1. Introduction to Functions

A function in C++ is a self-contained block of code that performs a specific task. Functions help in breaking a large program into smaller, manageable parts. Each function is designed to perform one particular operation, making the program easier to understand, debug, and maintain.

Functions promote **modularity**, **code reuse**, and **clarity**. Instead of writing the same code repeatedly, a function can be written once and called multiple times from different parts of the program.

---

2. Need for Functions in Programming

Functions are necessary in programming due to the following reasons:

- Reduce code duplication
- Improve readability
- Simplify debugging and testing
- Enhance program structure
- Support teamwork in large projects

Without functions, programs become lengthy, confusing, and difficult to maintain.

---

3. Types of Functions in C++

C++ supports two main types of functions:

1. **Built-in Functions**
   These are predefined functions provided by C++ libraries, such as:
   - cout, cin
   - sqrt(), pow()
   - strlen()
2. **User-defined Functions**
   These functions are created by programmers to perform specific tasks according to program requirements.

---

4. Components of a Function

A function in C++ consists of three main parts:

1. **Function Declaration (Prototype)**
2. **Function Definition**
3. **Function Call**

Each component plays an important role in function execution.

5. Function Declaration (Prototype)

A function declaration tells the compiler about the function name, return type, and parameters.

Syntax

```
return_type function_name(parameter_list);
```

Example

```
int add(int, int);
```

Function declarations are usually written before the main() function.

---

6. Function Definition

The function definition contains the actual code that executes when the function is called.

Syntax

```
return_type function_name(parameters)
{
   statements;
   return value;
}
```

Example

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

---

7. Function Call

A function call is used to execute the function.

Example

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

When a function is called:

- Control transfers to the function
- Statements inside the function execute
- Control returns to the calling function

---

8. Types of User-Defined Functions

User-defined functions are classified into four types:

1. No arguments, no return value
2. Arguments, no return value
3. No arguments, return value
4. Arguments and return value

Example

```
void display();        // no argument, no return
int square(int x);     // argument and return value
```

---

9. Function Arguments and Parameters

- **Parameters** are variables defined in the function declaration
- **Arguments** are values passed during function call

Example

```
void show(int x)   // parameter
{
   cout << x;
}

show(10);        // argument
```

---

10. Call by Value

In call by value, a copy of the argument is passed to the function.

Example

```
void change(int x)
{
   x = 20;
}
```

Changes made inside the function do not affect the original variable.

---

11. Call by Reference

In call by reference, the address of the variable is passed.

Example

```
void change(int &x)
{
   x = 20;
}
```

Changes made inside the function affect the original variable.

---

12. Default Arguments

Default arguments allow a function to use predefined values if arguments are not provided.

Example

```
int add(int a, int b = 5)
{
   return a + b;
}
```

---

13. Inline Functions

Inline functions reduce function call overhead by replacing the function call with function code.

Example

```
inline int square(int x)
{
   return x * x;
}
```

---

14. Recursive Functions

A recursive function is a function that calls itself.

Example

```
int fact(int n)
{
   if (n == 0)
      return 1;
   else
      return n * fact(n - 1);
}
```

Recursion must have a **base condition** to stop execution.

---

15. Advantages of Functions

- Code reuse
- Better organization
- Easy debugging
- Reduced complexity
- Improved readability

16. Limitations of Functions

- Function calls add overhead
- Poor design can increase complexity
- Excessive parameters reduce clarity

---

17. Applications of Functions

Functions are used in:

- Mathematical calculations
- File handling
- Game development
- Banking systems
- Scientific programs

---

18. Best Practices for Using Functions

- Use meaningful function names
- Keep functions small
- Avoid global variables
- Use comments
- Follow proper indentation

---

19. Difference Between main() and User-Defined Functions

main()	User-defined Function
Program execution starts here	Called from main
Only one main function	Multiple allowed
Mandatory	Optional

---

20. Conclusion

Functions are a core concept in C++. They help divide complex problems into simpler parts and promote reusable, readable, and efficient code. Understanding functions is essential for writing structured and professional C++ programs.