Operators and Expressions

INTRODUCTION

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. They usually form a part of mathematical or logical expressions.

C# supports a rich set of operators. C# operators can be classified into a number of related categories as below:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators.
7. Bitwise operators
8. Special operators

ARITHMETIC OPERATORS

C# provides all the basic arithmetic operators. They are listed in Table 5.1

The operators + , – , * and / all work the same way as they do in other languages

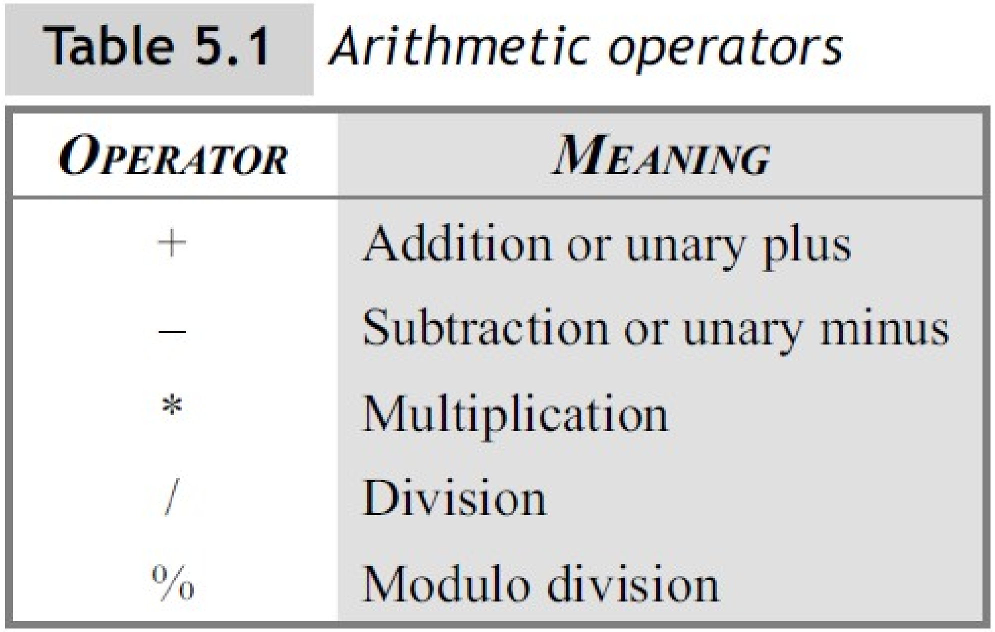

Table 5.1 *Arithmetic operators*

OPERATOR	MEANING
+	Addition or unary plus
–	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Program 5.1 | FLOATING-POINT ARITHMETIC

```
class FloatPoint
{
      public static void Main()
      {
          float a = 20.5F, b = 6.4F;
          System.Console.WriteLine("  a = " + a);
          System.Console.WriteLine("  b = " + b);
          System.Console.WriteLine("  a+b = " + (a+b));
          System.Console.WriteLine("  a-b = " + (a-b));
          System.Console.WriteLine("  a*b = " + (a*b));
          System.Console.WriteLine("  a/b = " + (a/b));
          System.Console.WriteLine("  a%b = " + (a%b));
      }
}
```

The output of Program 5.1 is follows:

```
a = 20.5
b = 6.4
a+b = 26.9
a-b = 14.1
a*b = 131.2
a/b=3.203125
a%b = 1.3
```

Table 5.2 *Relational operators*

OPERATOR	MEANING
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Table 5.3 *Relational expressions*

EXPRESSION	VALUE
4.5 <= 10	true
4.5 < –10	false
–35 >= 0	false
10 < 7+5	true
5.0 != 5	false
a + b == c+d	true*

*only if the sum of the values of a and b is equal to the sum of the values of c and d.

