C Functions

A function in C is a set of statements that when called perform some specific tasks. It is the basic building block of a C program that provides modularity and code reusability.

Syntax of Functions in C

The syntax of function can be divided into 3 aspects:

1. Function Declaration
2. Function Definition
3. Function Calls

Function Declarations

In a function declaration provide the function name, its return type, and the number and type of its parameters. A function declaration tells the compiler that there is a function with the given name defined in the program.

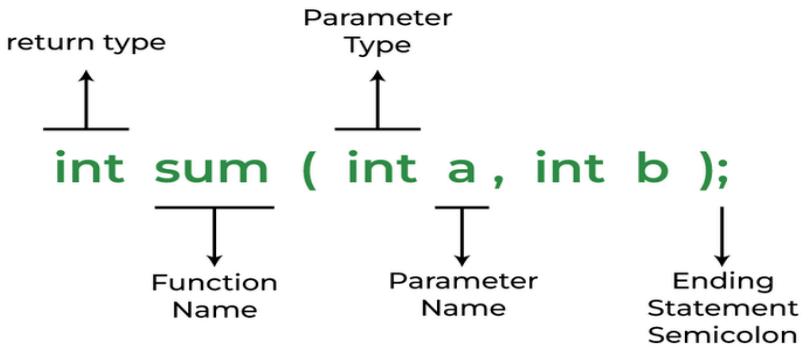
Syntax: return_type function_name (parameter_1, parameter_2);

The parameter name is not mandatory while declaring functions. It can also declare the function without using the name of the data variables.

Example

int sum(int a, int b); // Function declaration with parameter names

int sum(int , int); // Function declaration without parameter names

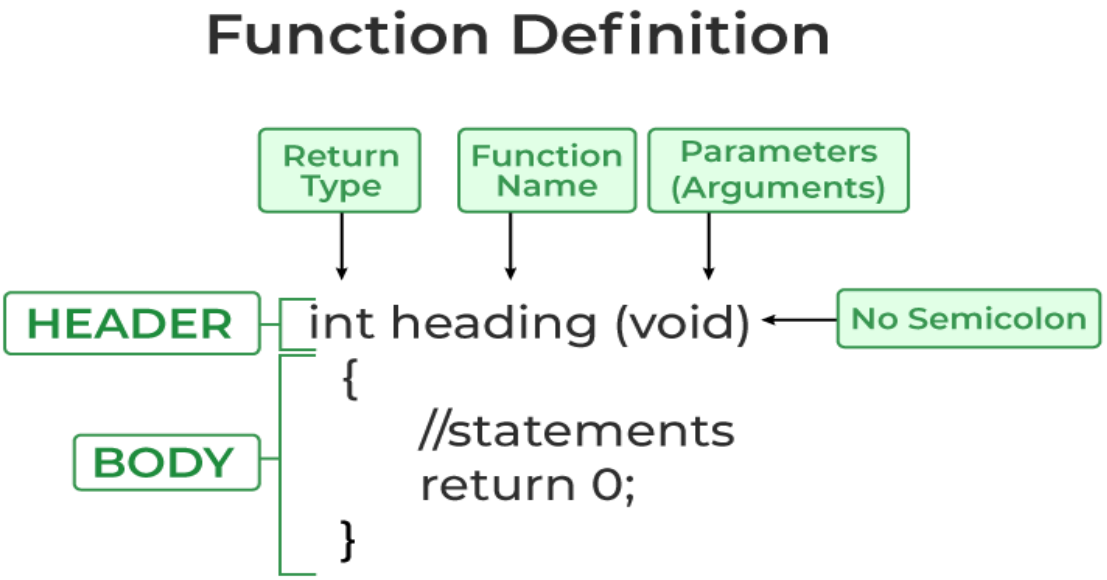


Function Definition

The function definition consists of actual statements which are executed when the function is called.

