Functions in C

21 February 2026 20:26

What is a function in C?
A function is a self-contained block of code that performs a specific task. Instead of writing the same logic repeatedly, we define it once and reuse it whenever needed.

In structured programming, functions improve:
- Modularity
- Readability
- Debugging
- Reusability

C language provides:

- Library functions-> printf(), scanf(), sqrt() and many more
- User defined functions-> Created by the programmer

A user defined function (UDF) is a function written by the programmer to perform a specific operation not provided by default in the C standard library.

General structure:

```
return_type function_name(<parameter-list>){
  //body of the function
  return value; //optional
}
```

Function declaration(prototype)

```
return_type function_name(<parameter-list>);
```

Example:

```
int add(int a, int b);
```

parameter(s)
return_type
function_name

```
int add(int a, int b){
   int r = a + b;
   return r;
}
```

Types of User-defined functions in C
Typ1-1: No Arguments, No return value
Function performs task but does not take input or return result.

```
#include<stdio.h>
void greet(){
    printf("\nWelcome to SigmaPi Academy");
}
```

return type void means my function will not return any value to the caller.

```
int main(int argc, char const *argv[])
{
    greet(); //function call
    return 0;
}
```

Output:
Welcome to SigmaPi Academy

Type-2: Arguments, No return value
Function takes input but prints the result directly.

```
#include<stdio.h>
void printSquare(int n){
    printf("\nSquare = %d", (n*n));
}
int main(int argc, char const *argv[])
{
    int l;
    printf("\nEnter the int value: ");
    scanf("%d", &l);
    printSquare(l);
    printSquare(5);
    return 0;
}
```

Output:

```
Enter the int value: 9

Square = 81
Square = 25
```

Type 3: No Arguments, with return value
Function calculates and returns a value.

```
#include<stdio.h>
int getNumber(){
    int n;
    printf("\nEnter number: ");
    scanf("%d", &n);
    return n;
}
int main(int argc, char const *argv[])
{
    int value = getNumber();
    printf("You entered %d", value);
    return 0;
}
```

Output:

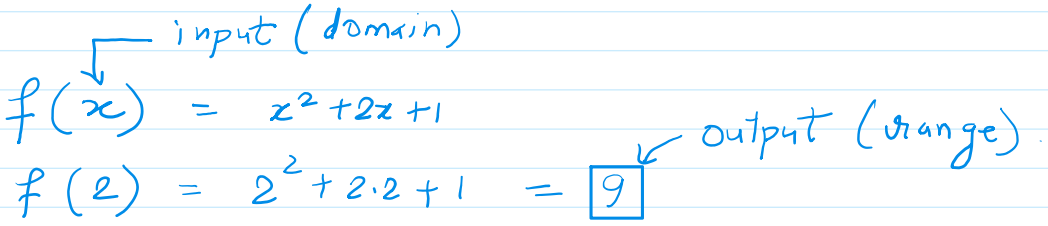
```
Enter number: 10
You entered 10
```

Type 4: Arguments with Return Value(Mostly used)
This is the standard professional model.

```c
#include<stdio.h>
int multiply(int, int); //prototype
int main(int argc, char const *argv[])
{
    int result = multiply(4, 7);
    printf("\nProduct = %d", result);
    int m, n;
    printf("\nEnter the value of m: ");
    scanf("%d", &m);

    printf("\nEnter the value of n: ");
    scanf("%d", &n);
    result = multiply(m, n);
    printf("\nProduct = %d", result);
    printf("\nProduct = %d", multiply(9,17));
    return 0;
}
int multiply(int x, int y){
    return x * y;
}
```

Output:

```
Product = 28
Enter the value of m: 0

Enter the value of n: 100

Product = 0
Product = 153
```



Real-Life Analogy
Think of a function as a robot module in your robotics curriculum:

- **Input->sensor data**
- **Processing->internal logic**
- **Output->action**

Function breaks the complex system into reusable blocks:
Example:

- moveForward()
- turnLeft()
- stopMotor()

WAP to find all 3 digit twisted prime. A prime number when it is revered it maintain the prime property in it. For example: 13 is a prime number, but if you reverse it will become 31, and 31 is also prime. Hence 13 is a twisted prime.
107, 701

```
int isPrime(int x){
    if(x > 2 && x % 2 == 0 || x == 1){
        return 0;
    }
    int d;
    for(d = 3; d <= sqrt(x); d +=2){
        if(x % d == 0){
            return 0;
        }
    }
    return 1; //x is prime
}
```

efficient code

```
int count = 0, i
for(i=1; i<=x; i++){
    if(x%i==0){
        count++;
    }
}
if(count==2)
    return 1;
else
    return 0;
```

not efficient.

```
int reverse(int n){
    int rev = 0;
    for(; n; rev = rev * 10 + n % 10, n/=10);
    return rev;
}
```

when the value of n becomes 0 the condition is false.

```
#include<stdio.h>
#include<math.h>
//prototypes
int isPrime(int);
int reverse(int);
int main(int argc, char const *argv[])
{
    int i;
    for(i = 101; i < 998; i += 2){
        if(isPrime(i) && isPrime(reverse(i))){
            printf("%d ", i);
        }
    }
    return 0;
}
int isPrime(int x){
```

```
    if(x > 2 && x % 2 == 0 || x == 1){
        return 0;
    }
    int d;
    for(d = 3; d <= sqrt(x); d +=2){
        if(x % d == 0){
            return 0;
        }
    }
    return 1; //x is prime
}
int reverse(int n){
    int rev = 0;
    for(;n; rev = rev * 10 + n % 10, n/=10);
    return rev;
}
```

Output:

```
101 107 113 131 149 151 157 167 179 181 191 199 311 313 337 347 353 359 373 383 389
701 709 727 733 739 743 751 757 761 769 787 797 907 919 929 937 941 953 967 971 983
991
```