CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 14

Functions in C (1)

Prepared by

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering

Jatiya Kabi Kazi Nazrul Islam University, Bangladesh

www.mijanrahman.com

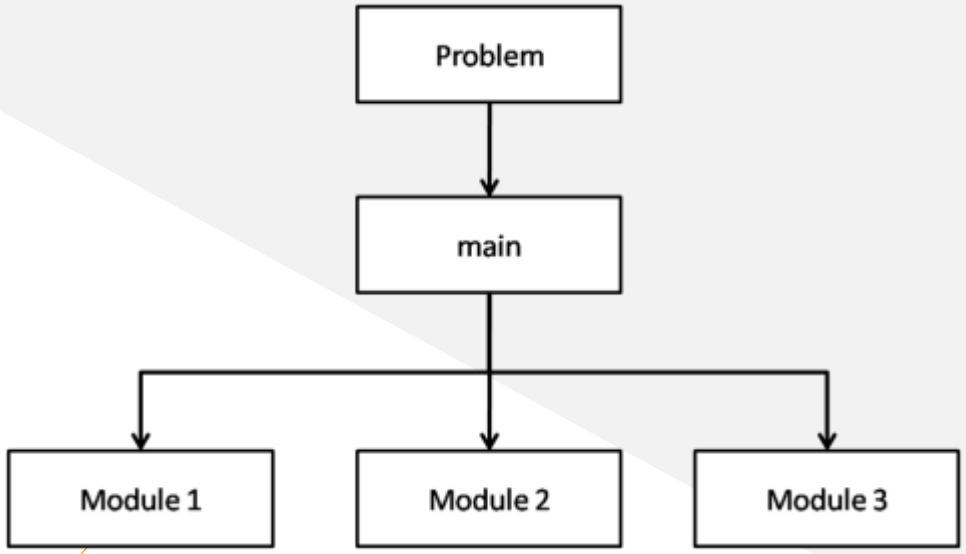
Contents

FUNCTIONS IN C

- Top-Down Modular Programming using Functions
- Functions in C
- Why do we need function?
- Types of Functions
- Elements of User-defined Function
- Function Definition
- Function Declaration
- Function Calling
- Parameters passing to Functions
- Main Function
- Library Functions

Top-Down Modular Programming using Functions

- In top down approach, we use following approach to solve any problem:
  1. First we will make a high level design of a problem statement.
  2. After that we will write the main function.
  3. From the main function we will call the sub functions. Generally, the large program is divided into small sub-functions (each function will do a specific task) which improves the modularity of a program.
  4. Later we will implement all the sub functions based on requirements.
- **That's why C is called the top down approach.**



Top-Down Modular Programming using Functions

- Example: Write a c program to implement a simple calculator.

- **High-Level Design:**
  - declare two integers
  - scan the values from the user
  - call addition
  - call subtraction
  - call multiplication
  - call division
- addition, subtraction, multiplication, division are sub functions.

Top-Down Modular Programming using Functions

- Write main function and call sub functions.

```c
int main()
{
    int a,b;

    scanf("%d%d", &a, &b);

    add(a,b);
    sub(a,b);
    mul(a,b);
    div(a,b);

    return 0;
}
```

Main function

```c
void add(int a, int b)
{
    printf("%d + %d = %d", a+b);
}

void sub(int a, int b)
{
    printf("%d - %d = %d", a-b);
}

void mul(int a, int b)
{
    printf("%d * %d = %d", a*b);
}

void div(int a, int b)
{
    if(b != 0) //division by zero is undefined.
        printf("%d / %d = %d", a/b);
}
```

Implement the sub functions

Functions

- **One of the strengths of C language is C functions.**
- In C, we can divide a large program into the basic building blocks known as **function**. The function contains the set of programming statements enclosed by {}.
- A function can be called multiple times to provide reusability and modularity to the C program.
- In other words, we can say that **the collection of functions creates a program**. The function is also known as ***procedure*** or ***subroutine*** in other programming languages.

