For Loop in C

Most programming languages including C support the **for** keyword for constructing a loop. In C, the other loop-related keywords are **while** and **do-while**. Unlike the other two types, the **for** loop is called an **automatic loop**, and is usually the first choice of the programmers.

The **for loop** is an entry-controlled loop that executes the statements till the given condition. All the elements (initialization, test condition, and increment) are placed together to form a **for loop** inside the parenthesis with the **for** keyword.

Syntax of for Loop

The syntax of the **for** loop in C programming language is –

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

Control Flow of a For Loop

Here is how the control flows in a "for" loop –

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.

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 control jumps to the next statement just after the "for" loop.

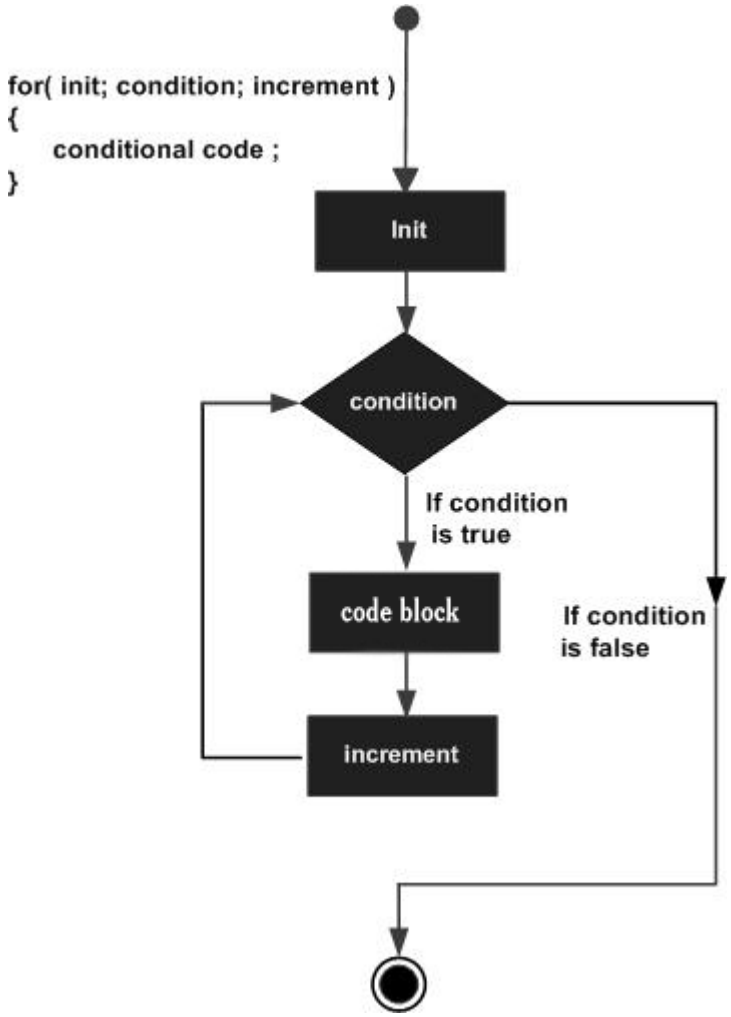
After the body of the "for" loop executes, the control flow jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

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 the condition). After the condition becomes false, the "for" loop terminates.

Explore our **latest online courses** and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Flowchart of for Loop

The following flowchart represents how the **for** loop works –



Developers prefer to use **for** loops when they know in advance how many number of iterations are to be performed. It can be thought of as a shorthand for **while** and **do-while** loops that increment and test a loop variable.

The **for** loop may be employed with different variations. Let us understand how the **for** loop works in different situations.

Example: Basic for Loop

This is the most basic form of the **for** loop. Note that all the three clauses inside the parenthesis (in front of the **for** keyword) are optional.

</> Open Compiler

```
#include <stdio.h>

int main(){
   int a;
```

```
   // for loop execution
   for(a = 1; a <= 5; a++){
      printf("a: %d\n", a);
   }

   return 0;
}
```

Output

Run the code and check its output –

```
a: 1
a: 2
a: 3
a: 4
a: 5
```

Initializing for Loop Counter Before Loop Statement

The initialization step can be placed above the header of the **for** loop. In that case, the **init** part must be left empty by putting a semicolon.