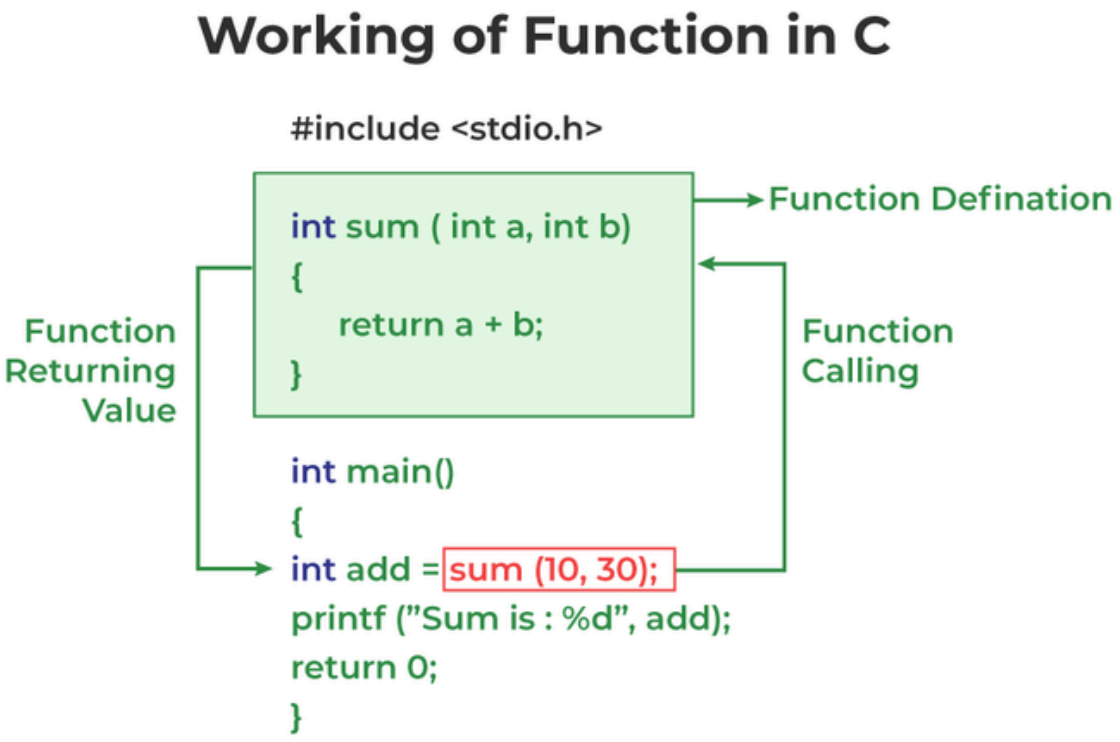
A C function is generally defined and declared in a single step because the function definition always starts with the function declaration so we do not need to declare it explicitly. The below example serves as both a function definition and a declaration.

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



Function Call

A function call is a statement that instructs the compiler to execute the function. The function name and parameters in the function call.



Example of C Function

```
int sum(int a, int b)
{
  return a + b;
}
int main()
{
          int add = sum(10, 30);
         printf("Sum is: %d", add);
  return 0;
}
```

Output

Sum is: 40

Function Return Type

Function return type tells what type of value is returned after all function is executed. When we don't want to return a value, we can use the void data type.

Example: int func(parameter_1,parameter_2);

The above function will return an integer value after running statements inside the function.

Function Arguments

Function Arguments (also known as Function Parameters) are the data that is passed to a function.

Example: int function_name(int var1, int var2);

Conditions of Return Types and Arguments

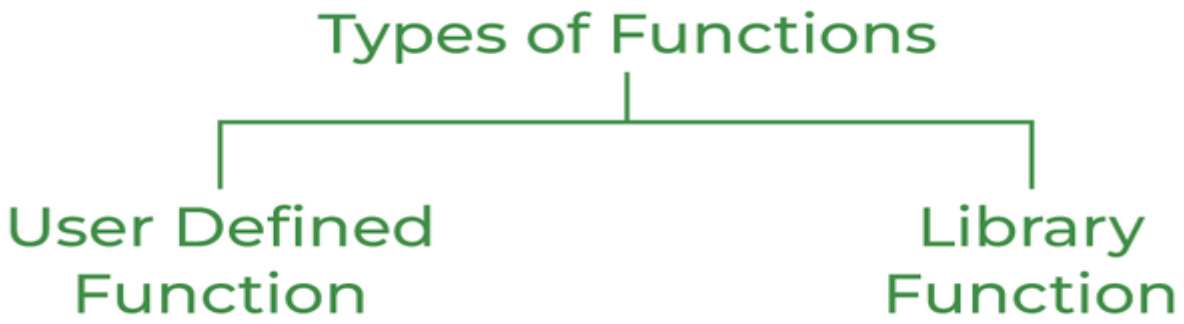
In C programming language, functions can be called either with or without arguments and might return values. They may or might not return values to the calling functions.

