University of Batna 2
Institute of hygiene and security
1st year LMD

Chapter 6: C functions

Introduction

- Like all programming languages, C allows you to break a program into **several parts** called "modules" or functions. A function is a block of code which only **runs when it is called**.
- Dividing a complex problem into smaller chunks makes our program easy to understand and reusable.
- Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Types of function

- There are two types of function:

- **Standard Library Functions:** Predefined in C
- **User-defined Function:** Created by users
- In this tutorial, we will focus mostly on user-defined functions.

C Library Functions

- Library functions are the built-in functions in C programming.
- Programmers can use library functions by invoking the functions directly; they don't need to write the functions themselves.
- Some common library functions in C are **`sqrt()`**, **`abs()`**, **`isdigit()`**, etc.
- In order to use library functions, we usually need to include the header file in which these library functions are defined.
- For instance, in order to use mathematical functions such as **`sqrt()`** and **`abs()`**, we need to include the **`header`** file **`cmath`**.

Commonly used functions in the cmath library

- **Basic Arithmetic Functions:**

  - **`abs(x)`:** Absolute value of a number.
  - **`fmod(x, y)`:** Remainder of division of x by y.
  - **`modf(x, &y)`:** Breaks x into its fractional and integer parts.

Commonly used functions in the cmath library

- **Power and Exponential Functions:**
  - **`pow(x, y)`:** Returns x raised to the power y.
  - **`sqrt(x)`:** Square root of x.
  - **`cbrt(x)`:** Cube root of x.
  - **`exp(x)`:** Exponential function e^x.
  - **`exp2(x)`:** Exponential function 2^x.
  - **`log(x)`:** Natural logarithm (base e) of x.
  - **`log10(x)`:** Logarithm with base 10 of x.
  - **`log2(x)`:** Logarithm with base 2 of x.

Commonly used functions in the cmath library

- **Trigonometric Functions:**
  - **`sin(x)`:** Sine of x (x in radians).
  - **`cos(x)`:** Cosine of x (x in radians).
  - **`tan(x)`:** Tangent of x (x in radians).
  - **`asin(x)`:** Arc sine of x (result in radians).
  - **`acos(x)`:** Arc cosine of x (result in radians).
  - **`atan(x)`:** Arc tangent of x (result in radians).
  - **`atan2(y, x)`:** Arc tangent of y/x (result in radians).

Commonly used functions in the cmath library

- **Hyperbolic Functions:**
  - **`sinh(x)`:** Hyperbolic sine of x.
  - **`cosh(x)`:** Hyperbolic cosine of x.
  - **`tanh(x)`:** Hyperbolic tangent of x.
  - **`asinh(x)`:** Inverse hyperbolic sine of x.
  - **`acosh(x)`:** Inverse hyperbolic cosine of x.
  - **`atanh(x)`:** Inverse hyperbolic tangent of x.

Commonly used functions in the cmath library

- **Rounding and Fractional Functions:**
  - **`ceil(x)`:** Smallest integer greater than or equal to x.
  - **`floor(x)`:** Largest integer less than or equal to x.
  - **`round(x)`:** Rounds x to the nearest integer.
  - **`trunc(x)`:** Truncates the fractional part of x (rounds towards zero).
  - **`fract(x)`:** Fractional part of x.

Commonly used functions in the cmath library

- **Other Functions:**
  - **`isnan(x)`:** Checks if x is NaN (Not a Number).
  - **`isinf(x)`:** Checks if x is infinity.
  - **`isfinite(x)`:** Checks if x is finite.
  - **`lgamma(x)`:** Natural logarithm of the absolute value of the Gamma function.
  - **`tgamma(x)`:** Gamma function of x.

Commonly used functions in the cmath library

- Example:

```
#include <stdio.h>
#include <math.h>

int main() {
    double number, squareRoot;
    number = 25.0;
    squareRoot = sqrt(number);
    printf("Square root of %.1f = %.1f\n", number, squareRoot);
    return 0;
}
```

C User-defined Function

- C allows the programmer to define their own function.
- A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).
- When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

