Functions in C Language

By

Prof. Muhammad Iqbal Bhat

Department of Higher Education

Government Degree College Beerwah

Topics:

1. Introduction and Need of Functions
2. Math Library Functions
3. Function Declaration (prototype)
4. Function Definition
5. Function Call

Functions in C language

Experience has shown that the best way to develop and maintain a program is to construct it from smaller pieces, each of which is more manageable than the original program.

This technique is called divide and conquer

In C, we use functions to modularize programs by combining the new functions you write with prepackaged C standard library functions.

The C standard library provides a rich collection of functions for performing common mathematical calculations, string manipulations, character manipulations, input/output and many other useful operations.

Familiarize yourself with the rich collection of C standard library functions to help reduce program-development time.

Math Library Functions:

C's math library functions (header **math.h**) allow you to perform common mathematical calculations.


```
printf("%.2f", sqrt(900.0));
```

Math Library Functions:

Function	Description	Example
`sqrt(x)`	square root of x	`sqrt(900.0)` is `30.0` `sqrt(9.0)` is `3.0`
`cbrt(x)`	cube root of x (C99 and C11 only)	`cbrt(27.0)` is `3.0` `cbrt(-8.0)` is `-2.0`
`exp(x)`	exponential function e^x	`exp(1.0)` is `2.718282` `exp(2.0)` is `7.389056`
`log(x)`	natural logarithm of x (base e)	`log(2.718282)` is `1.0` `log(7.389056)` is `2.0`

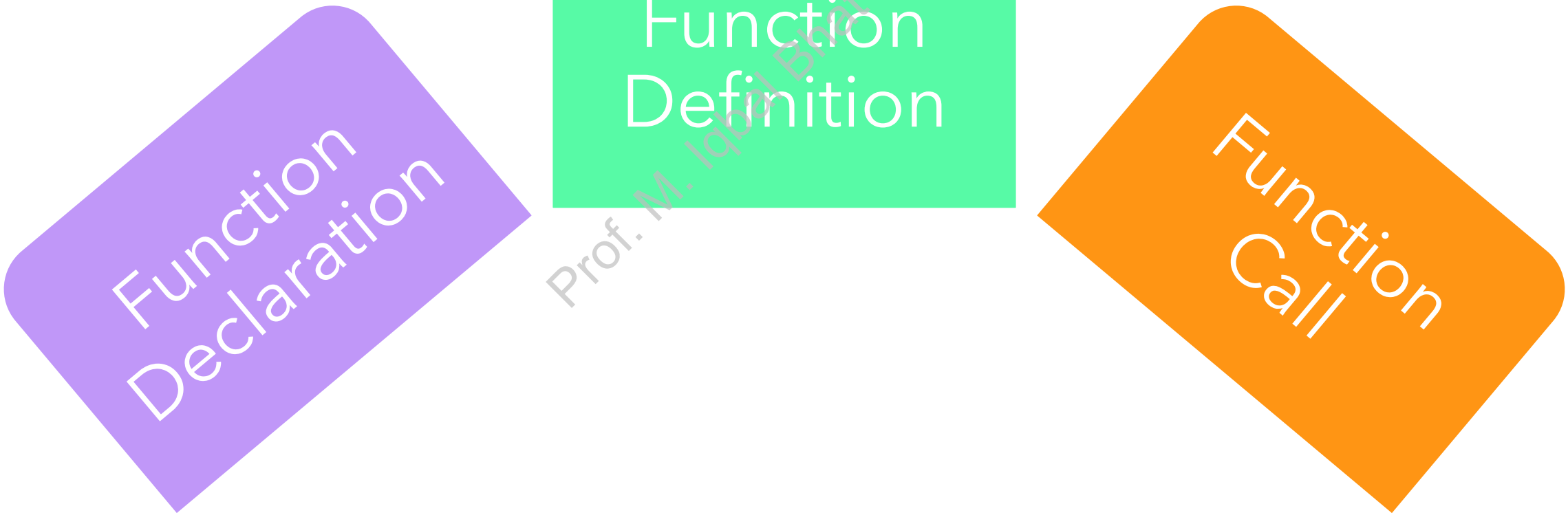
`log10(x)`	logarithm of x (base 10)	`log10(1.0)` is `0.0`
		`log10(10.0)` is `1.0`
		`log10(100.0)` is `2.0`

`fabs(x)`	absolute value of x as a floating-point number	`fabs(13.5)` is `13.5`
		`fabs(0.0)` is `0.0`
		`fabs(-13.5)` is `13.5`

Math Library Functions:

