**Function**

In C, a function is a block of code that performs a specific task. Functions allow you to break down a large program into smaller, more manageable sections. A function typically has a name, a return type, and optional parameters (inputs) that it operates on.

**Structure of a Function**

return\_type function\_name(parameter\_list) {

// Body of the function

// Code to be executed

return value; // if return\_type is not void

}

**return\_type**: Specifies the type of data the function will return (e.g., int, float, void if no return value).

**function\_name**: The name of the function.

**parameter\_list**: A list of variables passed into the function (optional). These are used to provide data to the function.

**body of the function**: This contains the actual code that defines what the function does.

**return**: (Optional) This statement is used to return a value to the calling function if the return type is not void.

**Types of Functions in C**

1. **Built-in Functions**: Functions that are provided by the C standard library, such as printf(), scanf(), strlen(), etc.
2. **User-defined Functions**: Functions that you define yourself to perform specific tasks.

**Simple Function Example**

Let's start with a basic example where we define a function that adds two integers.

#include <stdio.h>

// Function declaration

int add(int, int);

int main() {

int result;

result = add(5, 3); // Function call

printf("Sum: %d\n", result);

return 0;

}

// Function definition

int add(int a, int b) {

return a + b;

}

**Explanation:**

* add is a user-defined function that takes two int arguments and returns their sum as an int.
* In the main() function, we call the add() function with 5 and 3 as arguments, and it returns 8, which is then printed.

2. **Function Without Return Value (void)**

In C, you can have functions that do not return a value. These are declared with the void return type.

#include <stdio.h>

// Function declaration

void greet(void);

int main() {

greet(); // Function call to greet()

return 0;

}

// Function definition

void greet(void) {

printf("Hello, welcome to C programming!\n");

}

**3. Function with Multiple Parameters**

A function can take multiple parameters, which are passed in when calling the function.

#include <stdio.h>

// Function declaration

float average(int, int, int);

int main() {

float avg = average(10, 20, 30); // Function call with three arguments

printf("Average: %.2f\n", avg);

return 0;

}

// Function definition

float average(int a, int b, int c) {

return (a + b + c) / 3.0; // Calculates the average

}

**Explanation:**

* The function average takes three integer arguments and returns the average as a float.
  1. **Function with Return Value**

Here’s an example of a function that calculates the factorial of a number using recursion.

#include <stdio.h>

// Function declaration

int factorial(int n);

int main() {

int num = 5;

printf("Factorial of %d is %d\n", num, factorial(num));

return 0;

}

// Function definition

int factorial(int n) {

if (n == 0) {

return 1; // Base case

} else {

return n \* factorial(n - 1); // Recursive call

}

}

**Explanation:**

* The factorial function is defined recursively. If n is 0, it returns 1 (base case). Otherwise, it calls itself with n-1 and multiplies the result by n.

factorial(5)

|

v

5 \* factorial(4)

|

v

4 \* factorial(3)

|

v

3 \* factorial(2)

|

v

2 \* factorial(1)

|

v

1 \* factorial(0)

|

v

return 1

As each recursive call reaches the base case (factorial(0)), it begins to return the results, multiplying each return value by the current value of n until the final result is computed.

Now, the recursive calls start returning the results:

* **Return 1: factorial(1) = 1 \times 1 = 1**
* **Return 2: factorial(2) = 2 \times 1 = 2**
* **Return 3: factorial(3) = 3 \times 2 = 6**
* **Return 4: factorial(4) = 4 \times 6 = 24**
* **Return 5: factorial(5) = 5 \times 24 = 120**

Finally, the factorial(5) call returns 120, which is the result.