Example

</> Open Compiler

```
#include <stdio.h>

int main(){
   int a = 1;

   // for loop execution
   for( ; a <= 5; a++){
      printf("a: %d\n", a);
   }
   return 0;
}
```

Output

You still get the same output −

```
a: 1
a: 2
a: 3
a: 4
a: 5
```

Updating Loop Counter Inside for Loop Body

You can also put an empty statement in place of the increment clause. However, you need to put the increment statement inside the body of the loop, otherwise it becomes an infinite loop.

Example

</> Open Compiler

```
#include <stdio.h>

int main(){

   int a;

   // for loop execution
   for(a = 1; a <= 5; ){
      printf("a: %d\n", a);
      a++;
   }
   return 0;
}
```

Output

Here too, you will get the same output as in the previous example −

```
a: 1
a: 2
```

```
a: 3
a: 4
a: 5
```

Using Test Condition Inside for Loop Body

You can also omit the second clause of the test condition in the parenthesis. In that case, you will need to terminate the loop with a **break** statement, otherwise the loop runs infinitely.

Example

</> Open Compiler

```
#include <stdio.h>

int main(){
   int a;

   // for loop execution
   for(a = 1; ; a++){
      printf("a: %d\n", a);
      if(a == 5)
      break;
   }
   return 0;
}
```

Output

On executing this code, you will get the following output –

```
a: 1
a: 2
a: 3
a: 4
a: 5
```

Using for Loops with Multiple Counters

There may be initialization of more than one variables and/or multiple increment statements in a **for** statement. However, there can be only one test condition.

Example

```
#include <stdio.h>

int main(){
   int a, b;

   // for loop execution
   for(a = 1, b = 1; a <= 5; a++, b++){
      printf("a: %d b: %d a*b: %d\n", a, b, a*b);
   }

   return 0;
}
```

Output

When you run this code, it will produce the following output –

```
a: 1 b: 1 a*b: 1
a: 2 b: 2 a*b: 4
a: 3 b: 3 a*b: 9
a: 4 b: 4 a*b: 16
a: 5 b: 5 a*b: 25
```

Decrement in for Loop

You can also form a decrementing **for** loop. In this case, the initial value of the looping variable is more than its value in the test condition. The last clause in the for statement uses decrement operator.

Example

The following program prints the numbers 5 to 1, in decreasing order –

```c
#include <stdio.h>
int main(){
   int a;

   // for loop execution
   for(a = 5; a >= 1; a--){
      printf("a: %d\n", a);
   }

   return 0;
}
```

Output

Run the code and check its output –

```
a: 5
a: 4
a: 3
a: 2
a: 1
```

Traversing Arrays with for Loops

For loop is well suited for traversal of one element of an array at a time. Note that each element in the array has an incrementing index starting from "0".

Example

</> Open Compiler

```c
#include <stdio.h>

int main(){
   int i;
   int arr[] = {10, 20, 30, 40, 50};

   // for loop execution
   for(i = 0; i < 5; i++){
```

```
      printf("a[%d]: %d\n", i, arr[i]);
   }

   return 0;
}
```

Output

When you run this code, it will produce the following output –

```
a[0]: 10
a[1]: 20
a[2]: 30
a[3]: 40
a[4]: 50
```

Example: Sum of Array Elements Using for Loop

The following program computes the average of all the integers in a given array.

</> Open Compiler

```
#include <stdio.h>
int main(){
   int i;
   int arr[] = {10, 20, 30, 40, 50};
   int sum = 0;
   float avg;

   // for loop execution
   for(i=0; i<5; i++){
      sum += arr[i];
   }
   avg = (float)sum / 5;
   printf ("Average = %f", avg);

   return 0;
}
```

Output

Run the code and check its output −

```
Average = 30.000000
```

Example: Factorial Using for Loop

The following code uses a **for** loop to calculate the factorial value of a number. Note that the factorial of a number is the product of all integers between 1 and the given number. The factorial is mathematically represented by the following formula −

```
x! = 1 * 2 * . . . * x
```

Here is the code for computing the factorial −

</> Open Compiler

```
#include <stdio.h>

int main(){

   int i, x = 5;
   int fact = 1;

   // for loop execution
   for(i=1; i<= x; i++){
      fact *= i;
   }
   printf("%d != %d", x, fact);

   return 0;
}
```

Output

When you run this code, it will produce the following output −

```
5! = 120
```

The for loop is ideally suited when the number of repetitions is known. However, the looping behaviour can be controlled by the **break** and **continue** keywords inside the body of the **for** loop. Nested **for** loops are also routinely used in the processing of two dimensional arrays.