**Loops - For - While - Do While - Jump Statements In C# Programming**

* Loops allow you to execute a single statement or a block of statements repetitively.
* The most common uses of loops include displaying a series of numbers and taking repetitive input.
* In software programming, a loop construct contains a condition that helps the compiler identify the number of times a specific block will be executed.
* If the condition is not specified, the loop continues infinitely and is termed as an infinite loop.
* The loop constructs are also referred to as iteration statements.

**C# supports four types of loop constructs such as:**

1. For loop
2. While loop
3. Do-while loop
4. Foreach loop

**The For Loop**

* The for statement is similar to the while statement in its function.
* The statements within the body of the loop are executed as long as the condition is true.
* Here too, the condition is checked before the statements are executed.

**There are 3 things in FOR LOOP:**

1. Initialization
2. Condition
3. Increment/decrement

**The following is the syntax of the for loop:**

for (**initialization**; **condition**; **increment**/**decrement**)

{

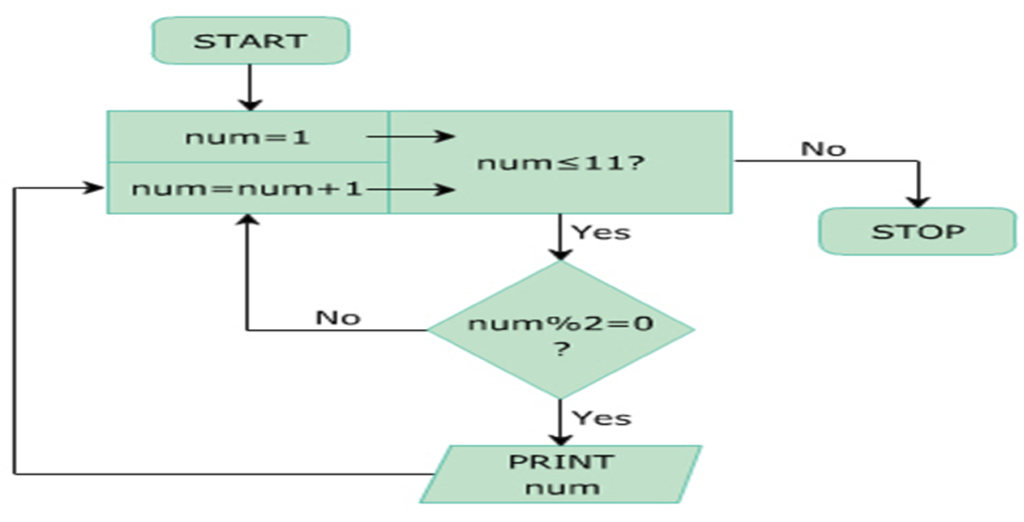
// one or more statements;

}

**where,**

* **initialization:** Initializes the variable(s) that will be used in the condition.
* **condition:** Comprises the condition that is tested before the statements in the loop are executed.
* **increment/decrement:** Comprises the statement that changes the value of the variable(s) to ensure that the condition specified in the condition section is reached. Typically, increment and decrement operators such as ++, -- and shortcut operators such as += or -= are used in this section. Note that there is no semicolon at the end of the increment/decrement expressions.

**The working of the for loop can be depicted using the following flowchart:**



**The following code displays even numbers from 1 to 10 using the for loop:**

int num;

Console.WriteLine(“Even Numbers”);

for (num = 1; num <= 11; num++) {

if ((num % 2) == 0)

{

Console.WriteLine(num);

}

}

**In the above code:**

* **num** is declared as an integer variable and it is initialized to value 1 in the for statement.
* The condition specified in the for statement is checked for value of num to be less than or equal to 11.
* If this condition is true, value of num is divided by 2 and the remainder is checked to see if it is 0.
* If this condition is true, the control is passed to the for statement again.
* Here, the value of num is incremented and the condition is checked again.
* When the value of num becomes 12, the condition of the for loop becomes false and the loop terminates.

**The while Loop**

* The while loop is used to execute a block of code repetitively as long as the condition of the loop remains true.
* The while loop consists of the while statement, which begins with the while keyword followed by a boolean condition.
* If the condition evaluates to true, the block of statements after the while statement is executed.
* After each iteration, the control is transferred back to the while statement and the condition is checked again for another round of execution.
* When the condition is evaluated to false, the block of statements following the while statement is ignored and the statement appearing after the block is executed by the compiler.

