

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

- 1) It provides code reusability.
- 2) Using loops, we do not need to write the same code again and again.
- 3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in **C language** that is given below:

1. do while
2. while
3. for

do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

The syntax of **do-while loop in c language** is given below:

1. **do**{
2. **//code to be executed**
3. **}while**(condition);

Flowchart and Example of do-while loop

while loop in C

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

The syntax of while loop in c language is given below:

```
while(condition){  
//code to be executed  
}
```

Flowchart and Example of while loop

for loop in C

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

The syntax of for loop in c language is given below:

1. **for**(initialization;condition;incr/decr){
2. //code to be executed
3. }

do while loop in C

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

do while loop syntax

The syntax of the C language do-while loop is given below:

```
do{  
//code to be executed
```

```
}while(condition);
```

Example 1

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
void main ()
```

```
{
```

```
    char c;
```

```
    int choice,dummy;
```

```
    do{
```

```
        printf("\n1. Print Hello\n2. Print Javatpoint\n3. Exit\n");
```

```
        scanf("%d",&choice);
```

```
        switch(choice)
```

```
        {
```

```
            case 1 :
```

```
                printf("Hello");
```

```
                break;
```

```
            case 2:
```

```
                printf("Javatpoint");
```

```
                break;
```

```
            case 3:
```

```
                exit(0);
```

```
            break;
```

```
            default:
```

```
                printf("please enter valid choice");
```

```
        }
```

```
        printf("do you want to enter more?");
```

```
        scanf("%d",&dummy);
```

```
        scanf("%c",&c);
```

```
    }while(c=='y');
```

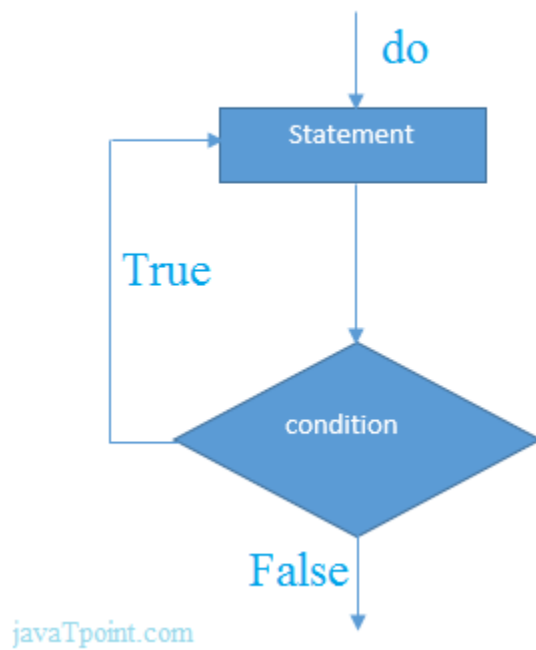
```
}
```

Output

```
1. Print Hello
2. Print Javatpoint
3. Exit
1
Hello
do you want to enter more?
Y

1. Print Hello
2. Print Javatpoint
3. Exit
2
Javatpoint
do you want to enter more?
n
```

Flowchart of do while loop



while loop in C

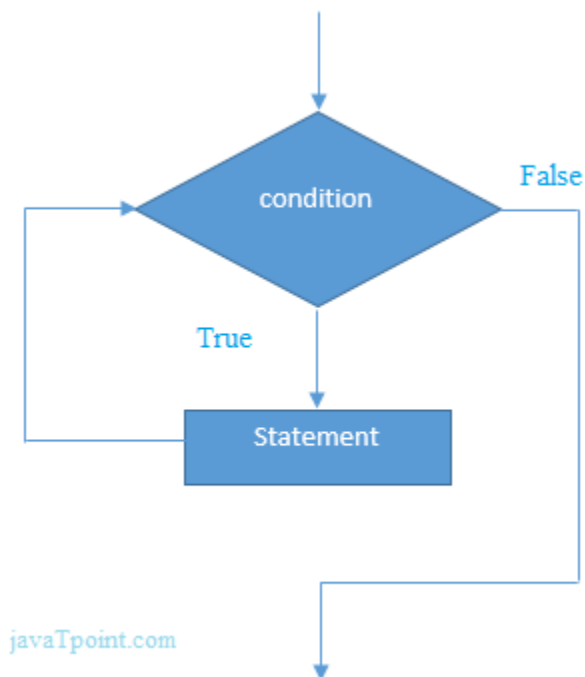
While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax of while loop in C language

The syntax of while loop in c language is given below:

```
while(condition){  
//code to be executed  
}
```

Flowchart of while loop in C



Example of the while loop in C language

Let's see the simple program of while loop that prints table of 1.