Function	Description	Example
`ceil(x)`	rounds x to the smallest integer not less than x	`ceil(9.2)` is `10.0` `ceil(-9.8)` is `-9.0`
`floor(x)`	rounds x to the largest integer not greater than x	`floor(9.2)` is `9.0` `floor(-9.8)` is `-10.0`
`pow(x, y)`	x raised to power y (x^y)	`pow(2, 7)` is `128.0` `pow(9, .5)` is `3.0`
`fmod(x, y)`	remainder of x/y as a floating-point number	`fmod(13.657, 2.333)` is `1.992`
`sin(x)`	trigonometric sine of x (x in radians)	`sin(0.0)` is `0.0`
`cos(x)`	trigonometric cosine of x (x in radians)	`cos(0.0)` is `1.0`
`tan(x)`	trigonometric tangent of x (x in radians)	`tan(0.0)` is `0.0`

Dealing with functions in C



Function Declaration:

A function declaration (aka function prototype) is used to inform the compiler about the name, return type, and arguments of a function.

This information helps the compiler ensure that the function is used correctly throughout the program.

Syntax:

```
return_type function_name(parameter_type parameter_name, ...);
```

Example:


```
#include <stdio.h>

int square(int number); // function prototype
```

Benefits of Function Declaration:

Helps in catching errors early: Function declaration ensures that the function is used correctly throughout the program, and any errors in the function signature can be caught early in the compilation process.

Enhances readability and reusability: Function declaration makes the code more readable and modular by breaking it down into smaller, manageable pieces that can be reused throughout the program.

Facilitates teamwork: In large projects with multiple developers, function declaration can help team members understand the purpose and functionality of each function, making collaboration easier.

Function Definition:

Function definition is a block of code that specifies the implementation of a function. It provides the instructions for the function to execute when called.

Syntax:

```
return_type function_name(parameter_list)
{
    // function body return expression;
}
```



```
// square function definition returns the square of its parameter
int square(int number) { // number is a copy of the function's argument
   return number * number; // returns square of number as an int
}
```

Benefits of Function Definition:

- Code Reusability: By defining a function, we can reuse the code multiple times within the program, which reduces the amount of code duplication, making it easier to maintain and modify.
- Readability: Function definition helps in breaking down complex code into smaller, more manageable pieces, making it easier to understand and maintain.
- Modularity: By defining a function, we can create modular code that can be tested, debugged and maintained separately from the rest of the program, which makes it easier to detect and fix issues within the code.

Function Call:

A function call is a statement that transfers control from the current program to a specific function.

It involves passing arguments to the function and receiving a return value from the function.

Syntax:

```
return_value = function_name(argument_list);
```



```
printf("%d  ", square(x)); // function call
```

Examples of Functions:

```c
#include <stdio.h>

int add(int,int);      //function declaration
int main() {
  int num1, num2;
  printf("Enter two numbers: ");
  scanf("%d %d", &num1, &num2);
  int sum = add(num1, num2);          //Function Call
  printf("The sum of %d and %d is %d", num1, num2, sum);
  return 0;
}
//Function Definition
int add(int x, int y) {
  return x + y;
}
```

Examples of Functions:

```
#include <stdio.h>

int factorial(int);
int main() {
  int num = 5;
  int fact = factorial(num);
  printf("The factorial of %d is %d", num, fact);
  return 0;
}
int factorial(int n) {
  if(n == 0 || n == 1) {
    return 1;
  } else {
    return n * factorial(n-1);
  }
}
```

Examples of Functions:

```
#include <stdio.h>

int isPrime(int);
int main() {
  int num = 7;
  if(isPrime(num)) {
    printf("%d is a prime number",
num);
  } else {
    printf("%d is not a prime
number", num);
  }
  return 0;
}
```

```
int isPrime(int num) {
  if(num < 2) {
    return 0;
  }
  for(int i = 2; i <= num/2; i++) {
    if(num % i == 0) {
      return 0;
    }
  }
  return 1;
}
```

Examples of Functions:

```
#include <stdio.h>
float celsiusToFahrenheit(float);
float fahrenheitToCelsius(float)
int main() {
  float tempInCelsius = 25;
  float tempInFahrenheit =
celsiusToFahrenheit(tempInCelsius);
  printf("%.2f degrees Celsius is %.2f degrees
Fahrenheit\n", tempInCelsius, tempInFahrenheit);


  tempInFahrenheit = 77;
  tempInCelsius =
fahrenheitToCelsius(tempInFahrenheit);
  printf("%.2f degrees Fahrenheit is %.2f degrees
Celsius", tempInFahrenheit, tempInCelsius);
  return 0;
}
```

```
float celsiusToFahrenheit(float celsius) {
  return (celsius * 9/5) + 32;
}


float fahrenheitToCelsius(float fahrenheit)
{
  return (fahrenheit - 32) * 5/9;
}
```

Questions?