Introduction:

A function is a block of code that performs a specific task.

Suppose, you need to create a program to create a circle and color it. You can create two functions to solve this problem:

- create a circle function
- create a color function

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting 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 a overhead in a C program.

Types of function

There are two types of function in C programming:

- Standard library functions
- User-defined functions

Standard library functions

The standard library functions are built-in functions in C programming.

These functions are defined in header files. For example,

- The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file.

Hence, to use the printf() function, we need to include the stdio.h header file using #include <stdio.h>.

- The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

User-defined function

You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

How user-defined function works?

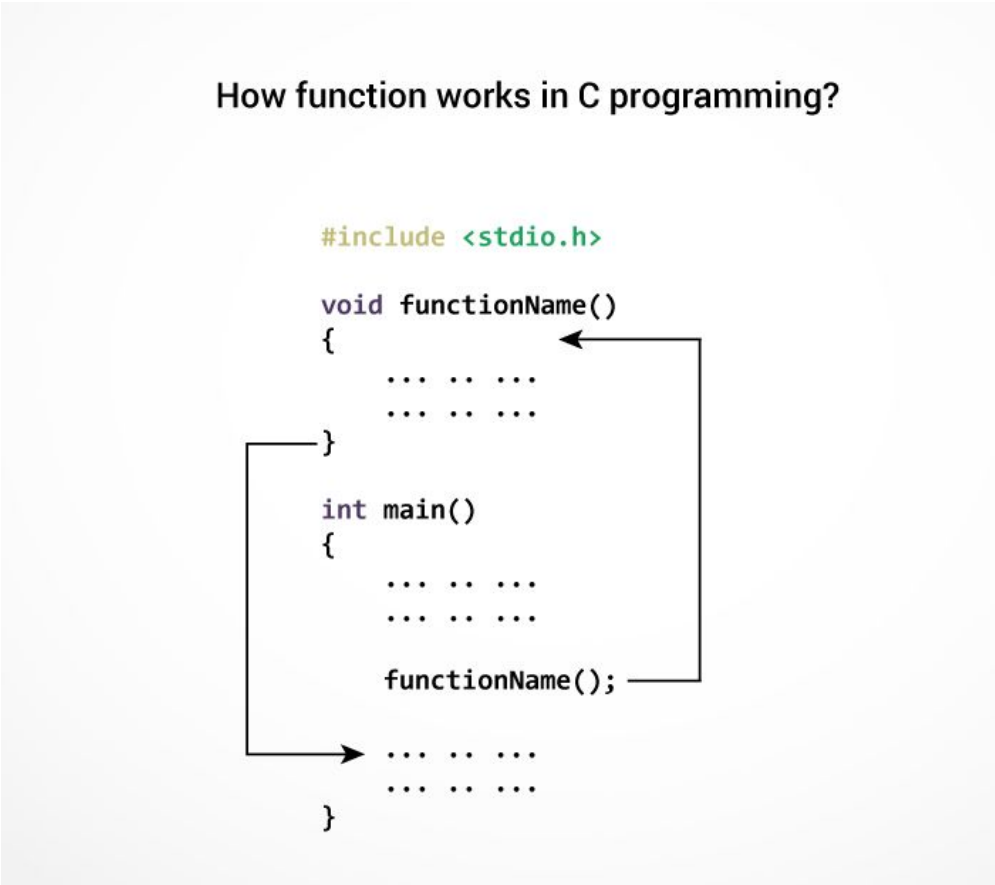
```
#include <stdio.h>
void functionName()
{
    ... .. ...
    ... .. ...
}

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

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.



Example: User-defined function

Here is an example to add two integers. To perform this task, we have created an user-defined addNumbers().

```
#include <stdio.h>
int addNumbers(int a, int b);         // function prototype