```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d \n",i);
i++;
}
return 0;
}
```

Output

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

Program to print table for the given number using while loop in C

```
#include<stdio.h>
int main(){
int i=1,number=0,b=9;
printf("Enter a number: ");
scanf("%d",&number);
while(i<=10){
printf("%d \n",(number*i));
i++;
}
return 0;
}
```

```
}
```

Output

```
Enter a number: 50
50
100
150
200
250
300
350
400
450
500
Enter a number: 100
100
200
300
400
500
600
700
800
900
1000
```

Properties of while loop

- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

Example 1

```
#include<stdio.h>
void main ()
{
    int j = 1;
    while(j+=2,j<=10)
    {
        printf("%d ",j);
    }
    printf("%d",j);
}
```

Output

```
3 5 7 9 11
```

Example 2

```
#include<stdio.h>
void main ()
{
    while()
    {
        printf("hello Javatpoint");
    }
}
```

Output

```
compile time error: while loop can't be empty
```

Example 3

```
#include<stdio.h>
void main ()
{
    int x = 10, y = 2;
    while(x+y-1)
```



```
{
    printf("%d %d",x--,y--);
}
}
```

Output

```
infinite loop
```

Infinitive while loop in C

If the expression passed in while loop results in any non-zero value then the loop will run the infinite number of times.

```
while(1){
//statement
}
```

for loop in C

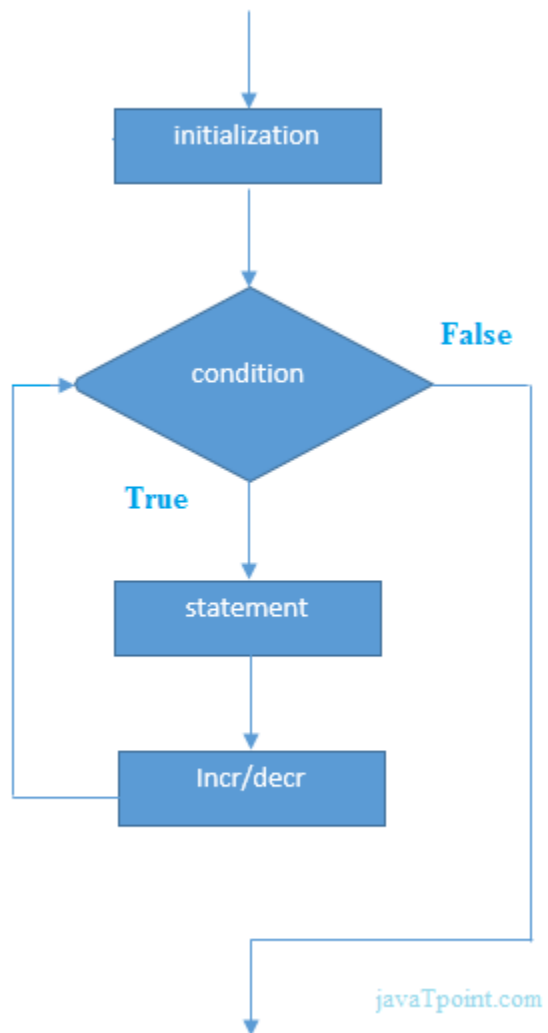
The **for loop in C language** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax of for loop in C

The syntax of for loop in c language is given below:

```
for(Expression 1; Expression 2; Expression 3){
//code to be executed
}
```

Flowchart of for loop in C



C for loop Examples

Let's see the simple program of for loop that prints table of 1.