Create a Function

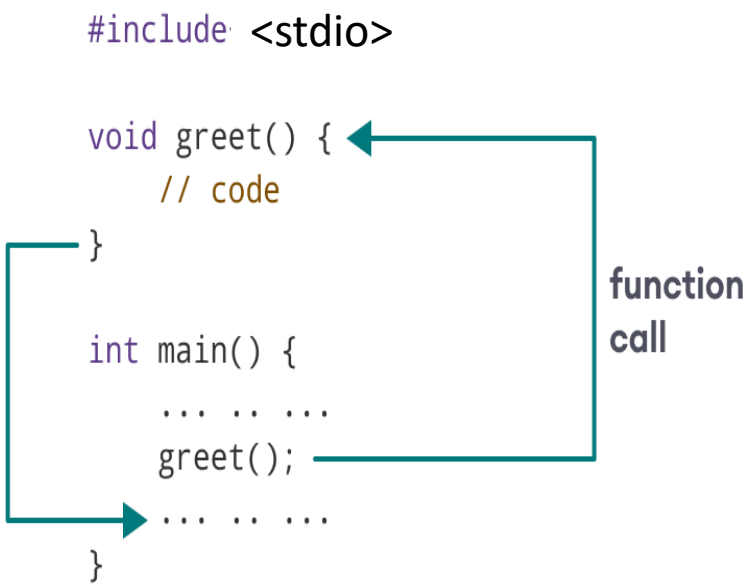
- To create (often referred to as declare) a function, specify the name of the function, followed by parentheses ():
- **Syntax**

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

- Where,
  - **`returnType`** : the return type of the value.
  - **`functionName`** : the name of the function.
  - **`parameter1, parameter2,...`**: a list of parameters for the function, separated by commas; each parameter is defined by a type and an identifier.
- Each time the function is called, the body of the function is executed.

Call a function

- Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called. It is always the main function that triggers the execution
- When the function is called, the program executes all the instructions within the function's body, and then resumes the program right after the function call.
- A function can be called multiple times in the program



C User-defined Function: Example

```
#include <stdio.h>
int add(int a, int b) {
    return a + b;
}

int main() {
    int x, y;
    printf("Enter the first number : ");
    scanf("%d", &x);
    printf("Enter the second number : ");
    scanf("%d", &y);

    printf("Sum of %d and %d is %d\n", x,y,add(x,y));

    return 0;
}
```

Parameter Passing to Functions

- The names of the parameters in the function header are called **formal parameters**. Their role is to allow the function to describe what it needs to do. For example, in the above program a and b are formal parameters.

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

formal parameters

Parameter Passing to Functions

- The parameters passed to the function are called **actual parameters or arguments**. For example, in the previous program, x and y are actual parameters.

```
add(x,y);
```

actual parameters

The return statement

- A **return** statement ends the processing of the current function and returns control to the caller of the function.
- **Syntax**

```
return expression;
```

- **`expression`**: must be of the same type as the return type of the function.
- **`expression`** can be:
  - a value; e.g., return(12);
  - a variable; e.g., return(a);
  - an expression; e.g., return(x+2*y-4);

Example :

```
#include <stdio.h>
int F1(int a, int b) {
    int s = 0;
    for (int i = 1; i <= b; i++) {
        s = s + a;
    }
    return s;
}
int F2(int c, int d) {
    int f = 0;
    while (c >= d) {
        c = c - d;
        f++;
    }
    return f;
}
int main() {
    int x = F1(4, 5);
    int y = F2(15, 5);
    printf("Return of F1(4,5) %d\n", x);
    printf("Return of F2(15,5) %d\n", y);
    return 0;}
```

Fuction without a return value

- When a function does not return any result, this is indicated using the keyword **`void`**.
- Example: `void no_result(int n)`
- Such functions perform actions or operations but do not return any value to the caller.

Fuction without a return value: Example

```
#include <stdio.h>

void printSum(int num1, int num2) {
    int sum = num1 + num2;
    printf("The sum of %d and %d is:
%d\n", num1, num2, sum);
}

int main() {
    printSum(5, 3);
    return 0;
}
```

Variable scope

- it is important to learn how variables act inside and outside of functions. In C, variables are only accessible inside the region they are created. This is called scope.
- **Local Scope**
- A variable created inside a function belongs to the local scope of that function, and can only be used inside that function:
- A local variable cannot be used outside the function it belongs to.
- If you try to access it outside the function, an error occurs:

Variable scope

- **Global Scope**
- A variable created outside of a function, is called a global variable and belongs to the global scope.

- Global variables are available from within any scope, global and local:

Variable scope: Example

```
#include <stdio.h>
int globalVar = 100;
void displayVariables() {
    int localVar = 50;
    printf("Inside displayVariables function: \n");
    printf("Local variable = %d\n", localVar);
    printf("Global variable = %d\n", globalVar);
}
int main() {
    int localVar = 20;
    globalVar++;

    printf("Inside main function: \n");
    printf("Local variable = %d\n", localVar);
    printf("Global variable = %d\n", globalVar);

    displayVariables();

    return 0;}
```