Chapter 2

Basic C# Language

OPERATORS

OPERATORS

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- C# has rich set of built-in operators and provides the following type of operators:
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Misc Operators

Arithmetic Operators

- Following table shows all the arithmetic operators supported by C#. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

Arithmetic Operators

- Example

  The following example demonstrates all the arithmetic operators available in C#:

```
static void Main(string[] args)
        {
            int a = 21;
            int b = 10;
            int c;

            c = a + b;
            Console.WriteLine("Line 1 - Value of c is {0}", c);
            c = a - b;
            Console.WriteLine("Line 2 - Value of c is {0}", c);
            c = a * b;
            Console.WriteLine("Line 3 - Value of c is {0}", c);
            c = a / b;
            Console.WriteLine("Line 4 - Value of c is {0}", c);
            c = a % b;
            Console.WriteLine("Line 5 - Value of c is {0}", c);
            c = a++;
            Console.WriteLine("Line 6 - Value of c is {0}", c);
            c = a--;
            Console.WriteLine("Line 7 - Value of c is {0}", c);
            Console.ReadLine();
        }
```

OUTPUT

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 7 - Value of c is 22
```

Relational Operators

- Following table shows all the relational operators supported by C#. Assume variable **A** holds 10 and variable **B** holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Relational Operators

- Example

  The following example demonstrates all the relational operators available in C#:

```
static void Main(string[] args)
        {
            int a = 21;
            int b = 10;

            if (a == b)
            {
                Console.WriteLine("Line 1 - a is equal to b");
            }
            else
            {
                Console.WriteLine("Line 1 - a is not equal to b");
            }
            if (a < b)
            {
                Console.WriteLine("Line 2 - a is less than b");
            }
            else
            {
                Console.WriteLine("Line 2 - a is not less than b");
            }
            if (a > b)
            {
                Console.WriteLine("Line 3 - a is greater than b");
            }
            else
            {
                Console.WriteLine("Line 3 - a is not greater than b");
            }
            /* Lets change value of a and b */
            a = 5;
            b = 20;
            if (a <= b)
            {
                Console.WriteLine("Line 4 - a is either less than or equal to  b");
            }
            if (b >= a)
            {
                Console.WriteLine("Line 5-b is either greater than or equal to b");
            }
            Console.ReadLine();
        }
```

Logical Operators

- Following table shows all the logical operators supported by C#.
- Assume variable **A** holds Boolean value **true** and variable **B** holds Boolean value **false**, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Logical Operators

- Example

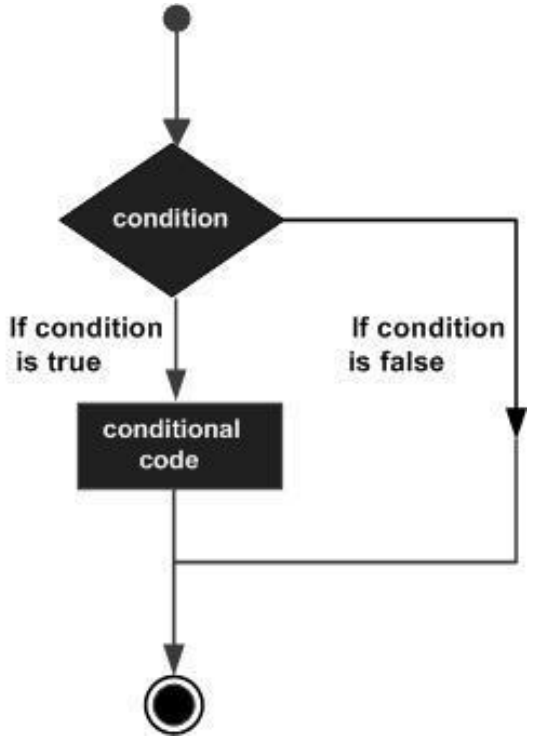
  The following example demonstrates all the logical operators available in C#:

