Loops In C:

It would recommend you to focus mostly on the theoratical concept, as you can learn or copy the syntax from any where, but the usage of right loop at the right place is crucial. In programming, we frequently need to perform an action, of again and again, with variations in the details each time. The mechanism, which meets this need, is the 'loop,' and loops' concept is the tutorial's main focus. The versatility of the computer lies in its ability to perform the set of instructions repeatedly. This involves repeating some code in the program, either a specified number of times or until a particular condition is satisfied. Loop control instructions are used to perform this repetitive operation.

Following are three types of loop in C programming:

- For loop
- While loop
- do-while loop

There are two kinds of loops:

1. **Entry Controlled loops:** In entry controlled loops, the test condition is evaluated before entering the loop body. The For Loop and While Loop are an example of entry controlled loops.
2. **Exit Controlled Loops**: In exit controlled loops, the test condition is tested at the end of the loop. The loop body will execute at least once, whether the test condition is true or false. The do-while loop is an example of an exit controlled loop.

What about an Infinite Loop?

An infinite loop also known as an endless loop occurs when a condition always evaluates to true. Usually, this is considered an error.

Sometimes, while executing a loop, it becomes necessary to jump out of the loop. For this, we will use the break statement or continue statement.

- **break statement**

When a break statement is encountered inside a loop whether it is a for loop or a while loop, the loop is terminated and the program continues with the statement immediately following the loop.

- **continue statement**

Using a continue statement in the loop will cause the control to go directly to the test-condition and then it will continue the loop process.

Do-While Loop:

A do-while loop executes the statements inside the body of the do-while loop before checking the condition. So if a condition is false in the first place, then they do while would run once. A **do-while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Unlike **for** and **while** loops, which test the loop condition first, then execute the code written inside the body of the loop, the **do-while** loop checks its condition at the end of the loop. Following is the syntax of the do-while loop in C programming.

```
do {
   statements );
} while( test condition );
```

If the test condition returns true, the flow of control jumps back up to do, and the set of statements in the loop executes again. This process repeats until the given test condition becomes false.

How does the do-while loop work?

- First, the body of the do-while loop is executed once. Only then, the test condition is evaluated.
- If the test condition returns true, the set of instructions inside the body of the loop is executed again, and the test condition is evaluated.
- This looping process goes on until the test condition becomes false.
- If the test condition returns false, then the loop terminates.

Example:

```
#include

int main()
{
    int num, index = 0;
    printf("Enter a number\n");
    scanf("%d", &num);
    do {
        printf("%d\n", index + 1);
        index = index + 1;
    } while (index < num);
•       return 0;
•   }
```

While Loop:

While loop is also called as a **pre-tested loop**. A while loop allows code to be executed multiple times, depending upon a given Boolean(true or false) condition. The while loop is mostly used in the case where the number of iterations is not known. If the number of iterations is known, then we use for loop.

The Syntax of while loop is:

```
while (condition test)
{
// Set of statements
}
```

The body of while loop can contain a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the test condition evaluates to true.

When the condition becomes false, the program control passes to the line immediately following the loop.

Example:

```
#include<stdio.h>

int main()
{
    int i = 0;
    while (i<54)
    {
        printf("%d\n", i);
        i = i+1;
    }
    return 0;
}
```

Explanation of the above program:-

1. We have **initialized** a variable i with value 0. This code will print from 0 to 4; hence the variable is initialized with value 0.
2. In a while loop, we have provided a **condition** (i<=5), which means the loop will execute the body until the value of i becomes 5. After that, the loop will be terminated.
3. In the body of a loop, we have a print function to print our number and an **increment operation** (i++) to increment the value per execution of a loop. This process will continue until the value becomes 5 and then it will print the number and then terminate the loop.

Properties of while loop:

- A conditional expression written in the brackets of while is used to check the condition. The Set of statements defined inside the while loop will execute until the given condition returns **false**.
- The condition will return **0** if it is **true**. The condition will be false if it returns any nonzero number.
- In the while loop, we cannot execute the loop until we do not specify the condition expression.
- It is possible to execute a while loop without any statements. This will give no **error**.
- We can have multiple conditional expressions in a while loop.
- Braces are optional if the loop body contains only one statement.

WHAT IS THE DIFFERENCE BETWEEN WHILE AND DO-WHILE LOOPS IN C?

While loop is executed when given test condition return true, whereas, do-while loop is executed for the first time irrespective the test condition is true or false, because the test condition is checked after executing while loop for the first time.

This difference between while and do-while will be more clear by the following program.

```
main( ) {
 while ( 2 < 1 )
printf ( "Hello World \n") ;
}
```

Here, since the condition fails the first time itself, the printf( ) statement will not get executed. Let's now write the same code using a do-while loop.

```
main( ) {
do {
printf ( "Hello World\n") ;
} while ( 2 < 1 ) ;
}
```

In this program, the printf( ) statement would be executed once, since first the body of the loop is executed, and then the test condition is evaluated.

For Loop:

Loop is one of the most important concepts in all programming languages as it simplifies complex problems and makes it easier to read and understand the code. Imagine a situation where you would have to print numbers from 1 to 1000. What would you do? Will you type in the printf() statement a thousand times? Using a **for loop**, we can perform this action in three statements.

The **"For" Loop** is used to repeat a specific code until a specific condition is satisfied. The for-loop statement is very specialized. We use for a loop when we know the number of iterations we want, whereas when we do not know about the number of iterations, we use while loop. Here is the syntax of the for loop in C programming.

The syntax of the for loop is:

```
for ( initialize counter ; test counter ; increment/decrement
counter)
 {
 //set of statements
 }
```

- **initialize counter**: It will initialize the loop counter value, i.e., i=0.
- **test counter**: It verifies whether the condition is true.
- **Increment/decrement counter**: Incrementing or decrementing the counter.
- **Set of statements**: Execute the set of statements.

Example:

```c
#include <stdio.h>

int main()
{
    int num = 10;
    int i;
    for(i = 0; i < num; i++) {
    printf("%d ",i);
}
    return 0;
}
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

Explanation of the above code:-

First, the initialization expression will initialize loop variables. The expression **i=0** executes once when the loop starts. Then the condition **i < num** is checked if it is **true**, then the statements inside the body of the loop are executed. After executing the statements inside the body, the control of the program is transferred to increment the variable by **1 (i++)**. The expression **i++** modifies the loop variables. Then the condition **i<num** is evaluated again. If the condition is still true, the body of the loop will execute once more. The for loop terminates when **i < num** becomes **false**.

Nested for loop:

Just as if statement, we can have for loop inside another for a loop. This is known as **nested for loop**. Similarly, while loop and do while loop can also be nested.

```c
for ( initialization; test condition; increment ) {
   for ( initialization; test condition; increment ) {
    // set of statements
   }
   // set of statements
}
```

Note: *there is no rule that a loop must be nested inside its own type. For loop can be nested inside the while loop and vice versa.*