Looping in C

Looping is the ability to repeat a block of statements a number of times.

Types of Loops

1. **Conditional Loops** – The loop stops when a condition is met. Examples:

   `while` loop

   `do while` loop

2. **Unconditional Loops** – The loop repeats a set/fixed number of times. Example:

   `for` loop

Relational Operators

Relational operators allow the comparison of expressions/values in conditions.

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

True and False in C

FALSE(0) – An expression evaluates to `0`.

TRUE(Non-Zero) – An expression evaluates to any **non-zero** integer (positive and negative).

For example

Let a<4 if a=3 then 3<4 Thus the expression is true and the loop continues.	Let a<4 if a=7 then 7<4 which is false Thus the expression is false and the loop exits

TYPES OF LOOPS IN C

Loops in c are used to execute a block of code repeatedly until a specified condition is met.
There are three main types of loops in C language i.e. **for loop, while loop and do-while loops.**

1.FOR LOOP

The for loop is used when the number of iterations is known.
It has three parts; initialization, condition and update.

Syntax

index.c

```
for (initialization: condition: update) {
  //Code to be executed
}
```

CodeImage

Explanation of Components:

1. Initialization:

This part is **executed only once at the beginning of the loop**.
It initializes a loop control variable (e.g., int i = 0).

2. Condition:

The loop executes **as long as** the **condition remains true**.
If the condition is false, the loop terminates. (e.g. i <= 10).

3. Update:
This part **modifies/changes** the loop control variable (**increments or decrements**) after each iteration. Examples:

- `i++` → **Increment by 1**
- `i--` → **Decrement by 1**
- `i += n` → **Increase by step `n`** (where `n` is an integer)
- `i *= n` → **Multiply by a factor `n`**

Example

index.c

```c
#include <stdio.h>

int main(){
    int sum = 10;
    for (int i=0; i<6; ++i){
        printf("Previous sum is %d\n", sum);
        sum = sum + i;
        printf("Current sum is %d\n", sum);
        printf("----------------------\n");
    }
    return 0;
}
```


index.c

```
Previous sum is 10
Current sum is 10
----------------------
Previous sum is 10
Current sum is 11
----------------------
Previous sum is 11
Current sum is 13
----------------------
Previous sum is 13
Current sum is 16
----------------------
Previous sum is 16
Current sum is 20
----------------------
Previous sum is 20
Current sum is 25
----------------------
```


index.c

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

int main(){
    double z=0;
    for (int x=100; x>=65; x-=5){
        double z=sqrt(x);
        printf("The square root of %d is %f\n",x,z);
    }
    return 0;
}
```


index.c

```
The square root of 100 is 10.000000
The square root of 95 is 9.746794
The square root of 90 is 9.486833
The square root of 85 is 9.219544
The square root of 80 is 8.944272
The square root of 75 is 8.660254
The square root of 70 is 8.366600
```


index.c

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

int main(){
    int product=1;
    for (int i=1;i<6;){
        product*=i++;
        printf("The product is %d\n", product);
        }
    return 0;
}
```


index.c

```
The product is 1
The product is 2
The product is 6
The product is 24
The product is 120
```

2.WHILE LOOP

While loops is used to repeatedly execute a block of code as long as the condition remains true.

Syntax

index.c

```c
while(condition) {
    // Code to execute
}
```

CodeImage

Example

index.c

```c
#include <stdio.h>

int main(){
    int i = 1;

    while (i <= 5) { //Loop continues as ling as i <= 5
    printf("i = %d\n", i); // Prints the current value of i
    i++; // increment i

    }

    return 0;
}
```

CodeImage

index.c

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

CodeImage

Explanation:

- The loop will start with i = 1 and print i = 1.
- Then i is incremented, and the condition i <= 5 is checked again.
- This continues until i becomes 6, at which point the condition is false, and the loop stops.

Key points:

- If the condition is initially false, the code inside the loop may not run at all.

Differences in for loop and while loop

Feature	For Loop	While Loop
Structure	Initialization, condition, increment in one line	Condition is separate from initialization and increment
When to Use	When the number of iterations is known	When the number of iterations is unknown or based on a condition
Compactness	More compact and concise	More flexible, but can be more verbose
Compactness	Checked before each iteration (condition)	Checked before each iteration (condition)
Initialization/Increment	Handled in the loop declaration	Handled manually inside the loop

3.Do-while loop

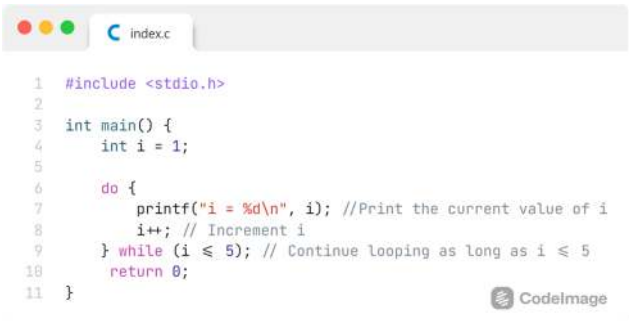
In C, a do-while loop is similar to a while loop, but with a key difference: the condition is checked **after** the block of code is executed. This means that the code inside the do block will always execute **at least once**, even if the condition is false the first time.

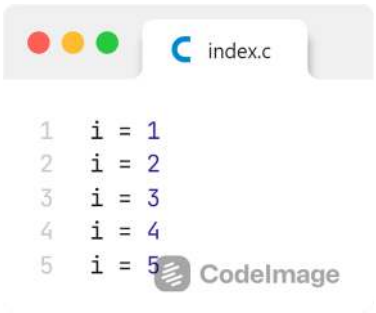
Syntax

Key points:

- The loop executes **at least once before** checking the condition.
- If the condition is **true** the loop continues.
- If the condition is **false** the loop stops.

Example





Explanation:

- The loop will first print i = 1 because the condition is checked after the first execution of the loop.
- Then i is incremented, and the condition i <= 5 is checked.
- This continues until i becomes 6, at which point the loop stops because the condition is now false.

Difference between while loop and do-while loop

FEATURE	WHILE LOOP	DO-WHILE LOOP
Execution check	Condition is checked first	Executes first, then checks condition
Minimum execution	May not execute at all	Executes atleast once
Best use case	When the loop may not run	When execution must happen at least once

Nested loops

A **nested loop** refers to a loop inside another loop.
It allows you to perform more complex repetitive tasks, such as;

- Iterating through multi-dimensional arrays.
- Working with patterns that require multiple levels of iteration.

Syntax

```
for (initialization; condition; update) {  // Outer loop
   for (initialization; condition; update) {  // Inner loop
      // Code to be executed
   }
}
```

- The outer loop controls the number of times the inner loop runs.
- The inner loop executes fully for each iteration of the outer loop.