- Function with no arguments and no return value
- Function with no arguments and with return value
- Function with argument and with no return value
- Function with arguments and with return value

Types of Functions

There are two types of functions in C:

1. **Library Functions**
2. **User Defined Functions**



1. Library Function

A library function is also referred to as a "built-in function". A compiler package already exists that contains these functions, each of which has a specific meaning and is included in the package. Built-in functions have the advantage of being directly usable without being defined, whereas user-defined functions must be declared and defined before being used.

For Example: pow(), sqrt(), strcmp(), strcpy() etc.

Advantages of C library functions

- C Library functions are easy to use and optimized for better performance.
- C library functions save a lot of time i.e, function development time.
- C library functions are convenient as they always work.

Example:

#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

2. User Defined Function

Functions that the programmer creates are known as User-Defined functions or “tailor-made functions”. User-defined functions can be improved and modified according to the need of the programmer.

Advantages of User-Defined Functions

- Changeable functions can be modified as per need.
- The Code of these functions is reusable in other programs.
- These functions are easy to understand, debug and maintain.

Example:

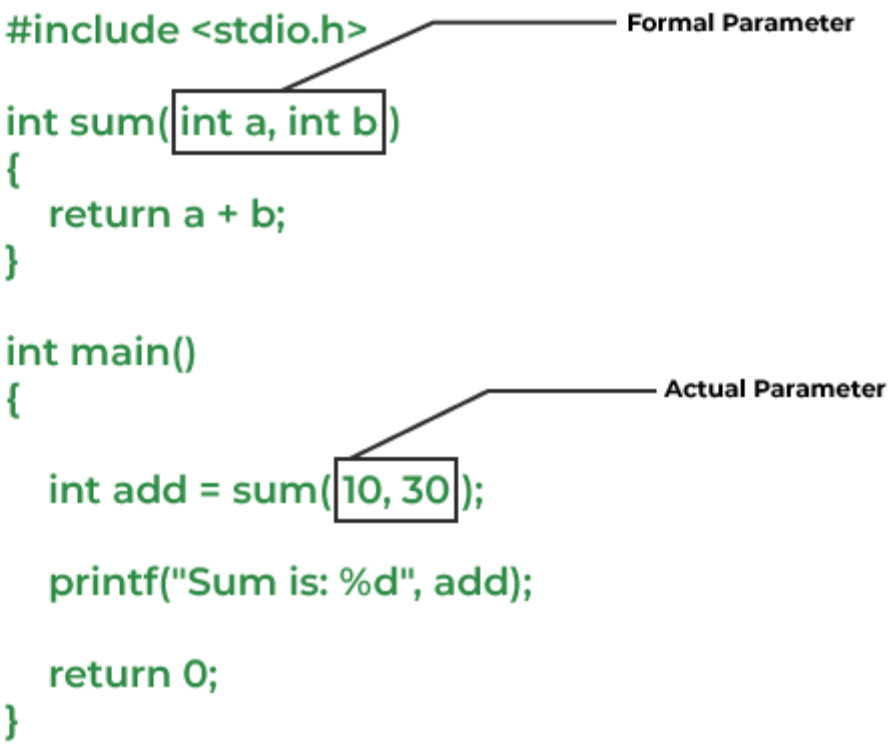
```
#include <stdio.h>
int sum(int a, int b)
{
        return a + b;
}
int main()
{
        int a = 30, b = 40;
        int res = sum(a, b);
        printf("Sum is: %d", res);
        return 0;
}
```

Output

Sum is: 70

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 in the function declaration.



Pass arguments to the C function in two ways:

1. Pass by Value
2. Pass by Reference

1. Pass by Value

Parameter 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.