```
#include<stdio.h>
int main(){
int i=0;
for(i=1;i<=10;i++){
```

```
printf("%d \n",i);  
}  
return 0;  
}
```

Output

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

C Program: Print table for the given number using C for loop

```
#include<stdio.h>  
int main(){  
int i=1,number=0;  
printf("Enter a number: ");  
scanf("%d",&number);  
for(i=1;i<=10;i++){  
printf("%d \n",(number*i));  
}  
return 0;  
}
```

Output

```
Enter a number: 2  
2  
4  
6  
8  
10  
12  
14  
16  
18  
20  
Enter a number: 1000
```

```
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
```

Properties of Expression 1

- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in Expression 1.
- Expression 1 is optional.
- In C, we can not declare the variables in Expression 1. However, It can be an exception in some compilers.

Example 1

```
#include <stdio.h>
int main()
{
    int a,b,c;
    for(a=0,b=12,c=23;a<2;a++)
    {
        printf("%d ",a+b+c);
    }
}
```

Output

```
35 36
```

Example 2

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

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

Output

```
1 2 3 4
```

Properties of Expression 2

- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- Expression 2 is optional.
- Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

Example 1

```
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<=4;i++)
    {
        printf("%d ",i);
    }
}
```

output

```
0 1 2 3 4
```

Example 2

```
#include <stdio.h>
int main()
{
    int i,j,k;
    for(i=0,j=0,k=0;i<4,k<8,j<10;i++)
    {
        printf("%d %d %d\n",i,j,k);
        j+=2;
        k+=3;
    }
}
```

Output

```
0 0 0
1 2 3
2 4 6
3 6 9
4 8 12
```

Example 3

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

Output

```
infinite loop
```

Properties of Expression 3

- Expression 3 is used to update the loop variable.
- We can update more than one variable at the same time.
- Expression 3 is optional.

Example 1

1. `#include<stdio.h>`
2. `void main ()`
3. `{`
4. `int i=0,j=2;`
5. `for(i = 0;i<5;i++,j=j+2)`
6. `{`
7. `printf("%d %d\n",i,j);`
8. `}`
9. `}`

Output

```
0 2
1 4
2 6
3 8
4 10
```

Loop body

The braces `{}` are used to define the scope of the loop. However, if the loop contains only one statement, then we don't need to use braces. A loop without a body is possible. The braces work as a block separator, i.e., the value variable declared inside for loop is valid only for that block and not outside. Consider the following example.

1. `#include<stdio.h>`
2. `void main ()`
3. `{`
4. `int i;`
5. `for(i=0;i<10;i++)`

```
6.  {
7.    int i = 20;
8.    printf("%d ",i);
9.  }
10.}
```

Output

```
20 20 20 20 20 20 20 20 20 20
```

Infinite for loop in C

To make a for loop infinite, we need not give any expression in the syntax. Instead of that, we need to provide two semicolons to validate the syntax of the for loop. This will work as an infinite for loop.

```
#include<stdio.h>
void main ()
{
    for(;;)
    {
        printf("welcome to CCSU");
    }
}
```

If you run this program, you will see above statement infinite times

Nested Loops in C

C supports nesting of loops in C. **Nesting of loops** is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define **'while'** loop inside a **'for'** loop.

Syntax of Nested loop

```
Outer_loop
{
    Inner_loop
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

Outer_loop and **Inner_loop** are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

Nested for loop

The nested for loop means any type of loop which is defined inside the 'for' loop.

```
for (initialization; condition; update)
{
    for(initialization; condition; update)
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

Example of nested for loop

```
#include <stdio.h>
int main()
{
    int n;// variable declaration
    printf("Enter the value of n :");
    // Displaying the n tables.
    for(int i=1;i<=n;i++) // outer loop
    {
        for(int j=1;j<=10;j++) // inner loop
```

```

{
    printf("%d\t",(i*j)); // printing the value.
}
printf("\n");
}

```

Explanation of the above code

- First, the 'i' variable is initialized to 1 and then program control passes to the $i \leq n$.
- The program control checks whether the condition ' $i \leq n$ ' is true or not.
- If the condition is true, then the program control passes to the inner loop.
- The inner loop will get executed until the condition is true.
- After the execution of the inner loop, the control moves back to the update of the outer loop, i.e., $i++$.
- After incrementing the value of the loop counter, the condition is checked again, i.e., $i \leq n$.
- If the condition is true, then the inner loop will be executed again.
- This process will continue until the condition of the outer loop is true.

Output:

