Operators and Expressions

Operators and Expressions

Operators are the symbols that represent a specific mathematical or logical processing in programming. Operators specify which operations to perform on operands. The following are some of the operators used in C#:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators – These are operators used in performing mathematical operations on numerical values. *Table 1* shows the basic arithmetic operators on C#.

Assume the following variable declarations: `int a = 11, b = 5;`

Operator	Description	Example	Return value
+	Adds two (2) operands	`a + b`	16
-	Subtracts the second operand from the first operand	`a – b`	6
*	Multiplies both operands	`a * b`	55
/	Divides the first operand by second operand	`a / b`	2
%	Modulus operator; returns the remainder after dividing first operand by second operand	`a % b`	1

Table 1. C# arithmetic operators (Gaddis, 2016)

Relational Operators – These are used to determine the relationship between operands of numeric values and generate a decision on that base. These return Boolean values `true` and `false`. *Table 2* shows the relational operators on C#.

Assume the following variable declarations: `int a = 11, b = 5;`

Operator	Description	Example	Return value
>	Determines if the value of the first operand is greater than the value of the second operand—if yes, it returns `true`; otherwise, `false`.	`a > b`	`true`
<	Determines if the value of the first operand is less than the value of the second operand—if yes, it returns `true`; otherwise, `false`.	`a < b`	`false`
>=	Determines if the value of the first operand is greater than or equal to the value of the second operand—if yes, it returns `true`; otherwise, `false`.	`a >= b`	`true`
<=	Determines if the value of the first operand is less than or equal to the value of the second operand—if yes, it returns `true`; otherwise, `false`.	`a <= b`	`false`
==	Determines if the two (2) operands are equal or not—if the values are equal, it returns `true`; otherwise, `false`.	`a == b`	`false`
!=	Determines if the two (2) operands are equal or not—if the values are not equal, it returns `true`; otherwise, `false`.	`a != b`	`true`

Table 2. C# relational operators (Gaddis, 2016)

Logical Operators – These are operators used in performing logical operation. These operate on logical expressions and returns a Boolean value. *Table 3* shows the logical operators on C#.

Assume the following variable declarations: `bool A = true, B = false;`

Operator	Description	Example	Return value
&&	Logical AND operator. This returns `true` only if both operands are `true`; otherwise, `false`.	`A && B`	`false`
||	Logical OR operator. This returns `true` if one (1) of the two (2) operands is `true`; otherwise, `false`.	`A || B`	`true`
^	Logical XOR operator. This returns `true` if only one (1) operand is `true`; otherwise, `false`.	`A ^ B`	`true`
!	Logical NOT operator. This reverses the value of a Boolean variable. If the value of the Boolean variable is `true`, the NOT operator will make it `false` and vice versa.	`!A`	`false`

Table 3. C# logical operators (Gaddis, 2016)

Assignment Operators – These are used to assign a value or the result of an expression to a variable. *Table 4* shows the assignment operators on C#.

Assume the following variable declarations: `int a = 2, b = 5;`

Operator	Description	Example	Value
=	This assigns a value of a variable or expression to the variable on its left side.	b = a	2
+=	This adds left operand to the first operand and assigns the result to the first operand.	b+=a	7
-=	This subtracts the second operand from the first operand and assigns the result to the first operand.	b-=a	3
*=	This multiplies both operands and assigns the result to the first operand.	b*=a	10
/=	This divides the first operand by the second operand and assigns the result to the first operand.	b/=a	2
%=	This assigns the remainder result to the first operand after dividing the first operand by the second operand.	b%=a	1
++	This adds 1 to the first operand and assigns the result to the first operand.	b++	6
--	This subtracts 1 from the first operand and assigns the result to the first operand.	b--	4

Table 4. C# assignment operators (Gaddis, 2016)

An **expression** in C# is a combination of operands (or variables) and operators that can be evaluated to a single value.

If all operands in an expression are integers, the expression evaluates to an integer value. For example:

```
int x = 10 + 5 * 2; //evaluates to 20.
```

If an expression contains a floating-point value, it evaluates to a floating-point value. For example:

```
double y = 10 + 5 * 2.0; //evaluates to 20.0.
```

Precedence and Associativity

The **operator precedence and associativity** defines a set of rules indicating the order in which the operator should be evaluated in an expression. When two (2) or more operators are used in an expression, the operators with the higher precedence are executed first, followed by the operators of lower precedence. *Table 5* displays common C# operators' precedence and their associativity. The operators in the table are arranged in order of precedence from highest to lowest.

Precedence	Operators	Associativity
Highest	!, ++, --, ()	Left to right
Intermediate	*, /, %	Left to right
	+, -	Left to right
	<, <=, >, >=	Left to right
	==, !=	Left to right
	^	Left to right
	&&, ||	Left to right
Lowest	=, *=, /=, %=, +=, -=	Right to left

Table 5. Common C# operators' precedence and associativity (Harwani, 2015)

Consider the following expression: `int a = 2 + 3 * 4;`

The precedence of the multiplication operator is higher than the plus operator, and the assignment operator has the lowest precedence. Therefore, `3 * 4` is evaluated first, and the result is added to 2.

When operators of the same precedence are used in a single expression, they are evaluated from left to right.

The *Math* Class

The **`System.Math`** class includes several methods that perform a variety of calculations that can be used in a program. *Table 6* shows some of methods provided by `Math` class.

Method	Description
Pow()	Raises a number to the given power.

`Exp()`	Raises the constant *e* to the given power.
`Log()`	Returns the natural and 10-based logarithm, respectively.
`Sqrt()`	Returns the square root of the given number.
`Sign()`	Returns the sign of the specified number.
`Abs()`	Returns the absolute value of a given number.
`Ceiling()`	Returns the smallest integral value larger than or equal to a fractional number.
`Floor()`	Returns the largest integral value smaller than or equal to the given fractional number.
`Round()`	Returns the number after rounding it to the nearest integral value.
`Truncate()`	Returns the number after removing its fractional part.
`Sin()`	Returns the sine of the specified angle.
`Cos()`	Returns the cosine of the specified angle.
`Tan()`	Returns the tangent of the specified angle.

Table 6. C# Math class methods (Harwani, 2015)

The methods of `Math` class are static methods, which means they are used with the class name without instantiating an object. For example: `double answer = Math.Pow(2, 3); //returns 8`

REFERENCES:

Deitel, P. & Deitel, H. (2015). *Visual C# 2012 how to program* (5th Ed.). USA: Pearson Education, Inc.

Gaddis, T. (2016). *Starting out with visual C#* (4th Ed.). USA: Pearson Education, Inc.

Harwani, B. (2015). *Learning object-oriented programming in C# 5.0*. USA: Cengage Learning PTR.