Functions

- **How a function works in C programming?**
- **Declaration:** Before using a function in the program, we need to declare it. A function declaration tells the compiler about the function's name, return type, and parameters (if any). This is also called a function prototype. Example:

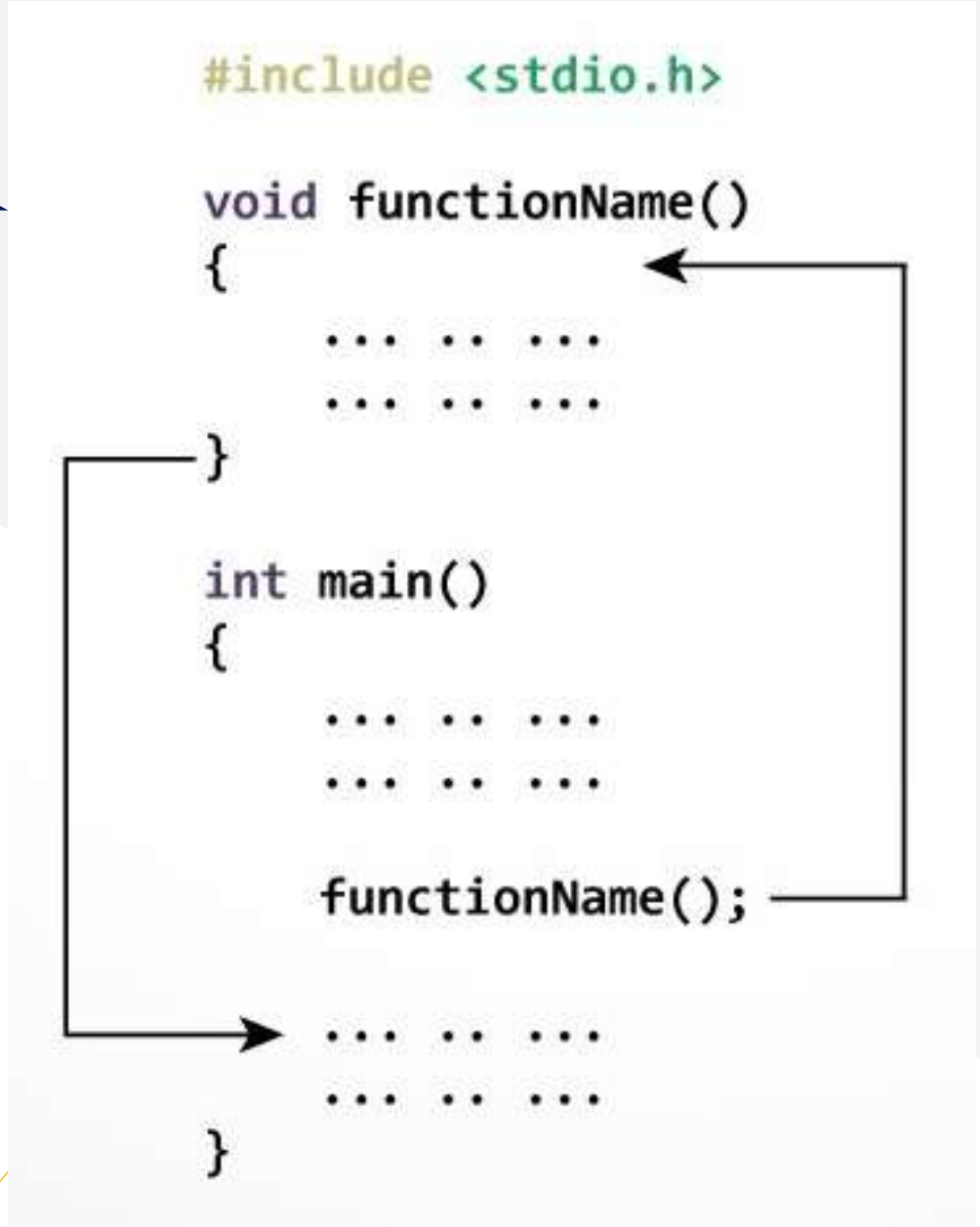
```
int add(int a, int b);
```

- **Definition:** The function definition contains the actual implementation of the function. It specifies what the function does when called. Example:

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

- **Calling:** To use a function, we call it by its name, providing arguments if necessary. When a function is called, the control transfers to the function definition. Example:

```
int sum = add(3, 5);
```

