CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 15

Functions in C (2)

Prepared by

Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering

Jatiya Kabi Kazi Nazrul Islam University, Bangladesh

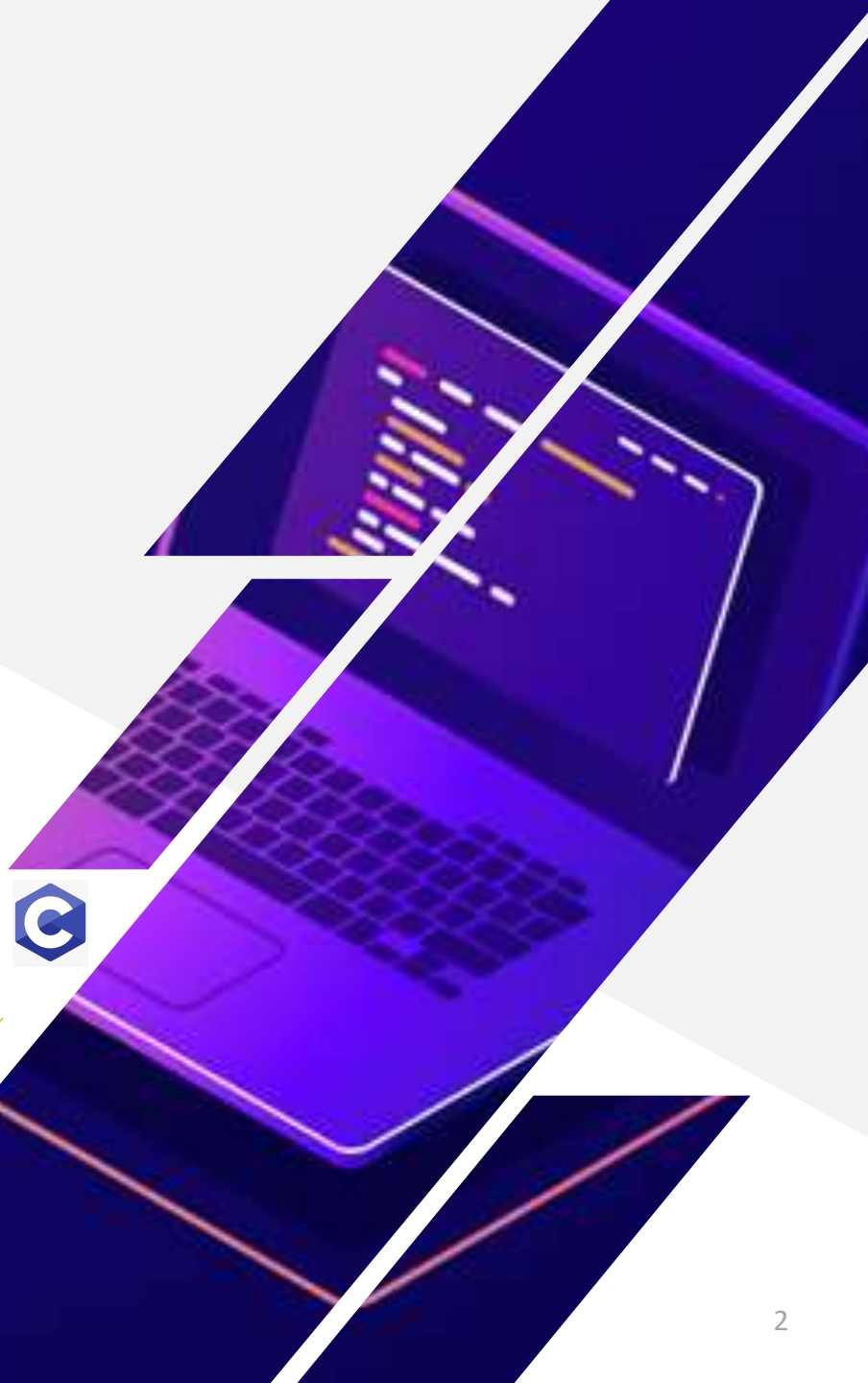
www.mijanrahman.com

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
- Category of Functions
- Parameters passing to Functions
- Main Function
- Library Functions



Function Declarations

- Like variables, all function in a C program must be declared, before they invoked. **A function declaration, also known as function prototype, consists of four parts**, such as-
  - **Function Type (Return Type)**
  - **Function Name**
  - **Parameter List**
  - **Terminating Semicolon (;)**

Function Declarations

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts:

```
return-type function-name( parameter list );
```

- For example:

```
int max(int num1, int num2);
```

- Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration:

```
int max(int, int);
```

- **Function declaration is required when you define a function in one source file and you call that function in another file.** In such case, you should declare the function at the top of the file calling the function.