```
static void Main(string[] args)
        {
            bool a = true;
            bool b = true;
            if (a && b)
            {
                Console.WriteLine("Line 1 - Condition is true");
            }
            if (a || b)
            {
                Console.WriteLine("Line 2 - Condition is true");
            }
            /* lets change the value of  a and b */
            a = false;
            b = true;
            if (a && b)
            {
                Console.WriteLine("Line 3 - Condition is true");
            }
            else
            {
                Console.WriteLine("Line 3 - Condition is not true");
            }
            if (!(a && b))
            {
                Console.WriteLine("Line 4 - Condition is true");
            }
            Console.ReadLine();
        }
```

DECISION MAKING

DECISION MAKING

- Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program



DECISION MAKING

- C# provides following types of decision making statements.

Statement	Description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s).

if Statement

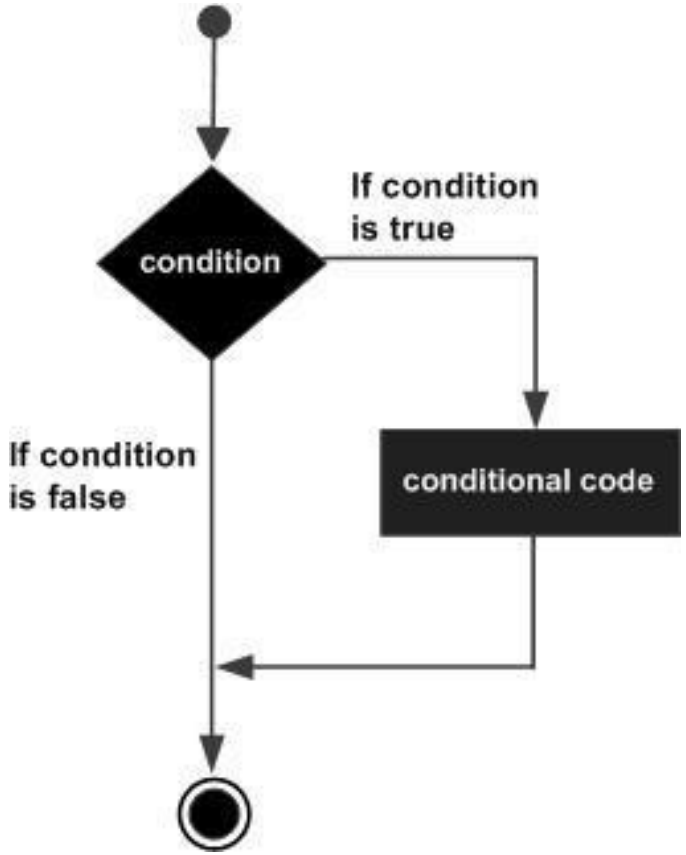
- An **if** statement consists of a boolean expression followed by one or more statements.
- **Syntax**
  The syntax of an if statement in C# is:

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
```

- If the boolean expression evaluates to **true,** then the block of code inside the if statement is executed.
- If boolean expression evaluates to **false,** then the first set of code after the end of the if statement (after the closing curly brace) is executed.

if Statement

- Flow Diagram



if Statement

- Example

```
static void Main(string[] args)
        {
            /* local variable definition */
            int a = 10;

            /* check the boolean condition using if statement */
            if (a < 20)
            {
                /* if condition is true then print the following */
                Console.WriteLine("a is less than 20");
            }
            Console.WriteLine("value of a is : {0}", a);
            Console.ReadLine();
        }
```

OUTPUT

```
a is less than 20;
value of a is : 10
```

if...else Statement

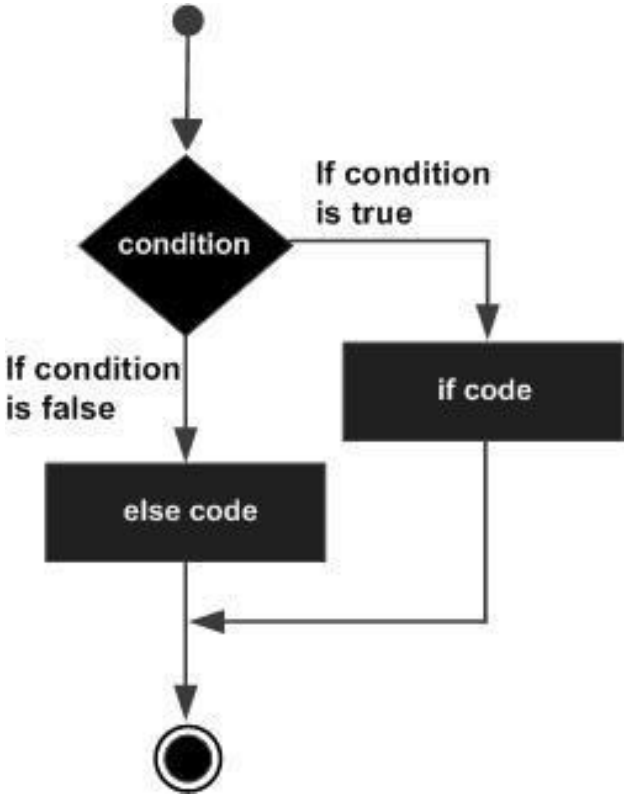
- An **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.
- **Syntax**