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

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

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

  return 0;
```

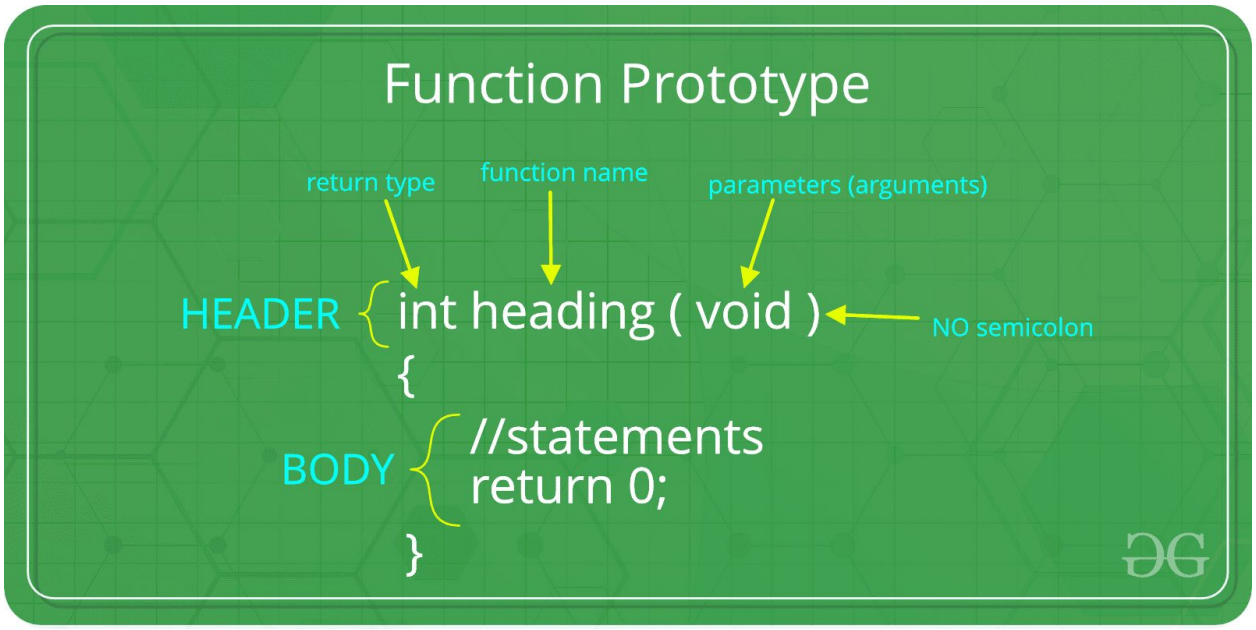
```
}

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

Output:

Write on your own…

Function Prototype:

A function in C can be called either with arguments or without arguments. These functions may or may not return values to the calling functions. All C functions can be called either with arguments or without arguments in a C program. Also, they may or may not return any values. Hence the function prototype of a function in C is as below:



Types of Function in C according to return value and arguments

Depending upon the presence of arguments and the return values, user defined functions can be classified into five categories.

1. **Function with no arguments and no return values**
2. **Function with no arguments and one return value**
3. **Function with arguments and no return values**
4. **Function with arguments and one return value**

1: Function with no arguments and no return value

Function with no argument means the called function does not receive any data from calling function and **Function with no return value** means calling function does not receive any data from the called function. So there is no data transfer between calling and called function.

C program to calculate the area of square using the function with no arguments and no return values

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

Explanation

In the above program, area( ); function calculates area and no arguments are passed to this function. The return type of this function is void and hence return nothing.

2: Function with no arguments and one return value

As said earlier function with no arguments means called function does not receive any data from calling function and function with one return value means one result will be sent back to the caller from the function.

C program to calculate the area of square using the function with no arguments and one return values

```
#include <stdio.h>
int area();   //function prototype with return type int
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;
}
```

Explanation

In this function int area( ); no arguments are passed but it returns an integer value square_area.

3: Function with arguments and no return values

Here function will accept data from the calling function as there are arguments, however, since there is no return type nothing will be returned to the calling program. So it's a one-way type communication.

C program to calculate the area of square using the function with arguments and no return values

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

Explanation

In this function, the integer value entered by the user in square_side variable is passed to area(); .The called function has void as a return type as a result, it does not return value.

This program calculates the area of a square and prints the result.

4: Function with arguments and one return value

Function with arguments and one return value means both the calling function and called function will receive data from each other. It's like a dual communication.

C program to calculate the area of square using the function with arguments and one return values

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

Types of Function calls in C

If the function does not have any arguments, then to call a function you can directly use its name. But for functions with arguments, we can call a function in two different ways, based on how we specify the arguments, and these two ways are:

- Call by Value
- Call by Reference

1. CALL BY VALUE:

- In call by value method, the value of the variable is passed to the function as parameter.
- The value of the actual parameter can not be modified by formal parameter.
- Different Memory is allocated for both actual and formal parameters. Because, the value of actual parameter is copied to formal parameter.

