Infinite Loop in C

What is infinite loop?

An infinite loop is a looping construct that does not terminate the loop and executes the loop forever. It is also called an **indefinite** loop or an **endless** loop. It either produces a continuous output or no output.

When to use an infinite loop

An infinite loop is useful for those applications that accept the user input and generate the output continuously until the user exits from the application manually. In the following situations, this type of loop can be used:

* All the operating systems run in an infinite loop as it does not exist after performing some task. It comes out of an infinite loop only when the user manually shuts down the system.
* All the servers run in an infinite loop as the server responds to all the client requests. It comes out of an indefinite loop only when the administrator shuts down the server manually.
* All the games also run in an infinite loop. The game will accept the user requests until the user exits from the game.

We can create an infinite loop through various loop structures. The following are the loop structures through which we will define the infinite loop:

* for loop
* while loop
* do-while loop
* go to statement
* C macros

For loop

Let's see the **infinite 'for'** loop. The following is the definition for the **infinite** for loop:

1. **for**(; ;)
2. {
3. // body of the for loop.
4. }

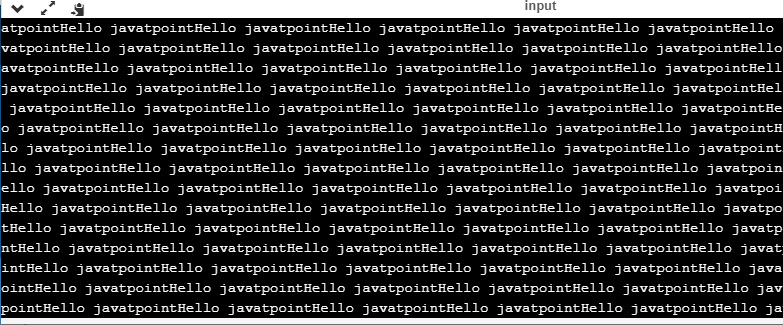
As we know that all the parts of the **'for' loop** are optional, and in the above for loop, we have not mentioned any condition; so, this loop will execute infinite times.

**Let's understand through an example.**

1. #include <stdio.h>
2. **int** main()
3. {
4. **for**(;;)
5. {
6. printf("Hello javatpoint");
7. }
8. **return** 0;
9. }

In the above code, we run the 'for' loop infinite times, so **"Hello javatpoint"** will be displayed infinitely.

**Output**



while loop

Now, we will see how to create an infinite loop using a while loop. The following is the definition for the infinite while loop:

1. **while**(1)
2. {
3. // body of the loop..
4. }

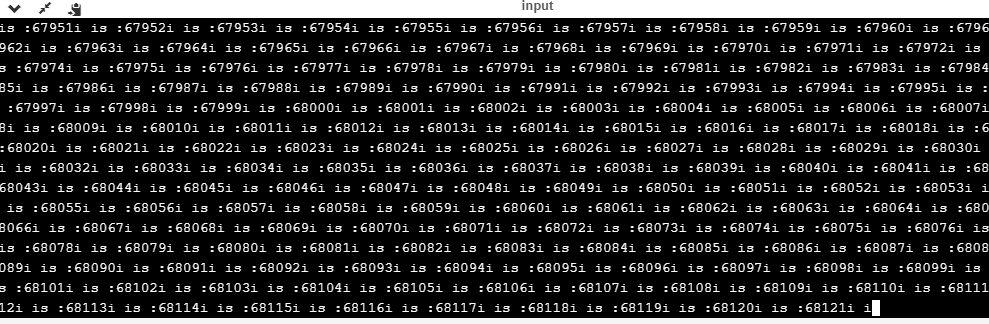
In the above while loop, we put '1' inside the loop condition. As we know that any non-zero integer represents the true condition while '0' represents the false condition.

**Let's look at a simple example.**

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** i=0;
5. **while**(1)
6. {
7. i++;
8. printf("i is :%d",i);
9. }
10. **return** 0;
11. }

In the above code, we have defined a while loop, which runs infinite times as it does not contain any condition. The value of 'i' will be updated an infinite number of times.

**Output**



**do..while loop**

The **do..while** loop can also be used to create the infinite loop. The following is the syntax to create the infinite **do..while** loop.

1. **do**
2. {
3. // body of the loop..
4. }**while**(1);

The above do..while loop represents the infinite condition as we provide the '1' value inside the loop condition. As we already know that non-zero integer represents the true condition, so this loop will run infinite times.

**goto statement**

We can also use the goto statement to define the infinite loop.

1. infinite\_loop;
2. // body statements.
3. **goto** infinite\_loop;

In the above code, the goto statement transfers the control to the infinite loop.