**The following is the syntax of the while loop:**

while (condition)

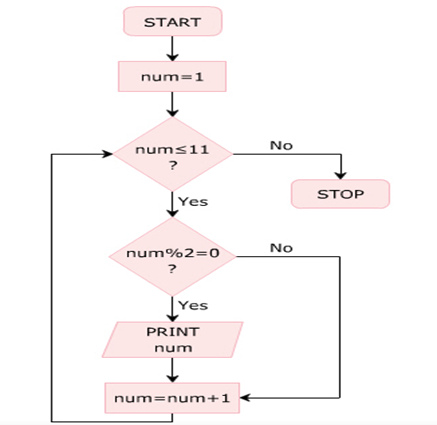
{

// one or more statements;

}

﻿

**The following figure depicts an example of a while loop using a flowchart:**



**The following code displays even numbers from 1 to 10 using the while loop:**

public int num = 1;

Console.WriteLine(“Even Numbers”);

while (num <= 11)

{

if ((num % 2) == 0)

{

Console.WriteLine(num);

}

num = num + 1;

}

﻿

**In the Above code:**

* num is declared as an integer variable and initialized to value 1.
* The condition in the while loop is checked, which specifies that the value of num variable should be less than or equal to 11.
* If this condition is true, the value of the **num** variable is divided by 2 and the remainder is checked to see if it is 0.
* If the remainder is 0, the value of the variable **num** is displayed in the console window and the variable **num** is incremented by 1.
* Then, the program control is passed to the while statement to check the condition again.
* When the value of **num** becomes 12, the while loop terminates as the loop condition becomes false.

**Output**

**Even Numbers**  
2  
4  
6  
8  
10

**Nested for Loops**

* The nested for loop consists of multiple for statements. When one for loop is enclosed inside another for loop, the loops are said to be nested.
* The for loop that encloses the other for loop is referred to as the outer for loop whereas the enclosed for loop is referred to as the inner for loop.
* The outer for loop determines the number of times the inner for loop will be invoked.

**Nested while Loop**

* A while loop can be created within another while loop to create a nested while loop structure.
* The following code demonstrates the use of nested while loops to create a geometric pattern:

using System;

class Pattern

{

static void Main(string[] args)

{

int i = 0;

int j;

while (i <= 5)

{

j = 0;

while (j <= i)

{

Console.Write("\*");

j++;

}

Console.WriteLine();

i++;

}

}

}

﻿

**The do-while Loop**

* The do-while loop is similar to the while loop; however, it is always executed at least once without the condition being checked.
* The loop starts with the do keyword and is followed  
  by a block of executable statThe while statement along with the condition appears at the end of this block.ements.
* The while statement along with the condition appears at the end of this block.
* The statements in the do-while loop are executed as long as the specified condition remains true.
* When the condition evaluates to false, the block of statements after the do keyword are ignored and the immediate statement after the while statement is executed.

**The following is the syntax of the do-while loop:**

do

{

// one or more statements;

} while (condition);

﻿

**The following code displays even numbers from 1 to 10 using the do-while loop:**

int num = 1;

Console.WriteLine(“EvenNumbers”);   
 do

{

if ((num % 2) == 0)   
 {

Console.WriteLine(num);

}

num = num + 1;

} while (num <= 11);

In the above code:

* num is declared as an integer variable and is initialized to value 1.
* In the do block, without checking any condition, the value of num is first divided by 2 and the remainder is checked to see if it is 0.
* If the remainder is 0, the value of num is displayed and it is then incremented by 1.
* Then, the condition in the while statement is checked to see if the value of num is less than or equal to 11.
* If this condition is true, the do-while loop executes again.
* When the value of num becomes 12, the do-while loop terminates.