Function Declarations

- A few acceptable forms of function **declaration** are:

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

- When a function does not take any parameter and does not return any value, its prototype is written as:

```
void display(void);
```

Function Calls

- While creating a C function, you give a definition of what the function has to do.
- **To use a function, you will have to call that function to perform the defined task.**
- When a program calls a function, the program control is transferred to the called function.
- **A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.**

- **For Example:**

```
int max(int x, int y)
{
        ----------------------
        ----------------------
}
int main()
{
        int a, b;
        ----------------------
        max(a,b);              //function calling
        ----------------------
}
```

Function Calls

- There are many different ways to call a function. Listed below are some of the ways the function **mul** can be invoked:

```
mul(10, 5)
mul(m, 5)
mul(10, n)
mul(m, n)
mul(m+5, 10)
mul(10, mul(m, n))
mul(exp1, exp2)
```

Function Calls

- **A function which returns a value can be used in expressions like any other variable.** Each of the following statements is valid:

```
printf("%d\n", mul(a, b);
y = mul(a, b) / (a+b);
if (mul(a, b)>total)
     print("large");
```

- However, a function cannot be used on the right side of an assignment statement. For instance,

```
mul(a, b) = 15;   is invalid.
```

Function Calls

- To call a function, we simply need to pass the required parameters along with the function name, and if the function returns a value, then we can store the returned value.

- **Example:**
- **Write a C Program To Swap Two Numbers using Function.**

```c
#include<stdio.h>

void swap(int, int);    //function declaration

int main()
{
    int a, b;

    printf("Enter values for a and b\n");
    scanf("%d%d", &a, &b);

    printf("\n\nBefore swapping: a = %d and b = %d\n", a, b);

    swap(a, b);    //function calling

    return 0;
}

void swap(int x, int y)    //function definition
{
    int temp;

    temp = x;
    x    = y;
    y    = temp;

    printf("\nAfter swapping: a = %d and b = %d\n", x, y);
}
```

Output :

```
Enter values for a and b
20
50


Before swapping: a = 20 and b = 50
After swapping: a = 50 and b = 20
```

Category of Functions

- A function, depending on whether arguments (or parameters) are present or not and whether a value is returned or not, may belong to one of the following categories:
  - Category 1: Functions with no arguments and no return values.
  - Category 2: Functions with arguments and no return values.
  - Category 3: Functions with arguments and one return value.
  - Category 4: Functions with no arguments, but return a value.
  - Category 5: Functions that return multiple values.

Category of Functions

- Category 1: Functions with no arguments and no return values.

- In C, we can define functions that don't take any arguments and don't return any value. Such functions are commonly used for tasks that don't require input parameters or for performing actions without needing to produce any specific result.

For example:

```c
#include <stdio.h>

// Function declaration
void greet();

int main() {
    // Function call
    greet();
    return 0;
}

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

Terminal

```
Hello, world!
```

Category of Functions

- Category 2: Functions with arguments and no return values.

- In C, we can define functions that accept arguments (also called parameters) but don't return any value. These functions are used when we need to perform some actions or operations on the provided arguments without needing to produce a specific result.

Here's an example:

```c
#include <stdio.h>

// Function declaration
void add(int num1, int num2);

int main() {
    int x, y;
    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);
    // Function call
    add(x, y);

    return 0;
}

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

Terminal

```
Enter two numbers: 10 20
The sum of 10 and 20 is: 30
```

Category of Functions

- Category 3: Functions with arguments and one return value.

- In C programming, functions can have arguments (parameters) and can also return a single value. This allows you to pass data into the function, perform some operations on that data, and then return a result back to the calling code.

Here's an example:

```c
#include <stdio.h>

// Function declaration
int add(int num1, int num2);

int main() {
    int x, y;
    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);

    // Function call
    int result = add(x, y);

    printf("The sum of %d and %d is: %d\n", x, y, result);

    return 0;
}

// Function definition
int add(int num1, int num2) {
    int sum = num1 + num2;
    return sum;
}
```

Terminal

```
Enter two numbers: 100 200
The sum of 100 and 200 is: 300
```

Category of Functions

- Category 4: Functions with no arguments, but return a value.

