Functions in C

Functions in C are reusable blocks of code that improve organization and reduce redundancy. A function consists of:

- A **return type** (specifying the data type of the return value, if any)
- A **name** (identifier for calling the function)
- **Parameters** (optional inputs for processing)
- A **body** (the block of code that defines the function's behavior)

To use a function, you typically declare it first (**function prototype**), define it later, and then call it in main() or other functions.

Types of Functions in C

There are four main types of functions:

1. Without Parameters and Without a Return Value

Used for tasks that don’t require input or output, like printing a message.

```
#include <stdio.h>

void greet() {
    printf("Hello, World!\n");
}

int main() {
    greet();
    return 0;
}
```

2. With Parameters and Without a Return Value

Takes input but doesn’t return a result, useful for operations like displaying values.

```
#include <stdio.h>

void displayNumber(int num) {
    printf("Number: %d\n", num);
}
```

```
int main() {
    displayNumber(7);
    return 0;
}
```

3. Without Parameters and With a Return Value

Returns a result but doesn’t take input, often used for fetching predefined values.

```
#include <stdio.h>
int getNumber() {
    return 10;
}
int main() {
    int num = getNumber();
    printf("Returned Number: %d\n", num);
    return 0;
}
```

4. With Parameters and With a Return Value

The most flexible type, used for calculations and data processing.

```
#include <stdio.h>
float divide(float a, float b) {
    return a / b;
}
int main() {
    float result = divide(10, 2);
    printf("Result: %.2f\n", result);
    return 0;
}
```

5. Recursion (A Function Calling Itself)

Recursion is a special case where a function calls itself, commonly used in problems like factorial calculation.

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial(n - 1);
}

int main() {
    printf("Factorial of 5: %d\n", factorial(5));
    return 0;
}
```

Performing Arithmetic Operations with Numeric Variables in C

C supports basic arithmetic operations:

- **Addition (+)**
- **Subtraction (-)**
- **Multiplication (*)**
- **Division (/)**
- **Modulus (%)** (returns the remainder of integer division)

Notes:

- Integer division truncates decimals. To get precise results, use float or double.
- Operator precedence follows standard math rules: multiplication and division are evaluated before addition and subtraction.
- Use parentheses () to explicitly define execution order.

- Shorthand operators (+=, -=, *=, /=, %=) simplify calculations by updating variables directly.

Example:

```
#include <stdio.h>

int main() {
   int a = 10, b = 3;
   printf("Addition: %d\n", a + b);
   printf("Subtraction: %d\n", a - b);
   printf("Multiplication: %d\n", a * b);
   printf("Division: %f\n", (float)a / b);
   printf("Modulus: %d\n", a % b);
   return 0;
}
```

Recursion in C

Recursion is a programming technique where a function calls itself to solve a problem. It is commonly used in problems that can be broken down into smaller subproblems of the same type.

How Recursion Works

A recursive function must have:

1. **Base Case** – A condition that stops the recursion.
2. **Recursive Case** – A call to the function itself with a smaller problem.

Types of Recursion in C

1. Direct Recursion

A function calls itself directly within its definition.

```
void directRecursion(int n) {
   if (n == 0) return;
```

```
    printf("%d ", n);
    directRecursion(n - 1);
}
```

2. Indirect Recursion

A function calls another function, which then calls the first function again.

```
void functionA(int n);
void functionB(int n) {
    if (n == 0) return;
    printf("%d ", n);
    functionA(n - 1);
}
void functionA(int n) {
    if (n == 0) return;
    printf("%d ", n);
    functionB(n - 1);
}
```

3. Tail Recursion

The recursive call is the last operation in the function, allowing optimizations.

```
int tailRecursion(int n, int result) {
    if (n == 0) return result;
    return tailRecursion(n - 1, n * result);
}
```

4. Head Recursion

The recursive call occurs before any other operations.

```
void headRecursion(int n) {
    if (n == 0) return;
```

```
    headRecursion(n - 1);
    printf("%d ", n);
}
```

5. Tree Recursion

A function makes more than one recursive call within itself.

```
void treeRecursion(int n) {
    if (n == 0) return;
    printf("%d ", n);
    treeRecursion(n - 1);
    treeRecursion(n - 1);
}
```

6. Nested Recursion

A recursive function calls itself as an argument within the same function.

```
int nestedRecursion(int n) {
    if (n > 100) return n - 10;
    return nestedRecursion(nestedRecursion(n + 11));
}
```