1.3 Decision making and branching
1.4 Decision making and looping

1.3 Decision making and branching

A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

The conditional statements of C#:

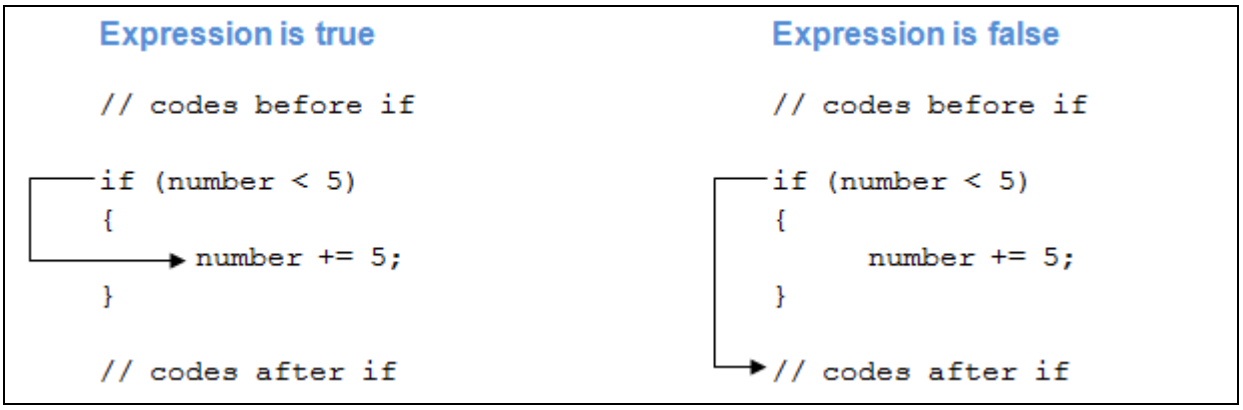
- Simple if
- if-else
- if-else-if
- Nested if
- Switch

Simple if
C# if statement will execute a block of code if the given condition is true.

syntax

```
if (boolean-expression)
{
        // statements executed if boolean-expression is true
}
```

- The boolean-expression will return either true or false.
- If the boolean-expression returns true, the statements inside the body of if (inside {...}) will be executed.
- If the boolean-expression returns false, the statements inside the body of if will be ignored.



```
using System;

namespace Conditional
{
	class IfStatement
	{
		public static void Main(string[] args)
		{
			int number = 2;
			if (number < 5)
			{
				Console.WriteLine("{0} is less than 5", number);
			}

			Console.WriteLine("This statement is always executed.");
		}
	}
}
```

Output

```
2 is less than 5
```

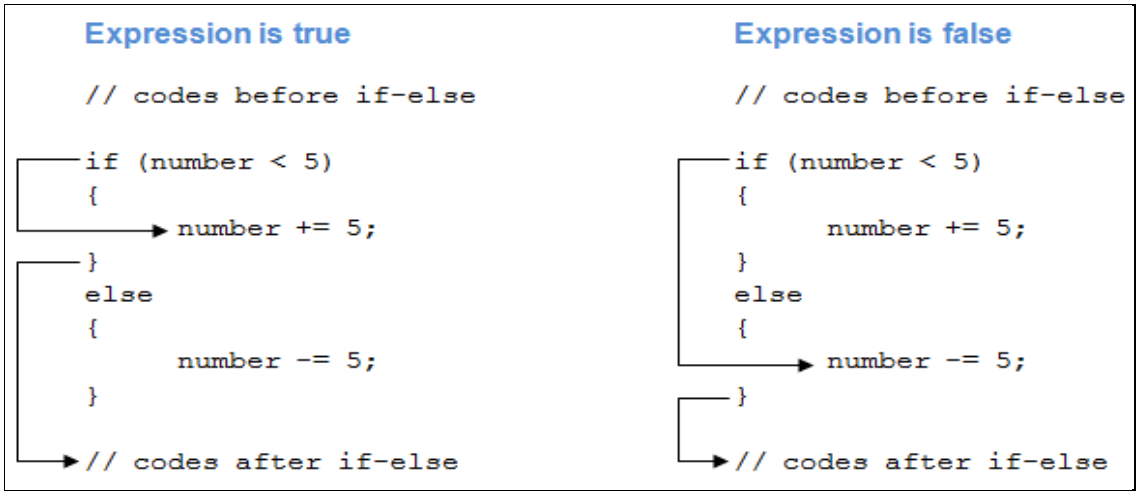
.

