Loops

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

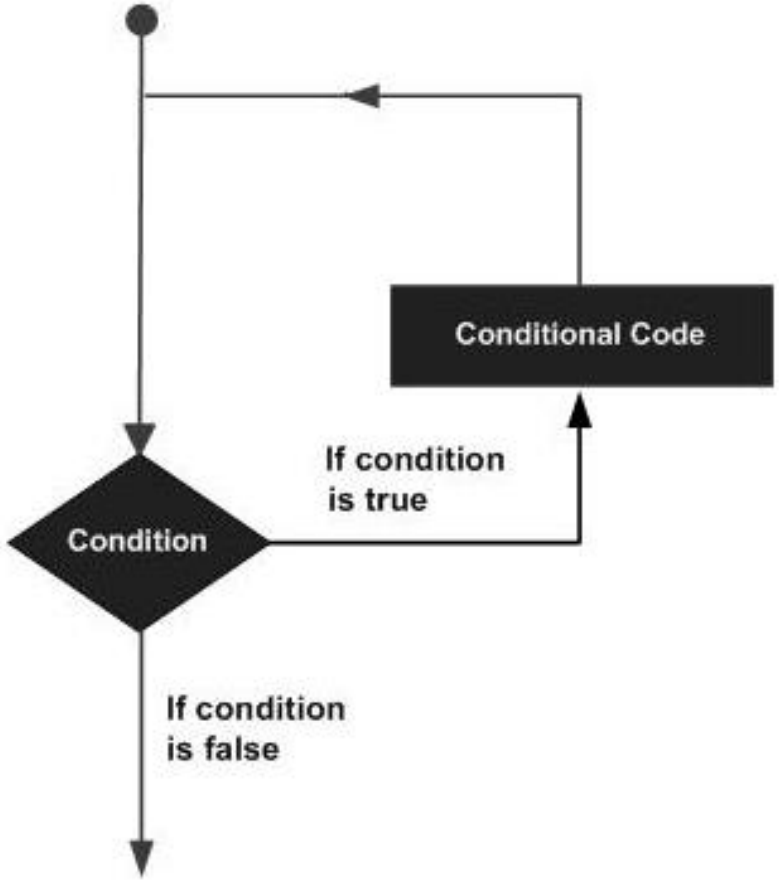
25 March 2021

Introduction to Loops

- When a block of code needs to be executed several number of times.
- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times.

Introduction to Loops

- General form of a loop statement in most of the programming languages –



Types of Loops

- while loop – Repeats a statement or group of statements while a given condition is true.
  It tests the condition before executing the loop body.
- for loop – Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- do..while loop - It is more like a while statement, except that it tests the condition at the end of the loop body.
- Nested loop - one or more loops inside any other while, for, or do..while loop.

while Loop

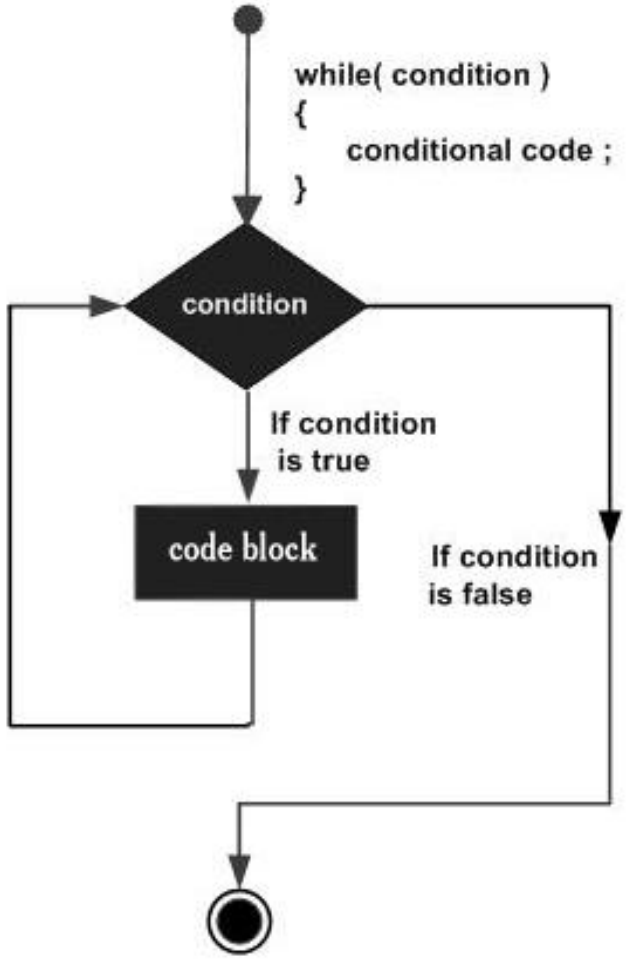
- Repeatedly executes a target statement as long as a given condition is true.
- Syntax:

```
while(condition) {
    statement(s);
}
```

