Loop Understanding in C

Loops in C let you repeat actions without writing the same code again and again. They help make programs more efficient, cleaner, and easier to read. This guide explains loops not just by showing the syntax but by walking you through them in plain language — like a conversation with a teacher. You'll see how each loop works, what makes it unique, and where you might use it in real life.

1. For Loop

Think of a for loop like counting steps on a staircase. You know exactly how many steps there are, so you keep walking until you reach the top. In C, a for loop is perfect for situations where you already know how many times you want something to happen. You set a starting point, a stopping point, and how you're going to move between them.

```
for(initialization; condition; increment/decrement) {
    // code to repeat
}
#include <stdio.h>

int main() {
    for(int i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

This prints numbers from 1 to 5 because we told it to start at 1, keep going until 5, and add one each time.

2. While Loop

Now imagine you're filling a bottle with water — you don't know exactly how many cups it will take. You keep pouring until the bottle is full. That's like a while loop: you don't know how many times it will run, you just have a condition and you keep going until it's no longer true.

```
while(condition) {
    // code to repeat
}
#include <stdio.h>

int main() {
    int i = 1;
    while(i <= 5) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

We start at 1 and keep printing numbers while i <= 5. Once i becomes 6, the loop stops.

3. Do-While Loop

Sometimes you want to do something at least once before checking a condition — like tasting soup before deciding if it needs more salt. A do-while loop runs first, then checks the condition, making sure your code executes at least one time.

```
do {
    // code to repeat
} while(condition);
#include <stdio.h>
```

```
int main() {
    int i = 1;
    do {
        printf("%d ", i);
        i++;
    } while(i <= 5);
    return 0;
}
```
This prints 1 through 5. Even if i started greater than 5, it would still run once.

4. Nested For Loop

Nested loops are loops inside loops — like rows and columns in a spreadsheet. The outer loop handles one dimension (like rows), while the inner loop handles the other (like columns). They're great for patterns, tables, and grids.

```
#include <stdio.h>

int main() {
    for(int i = 1; i <= 3; i++) {
        for(int j = 1; j <= 3; j++) {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```
This prints a 3x3 grid of stars. The outer loop creates rows, and the inner loop prints stars in each row.

5. Additional Practice Examples

Here are some small programs you can try yourself. They're like mini-exercises to help you get comfortable with loops. Look at how each example uses a different loop to solve a real-world style problem.

a) Sum of first 10 natural numbers

```
#include <stdio.h>

int main() {
    int sum = 0;
    for(int i = 1; i <= 10; i++) {
        sum += i;
    }
    printf("Sum = %d\n", sum);
    return 0;
}
```

b) Print even numbers from 1 to 20

```
#include <stdio.h>

int main() {
    int i = 2;
    while(i <= 20) {
        printf("%d ", i);
        i += 2;
    }
    return 0;
}
```

c) Factorial of a number using do-while

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

```c
int main() {
    int num = 5, fact = 1;
    int i = 1;
    do {
        fact *= i;
        i++;
    } while(i <= num);
    printf("Factorial of %d is %d\n", num, fact);
    return 0;
}
```

d) Multiplication table of 5 using for loop

```c
#include <stdio.h>

int main() {
    for(int i = 1; i <= 10; i++) {
        printf("5 x %d = %d\n", i, 5*i);
    }
    return 0;
}
```

e) 4x4 number grid using nested loops

```c
#include <stdio.h>

int main() {
    for(int i = 1; i <= 4; i++) {
        for(int j = 1; j <= 4; j++) {
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```