**Output**

**Even Numbers**  
2  
4  
6  
8  
10

**Foreach Loop**

* The **Foreach** loop is specifically used with arrays or collections.
* The **Foreach** loop is used to retrieve the elements from an array or collection.
* The **Foreach** loop automatically terminates when all the elements of an array or collection is retrieved.
* The **Foreach** Loop sometimes called **Array's Loop**

**Syntax of Foreach Loop**

foreach (<datatype><identifier> in <Collection>)

{

// one or more statements;

}

﻿

**where,**

* **datatype**: Specifies the data type of the elements in the list.
* **identifier**: Is an appropriate name for the collection of elements.
* **list**: Specifies the name of the list.

**The following figure displays the employee names using the foreach loop:**

string[] employeeNames = { “Amit”, “Ali”, “Ishan”, “Amir” };

Console.WriteLine(“Employee Names”);

foreach (string names in employeeNames)

{

Console.WriteLine(“{0} “, names);

}

﻿

**In the above code:**

* The list of employee names is declared in an array of string variables called **employeeNames**.
* In the foreach statement, the data type is declared as string and the identifier is specified as names.
* This variable refers to all values from the **employeeNames** array.
* The foreach loop displays names of the employees in the order in which they are stored.

**Output**

**Employee Names**  
Amit  
Ali  
Ishan  
Amir

**Jump Statements in C#**

* Jump statements are used to transfer control from one point in a program to another.
* There will be situations where you need to exit out of a loop prematurely and continue with the program.
* In such cases, jump statements are used. Jump statement unconditionally transfer control of a program to a different location.
* The location to which a jump statement transfers control is called the target of the jump statement.

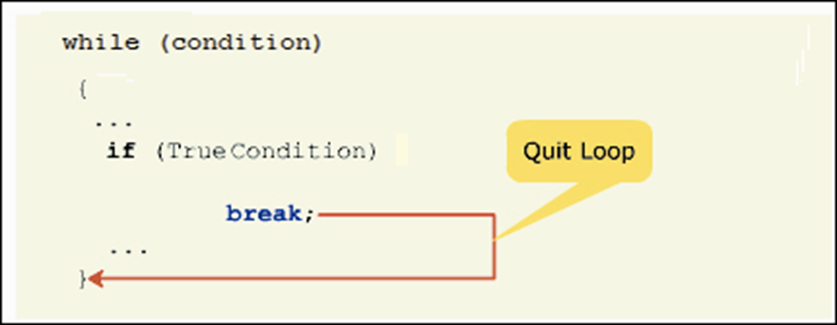
**C# supports four types of jump statements. These are as follows:**

1. break
2. continue
3. goto
4. return

**Break Statement**

* The break statement is used in the selection and loop constructs.
* It is most widely used in the switch…case construct and in the for and while loops.
* The break statement is denoted by the break keyword. In the switch…case construct, it is used to terminate the execution of the construct.
* In loops, it is used to exit the loop without testing the loop condition.
* In this case, the control passes to the next statement following the loop.

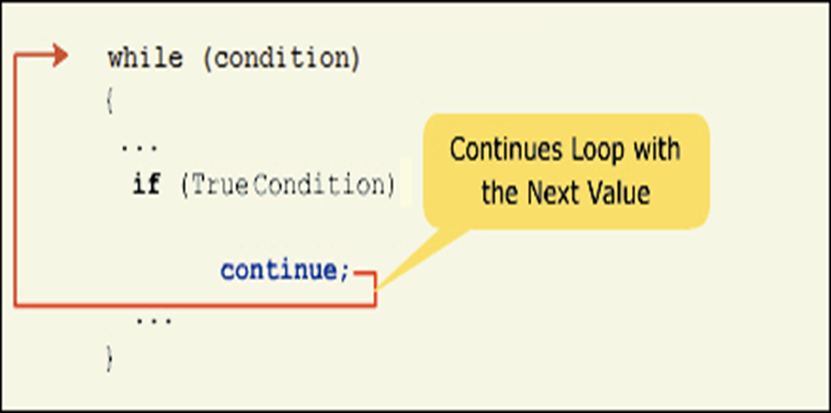
**The following figure depicts the break statement:**



**Continue Statement**

* The continue statement is most widely used in the loop constructs and is denoted by the continue keyword.
* The continue statement is used to end the current iteration of the loop and transfer the program control back to the beginning of the loop.
* The statements of the loop following the continue statement are ignored in the current iteration.