if...else Statement

The if statement in C# may have an optional else statement. The block of code inside the else statement will be executed if the expression is evaluated to false.

Syntax

```
if (boolean-expression)
{
	// statements executed if boolean-expression is true
}
else
{
	// statements executed if boolean-expression is false
}
```



using System;

```
namespace Conditional
{
        class IfElseStatement
        {
                public static void Main(string[] args)
                {
                        int number = 12;

                if (number < 5)
                {
                        Console.WriteLine("{0} is less than 5", number);
                }
                else
                {
                         Console.WriteLine("{0} is greater than or equal to 5", number);
                }

                         Console.WriteLine("This statement is always executed.");
                }
        }
}
```

Output

```
12 is greater than or equal to 5
This statement is always executed.
```

if...else if Statement

When we have only one condition to test, if and if- else statement works fine. But what if we have a multiple condition to test and execute one of the many block of code.

Syntax

```
if (boolean-expression-1)
{
        // statements executed if boolean-expression-1 is true
}
else if (boolean-expression-2)
{
        // statements executed if boolean-expression-2 is true
}
else if (boolean-expression-3)
{
        // statements executed if boolean-expression-3 is true
}
.
.
.
else
{
        // statements executed if all above expressions are false
}
```

```
using System;

namespace Conditional
{
        class IfElseIfStatement
  {
             public static void Main(string[] args)
   {
         int number = 12;
      if (number < 5)
   {
      Console.WriteLine("{0} is less than 5", number);

   }
else if (number > 5)
  {
       Console.WriteLine("{0} is greater than 5", number);
```

```
  }
else
  {
Console.WriteLine("{0} is equal to 5");
     }
    }
  }
}
```

Output

12 is greater than 5

Nested if...else Statement

An if...else statement can exist within another if...else statement.
Such statements are called nested if...else statement.

Syntax:

```
if (boolean-expression)
{
        if (nested-expression-1)
        {
                // code to be executed
        }
        else
        {
        // code to be executed
        }
}
else
{
        if (nested-expression-2)
        {
                // code to be executed
        }
        else
        {
                // code to be executed
        }
}
```

```
namespace Conditional
{
class Nested
        {
        public static void Main(string[] args)
        {
                int first = 7, second = -23, third = 13;
                if (first > second)
                {
                        if (firstNumber > third)
                        {
                                Console.WriteLine("{0} is the largest", first);
                        }
                        else
                        {
                                Console.WriteLine("{0} is the largest", third);
                        }
                }
                else
                {
                        if (second > third)
                        {
                                Console.WriteLine("{0} is the largest", second);
                        }
                        else
                        {
                                Console.WriteLine("{0} is the largest", third);
                        }
                }
          }
     }
}
```

Output

```
13 is the largest
```

Switch statement

Switch statement can be used to replace the if...else if statement in C#.
The advantage of using switch over if...else if statement is the codes will look much cleaner and readable with switch.

