Loops

By: **Annamalai A**

Have you ever noticed tasks which involves doing something repeatedly for a number of times to achieve the desired result? This is called looping.

Looping is basically doing something repeatedly for a number of times until a condition is met. Say you're walking your way back home and you are at a street. Now you have to turn at the end of the street after going forward to reach your home. It can be said as "Walk 1 step forward until you reach the end of the street".

In programming (C) there are two ways to write loops — `for` and `while` loops.

`for` Loop

This loop takes in three statements — Initialization, condition and update, and runs a set of statements using the three given statements.

It creates a temparory variable that only lasts for the duration of the loop, beyond which it becomes undefined. The block of code written inside this loop is executed *for* all values of the temparory variables.

Here is the structure:

```
for (/* Initialize */; /* Condition */; /* Update */) {
    // Set of statements to be repeated
}
```

Here is an example where we want the compiler to write the first 5 natural numbers

```
#include <stdio.h>

void main() {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
```

```
    }
}
```

Here is an example where we want the compiler to print the sum of first 5 natural numbers

```
#include <stdio.h>

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

Here is an example where we want the compiler to print the first 10 negative integers

```
#include <stdio.h>

void main() {
    for (int i = -1; i >= -10; i--) {
        printf("%d\n", i);
    }
}
```

Here is an example where we want the compiler to print the first `n` even numbers

```
#include <stdio.h>

void main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i <= 2*n; i += 2) {
        printf("%d\n", i);
```

```
    }
}
```

`while` Loop

This kind of loop runs a block of code repeatedly until a specific condition is met. Here is the structure:

```
while (/* Condition to be met */) {
    // Set of statements to be executed repeatedly
}
```

This loop does not create any temparory vairable. Hence, the programmers must define the variable before using it in the condition, and also write the updating statement in the loop. Ignoring the update statement leads to wierd and unexpected problems, which will be mentioned here later. However, a `while` loop is not always (and preferrably not) used for such purposes.

Here is an example where we want the compiler to write the first 5 natural numbers

```
#include <stdio.h>

void main() {
    int i = 1;
    while (i != 5) {
        printf("%d", i);
        i++
    }
}
```

Here is an example where we want the compiler to print the sum of first 5 natural numbers

```
#include <stdio.h>

void main() {
    int sum = 0, i = 1;
```

```
    while (int i <= 5) {
        printf("%d", sum);
        i++;
    }
}
```

Here is an example where we want the compiler to print the first 10 negative integers

```
#include <stdio.h>

void main() {
    int i = -1;
    while (i >= -10) {
        printf("%d\n", i);
        i--;
    }
}
```

Here is an example where we want the compiler to print the first `n` even numbers

```
#include <stdio.h>

void main() {
    int n;
    scanf("%d", &n);

    int i = 0;
    while (i <= 2*n);
        printf("%d\n", i);
        i += 2;
    }
}
```

`do...while` Loop

There is another way to write `while` loops, and it is the `do...while` loop.

This loop is just like a while loop, but unlike the `while` loop which checks if the condition is true before executing the statements, the `do...while` loop executes the statements first and then checks if the condition is true to go to the next iteration.

```
do {
   // Set of statements to be executed
} while (/* Condition to be met */)
```

This loop executed the set of statements at least once, and are really great for menu driven programs, where the menu is opened irrespective of what the user wants for the first time.

Nested Loops

Just like nested `if...else` statements, nested loops means to create loops inside loops (maybe even a nested loop inside a loop and so on).

This is really useful when we want to keep repeating a subprocess over and over inside a process that is being repeated. For example, if we are searching an element in a matrix, then we go through cell by cell. But how? We go through every column in one row, and the same for each row, until the element is found.

Nested loops work great for finding or creating elements of multiple dimensions, but it can be used for other cases very well.

For example, consider a matrix with 4 rows and 3 columns, and we want the compiler to print which row and column it is reading. Here is how it could be done:

```
#include <stdio.h>

int main() {
   for (int i = 1; i <= 4; i++) { // Row
      for (int j = 1; j <= 3; j++) { // Column
         printf("Row %d; Column %d", i, j);
      }
   }
   return 0;
}
```

`continue` and `break`

These are two keywords that works only with loops. Using this statement anywhere else will throw an error. These statements are used in loops to jump or skip iterations.

The `continue` statement is used to jump to the next iteration by skipping all the statements after this in the current iteration. It basically skips the current iteration and goes to the next one.

The `break` statement is used to completely exit the loop, skipping all statements after this in the current iteration and all the upcoming iterations. It basically terminates the loop.

They go really well with decision making inside the loop.

```
#include <stdio.h>

void main() {
    int i = 1;
    while (true) {
        if (i == 3) { // Skip iteration 3
            continue;
        }

        if (i == 5) { // Terminate the loop at iteration 5
            break;
        }

        printf("%d", i);
    }

    i++;
}
```

Common Errors

When it comes to writing loops, there can be many logical and sntax errors that are commonly made by many programmers, from beginner to experts.

Infinite Looping

When conditions are defined in such a way that they are never met or never become false or the variable in the condition is never updated after execute the statements once.

Loops become endless and the rest of the code becomes unreachable. In most cases, the system becomes slow and irresponsive, and the program might crash. Here are examples which cause loops to never end:

```
while (true) {
    printf("Crashes! Condition is always true!");
}
```

```
int i = 5;
while (i == 5) {
    printf("No update! Crash!");
}
```

```
int i = 5;
while (i = 10) {
    printf("i is assigned in condition space and is always true! Crash!");
}
```

```
for (int i = 0; i < 4; i--) {
    print("i is always less than 4 and is always true! Crash!");
}
```

Of-by-one Error

Often we might give the condition wrong and the loop runs for one extra time, which throws an error in certain situations, such as traversing through arrays (lists) or matrices, or gives incorrect results.

```
for (int i = 0; i <= 10; i++);
```

Here, we are expecting 10 iterations, but the condition has been written for 11.

Wrong Usage of Jump Statements

Using `continue` and `break` wrong within the loop itself causes many wierd problems and errors.

If we use these statements in the middle of the block with no decisions, the rest of the code in the block becomes unreachable at any iteration. Although it does not throw an error, it creates unexpected situations and the program does not behave as intended.