  The syntax of an **if...else** statement in C# is:

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
  /* statement(s) will execute if the boolean expression is false */
}
```

if...else Statement

- Flow Diagram



if...else Statement

- Example

```
static void Main(string[] args)
        {
            /* local variable definition */
            int a = 100;

            /* check the boolean condition */
            if (a < 20)
            {
                /* if condition is true then print the following */
                Console.WriteLine("a is less than 20");
            }
            else
            {
                /* if condition is false then print the following */
                Console.WriteLine("a is not less than 20");
            }
            Console.WriteLine("value of a is : {0}", a);
            Console.ReadLine();
        }
```

OUTPUT

```
a is not less than 20;
value of a is : 100
```

The if...else if...else Statement

- An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.
- When using if, else if and else statements there are few points to keep in mind.
  - An if can have zero or one else's and it must come after any else if's.
  - An if can have zero to many else if's and they must come before the else.
  - Once an else if succeeds, none of the remaining else if's or else's will be tested.

The if...else if...else Statement

- **Syntax**

  The syntax of an **if...else if...else** statement in C# is:

```
if(boolean_expression 1)
{
   /* Executes when the boolean expression 1 is true */
}

else if( boolean_expression 2)
{
   /* Executes when the boolean expression 2 is true */
}

else if( boolean_expression 3)
{
   /* Executes when the boolean expression 3 is true */
}

else
{
   /* executes when the none of the above condition is true */
}
```

The if...else if...else Statement

- Example

```
static void Main(string[] args)
        {
            /* local variable definition */
            int a = 100;

            /* check the boolean condition */
            if (a == 10)
            {
                /* if condition is true then print the following */
                Console.WriteLine("Value of a is 10");
            }
            else if (a == 20)
            {
                /* if else if condition is true */
                Console.WriteLine("Value of a is 20");
            }
            else if (a == 30)
            {
                /* if else if condition is true  */
                Console.WriteLine("Value of a is 30");
            }
            else
            {
                /* if none of the conditions is true */
                Console.WriteLine("None of the values is matching");
            }
            Console.WriteLine("Exact value of a is: {0}", a);
            Console.ReadLine();
        }
```

OUTPUT

```
None of the values is matching
Exact value of a is: 100
```

Nested if Statements

- It is always legal in C# to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).
- **Syntax**
  The syntax for a **nested if** statement is as follows:

```
if( boolean_expression 1)
{
   /* Executes when the boolean expression 1 is true */

   if(boolean_expression 2)
   {
      /* Executes when the boolean expression 2 is true */
   }
}
```

- You can nest **else if...else** in the similar way as you have nested *if statement.*

Nested if Statements

- Example

```
static void Main(string[] args)
        {
            //* local variable definition */
            int a = 100;
            int b = 200;

            /* check the boolean condition */
            if (a == 100)
            {
                /* if condition is true then check the following */
                if (b == 200)
                {
                    /* if condition is true then print the following */
                    Console.WriteLine("Value of a is 100 and b is 200");
                }
            }
            Console.WriteLine("Exact value of a is : {0}", a);
            Console.WriteLine("Exact value of b is : {0}", b);
            Console.ReadLine();
        }
```

OUTPUT

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

Extra if statement

- Try This