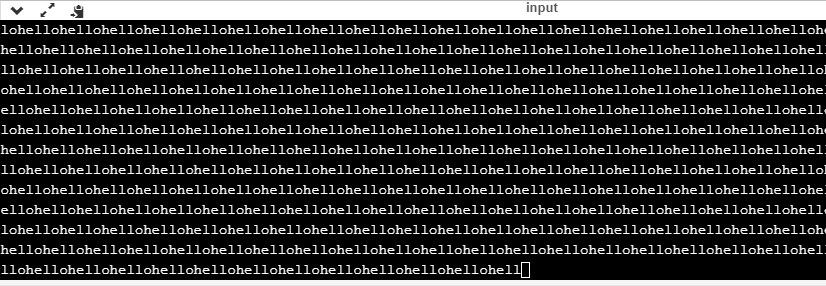
**Macros**

We can also create the infinite loop with the help of a macro constant. Let's understand through an example.

1. #include <stdio.h>
2. #define infinite for(;;)
3. **int** main()
4. {
6. infinite
7. {
8. printf("hello");
9. }
11. **return** 0;
12. }

In the above code, we have defined a macro named as 'infinite', and its value is 'for(;;)'. Whenever the word 'infinite' comes in a program then it will be replaced with a 'for(;;)'.

**Output**



Till now, we have seen various ways to define an infinite loop. However, we need some approach to come out of the infinite loop. In order to come out of the infinite loop, we can use the break statement.

**Let's understand through an example.**

1. #include <stdio.h>
2. **int** main()
3. {
4. **char** ch;
5. **while**(1)
6. {
7. ch=getchar();
8. **if**(ch=='n')
9. {
10. **break**;
11. }
12. printf("hello");
13. }
14. **return** 0;
15. }

In the above code, we have defined the while loop, which will execute an infinite number of times until we press the key 'n'. We have added the 'if' statement inside the while loop. The 'if' statement contains the break keyword, and the break keyword brings control out of the loop.

**Unintentional infinite loops**

Sometimes the situation arises where unintentional infinite loops occur due to the bug in the code. If we are the beginners, then it becomes very difficult to trace them. Below are some measures to trace an unintentional infinite loop:

* We should examine the semicolons carefully. Sometimes we put the semicolon at the wrong place, which leads to the infinite loop.

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** i=1;
5. **while**(i<=10);
6. {
7. printf("%d", i);
8. i++;
9. }
10. **return** 0;
11. }

In the above code, we put the semicolon after the condition of the while loop which leads to the infinite loop. Due to this semicolon, the internal body of the while loop will not execute.

* We should check the logical conditions carefully. Sometimes by mistake, we place the assignment operator (=) instead of a relational operator (= =).

1. #include <stdio.h>
2. **int** main()
3. {
4. **char** ch='n';
5. **while**(ch='y')
6. {
7. printf("hello");
8. }
9. **return** 0;
10. }

In the above code, we use the assignment operator (ch='y') which leads to the execution of loop infinite number of times.

* We use the wrong loop condition which causes the loop to be executed indefinitely.

1. #include <stdio.h>
2. **int** main()
3. {
4. **for**(**int** i=1;i>=1;i++)
5. {
6. printf("hello");
7. }
8. **return** 0;
9. }

The above code will execute the 'for loop' infinite number of times. As we put the condition (i>=1), which will always be true for every condition, it means that "hello" will be printed infinitely.

* We should be careful when we are using the **break** keyword in the nested loop because it will terminate the execution of the nearest loop, not the entire loop.

1. #include <stdio.h>
2. **int** main()
3. {
4. **while**(1)
5. {
6. **for**(**int** i=1;i<=10;i++)
7. {
8. **if**(i%2==0)
9. {
10. **break**;
11. }
12. }
13. }
14. **return** 0;
15. }

In the above code, the while loop will be executed an infinite number of times as we use the break keyword in an inner loop. This break keyword will bring the control out of the inner loop, not from the outer loop.

* We should be very careful when we are using the floating-point value inside the loop as we cannot underestimate the floating-point errors.

1. #include <stdio.h>
2. **int** main()
3. {
4. **float** x = 3.0;
5. **while** (x != 4.0) {
6. printf("x = %f\n", x);
7. x += 0.1;
8. }
9. **return** 0;
10. }

In the above code, the loop will run infinite times as the computer represents a floating-point value as a real value. The computer will represent the value of 4.0 as 3.999999 or 4.000001, so the condition (x !=4.0) will never be false. The solution to this problem is to write the condition as (k<=4.0).