```

input
Enter the value of n : 3
1      2      3      4      5      6      7      8      9      10
2      4      6      8      10     12     14     16     18     20
3      6      9      12     15     18     21     24     27     30

...Program finished with exit code 0
Press ENTER to exit console.

```

Nested while loop

The nested while loop means any type of loop which is defined inside the 'while' loop.

```
while(condition)
{
    while(condition)
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

Example of nested while loop

```
#include <stdio.h>
int main()
{
    int rows; // variable declaration
    int columns; // variable declaration
    int k=1; // variable initialization
    printf("Enter the number of rows :"); // input the number of rows.
    scanf("%d",&rows);
    printf("\nEnter the number of columns :"); // input the number of columns.
    scanf("%d",&columns);
    int a[rows][columns]; //2d array declaration
    int i=1;
    while(i<=rows) // outer loop
    {
        int j=1;
        while(j<=columns) // inner loop
        {
            printf("%d\t",k); // printing the value of k.
            k++; // increment counter
            j++;
        }
        i++;
        printf("\n");
    }
}
```

```
}
```

Explanation of the above code.

- We have created the 2d array, i.e., `int a[rows][columns]`.
- The program initializes the 'i' variable by 1.
- Now, control moves to the while loop, and this loop checks whether the condition is true, then the program control moves to the inner loop.
- After the execution of the inner loop, the control moves to the update of the outer loop, i.e., `i++`.
- After incrementing the value of 'i', the condition (`i<=rows`) is checked.
- If the condition is true, the control then again moves to the inner loop.
- This process continues until the condition of the outer loop is true.

Output:

```
Enter the number of rows : 5
Enter the number of columns :3
1      2      3
4      5      6
7      8      9
10     11     12
13     14     15

...Program finished with exit code 0
Press ENTER to exit console.
```

Nested do..while loop

The nested do..while loop means any type of loop which is defined inside the 'do..while' loop.

do

```
{
```

do

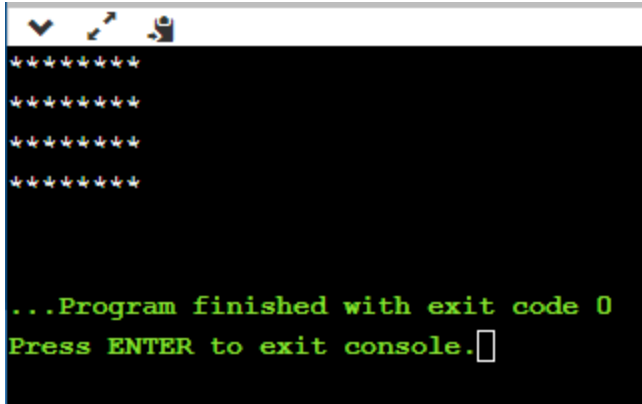
```
{
```

```
    // inner loop statements.
}while(condition);
// outer loop statements.
}while(condition);
```

Example of nested do..while loop.

```
#include <stdio.h>
int main()
{
    /*printing the pattern
    *****
    *****
    *****
    ***** */
    int i=1;
    do // outer loop
    {
        int j=1;
        do // inner loop
        {
            printf("*");
            j++;
        }while(j<=8);
        printf("\n");
        i++;
    }while(i<=4);
}
```

Output:



```
*****
*****
*****
*****

...Program finished with exit code 0
Press ENTER to exit console.█
```

Explanation of the above code.

- First, we initialize the outer loop counter variable, i.e., 'i' by 1.
- As we know that the do..while loop executes once without checking the condition, so the inner loop is executed without checking the condition in the outer loop.
- After the execution of the inner loop, the control moves to the update of the i++.
- When the loop counter value is incremented, the condition is checked. If the condition in the outer loop is true, then the inner loop is executed.
- This process will continue until the condition in the outer loop is true.