```
static void Main(string[] args)
        {
            Boolean a = false;
            if (1 == 1 ? a = true : a = false);
            Console.WriteLine(a);
            Console.ReadLine();
        }
```

Switch Statement

- A **switch** statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each **switch case.**
- **Syntax**

  The syntax for a **switch** statement in C# is as follows:

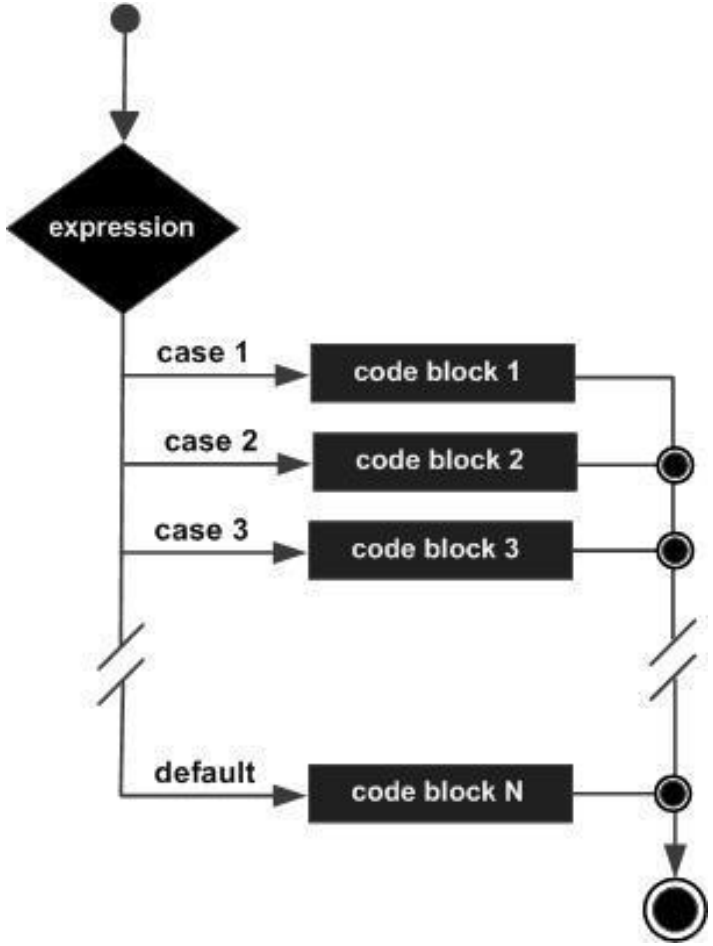
```
switch (expression)
            {
                case constant - expression:
                    statement(s);
                    break; /* optional */

                case constant - expression:
                    statement(s);
                    break; /* optional */

                /* you can have any number of case statements */
                default: /* Optional */
                    statement(s);
            }
```

Switch Statement

- The following rules apply to a **switch** statement:
  - The **expression** used in a **switch** statement must have an integral or enumerated type
  - You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
  - The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
  - When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
  - When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
  - Not every case needs to contain a **break.** If no **break** appears, the flow of control will *fall through to subsequent cases until a break is reached.*
  - A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case**.**

Switch Statement

- Flow Diagram



Switch Statement

```
static void Main(string[] args)
        {
            /* local variable definition */
            char grade = 'B';
            switch (grade)
            {
                case 'A':
                    Console.WriteLine("Excellent!");
                    break;
                case 'B':
                case 'C':
                    Console.WriteLine("Well done");
                    break;
                case 'D':
                    Console.WriteLine("You passed");
                    break;
                case 'F':
                    Console.WriteLine("Better try again");
                    break;
                default:
                    Console.WriteLine("Invalid grade");
                    break;
            }
            Console.WriteLine("Your grade is  {0}", grade);
            Console.ReadLine();
        }
```

OUTPUT