Note:

- Actual parameter – This is the argument which is used in function call.
- Formal parameter – This is the argument which is used in function definition

EXAMPLE PROGRAM FOR C FUNCTION (USING CALL BY VALUE):

- In this program, the values of the variables “m” and “n” are passed to the function “swap”.
- These values are copied to formal parameters “a” and “b” in swap function and used.

```
#include<stdio.h>
// function prototype, also called function declaration
void swap(int a, int b);

int main()
{
   int m = 22, n = 44;
   // calling swap function by value
   printf(" values before swap  m = %d \nand n = %d", m, n);
   swap(m, n);
}

void swap(int a, int b)
{
   int tmp;
   tmp = a;
```

```
    a = b;
    b = tmp;
    printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

OUTPUT:

```
values before swap m = 22

and n = 44

values after swap m = 44

and n = 22
```

2. CALL BY REFERENCE:

- In call by reference method, the address of the variable is passed to the function as parameter.
- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

EXAMPLE PROGRAM FOR C FUNCTION (USING CALL BY REFERENCE):

- In this program, the address of the variables “m” and “n” are passed to the function “swap”.
- These values are not copied to formal parameters “a” and “b” in swap function.
- Because, they are just holding the address of those variables.

- This address is used to access and change the values of the variables.

```
#include<stdio.h>
// function prototype, also called function declaration
void swap(int *a, int *b);
int main()
{
   int m = 22, n = 44;
   //  calling swap function by reference
   printf("values before swap m = %d \n and n = %d",m,n);
   swap(&m, &n);
}

void swap(int *a, int *b)
{
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
   printf("\n values after swap a = %d \nand b = %d", *a, *b);
}
```

OUTPUT:

```
values before swap m = 22

and n = 44

values after swap a = 44

and b = 22
```

Recursion

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
  recursion(); /* function calls itself */
}

int main() {
  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 factorial of a number, generating Fibonacci series, etc.

Example: Number Factorial

The following example calculates the factorial of a given number using a recursive function −

```
#include <stdio.h>

int factorial( int i) {

   if(i <= 1) {
      return 1;
   }
   return i * factorial(i - 1);
}

int  main() {
   int i = 12;
   printf("Factorial of %d is %d", i, factorial(i));
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Factorial of 12 is 479001600
```

Example: Fibonacci Series

The following example generates the Fibonacci series for a given number using a recursive function −

```
#include <stdio.h>

int fibonacci(int i) {

   if(i == 0) {
      return 0;
   }

   if(i == 1) {
      return 1;
   }
```

```
   return fibonacci(i-1) + fibonacci(i-2);
}

int  main()
{

   int i;

   for (i = 0; i < 10; i++) {
      printf("%d\t\n", fibonacci(i));
   }

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
0

1

1

2

3

5

8

13

21

34
```

Concept of Local, global and Static Variable

Local Variables

The variables which are declared inside the function, compound statement (or block) are called Local variables.

```
void function_1()
{
    int a = 1, b = 2;
}

void function_2()
{
    int a = 10, b = 20;
}
```

a and b are called local variables. They are available only inside the function in which they are defined (in this case function_1()). If you try to use these variables outside the function in which they are defined, you will get an error. Another important point is that variables a and b only exists until function_1() is executing. As soon as function function_1() ends variables a and b are destroyed.

Global Variables

The variables declared outside any function are called global variables. They are not limited to any function. Any function can access and modify global variables. Global variables are automatically initialized to 0 at the time of declaration. Global variables are generally written before main() function.

```c
#include<stdio.h>
void func_1();
void func_2();
int a, b = 10;  // declaring and initializing glob

int main()
{
    printf("Global a = %d\n", a);
    printf("Global b = %d\n\n", b);

    func_1();
    func_2();

    // signal to operating system program ran fine
    return 0;
}

void func_1()
{
    printf("From func_1() Global a = %d\n", a);
    printf("From func_1() Global b = %d\n\n", b);
}

void func_2()
{
    int a = 5;
    printf("Inside func_2() a = %d\n", a);
}
```

Expected Output:

```
Global a = 0
Global b = 10

From func_1() Global a = 0
From func_1() Global b = 10

Inside func_2() a = 5
```

Static variables

A Static variable is able to retain its value between different function calls. The static variable is only initialized once, if it is not initialized, then it is automatically initialized to 0. Here is how to declare a static variable.

```c
#include<stdio.h>
int fun()
{
  static int count = 0;
  count++;
  return count;
}

int main()
{
  printf("%d ", fun());
  printf("%d ", fun());
  return 0;
}
```

Output:

1 2