# Loops & Iteration

In C programming, there is often a need for repeating the same part of the code multiple times. For example, to print a text three times, we have to use printf() three times as shown in the code:

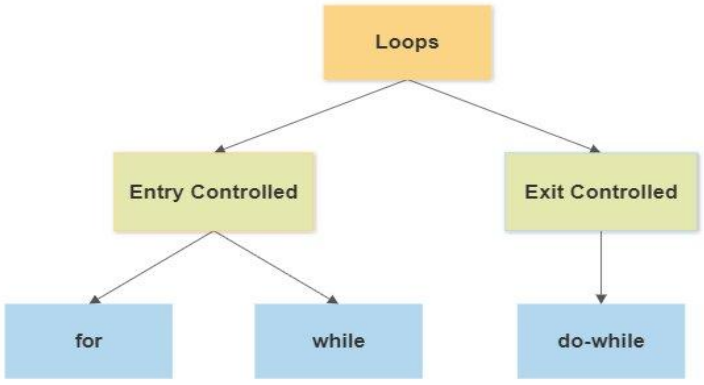
```
#include <stdio.h>

int main() {
   printf( "Hello\n");
   printf( "Hello\n");
   printf( "Hello");
   return 0;
}
```

Output:
Hello
Hello
Hello

But if we say to write this 20 times, it will take some time to write statement. Now imagine writing it 100 or 1000 times. Then it becomes a really hectic task to write same statements again and again. To solve such kind of problems, we have loops in programming languages.

**What are loops in C?**
**Loops** in C programming are used to repeat a block of code until the specified condition is met. It allows programmers to execute a statement or group of statements multiple times without writing the code again and again.

**Types of Loops in C**
There are 3 looping statements in C:



**for Loop**

**for loop** is an **entry-controlled** loop, which means that the condition is checked before the loop's body executes.

**Syntax**

```
for (initialization; condition; updation) {
    // body of for loop
}
```

The various parts of the for loop are:

- **Initialization:** Initialize the variable to some initial value.
- **Test Condition:** This specifies the test condition. If the condition evaluates to true, then body of the loop is executed. If evaluated false, loop is terminated.
- **Update Expression:** After the execution loop's body, this expression increments/decrements the loop variable by some value.
- **Body of Loop**: Statements to repeat. Generally enclosed inside **{} braces.**

## Example:

```
#include <stdio.h>

int main() {

    // Loop to print numbers from 1 to 5
    for (int i = 0; i < 5; i++) {
        printf( "%d ", i + 1);
    }

    return 0;
}
```

## Output

```
1 2 3 4 5
```

**while Loop**

A **while loop** is also an **entry-controlled loop** in which the condition is checked before entering the body.

**Syntax**

```
while (condition) {
    // Body of the loop
}
```

Only the **condition** is the part of **while loop** syntax, we have to initialize and update loop variable manually.

**Example:**

```
#include <stdio.h>

int main() {

    // Initialization expression

  int i = 0;

 // Test expression

  while(i <= 5) {

    printf("%d ", i + 1);

     // update expression

    i++;

  }

return 0;

}
```

## Output

```
1 2 3 4 5 6
```

**do-while Loop**

The do-while loop is an **exit-controlled loop**, which means that the condition is checked after executing the loop body. Due to this, the loop body will **execute at least once** irrespective of the test condition.

**Syntax**

```
do {
   // Body of the loop
} while (condition);
```

Like while loop, only the condition is the part of **do while loop** syntax, we have to do the initialization and updating of loop variable manually.

**Example:**

```
#include <stdio.h>

 int main() {

      // Initialization expression
```

```
  int i = 0;

    do

  {

    // loop body

    printf( "%d ", i);

   // Update expression

    i++;

    // Condition to check

  }  while (i <= 10);

   return 0;

}
```

## Output

```
0 1 2 3 4 5 6 7 8 9 10
```

**Infinite Loop**

An **infinite loop** is executed when the test expression never becomes false, and the body of the loop is executed repeatedly. A program is stuck in an Infinite loop when the condition is always true. Mostly this is an error that can be resolved by using Loop Control statements.

**Using for loop:**

```c
#include <stdio.h>

int main () {

        // This is an infinite for loop
        // as the condition expression
        // is blank
        for ( ; ; ) {
                printf("This loop will run forever.");
        }
        return 0;
}
```

**Output**

```
This loop will run forever.
This loop will run forever.
This loop will run forever.
...
```

**Using While loop:**

```c
#include <stdio.h>

int main()  {
        while (1)
                printf("This loop will run forever.\n");
        return 0;
}
```

**Output**

```
This loop will run forever.
This loop will run forever.
This loop will run forever.
...
```

**Using the do-while loop:**

```c
#include <stdio.h>

int main() {
        do {
                printf("This loop will run forever.");
        } while (1);

        return 0;
}
```

**Output**

```
This loop will run forever.
This loop will run forever.
This loop will run forever.
...
```

**Nested Loops**

Nesting loops means placing one loop inside another. The inner loop runs fully for each iteration of the outer loop. This technique is helpful when you need to perform multiple iterations within each cycle of a larger loop, like when working with a two-dimensional array or performing tasks that require multiple levels of iteration.

**Example:**

```c
#include <stdio.h>

int main() {
```

```
    // Outer loop runs 3 times
    for (int i = 0; i < 3; i++) {

        // Inner loop runs 2 times for each
        // outer loop iteration
        for (int j = 0; j < 2; j++) {
            printf("i = %d, j = %d\n", i, j);
        }
    }
    return 0;
}
```

## Output

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
```

**Loop Control Statements**

Loop control statements in C programming are used to change execution from its normal sequence.

| Name | Description |
|---|---|
| break | The break statement is used to terminate the loop statement. |
| continue | When encountered, the continue statement skips the remaining body and jumps to the next iteration of the loop. |
| goto | goto statement transfers the control to the labeled statement. |

**Example:**

```
#include <stdio.h>
int main() {
    for (int i = 0; i < 5; i++) {
        if (i == 3) {
            // Exit the loop when i equals 3
            break;
        }
        printf("%d ", i);
    }
    printf("\n");
    for (int i = 0; i < 5; i++) {
        if (i == 3) {
            // Skip the current iteration
            // when i equals 3
            continue;
        }
        printf("%d ", i);
    }
    printf("\n");
    for (int i = 0; i < 5; i++) {
        if (i == 3) {
            // Jump to the skip label when
            // i equals 3
            goto skip;
        }
```

```
        printf("%d ", i);
    }
    skip:
    printf("\nJumped to the 'skip' label %s",
    "when i equals 3.");
 return 0;
}
```

## Output

```
0 1 2
0 1 2 4
0 1 2
Jumped to the 'skip' label when i equals 3.
```