```
Well done
Your grade is B
```

Nested Switch Statement


```
static void Main(string[] args)
        {
            int a = 100;
            int b = 200;
            switch (a)
            {
                case 100:
                    Console.WriteLine("This is part of outer switch ");
                    switch (b)
                    {
                        case 200:
                            Console.WriteLine("This is part of inner switch ");
                            break;
                    }
                    break;
            }
            Console.WriteLine("Exact value of a is : {0}", a);
            Console.WriteLine("Exact value of b is : {0}", b);
            Console.ReadLine();
        }
```

OUTPUT

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

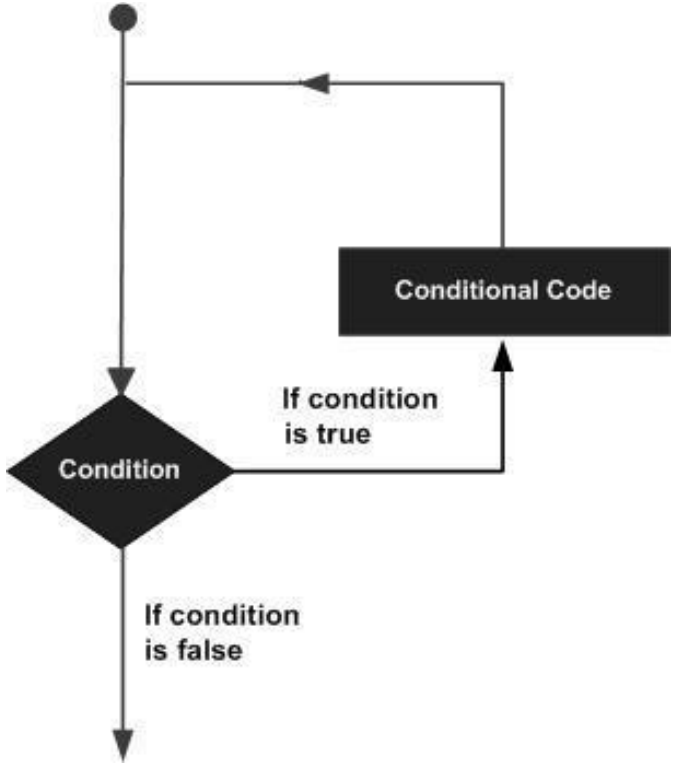
แบบฝึกหัด

- Write a program to input 5 number and show Max and Min numbers

```
Enter 5 Number
8
7
6
2
3
Max is 8
Min is 2
```

LOOPS

- A loop statement allows us to execute a statement or a group of statements multiple times and following is the general from of a loop statement in most of the programming languages:



LOOPS

- C# provides following types of loop to handle looping requirements

Loop Type	Description
while loop	It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do…while loop	It is similar to a while statement, except that it tests the condition at the end of the loop body.
nested loop	You can use one or more loops inside any another while, for or do…while loop.

While Loop

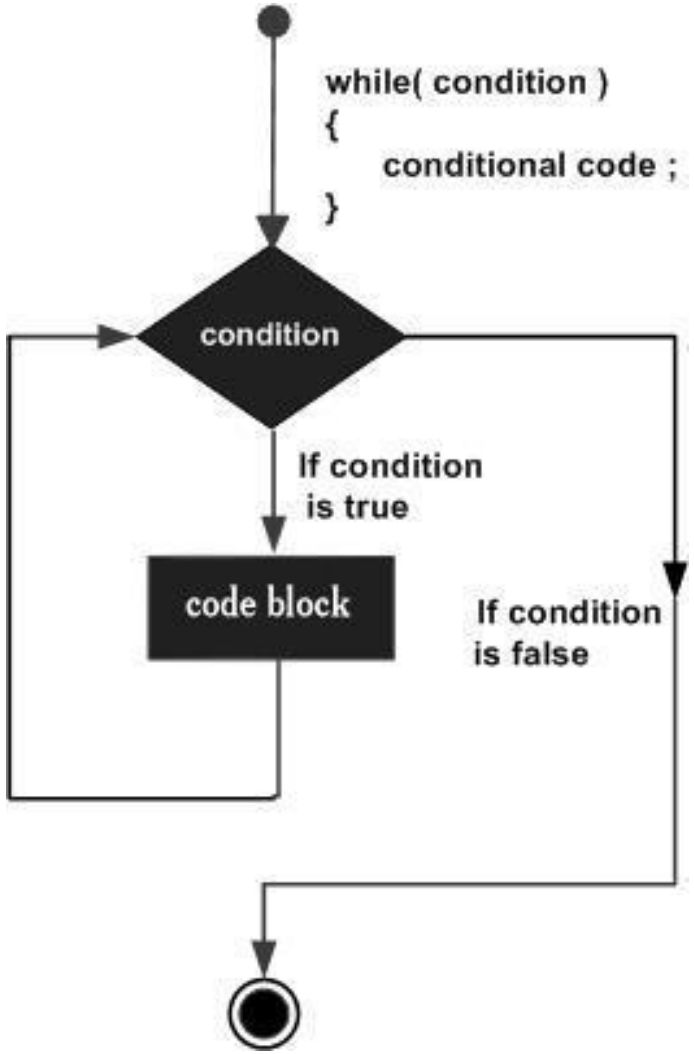
- A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true.
- **Syntax**
  - The syntax of a **while** loop in C# is:

```
while (condition)
{
     statement(s);
}
```

- **statement(s)** may be a single statement or a block of statements.
- The **condition** may be any expression, and true is any non-zero value.
- The loop iterates while the condition is true.

While Loop

- Flow Diagram



While Loop

- The *while loop* is that the loop might not ever run. When the condition is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

While Loop

- Example

```
static void Main(string[] args)
   {
            /* local variable definition */
            int a = 10;
            /* while loop execution */
            while (a < 20)
     {
                Console.WriteLine("value of a: {0}", a);
                a++;
     }
            Console.ReadLine();
   }
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

For Loop

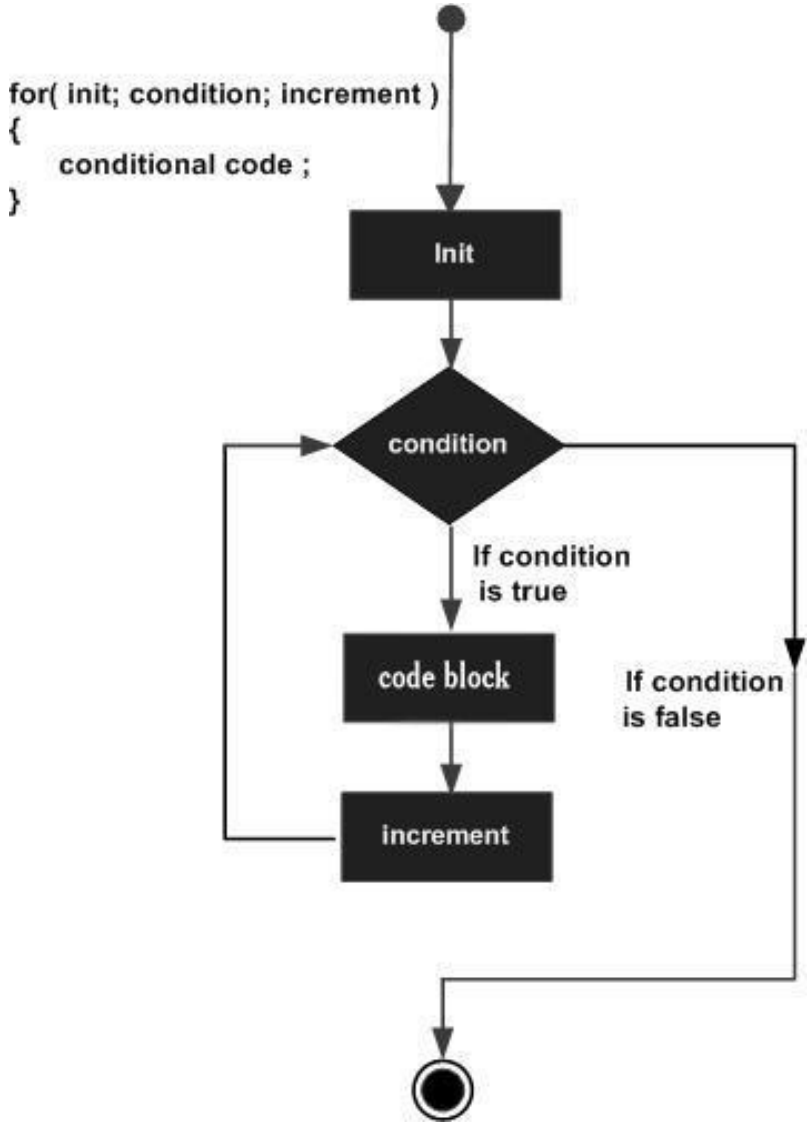
- A **for** loop is a repetition control structure that allows you to efficiently write a loopthat needs to execute a specific number of times.
- Syntax
  - The syntax of a **for** loop in C# is:

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

- Here is the flow of control in a for loop:
- 1. The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- 2. 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 flow of control jumps to the next statement just after the for loop.
- 3. After the body of the for loop executes, the flow of control 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.
- 4. 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 testing for a condition). After the condition becomes false, the for loop terminates.