LOGICAL OPERATORS

In addition to the relational operators, C# has six logical operators, which are given in Table 5.4

Table 5.4 *Logical operators*

OPERATOR	MEANING
&&	logical AND
||	logical OR
!	logical NOT
&	bitwise logical AND
|	bitwise logical OR
^	bitwise logical exclusive OR

Table 5.5 *Truth table*

op-1	*op-2*	VALUE OF THE EXPRESSION	
		op-1 && op-2	*op-1 || op-2*
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

ASSIGNMENT OPERATORS

Assignment operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator, '='. In addition, C# has a set of 'shorthand' assignment operators which are used in the form

v **op=** exp

where v is a variable, *exp* is an expression and *op* is a C# binary operator. The operator **op** = is known as the *shorthand assignment operator.*

Table 5.6 *Shorthand assignment operators*

STATEMENT WITH SIMPLE ASSIGNMENT OPERATOR	STATEMENT WITH SHORTHAND OPERATOR
a = a+1	a += 1
a = a–1	a –= 1
a = a*(n+1)	a *= n+1
a = a/(n+1)	a /= n+1
a = a%b	a %= b

INCREMENT AND DECREMENT OPERATORS

C# has two very useful operators not generally found in many other languages. These are the increment and decrement operators:

++ and - -

The operator ++ adds 1 to the operand while – – subtracts 1. Both are unary operators and are used in the following form:

```
++m; or m++;

- -m; or m- -;

++m; is equivalent to m = m + 1; (or m + = 1;)

- -m; is equivalent to m = m - 1; (or m - = 1;)
```

Program 5.3 INCREMENT OPERATOR ILLUSTRATED

```
class IncrementOperator
{
        public static void Main()
        {
              int m = 10, n = 20;
              System.Console.WriteLine(" m = " + m);
              System.Console.WriteLine(" n = " + n);
              System.Console.WriteLine(" ++m = " + ++m);
              System.Console.WriteLine(" n++ = " + n++);
              System.Console.WriteLine(" m = " + m);
              System.Console.WriteLine(" n = " + n);
        }
}
```

The output of Program 5.3 is as follows:

```
m = 10
n = 20
++m = 11
n++ = 20
m = 11
n = 21
```

Similar is the case when we use ++ (or – –) in subscripted variables. That is, the statement

```
a[i++] = 10
```

is equivalent to

```
a[i]= 10
i = i+1
```

BITWISE OPERATORS

C# supports operators that may be used for manipulation of data atbit level. These operators may be used for testing the bits or shifting them to the right or left. Bitwise operators may not be applied to floating-point data. Table 5.7 lists the bitwise logical and shift operators.

Table 5.7 *Bitwise operators*

OPERATOR	MEANING
&	bitwise logical AND
|	bitwise logical OR
^	bitwise logical XOR
~	one's complement
<<	shift left
>>	shift right

ARITHMETIC EXPRESSIONS

C# supports the following special operators.

is (relational operator)

as (relational operator)

typeof (type operator)

sizeof (size operator)

new (object creator)

.(dot) (member-access operator)

checked (overfl ow checking)

unchecked (prevention of overflow checking)

These operators will be discussed as and when they are encountered

ARITHMETIC EXPRESSIONS

An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language

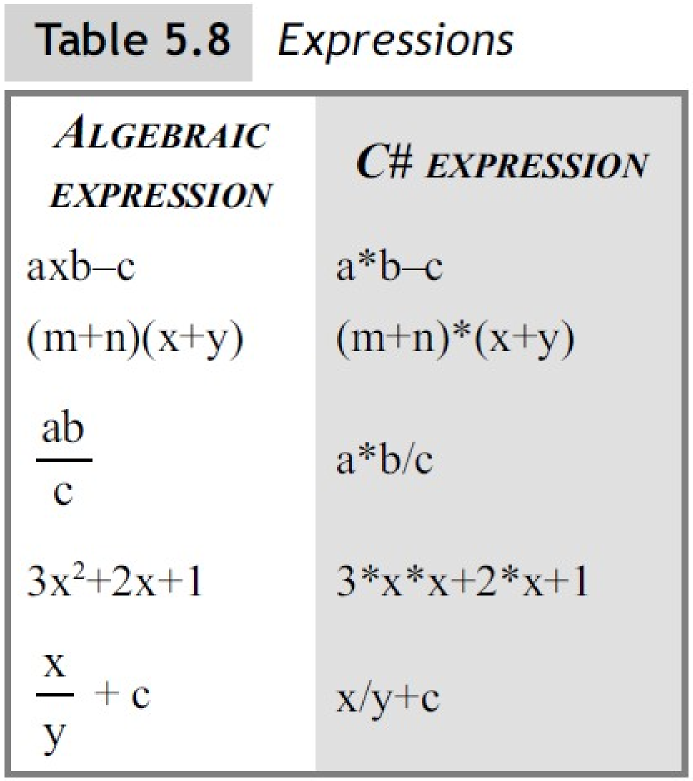

Table 5.8 *Expressions*

ALGEBRAIC EXPRESSION	C# EXPRESSION
axb–c	a*b–c
(m+n)(x+y)	(m+n)*(x+y)
$\frac{ab}{c}$	a*b/c
$3x^2+2x+1$	3*x*x+2*x+1
$\frac{x}{y} + c$	x/y+c