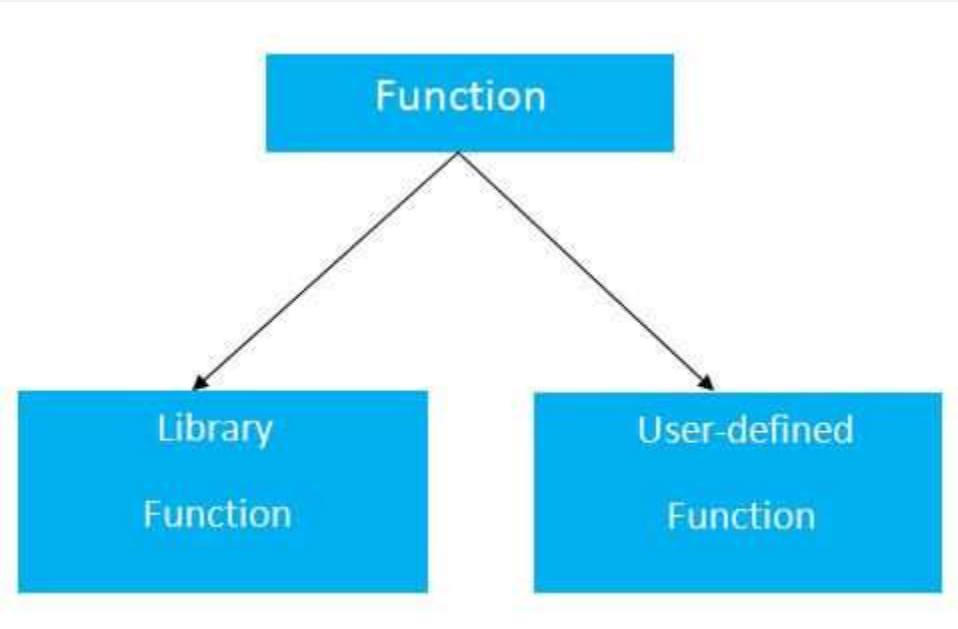


Why do we need functions?

- **By using functions, we can avoid rewriting same logic/code again and again in a program. Functions help us in reducing code redundancy.**
- If functionality is performed at multiple places in software, then rather than writing the same code, again and again, **we create a function and call it everywhere**. This also helps in maintenance as we have to change at one place if we make future changes to the functionality.
- **Functions make code modular.** Consider a big file having many lines of code. It becomes really simple to read and use the code if the code is divided into functions. **Thus, we can track a large C program easily when it is divided into multiple functions.**
- **Functions provide abstraction.** For example, we can use library functions without worrying about their internal working.
- **Reusability is the main achievement of C functions.**

Types of Functions

- **There are two types of functions in C programming:**
  - **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
  - **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.



Types of 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>.
- **Another example:** The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

Types of Functions

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

- In the example, the execution of a C program begins from the main() function.
- When the compiler encounters functionName();, control of the program jumps to:

```
void functionName()
```

How user-defined function works?

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

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

Elements of User-defined Function

- **There are three aspects of a C function: function declaration, function definition, and function call.**
- **Function declaration:**
  - A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.
- **Function call:**
  - Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.
- **Function definition:**
  - It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

Elements of User-defined Function

- **There are three aspects of a C function:**

SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

Defining a Function

- **The general form of a function definition** in C programming language is as follows –

```
return_type function_name(data_type parameter...){
//code to be executed
}
```

- It is also known as **the syntax of creating function** in C language.

Defining a Function

- **A function definition in C programming consists of a *function header* and a *function body***. Here are all the parts of a function:
- **Return Type: A function may return a value.** The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name: This is the actual name of the function.** The function name and the parameter list together constitute the function signature.
- **Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.** The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body: The function body contains a collection of statements** that define what the function does.

Defining a Function

- **Example:** This is a simple C program to demonstrate functions.

C Program using function:

```c
#include <stdio.h>

// An example function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
int max(int x, int y)
{
    if (x > y)
      return x;
    else
      return y;
}

// main function that doesn't receive any parameter and
// returns integer.
int main(void)
{
    int a = 10, b = 20;

    // Calling above function to find max of 'a' and 'b'
    int m = max(a, b);

    printf("m is %d", m);
    return 0;
}
```

Examples of Function Definition

1. Function with no parameters and no return value:

```c
void greet() {
    printf("Hello, world!\n");
}
```

- This function simply prints "Hello, world!" to the console.

2. Function with parameters and no return value:

```c
void multiply(int a, int b) {
    int result = a * b;
    printf("The result of multiplication is: %d\n", result);
}
```

- This function takes two integers as parameters, multiplies them, and prints the result.

Examples of Function Definition

3. Function with parameters and return value:

```
1 int add(int x, int y) {
2     return x + y;
3 }
```

- This function takes two integers as parameters, adds them, and returns the result.

4. Function with no parameters and return value:

```
1 int getRandomNumber() {
2     return rand() % 100; // Generates a random number between 0 and 99
3 }
```

- This function generates and returns a random number.

Examples of Function Definition

5. Function with array parameters:

```c
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

- This function takes an array and its size as parameters and prints all the elements of the array.

Examples of Function Definition

6. Function with pointer parameters:

```
1  void swap(int *x, int *y) {
2      int temp = *x;
3      *x = *y;
4      *y = temp;
5  }
```

- This function takes two integer pointers as parameters and swaps the values they point to.

Examples of Function Definition

7. Function with multi-dimensional array parameters:

```c
void printMatrix(int matrix[][3], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

- This function takes a 2D array and its dimensions as parameters and prints all the elements of the array.

THE END