- In C programming, functions can be defined without any arguments but can still return a value. This allows you to perform some computation or task within the function and then return a result to the calling code.

Here's an example:

```c
#include <stdio.h>

// Function declaration
int getRandomNumber();

int main() {
    // Function call
    int randomNumber = getRandomNumber();

    printf("Random number generated: %d\n", randomNumber);

    return 0;
}

// Function definition
int getRandomNumber() {
    // Generate and return a random number
    return rand() % 100;
}
```

Terminal

```
Random number generated: 83
```

Category of Functions

- Category 5: Functions that return multiple values.
- **In C, functions inherently return only one value. However, we can simulate the concept of returning multiple values by using pointers or structures.**
- Two common approaches to achieve this:
  - **Using Pointers:** Pass pointers to variables to the function, which will modify the values stored at those memory locations.
  - **Using Structures:** Define a structure that holds multiple values, and return an instance of that structure from the function.

Category of Functions

- Category 5: Functions that return multiple values.

- **Using Pointers:** Pass pointers to variables to the function

Here's an example:

```c
#include <stdio.h>
// Function declaration
void getMinMax(int arr[], int size, int *min, int *max);

int main() {
    int numbers[] = {30, 10, 50, 20, 40};
    int min, max;
    // Function call
    getMinMax(numbers, 5, &min, &max);

    printf("Minimum value: %d\n", min);
    printf("Maximum value: %d\n", max);
    return 0;
}

// Function definition
void getMinMax(int arr[], int size, int *min, int *max) {
    *min = *max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < *min)
            *min = arr[i];
        else if (arr[i] > *max)
            *max = arr[i];
    }
}
```

Terminal

```
Minimum value: 10
Maximum value: 50
```

Category of Functions

- Category 5: Functions that return multiple values.

- **Using Structures:** Define a structure that holds multiple values, and return an instance of that structure from the function.

Here's an example:

```c
#include <stdio.h>
// Define a structure to hold min and max values
struct MinMax {
    int min;
    int max;
};
// Function declaration
struct MinMax getMinMax(int arr[], int size);

int main() {
    int numbers[] = {300, 100, 500, 200, 400};
    // Function call
    struct MinMax result = getMinMax(numbers, 5);
    printf("Minimum value: %d\n", result.min);
    printf("Maximum value: %d\n", result.max);
    return 0;
}
// Function definition
struct MinMax getMinMax(int arr[], int size) {
    struct MinMax result;
    result.min = result.max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] < result.min)
            result.min = arr[i];
        else if (arr[i] > result.max)
            result.max = arr[i];
    }
    return result;
}
```

Terminal

```
Minimum value: 100
Maximum value: 500
```

Example:

- **Function with Array Parameters:**

- This C program uses a function that takes an array of numbers as a parameter, calculates the sum and average of those numbers, and returns the results using pointers.

Here's an example:

```c
#include <stdio.h>
// Function declaration
void SumAverage(int arr[], int size, int *sum, double *average);

int main() {
    int numbers[] = {10, 20, 30, 40, 50};
    int size = sizeof(numbers) / sizeof(numbers[0]);
    int sum;
    double average;

    // Function call
    SumAverage(numbers, size, &sum, &average);
    printf("Sum of numbers: %d\n", sum);
    printf("Average of numbers: %.2f\n", average);
    return 0;
}
// Function definition
void SumAverage(int arr[], int size, int *sum, double *average) {
    *sum = 0;
    for (int i = 0; i < size; i++) {
        *sum += arr[i];
    }
    *average = (double)(*sum) / size;
}
```

Terminal

```
Sum of numbers: 150
Average of numbers: 30.00
```

Example:

- **Function with Parameters and return:**

- This C program uses a function that calculates the principal amount after a period of time:

$$A = P\,(1+r/100)^t$$

Here's an example:

```c
#include <stdio.h>
#include <math.h>
// Function declaration
double Principal(double p, double iRate, int n);

int main() {
    double p = 1000.0; // Initial principal amount
    double iRate = 5.0; // Annual interest rate (in percentage)
    int n = 5; // Time in years

    // Function call
    double finalAmount = Principal(p, iRate, n);
    printf("Principal amount after %d years: %.2f\n", n, finalAmount);
    return 0;
}
// Function definition
double Principal(double p, double iRate, int n) {
    double rate = iRate / 100.0;
    // Calculate final amount
    double finalAmount = p * pow((1 + rate), n);
    return finalAmount;
}
```

Terminal

```
Principal amount after 5 years: 1276.28
```

THE END