Object-Oriented Programming Using C#

Operators

Applications use operators to process the data entered by a user.

- Operators in C# can be classified as follows:
  - Arithmetic operators
  - Arithmetic Assignment operators
  - Unary operators
  - Comparison operators
  - Logical operators

Arithmetic Operators

- Arithmetic operators are the symbols that are used to perform arithmetic operations on variables.
- The following table describes the commonly used arithmetic operators.

Operator	Description	Example
+	Used to add two numbers	X=Y+Z; If Y is equal to 20 and Z is equal to 2, X will have the value 22.
-	Used to subtract two numbers	X=Y-Z; If Y is equal to 20 and Z is equal to 2, X will have the value 18.
*	Used to multiply two numbers	X=Y*Z; If Y is equal to 20 and Z is equal to 2, X will have the value 40.
/	Used to divide one number by another	X=Y/Z; If Y is equal to 21 and Z is equal to 2, X will have the value 10. But, if Y is equal to 21.0 and Z is equal to 2, X will have the value 10.5.
%	Used to divide two numbers and return the remainder	X=Y%Z; If Y is equal to 21 and Z is equal to 2, X will contain the value 1.

Arithmetic Assignment Operators

- Arithmetic assignment operators are used to perform arithmetic operations to assign a value to an operand.
- The following table lists the usage and describes the commonly used assignment operators.

Operator	Usage	Description
=	X = 5;	Stores the value 5 in the variable X.
+=	X+=Y;	Same as: X = X + Y;
-=	X-=Y;	Same as: X = X - Y;
*=	X*=Y;	Same as: X = X * Y;
/=	X/=Y;	Same as: X = X / Y;
%=	X%=Y;	Same as: X = X % Y;

Unary Operators

- Unary operators are used to increment or decrement the value of an operand by 1.
- The following table explains the usage of the increment and decrement operators.

Operator	Usage	Description	Example
++	++Operand; (Preincrement operator) Or, Operand++; (Postincrement operator)	Used to increment the value of an operand by 1	Y = ++X; If the initial value of X is 5, after the execution of the preceding statement, values of both X and Y will be 6. Y = X++; If the initial value of X is 5, after the execution of the preceding statement, value of X will be 6 and the value of Y will be 5.
--	--Operand; (Predecrement operator) Or, Operand--; (Postdecrement)	Used to decrement the value of an operand by 1	Y = --X; If the initial value of X is 5, after the execution of the preceding statement, values of X and Y will be 4. Y = X--; If the initial value of X is 5, after the execution of the preceding statement, value of X will be 4 and the value of Y will be 5.

Comparison Operators

- Comparison operators are used to compare two values and perform an action on the basis of the result of that comparison.
- The following table explains the usage of commonly used comparison operators.

Operator	Usage	Description	**Example** (In the following examples, the value of X is assumed to be 20 and the value of Y is assumed to be 25)
<	expression1 < expression2	Used to check whether expression1 is less than expression2	bool Result; Result = X < Y; Result will have the value true.
>	expression1 > expression2	Used to check whether expression1 is greater than expression2	bool Result; Result = X > Y; Result will have the value false.
<=	expression1 <= expression2	Used to check whether expression1 is less than or equal to expression2	bool Result; Result = X <= Y; Result will have the value true.
>=	expression1 >= expression2	Used to check whether expression1 is greater than or equal to expression2	bool Result; Result = X >= Y; Result will have the value false.

Comparison Operators

Operator	Usage	Description	Example (In the following examples, the value of X is assumed to be 20 and the value of Y is assumed to be 25)
==	expression1 == expression2	Used to check whether expression1 is equal to expression2	bool Result; Result = X == Y; Result will have the value false.
!=	expression1 != expression2	Used to check whether expression1 is not equal to expression2	bool Result; Result = X != Y; Result will have the value true.

Logical Operators

- Logical operators are used to evaluate expressions and return a Boolean value.
- The following table explains the usage of logical operators.

Operator	Usage	Description	Example
&&	expression1 && expression2	Returns true if both expression1 and expression2 are true.	bool Result; string str1, str2; str1 = “Korea”; str2 = “France”; Result= ((str1==“Korea”) && (str2==“France”)) Console.WriteLine (Result .ToString()); The message displays True because str1 has the value “Korea” and str2 has the value “France”.
!	! expression	Returns true if the expression is false.	bool Result int x; x = 20; Result=(!(x == 10)) Console.WriteLine(Result.ToString()); The message displays True because the expression used returns true.