Example:

```
#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: 3, 2

2. Pass by Reference

The caller's actual parameters and the function's actual parameters refer to the same locations, so any changes made inside the function are reflected in the caller's actual parameters.

Example:

```
#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

Advantages of Functions in C

- The function can reduce the repetition of the same statements in the program.
- The function makes code readable by providing modularity to our program.
- There is no fixed number of calling functions it can be called as many times as you want.
- The function reduces the size of the program.
- Once the function is declared you can just use it without thinking about the internal working of the function.

Disadvantages of Functions in C

- Cannot return multiple values.
- Memory and time overhead due to stack frame allocation and transfer of program control.

String in C

A String in C is a collection of characters in a linear sequence. 'C' always treats a string a single data even though it contains whitespaces. A single character is defined using single quote representation. A string is represented using double quote marks.

Example: "Welcome to the world of programming!"

'C' provides standard library <string.h> that contains many functions which can be used to perform complicated operations easily on Strings in C.

Declaration of strings: Declaring a string is as simple as declaring a one-dimensional array.
Basic syntax for declaring a string.

Syntax: datatype str_name[size];

In the above syntax str_name is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store which is the Null character ('\0') used to indicate the termination of string which differs strings from normal character arrays.

Initializing a String: A string can be initialized in different ways.
To declare a string with name as str and initialize it with "NPSGDC".
1. char str[] = "NPSGDC";

2. char str[50] = "NPSGDC";

3. char str[] = {'N','P','S','G','D','C','\0'};

4. char str[14] = {'N','P','S','G','D','C','\0'};

C – String functions

1. STRLEN(): In C programming, we use strlen() to find the length of the string. String handling function strlen() returns the number of characters in string.
Syntax: integer_variable = strlen(string or string_variable);
Examples:

```
char str[30];
int len;
printf("Enter string:\n");
gets(str);
len = strlen(str);
printf("Length of given string is: %d", len);
```

2. STRCPY(): String handling function strcpy() is used to copy content of one string variable to another string variable i.e. strcpy(string2, string1) copies content of string1 to string2.
Syntax: integer_variable = strcpy (string2, string1);
Examples:

```
char str1[30], str2[30];
 printf("Enter string:\n");
 gets(str1);
 strcpy(str2, str1);
 printf("Coped string is: %s", str2);
```

3. STRCMP(): In C programming, string handling function strcmp() is used to compare two strings. This function returns 0 if two strings are same otherwise it returns some integer value other than 0.
Syntax: integer_variable = strcmp(string1, string2);
Examples:

```
char str1[40], str2[40];
 int d;
printf("Enter first string:\n");
gets(str1);
 printf("Enter second string:\n");
 gets(str2);
 d = strcmp(str1, str2);
 if(d==0)
 {
  printf("Given strings are same.");
 }
 else
 {
  printf("Given strings are different.");
 }
```

4. STRCAT(): String handling function strcat() is used to concatenate two strings. Concatenation is the process of merging content of one string to another string. Function strcat(str1, str2) concatenates content of string str2 after content of string str1.
Syntax: integer_variable = strcat(string1, string2);
Examples:

```
char str1[50], str2[50];
 printf("Enter first string:\n");
 gets(str1);
 printf("Enter second string:\n");
 gets(str2);
 strcat(str1,str2);
 printf("Concatenated string is: %s", str1);
```

5. STRREV(): String handling function strrev() is used to reverse string in C programming language. Function strrev(str1) reverses the content of string str1.
Syntax: integer_variable =strrev(string);
Examples:

```
char name[40];
printf("Enter your name: ");
gets(name);
strrev(name);
printf("Reversed name is: %s", name);
```

6. STRUPR(): String handling function strupr() is used to convert all lower case letter in string to upper case i.e. strlwr(str) converts all lower case letter in string **str** to upper case.
Syntax: integer_variable =strupr(string);

Examples:

```
char str[40];
 printf("Enter string:\n");
 gets(str);
 strupr(str);
 printf("String in uppercase is:");
 puts(str);
```

7. STRLWR(): String handling function strlwr() is used to convert all upper case letter in string to lower case i.e. strlwr(str) converts all upper case letter in string **str** to lowercase.

Syntax: integer_variable =strlwr(string);
Examples:

```
char str[40];
 printf("Enter string:\n");
 gets(str);
 strlwr(str);
 printf("String in lowercase is:");
 puts(str);
```