For Loop

- Flow Diagram



For Loop

- Example

```
static void Main(string[] args)
{
        /* for loop execution */
        for (int a = 10; a < 20; a = a + 1)
    {
            Console.WriteLine("value of a: {0}", a);
    }
            Console.ReadLine();
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Do..While Loop

- Unlike **for** and **while** loops, which test the loop condition at the start of the loop, the **do...while** loop checks its condition at the end of the loop.
- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

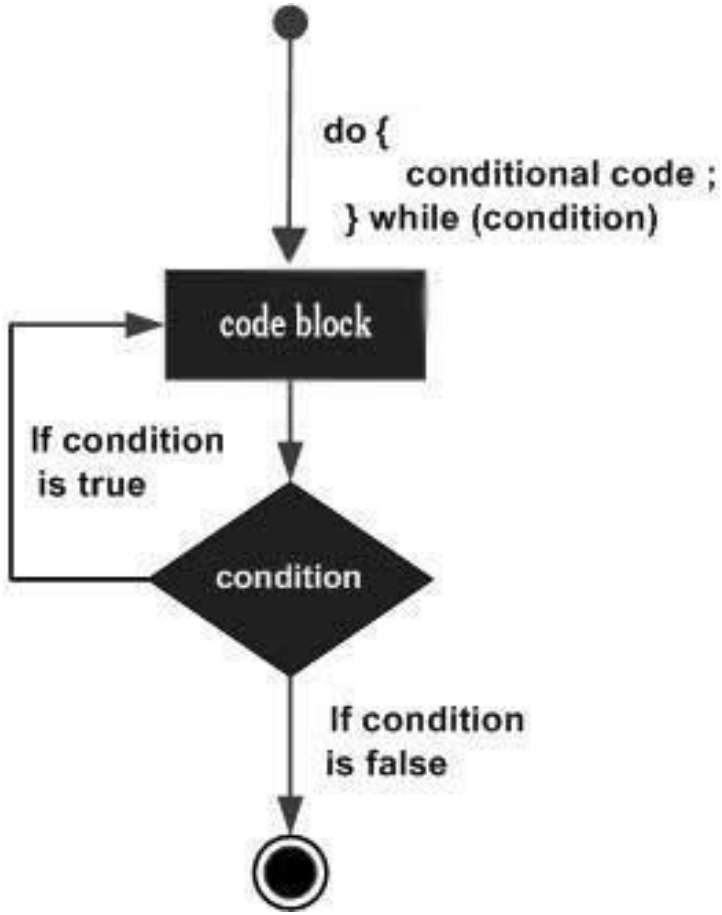
Do..While Loop

- Syntax
  - The syntax of a **do...while** loop in C# is:

```
do
{
     statement(s);

} while (condition);
```

Do..While Loop

- Flow Diagram



Do..While Loop

- Example

```
static void Main(string[] args)
{
      /* local variable definition */
      int a = 10;

      /* do loop execution */
      do
      {
          Console.WriteLine("value of a: {0}", a);
          a = a + 1;
      } while (a < 20);

          Console.ReadLine();
}
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Nested Loops

- C# allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.
- **Syntax**
  - The syntax for a **nested for loop** statement in C# is as follows:

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

Nested Loops

- The syntax for a **nested while loop** statement in C# is as follows:

```
while (condition)
{
      while (condition)
      {
            statement(s);
      }
      statement(s);
}
```

Nested Loops

- The syntax for a **nested do...while loop** statement in C# is as follows:

```
do
{
        statement(s);
        do
        {
            statement(s);
        } while (condition);

} while (condition);
```

Nested Loops

- Example
  - The following program uses a nested for loop to find the prime numbers from 2 to 100:

```
static void Main(string[] args)
        {
            /* local variable definition */
            int i, j;

            for (i = 2; i < 100; i++)
            {
                for (j = 2; j <= (i / j); j++)
                {
                    if ((i % j) == 0) break; // if factor found, not prime
                }
                if (j > (i / j))
                    Console.WriteLine("{0} is prime", i);
            }
            Console.ReadLine();
        }
```

OUTPUT

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Loop Control Statements

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- C# provides the following control statements. Click the following links to check their details.

