Trainer: Asif Nafees

1. Introduction to Function:
A function is a block of code designed to perform a specific task. It is a reusable code structure that can be called multiple times in a program, helping to make the program modular and organized.

Why Use Functions?
Reusability: Code can be reused without rewriting.
Modularity: Breaks the program into smaller, manageable parts.
Debugging: Makes it easier to find and fix errors.
Code Clarity: Increases readability and maintainability.

2. Types of Functions in C
i. Library Functions: Predefined functions in C's standard library (e.g., printf(), scanf(), sqrt()).
ii. User-defined Functions: Functions created by the programmer.

3. Function Syntax

```
return_type function_name(parameter_list) {
    // Function body
    // Code to be executed
    return value;
}
```

Explanation:
return_type: Data type of the value returned by the function (e.g., int, float, void).
function_name: Name of the function (identifier).
parameter_list: List of input parameters (arguments). It can be empty if no input is needed.
return: The value returned by the function (optional for void functions).

4. Function Declaration, Definition, and Call
Function Declaration (Prototype): Informs the compiler about the function name, return type, and parameters.
Function Definition: Contains the actual code of the function.
Function Call: Executes the function.

Example:

```
#include <stdio.h>
// Function Declaration
int add(int, int);
int main() {
    int num1 = 5, num2 = 10;
```

```
    int result = add(num1, num2); // Function Call
    printf("The sum is: %d\n", result);
    return 0;
}
// Function Definition
int add(int a, int b) {
    return a + b;
}
```

Output:
The sum is: 15
Explanation:
Declaration: int add(int, int);
Definition: int add(int a, int b) { return a + b; }
Call: add(num1, num2);

5. Types of User-defined Functions
There are four types of functions based on the return type and arguments:

1. Function with No Arguments and No Return Value

```
#include <stdio.h>
void greet() {
    printf("Hello, World!\n");
}
int main() {
    greet(); // Function Call
    return 0;
}
```

Output: Hello, World!

2. Function with Arguments and No Return Value

```
#include <stdio.h>
void display(int n) {
    printf("Number is: %d\n", n);
}
int main() {
    display(10);
    return 0;
}
```

Output: Number is: 10

3. Function with No Arguments but with Return Value

```
#include <stdio.h>
int getNumber() {
    return 100;
}
int main() {
    int num = getNumber();
    printf("Number is: %d\n", num);
    return 0;
}
```

Output: Number is: 100

4. Function with Arguments and Return Value

```
#include <stdio.h>
int multiply(int a, int b) {
    return a * b;
}
int main() {
    int result = multiply(4, 5);
    printf("Product is: %d\n", result);
    return 0;
}
```

Output: Product is: 20

6. Passing Arguments to Functions

There are two ways to pass arguments:

1. Pass by Value

A copy of the actual parameter is passed.

Changes made to the parameter inside the function do not affect the original value.

Example:

```
#include <stdio.h>
void modify(int x) {
    x = 50;
}
int main() {
    int num = 10;
    modify(num);
    printf("Value of num: %d\n", num); // Output: 10
    return 0;
```

```
}
```

2. Pass by Reference (Using Pointers)
The address of the actual parameter is passed.
Changes made to the parameter affect the original value.
Example:

```
#include <stdio.h>
void modify(int *x) {
    *x = 50;
}
int main() {
    int num = 10;
    modify(&num);
    printf("Value of num: %d\n", num); // Output: 50
    return 0;
}
```

7. Recursive Functions
A recursive function is a function that calls itself. It is useful for problems that can be divided into smaller, similar problems.
Example: Factorial Calculation

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
int main() {
    int num = 5;
    printf("Factorial of %d is: %d\n", num, factorial(num));
    return 0;
}
```

Output:
Factorial of 5 is: 120
Explanation:
The function factorial() calls itself until n becomes 0.
Base condition: if (n == 0), return 1.

8. Library Functions
These are predefined functions in C's standard library:
Mathematical Functions: sqrt(), pow(), abs()

String Functions: strlen(), strcpy(), strcmp()
Input/Output Functions: printf(), scanf()
Utility Functions: exit(), malloc(), free()

Example of Library Function:

```
#include <stdio.h>
#include <math.h>
int main() {
    double num = 16.0;
    double result = sqrt(num);
    printf("Square root of %.2f is %.2f\n", num, result);
    return 0;
}
```

Output:
Square root of 16.00 is 4.00

9. Advantages of Using Functions
Code Reusability: Avoids code duplication.
Simplifies Complex Problems: Breaks down complex tasks.
Enhances Readability and Maintainability: Organized and easier to understand code.
Improves Debugging: Errors can be traced easily in small functions.