EVALUATION OF EXPRESSIONS

Expressions are evaluated using an assignment statement of the form *variable = expression*; *variable* is any valid C# variable name. When the statement is encountered, the *expression* is evaluated fi rst and the result then replaces the previous value of the variable on the left-hand side. All variables used in the expression must be assigned values before evaluation is attempted. Examples of evaluation statements are

```
x = a*b-c;
y = b/c*a;
z = a-b/c+d;
```

PRECEDENCE OF ARITHMETIC OPERATORS

An arithmetic expression without any parentheses will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C#:

High priority * / %

Low priority + -

TYPE CONVERSIONS

We often encounter situations where there is a need to convert a data of one type to another before it is used in arithmetic operations or to store a value of one type into a variable of another type. For example, consider the code below:

```
byte b1 = 50;
byte b2 = 60;
byte b3 = b1 + b2;
```

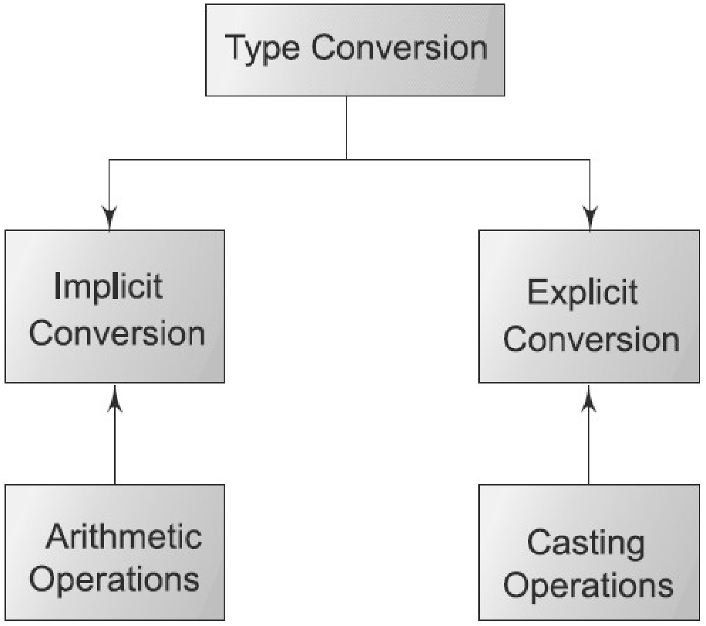

Fig. 5.1 *Type conversions in C#*

Explicit Conversions

There are many conversions that cannot be implicitly made between types. If we attempt such conversions, the compiler will give an error message. For example, the following conversions cannot be made implicitly:

- **int** to **short**
- **int** to **uint**
- **uint** to **int**
- **fl oat** to **int**
- **decimal** to any numeric type
- any numeric type to **char**

MATHEMATICAL FUNCTIONS

Often, we may need to use trigonometric functions and logarithms. The **System** namespace defines a class known as **Math** class with a rich set of static methods that makes math-oriented programming easy and efficient. It also contains two static members, **E** and **PI**, representing the values of mathematical constants e and p. Table 5.11 lists some of the mathematical methods contained in the **Math** class.

Table 5.11 *Main mathematical methods*

METHOD	DESCRIPTION
Sin ( )	sine of an angle in radians
Cos ( )	cosine of an angle in radians
Tan ( )	tangent of an angle in radians
Asin ( )	inverse of sine
Acos ( )	inverse of cosine
Atan ( )	inverse of tangent
Atan2 ( )	inverse tangent, with x and y co-ordinates specified
Sinh ( )	hyperbolic sine
Cosh ( )	hyperbolic cosine
Tanh ( )	hyperbolic tangent
Sqrt ( )	square root
Pow ( )	number raised to a given power
Exp ( )	exponential
Log ( )	natural logarithm (base e)
Log10 ( )	base 10 logarithm
Abs ( )	absolute value of a number
Min ( )	lower of two numbers
Max ( )	higher of two numbers
Sign ( )	sign of the number