Control Statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Loop Control Statements

- **Break Statement**

  The **break statement in C# has following two usage:**

  - 1. When the **break statement is encountered inside a loop, the loop is** immediately terminated and program control resumes at the next statement following the loop.
  - 2. It can be used to terminate a case in the **switch statement.**

- If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

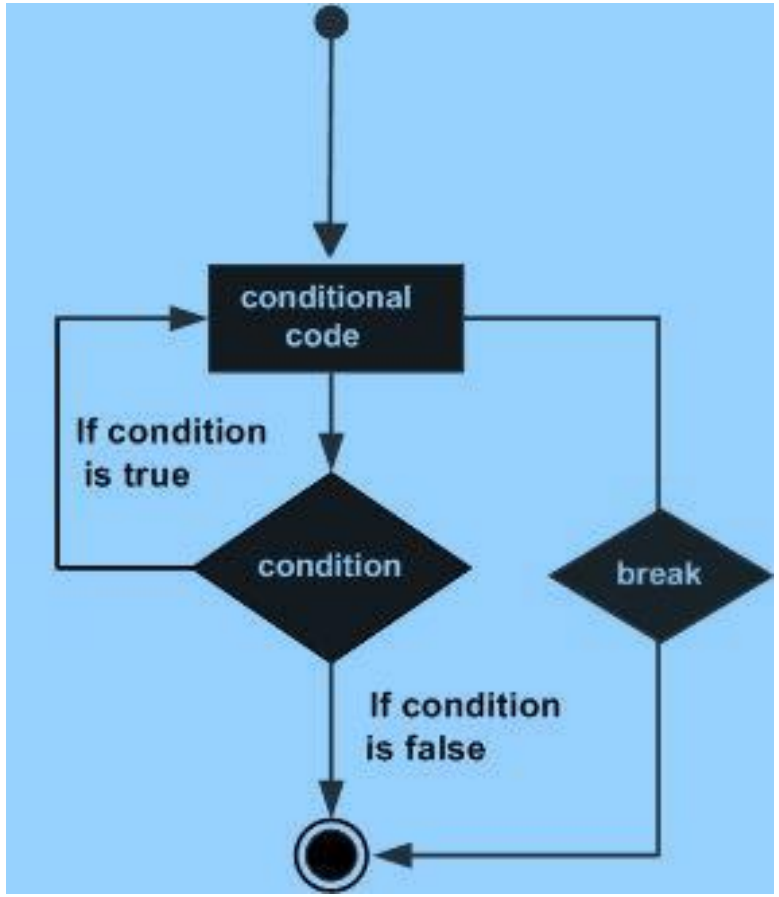
- **Syntax**

  - The syntax for a break statement in C# is as follows:

```
break;
```

Loop Control Statements

- Break Flow Diagram



Loop Control Statements

- Break Example

```
static void Main(string[] args)
        {
            /* local variable definition */
            int a = 10;

            /* while loop execution */
            while (a < 20)
            {
                Console.WriteLine("value of a: {0}", a);
                a++;
                if (a > 15)
                {
                    /* terminate the loop using break statement */
                    break;
                }
            }
            Console.ReadLine();
        }
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Loop Control Statements

- **Continue Statement**
- The **continue** statement in C# works somewhat like the **break** statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.
- **Syntax**
  - The syntax for a **continue** statement in C# is as follows:

```
continue;
```

Loop Control Statements

- Continue Example

```
static void Main(string[] args)
        {
            /* local variable definition */
            int a = 10;

            /* do loop execution */
            do
            {
                if (a == 15)
                {
                    /* skip the iteration */
                    a = a + 1;
                    continue;
                }
                Console.WriteLine("value of a: {0}", a);
                a++;

            } while (a < 20);

            Console.ReadLine();
        }
```

OUTPUT

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Infinite Loop

- A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose.
- Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.
- **Example**

```
static void Main(string[] args)
        {
            for (;;)
            {
                Console.WriteLine("Hey! I am Trapped");
            }
        }
```

แบบฝึกหัด

- Write a program to find sum of all integers greater than 100 and less than 200 that are divisible by 7

Result is : ????

CONCLUSION

การบ้านบทที่ 2

- จงเขียนโปรแกรมนับถอยหลังจากค่าที่ใส่เข้าไป เช่น 5 : 5 4 3 2 1