Logical Operators

Operator	Usage	Description	Example
||	expression1 || expression2	Returns true if either expression1 or expression2 or both of them are true.	bool Result string str1, str2; str1 = “Korea”; str2 = “England”; Result= ((str1==“Korea”) || (str2== “France”)) Console.WriteLine (Result .ToString()); The message displays True if either str1 has the value “Korea” or str2 has the value “France”.
^	expression1 ^ expression2	Returns true if either expression1 or expression2 is true. It returns false if both expression1 and expression2 are true or if both expression1 and expression2 are false.	bool Result; string str1, str2; str1 = “Korea”; str2= “France”; Result = (str1== “Korea”) ^ (str2== “France”); Console.WriteLine (Result .ToString()); The message False is displayed because both the expressions are true.

Using Conditional Constructs

- Conditional constructs allow the selective execution of statements, depending on the value of expression associated with them.
- The comparison operators are required for evaluating the conditions.
- The various conditional constructs are:
  - The `if…else` construct
  - The `switch…case` construct

The if…else Construct

- The `if…else` conditional construct is followed by a logical expression where data is compared and a decision is made on the basis of the result of the comparison.
- The following is the syntax of the `if…else` construct:

```
if (expression)
{
    statements;
}
else
{
    statements;
}
```

The if…else Construct

- The `if…else` constructs can be nested inside each other.
- When `if…else` construct is nested together, the construct is known as `cascading if…else` constructs.

The switch…case Construct

- The `switch…case` construct is used when there are multiple values for a variable.
- The following is the syntax of the `switch…case` construct:

```
switch (VariableName)
{
  case ConstantExpression_1:
    statements;
    break;
  case ConstantExpression_2:
    statements;
    break;
  default:
    statements;
    break;
}
```

Using Loop Constructs

- Loop structures are used to execute one or more lines of code repetitively.
- The following loop constructs are supported by C#:
  - The `while` loop
  - The `do…while` loop
  - The `for` loop

The while Loop

- The `while` loop construct is used to execute a block of statements for a definite number of times, depending on a condition.
- The following is the syntax of the `while` loop construct:

```
while (expression)
{
  statements;
}
```

