variable which should be stored in the CPU register. However, it is the compiler's choice whether or not; the variables can be stored in the register.
- We can store pointers into the register, i.e., a register can store the address of a variable.
- Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.

External

- The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.
- The variables declared as extern are not allocated any memory. It is only a declaration and intended to specify that the variable is declared elsewhere in the program.
- The default initial value of external integral type is 0 otherwise null.
- We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.
- An external variable can be declared many times but can be initialized at only once.
- If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.