Syntax

```
switch (variable/expression)
{
   case value1:
      // Statements executed if expression(or variable) = value1
      break;
   case value2:
      // Statements executed if expression(or variable) = value1
      break;
   ... ... ...
   ... ... ...
   default:
      // Statements executed if no case matches
}
```

- The switch statement evaluates the expression (or variable) and compare its value with the values (or expression) of each case (value1, value2, …). When it finds the matching value, the statements inside that case are executed.
- The switch expression is of integer type such as int, char, byte, or short, or of an enumeration type, or of string type.
- But, if none of the above cases matches the expression, the statements inside default block is executed.
- The default statement at the end of switch is similar to the else block in if else statement.
- However a problem with the switch statement is, when the matching value is found, it executes all statements after it until the end of switch block.
- To avoid this, we use break statement at the end of each case. The break statement stops the program from executing non-matching statements by terminating the execution of switch statement.

```
using System;

namespace Conditional
{
   class SwitchCase
   {
      public static void Main(string[] args)
      {
         char ch;
         Console.WriteLine("Enter an alphabet");
         ch = Convert.ToChar(Console.ReadLine());

         switch(Char.ToLower(ch))
         {
            case 'a':
               Console.WriteLine("Vowel");
               break;
            case 'e':
               Console.WriteLine("Vowel");
               break;
            case 'i':
               Console.WriteLine("Vowel");
               break;
            case 'o':
               Console.WriteLine("Vowel");
               break;
            case 'u':
               Console.WriteLine("Vowel");
               break;
            default:
               Console.WriteLine("Not a vowel");
               break;
         }
      }
   }
}
```

Output

```
Enter an alphabet
X
Not a vowel
```

1.4 Decision making and looping

Looping in a programming language is a way to execute a statement or a set of statements multiple times depending on the result of the condition to be evaluated to execute statements. The result condition should be true to execute statements within loops.

- while
- do-while
- for
- foreach

while

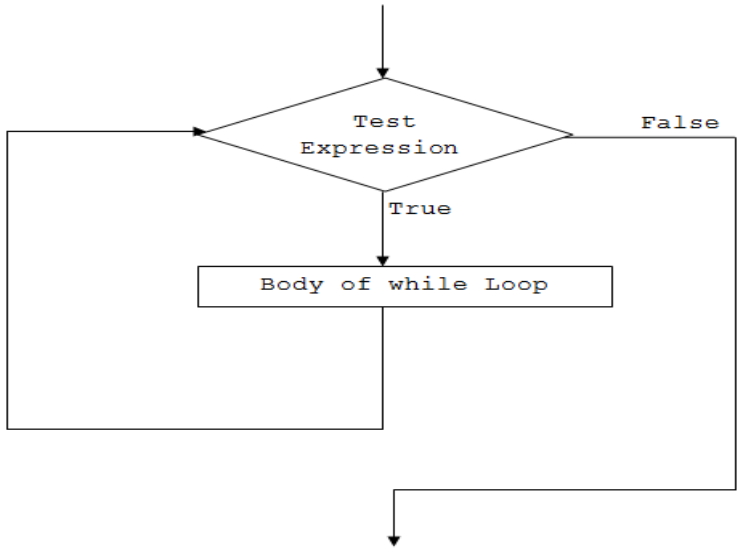
- The test condition is given in the beginning of the loop and all statements are executed till the given boolean condition satisfies when the condition becomes false, the control will be out from the while loop.
- The *while* keyword is used to create while loop in C#.

Syntax

```
while (test-expression)
{
        // body of while
}
```

- C# while loop consists of a *test-expression.*If the *test-expression* is evaluated to true,statements inside the while loop are executed, after execution, the *test-expression* is evaluated again.
- If the test-expression is evaluated to false, the while loop terminates .

Flowchart



```
using System;

namespace Loop
{
	class WhileLoop
	{
		public static void Main(string[] args)
		{
			int i=1;
			while (i<=5)
			{
				Console.WriteLine("C# For Loop: Iteration {0}", i);
				i++;
			}
		}
	}
}
```