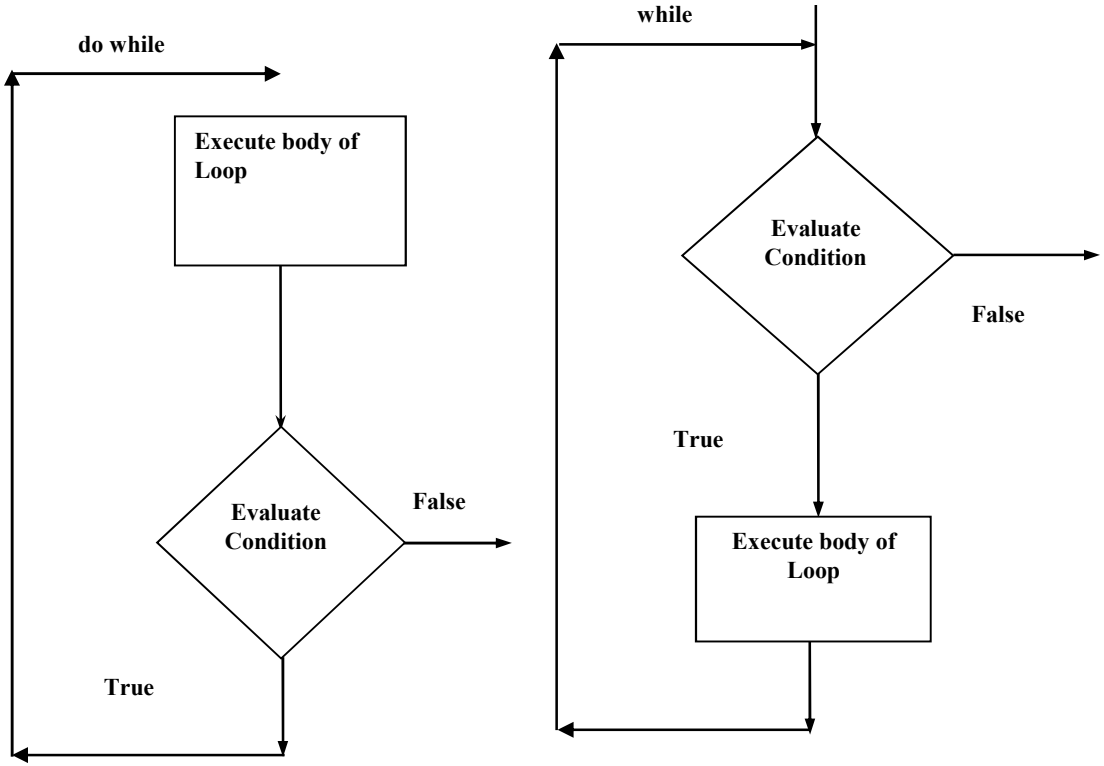
The do…while Loop

- The `do…while` loop construct is similar to the while loop construct.
- Both iterate until the specified loop condition becomes false.
- The following is the syntax of the `do…while` loop construct:

```
do
{
  statements;
}while(expression);
```

The do…while Loop

- The following figure shows the difference between the `do…while` and `while` loop construct.



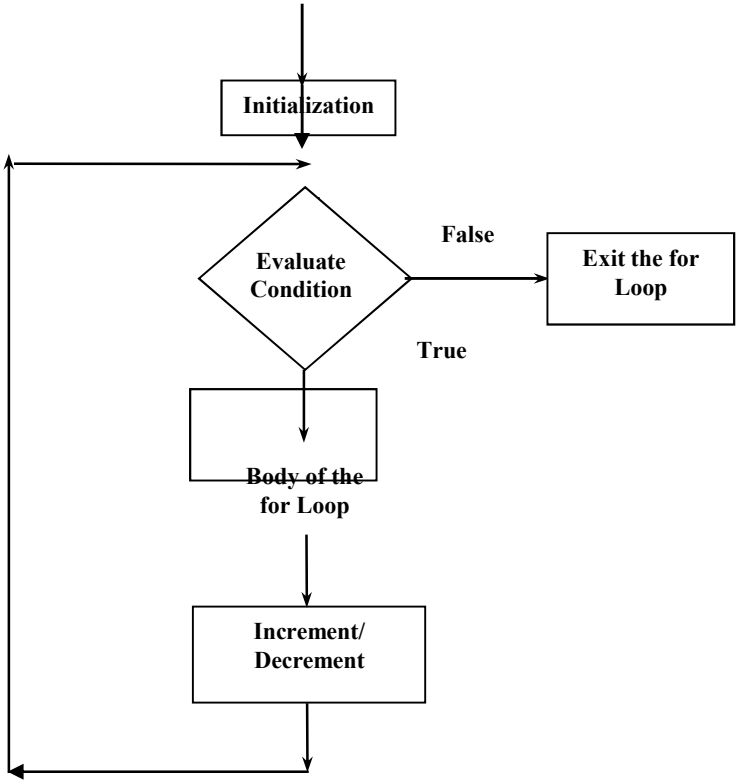
The for Loop

- The `for` loop structure is used to execute a block of statements for a specific number of times.
- The following is the syntax of the `for` loop construct:

```
for (initialization; termination;
 increment/decrement)
 {
     statements
 }
```

The for Loop

- The following figure shows the sequence of execution of a complete for loop construct.



The break and continue Statements

- The `break` statement is used to exit from the loop and prevents the execution of the remaining loop.
- The `continue` statement is used to skip all the subsequent instructions and take the control back to the loop.

Arrays in C#

An array is a group of variables of similar data types.

Example of declaring an integer array

```
int  []ary=new int [10];
```

or

```
int []ary;
ary=new int [10];
```

Assigning values to an array

```
ary[0]=10;
ary[1]=23;
ary[9]=125;
```

To display 2nd array element we write:

```
Console.WriteLine(ary[1]);
```

One more way to declare and initialize an array.

```
int [] ary={100,102,125,160,120};
```

Declaring a Two Dimensional Array

```
char [,]arr=new char[3,3];
arr[0,0]='x';
arr[0,1]='a';
arr[0,2]='2';
arr[2,1]='@';
```

To display value at 1st Row and 3rd Column we write:

```
Console.WriteLine(arr[0,2]);
```

One more way to declare and initialize a 2D array

```
int [,]ary= {
                {11,22,33},
                {44,55,66},
                {77,88,99}
            };
```