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.

```
#include <stdio.h>
int main()
{
    for(;;)
    {
        printf("Hello javatpoint");
    }
}
```

```
return 0;
}
```

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

Output



```
input
javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello
javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello
javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHell
javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHel
javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHe
o javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointH
lo javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpoint
llo javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpoi
ello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpoi
Hello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpo
tHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javatp
ntHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello javat
intHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello java
ointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello jav
pointHello javatpointHello javatpointHello javatpointHello javatpointHello javatpointHello ja
```

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:

```
while(1)
{
    // body of the loop..
}
```

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.

```
#include <stdio.h>
int main()
```



```

{
  int i=0;
  while(1)
  {
    i++;
    printf("i is :%d",i);
  }
return 0;
}

```

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

```

input
i is :67951i is :67952i is :67953i is :67954i is :67955i is :67956i is :67957i is :67958i is :67959i is :67960i is :67961i is :67962i is :67963i is :67964i is :67965i is :67966i is :67967i is :67968i is :67969i is :67970i is :67971i is :67972i is :67973i is :67974i is :67975i is :67976i is :67977i is :67978i is :67979i is :67980i is :67981i is :67982i is :67983i is :67984i is :67985i is :67986i is :67987i is :67988i is :67989i is :67990i is :67991i is :67992i is :67993i is :67994i is :67995i is :67996i is :67997i is :67998i is :67999i is :68000i is :68001i is :68002i is :68003i is :68004i is :68005i is :68006i is :68007i is :68008i is :68009i is :68010i is :68011i is :68012i is :68013i is :68014i is :68015i is :68016i is :68017i is :68018i is :68019i is :68020i is :68021i is :68022i is :68023i is :68024i is :68025i is :68026i is :68027i is :68028i is :68029i is :68030i is :68031i is :68032i is :68033i is :68034i is :68035i is :68036i is :68037i is :68038i is :68039i is :68040i is :68041i is :68042i is :68043i is :68044i is :68045i is :68046i is :68047i is :68048i is :68049i is :68050i is :68051i is :68052i is :68053i is :68054i is :68055i is :68056i is :68057i is :68058i is :68059i is :68060i is :68061i is :68062i is :68063i is :68064i is :68065i is :68066i is :68067i is :68068i is :68069i is :68070i is :68071i is :68072i is :68073i is :68074i is :68075i is :68076i is :68077i is :68078i is :68079i is :68080i is :68081i is :68082i is :68083i is :68084i is :68085i is :68086i is :68087i is :68088i is :68089i is :68090i is :68091i is :68092i is :68093i is :68094i is :68095i is :68096i is :68097i is :68098i is :68099i is :68100i is :68101i is :68102i is :68103i is :68104i is :68105i is :68106i is :68107i is :68108i is :68109i is :68110i is :68111i is :68112i is :68113i is :68114i is :68115i is :68116i is :68117i is :68118i is :68119i is :68120i is :68121i i

```

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.

```

do
{
  // body of the loop..
}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.

```
#include <stdio.h>
#define infinite for(;;)
int main()
{

    infinite
    {
        printf("hello");
    }

    return 0;
}
```

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(;;)'.

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.

```
#include <stdio.h>
int main()
{
int i=1;
while(i<=10);
{
printf("%d", i);
i++;
}
return 0;
}
```

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 (= =).

```
#include <stdio.h>
int main()
{
char ch='n';
while(ch='y')
{
printf("hello");
}
}
```

```
return 0;
}
```

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.

```
#include <stdio.h>
int main()
{
    for(int i=1;i>=1;i++)
    {
        printf("hello");
    }
return 0;
}
```

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.

```
#include <stdio.h>
int main()
{
    while(1)
    {
        for(int i=1;i<=10;i++)
        {
            if(i%2==0)
            {
                break;
            }
        }
    }
}
```

```
}  
    return 0;  
}
```

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.

```
#include <stdio.h>  
  
int main()  
{  
    float x = 3.0;  
    while (x != 4.0) {  
        printf("x = %f\n", x);  
        x += 0.1;  
    }  
    return 0;  
}
```

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).