- Here, **statement(s)** may be 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 condition is true.
- When the condition becomes false, the program control passes to the line immediately following the loop.

while Loop

- Flow diagram –



while Loop

- Note: A while loop might not execute at all.
- When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

while Loop

- Example:

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

for Loop

- A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
- Syntax:

```
for( init; condition; increment ) {
    statement(s);
}
```

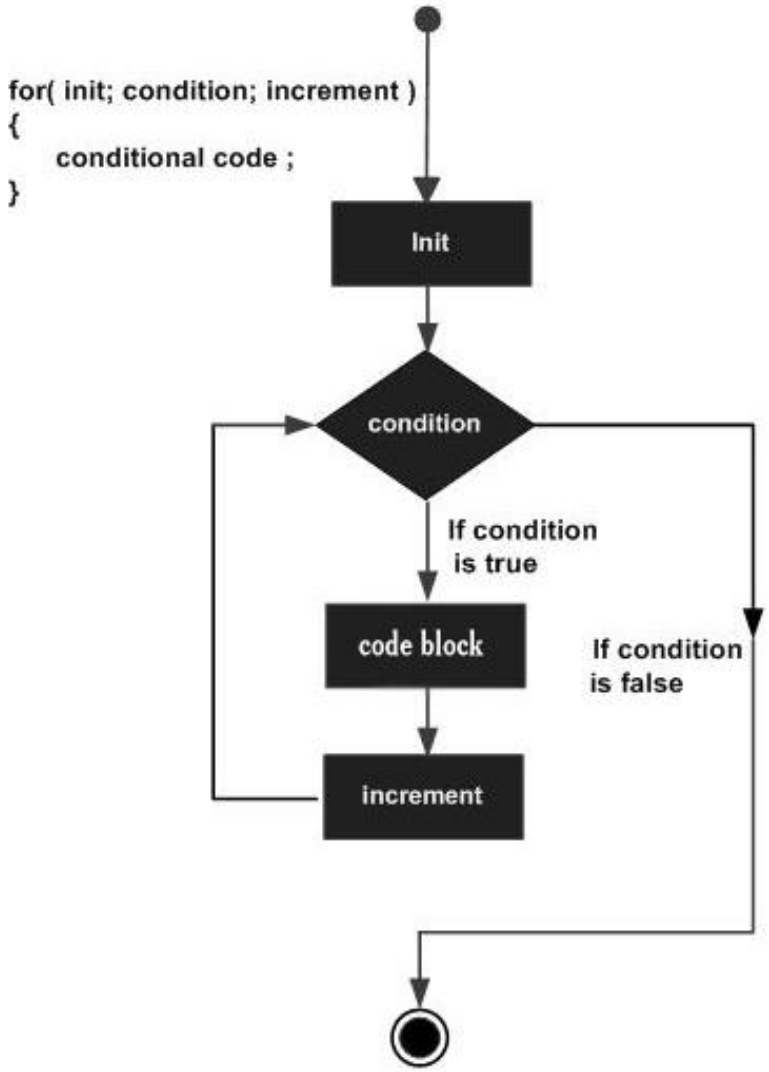
- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