**The following figure displays the working of the continue statement:**



**The following code displays the even numbers in the range of 1 to 10 using the for loop and the continue statement:**

Console.WriteLine(“Even numbers in the range of 1-10”);

for (int i=1; i<=10; i++)

{

if (i % 2 != 0)

{

continue;

}

Console.Write(i + “ “);

}

**In the above code:**

* i is declared as an integer and is initialized to value 1 in the for loop definition.
* In the body of the loop, the value of i is divided by 2 and the remainder is checked to see if it is equal to 0.
* If the remainder is zero, the value of i is displayed as the value is an even number.
* If the remainder is not equal to 0, the continue statement is executed and the program control is transferred to the beginning of the for loop.

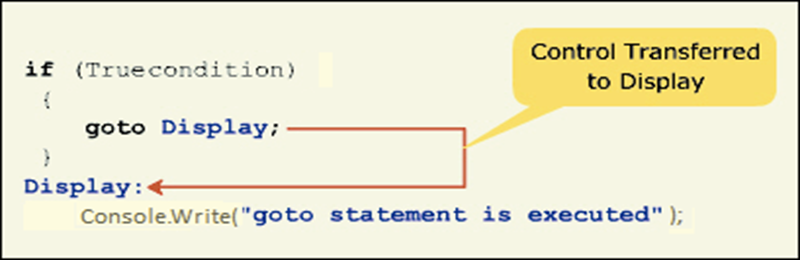
**Output**

Even numbers in the range of 1-10.  
2 4 6 8 10

**GoTo Statement**

* The goto statement allows you to directly execute a labeled statement or a labeled block of statements.
* A labeled block or a labeled statement starts with a label. A label is an identifier ending with a colon.
* A single labeled block can be referred by more than one goto statements.
* The goto statement is denoted by the goto keyword

**The following code displays the goto statement:**



**The following figure displays the output “Hello World” five times using the goto statement:**

int i = 0;

display:

Console.WriteLine(“Hello World”);

i++;

if (i < 5)

{

goto display;

}

**In the above code:**

* i is declared as an integer and is initialized to value 0.
* The program control transfers to the display label and the message “Hello World” is displayed.
* Then, the value of i is incremented by 1 and is checked to see if it is less than 5.
* If this condition evaluates to true, the goto statement is executed and the program control is transferred to the display label. If the condition evaluates to false, the program ends.

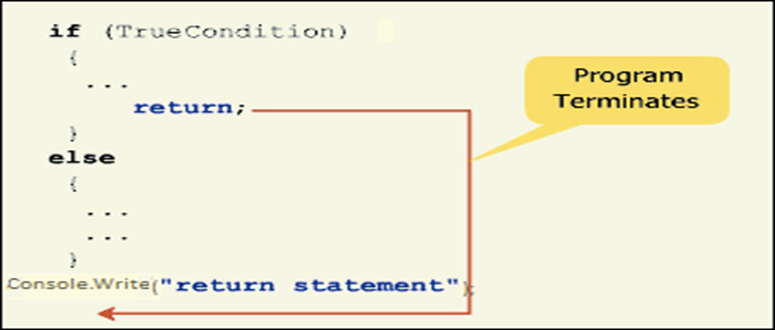
**Output**

Hello World  
Hello World  
Hello World  
Hello World  
Hello World

**The return Statement**

* The return statement is used to return a value of an expression or is used to transfer the control to the method from which the currently executing method was invoked.
* The return statement is denoted by the return keyword. The return statement must be the last statement in the method block.

**The following figure displays the working of the return statement:**



**The following code displays the cube of a number using the return statement:**

static void Main(string[] args)

{

int num = 23;

Console.WriteLine(“Cube of {0} = {1}”,num,Cube(num));

}

static int Cube(int n)

{

return (n \* n \* n);

}

﻿

In the above code:

* The variable num is declared as an integer and is initialized to value 23.
* The Cube() method is invoked by the Console.WriteLine() method.
* At this point, the program control passes to the Cube()method, which returns the cube of the specified value. The return statement returns the calculated cube value back to the Console.WriteLine() method, which displays the calculated cube of 23.

**Output**

Cube of 23 = 12167