Unit 4: Function

Functions are one of the most important concepts in C programming. They help in breaking large programs into smaller, manageable modules.

4.1 Function Concept

◇ Definition:

A **function** is a block of code that performs a specific task and can be reused whenever required.

◇ Why Use Functions?

- Code reusability
- Reduces repetition
- Makes program modular
- Easy debugging
- Improves readability

◇ Basic Structure of a Function:

```
return_type function_name(parameters)
{
    // body of function
    return value;
}
```

4.2 Function Prototype, Call and Definition

There are three main parts of a function:

1️⃣ Function Prototype (Declaration)

Tells the compiler about:

- Function name
- Return type
- Number and type of arguments

◇ Syntax:

```
return_type function_name(data_type, data_type);
```

Example:

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

2️⃣ Function Definition

Actual body of the function.

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

3️⃣ Function Call

Calling the function inside `main()`.

```
result = add(10, 20);
```

☑️ Complete Example:

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

```
// Function Prototype
int add(int, int);

int main()
{
    int result;
    result = add(10, 20);   // Function call
    printf("Sum = %d", result);
    return 0;
}

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

4.3 Different Ways of Using Function

Functions can be classified based on:

Arguments	Return Type	Type
No	No	1
Yes	No	2
No	Yes	3
Yes	Yes	4

◇ 1 No Arguments, No Return Value

```
#include <stdio.h>

void greet()
{
    printf("Hello World");
}

int main()
{
    greet();
```

```
    return 0;
}
```

◇ 2️⃣ Arguments, No Return Value

```
#include <stdio.h>

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

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

◇ 3️⃣ No Arguments, With Return Value

```
#include <stdio.h>

int getNumber()
{
    int num;
    printf("Enter number: ");
    scanf("%d", &num);
    return num;
}

int main()
{
    int n = getNumber();
    printf("You entered %d", n);
    return 0;
}
```

◇ 4️⃣ 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 = %d", result);
    return 0;
}
```

4.4 Call by Value and Call by Reference

◇ Call by Value

- Copy of actual parameter is passed.
- Changes inside function do NOT affect original value.

Example:

```
#include <stdio.h>

void change(int x)
{
    x = 100;
}

int main()
{
    int a = 10;
    change(a);
    printf("Value of a = %d", a);
```

```
    return 0;
}
```

◇ Output:

```
Value of a = 10
```

◇ Call by Reference

- Address of variable is passed.
- Changes affect original variable.
- Implemented using pointers.

Example:

```
#include <stdio.h>

void change(int *x)
{
    *x = 100;
}

int main()
{
    int a = 10;
    change(&a);
    printf("Value of a = %d", a);
    return 0;
}
```

◇ Output:

```
Value of a = 100
```

4.5 Recursion

◇ Definition:

Recursion is a process in which a function calls itself.

◇ Two Important Parts:

1. Base condition (Stopping condition)
2. Recursive call

◇ Example: Factorial Using Recursion

```
#include <stdio.h>

int factorial(int n)
{
    if(n == 0)
        return 1;        // Base case
    else
        return n * factorial(n - 1);  // Recursive call
}

int main()
{
    int num = 5;
    printf("Factorial = %d", factorial(num));
    return 0;
}
```

◇ Output:

```
Factorial = 120
```

C Program with Function

```c
#include <stdio.h>

// Function prototype
int square(int);

int main()
{
    int num = 4;
    printf("Square = %d", square(num));
    return 0;
}

// Function definition
int square(int x)
{
    return x * x;
}
```

Short Summary

Topic	Key Point
Function	Block of reusable code
Prototype	Declaration of function
Call	Invoking function
Definition	Actual body
Call by value	Copy passed
Call by reference	Address passed
Recursion	Function calls itself