for Loop

- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

for Loop

- Flow diagram -



for Loop

- Example -

```c
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

do...while Loop

- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.
- A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.
- Syntax:

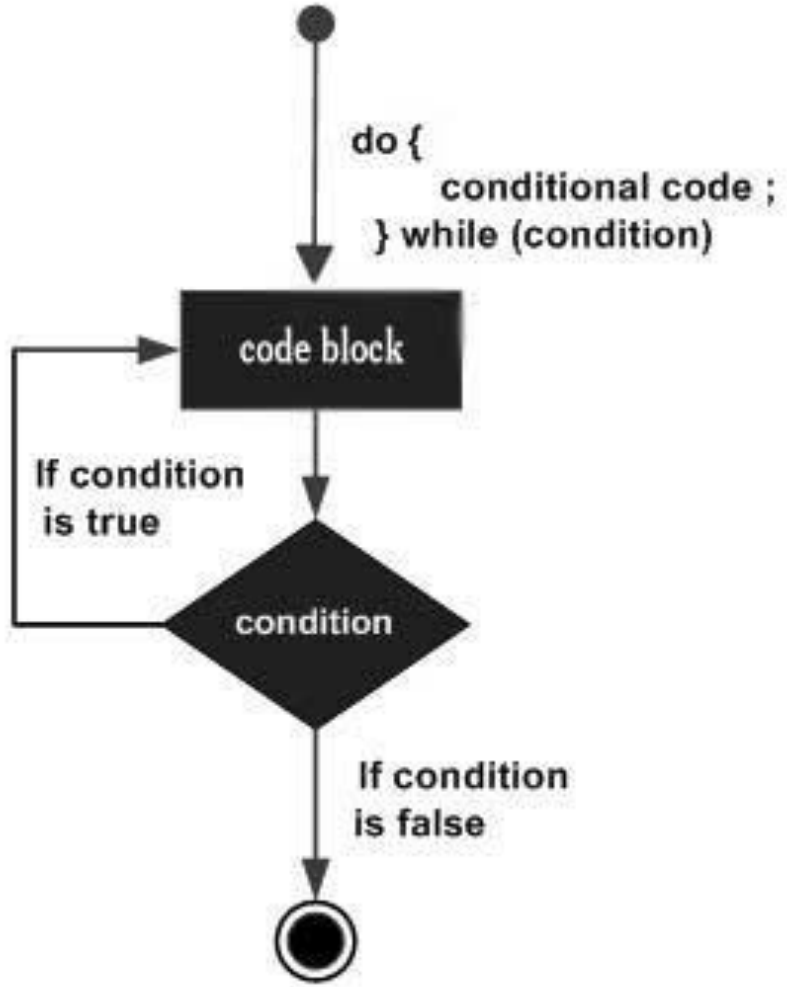
```
do {
    statement(s);
} while(condition);
```

do...while Loop

- Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.
- If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

do...while Loop

- Flow Diagram -



do...while Loop

- Example -

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {
      printf("value of a: %d\n", a);
      a = a + 1;
   }while( a < 20 );

   return 0;
}
```

Nested Loop

- C programming allows to use one loop inside another loop.
- Example:

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int i, j;

   for(i = 2; i<100; i++) {

      for(j = 2; j <= (i/j); j++)
      if(!(i%j)) break; // if factor found, not prime
      if(j > (i/j)) printf("%d is prime\n", i);
   }

   return 0;
}
```

Loop Control Statements

- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Types of loop control statements:
  - break
  - continue
  - goto

break statement

- break - Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
- Example:

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {

      printf("value of a: %d\n", a);
      a++;

      if( a > 15) {
         /* terminate the loop using break statement */
         break;
      }
   }

   return 0;
}
```

continue statement

- continue - Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- Example:

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         continue;
      }

      printf("value of a: %d\n", a);
      a++;

   } while( a < 20 );

   return 0;
}
```

goto statement

- goto - Transfers control to the labeled statement.
- Example:

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   LOOP:do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         goto LOOP;
      }

      printf("value of a: %d\n", a);
      a++;

   }while( a < 20 );

   return 0;
}
```

Infinite Loop

- A loop becomes an infinite loop if a condition never becomes false.
- The **for** loop is traditionally used for this purpose.
- Since none of the three expressions that form the 'for' loop are required, you can make an endless loop by leaving the conditional expression empty.
- Example:

```c
#include <stdio.h>

int main () {

   for( ; ; ) {
      printf("This loop will run forever.\n");
   }

   return 0;
}
```

Thank you!