Output

```
C# For Loop: Iteration 1
C# For Loop: Iteration 2
C# For Loop: Iteration 3
C# For Loop: Iteration 4
C# For Loop: Iteration 5
```

do-while

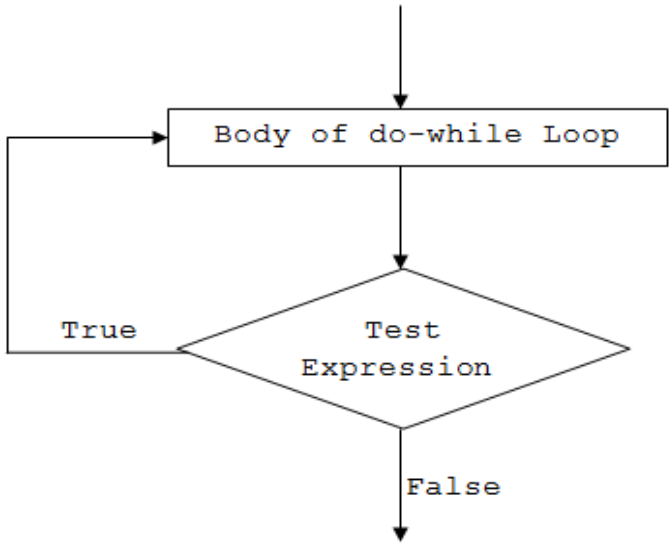
- The *do* and *while* keyword is used to create a do...while loop.
- It is similar to a while loop, however there is a major difference between them.
- In while loop, the condition is checked before the body is executed. It is the exact opposite in do...while loop, i.e. condition is checked after the body is executed.
- This is why, the body of do...while loop will execute at least once irrespective to the test-expression.

Syntax:

```
do
{
	// body of do while loop
} while (test-expression);
```

- The body of do...while loop is executed at first.Then the test-expression is evaluated.
- If the test-expression is true, the body of loop is executed.When the test-expression is false, do...while loop terminates.

Flowchart




```
using System;

namespace Loop
{
        class DoWhileLoop
        {
                public static void Main(string[] args)
                {
                        int i = 1, n = 5, product;

                        do
                        {
                                product = n * i;
                                Console.WriteLine("{0} * {1} = {2}", n, i, product);
                                i++;
                        } while (i <= 10);
                }
        }
}
```

Output

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
```

```
5 * 9 = 45
5 * 10 = 50
```

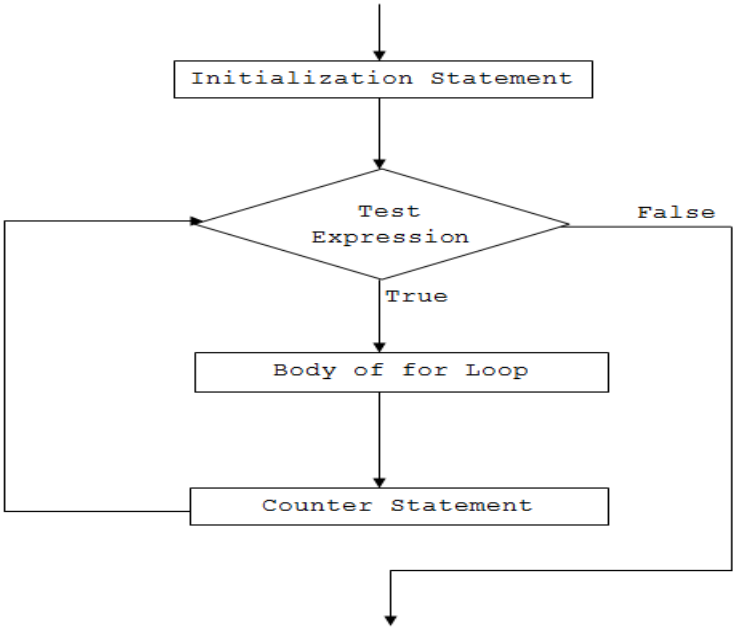
For loop

- The for keyword is used to create for loop in C#.
- C# for loop has three statements: **initialization, condition and iterator.**
- The initialization statement is executed at first and only once. Here, the variable is usually declared and initialized.
- Then, the condition is evaluated. The condition is a boolean expression, i.e. it returns either true or false.
- If the condition is evaluated to true.The statements inside the for loop are executed.
- Then, the iterator statement is executed which usually changes the value of the initialized variable.
- Again the condition is evaluated.
- The process continues until the condition is evaluated to false.
- If the condition is evaluated to false, the for loop terminates.

Syntax

```
for (initialization; condition; iterator)
{
        // body of for loop
}
```

for Loop Flowchart



```
using System;

namespace Loop
{
	class ForLoop
	{
		public static void Main(string[] args)
		{
			for (int i=1; i<=5; i++)
			{
				Console.WriteLine("C# For Loop: Iteration {0}", i);
			}
		}
	}
}
```

Output

```
C# For Loop: Iteration 1
C# For Loop: Iteration 2
C# For Loop: Iteration 3
C# For Loop: Iteration 4
C# For Loop: Iteration 5
```

Multiple expressions inside a for loop

We can also use multiple expressions inside a for loop. It means we can have more than one initialization and/or iterator statements within a for loop.

```
using System;

namespace Loop
{
	class ForLoop
	{
		public static void Main(string[] args)
		{
			for (int i=0, j=0; i+j<=5; i++, j++)
			{
				Console.WriteLine("i = {0} and j = {1}", i,j);
			}
		}
	}
}
```

Output

```
i = 0 and j = 0
i = 1 and j = 1
i = 2 and j = 2
```

foreach

- Here iterable-item can be an array or a class of collection.
- The *in* keyword used along with foreach loop is used to iterate over the iterable-item. The *in* keyword selects an item from the iterable-item on each iteration and store it in the variable element.
- On first iteration, the first item of iterable-item is stored in element. On second iteration, the second element is selected and so on.
- The number of times the foreach loop will execute is equal to the number of elements in the array or collection.

```
foreach (element in iterable-item)
{
   // body of foreach loop
}
```

```
using System;

namespace Loop
{
   class ForEachLoop
   {
      public static void Main(string[] args)
      {
         char[] myArray = {'H','e','l','l','o'};

         foreach(char ch in myArray)
         {
            Console.WriteLine(ch);
         }
      }
   }
}
```

Output

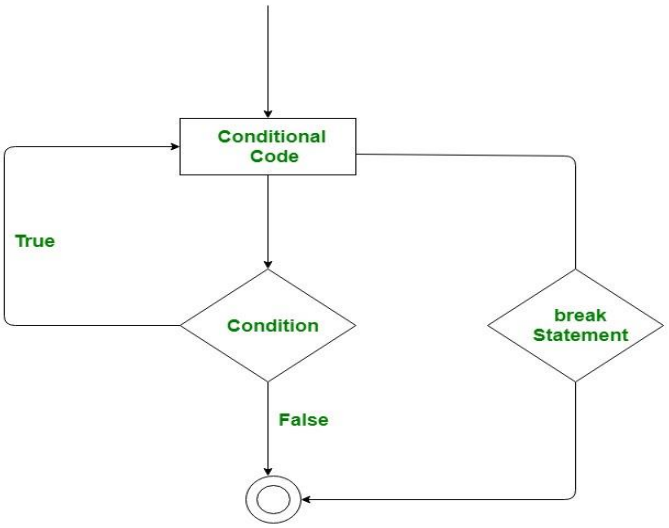
```
H
e
l
```

l
o

Jump statements
Jump statements are used to transfer control from one point to another point in the program due to some specified code while executing the program.

- break
- continue
- goto
- return

break statement
The break statement is used to terminate the loop or statement in which it present. After that, the control will pass to the statements that present after the break statement, if available. If the break statement present in the nested loop, then it terminates only those loops which contains break statement.
Flowchart:




```
using System;

class Program
 {

   static public void Main()
   {
```

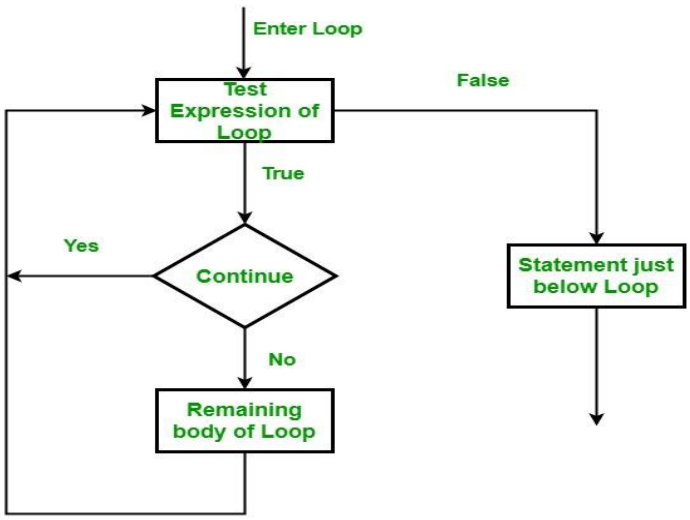
```
 for (int i = 1; i < 4; i++)
       {
          if (i == 3)
             break;

          Console.WriteLine("Hello World");
       }
    }
}
```

Output:

```
Hello World
Hello World
```

continue statement

This statement is used to skip over the execution part of the loop on a certain condition. After that, it transfers the control to the beginning of the loop. Basically, it skips its following statements and continues with the next iteration of the loop.




```
using System;
class Program
 {
    public static void Main()
   {
      for (int i = 1; i <= 10; i++)
{
        if (i == 4)
            continue;
```

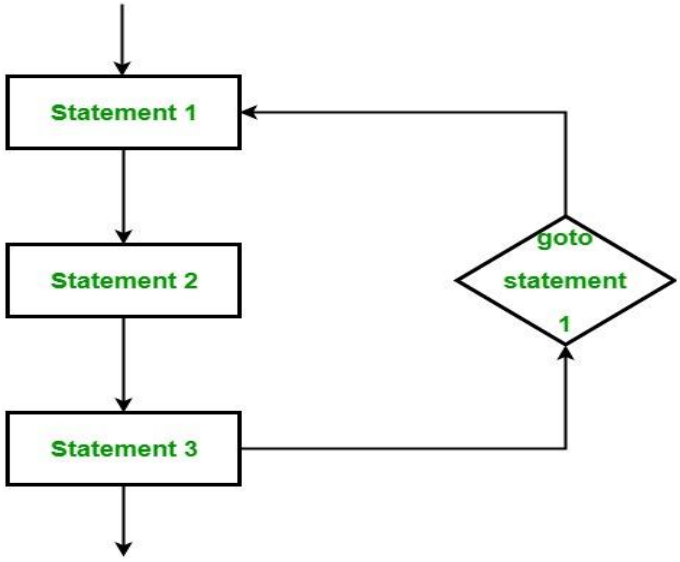
```
            Console.WriteLine(i);
        }
    }
}
```

Output:

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

goto statement

This statement is used to transfer control to the labeled statement in the program. The label is the valid identifier and placed just before the statement from where the control is transferred.




```
using System;

class Geeks {

    public static void Main()
```

```
    {
        int number = 20;
        switch (number) {

        case 5:
            Console.WriteLine("case 5");
            break;
        case 10:
            Console.WriteLine("case 10");
            break;
        case 20:
            Console.WriteLine("case 20");

            // goto statement transfer
            // the control to case 5
            goto case 5;

        default:
            Console.WriteLine("No match found");
            break;
        }
    }
}
```

Output:

```
case 20
case 5
```

return statement

This statement terminates the execution of the method and returns the control to the calling method. It returns an optional value. If the type of method is void, then the return statement can be excluded.

```
using System;
 class Geeks
{
    static int Addition(int a)
    {
  int add = a + a;
        return add;
    }
```

```
    static public void Main()
    {
        int number = 2;
         int result = Addition(number);
        Console.WriteLine("The addition is {0}", result);
    }
}
```

Output:

```
The addition is 4
```