Ch 5:Operators and Expressions

By: Ansari Rafiya

Operators

- *Operators is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.*

Operators are classified into

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

Operator	Meaning
+	Addition or unary plus
-	Subtraction
*	Multiplication
/	Division
%	Modulo division

For e.g,

```
Console.Writeline(“ a+b= ” +(a+b));
```

Assignment Operators

- This Operator are used to assign the value of an expression to a variable.

Syntax:

V **op**=exp

V is a variable,**exp** is an expression and **op** is a binary operator

e.g

x+=y+1; or **x=x+(y+1);**

Assignment Operators

Assignment operator	Sample expression	Explanation
+=	c += 7	c = c + 7
-=	d -= 4	d = d - 4
*=	e *= 5	e = e * 5
/=	f /= 3	f = f / 3
%=	g %= 2	g = g % 2

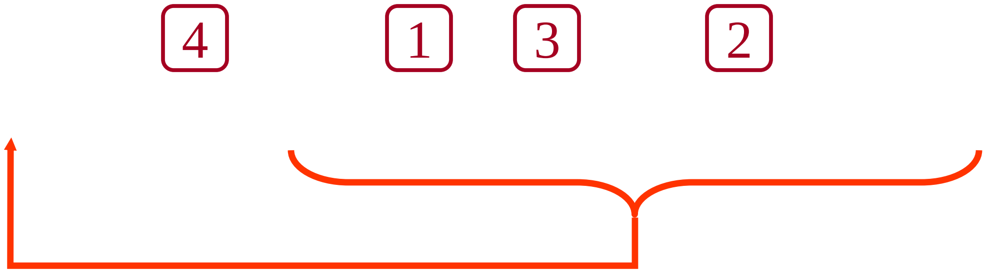
```
count  =  count + 1; // these two are equivalent
count ++;

count = count - 1; // these two are equivalent
count --;
```

Assignment Revisited

- You can consider assignment as another operator, with a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

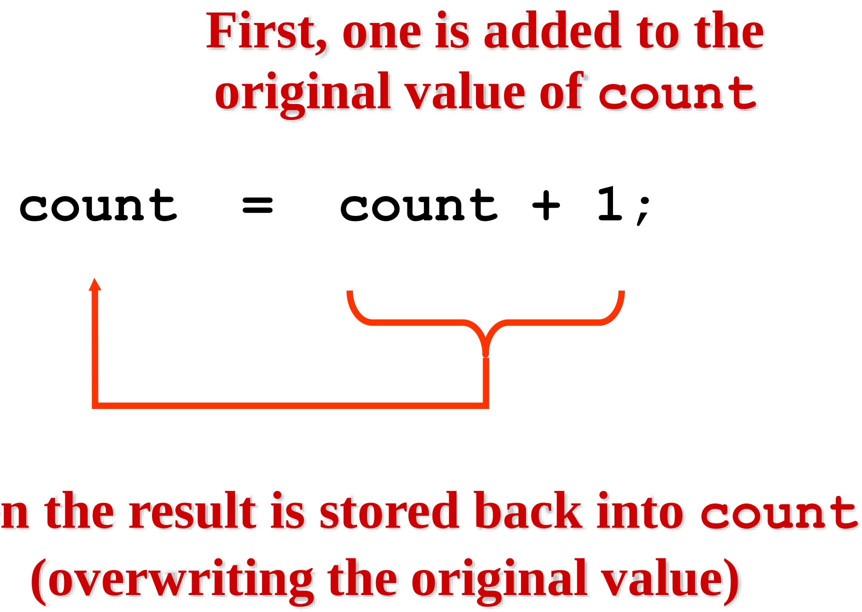
```
answer  =  sum / 4 + MAX * lowest;
```



Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable



First, one is added to the original value of `count`

```
count  =  count + 1;
```

Then the result is stored back into `count` (overwriting the original value)

Increment and Decrement Operators

Operator	Called	Sample expression	Explanation
`++`	preincrement	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	postincrement	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	predecrement	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	postdecrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 4.13 The increment and decrement operators.

Program: Increment.cs

Outline

Increment.cs

```
// Fig. 4.14: Increment.cs
// Preincrementing and postincrementing

using System;

class Increment
{
   static void Main(string[] args)
   {
      int c;

      c = 5;
      Console.WriteLine( c );   // print 5
      Console.WriteLine( c++ ); // print 5 t
      Console.WriteLine( c );   // print 6

      Console.WriteLine();      // skip a line

      c = 5;
      Console.WriteLine( c );   // print
      Console.WriteLine( ++c ); // preincr
      Console.WriteLine( c );   // print 6

   } // end of method Main

} // end of class Increment
```

Declare variable c

Set c equal to 5

Display c (5)

Display c (5) then add 1

Display c (6)

c is set to 5

Display c (5)

Add 1 then display c (6)

Display c (6)

Program Output

```
5
5
6

5
6
6
```

Swapping Values of Two Variables

- How about?

```
x  =  y;
y  =  x;
```

Value stored in

x	y
a	b
b	b
b	b

- Use two temporaries:

```
t1  =  x;
t2  =  y;
x   =  t1;
y   =  t2;
```

Swapping Values of Two Variables

- Just one temporary:

```
t1  =  x;
x   =  y;
y   =  t1;
```

- No temporaries!

~~x = x + y;~~
~~y = x - y;~~
~~x = x - y;~~

Value stored in

x	y
a	b
a+b	b
a+b	a
b	a

Don’t write such code!!

Relational Operators

- Acondition often uses one of C#’s *equality operators (==, !=)* or *relational operators (<, >, <=, >=)*, which all return boolean results:

==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

Equality and Relational Operators

Standard algebraic equality operator or relational operator	C# equality or relational operator	Example of C# condition	Meaning of C# condition
Equality operators			
=	`==`	`x == y`	**x** is equal to **y**
≠	`!=`	`x != y`	**x** is not equal to **y**
Relational operators			
>	`>`	`x > y`	**x** is greater than **y**
<	`<`	`x < y`	**x** is less than **y**
≥	`>=`	`x >= y`	**x** is greater than or equal to **y**
≤	`<=`	`x <= y`	**x** is less than or equal to **y**

Equality and relational operators.

Note the difference between the equality operator (==) and the assignment operator (=)

Question: if (grade = 100)
Console.WriteLine(“Great!”);

Program: Comparison.cs

Outline

Comparison.cs

```
// Comparison.cs
// Using if statements, relational operators and equality
// operators.

using System;

class Comparison
{
   static void Main( string[] args )
   {
      int number1,              // first number to compare
          number2;              // second number to compare

      // read in first number from user
      Console.Write( "Please enter first integer: " );
      number1 = Int32.Parse( Console.ReadLine() );

                                om user
                                tor second integer: " );



      if ( r
         Co                                           + number2 );

      if ( num
         Console.WriteLine( n

      if ( number1 < number2 )
         Console.WriteLine( number1 + " < " + number2 );

      if ( number1 > number2 )
         Console.WriteLine( number1 + " > " +  number2 );

```

If number1 is the same as number2 this line is preformed

Combining these two methods eliminates temporary string variable.

If number1 does not equal number2 th

If number1 is less than number2 the program wil

If number1 is greater than number2 this line will be preformed

Comparison.cs

```
         if ( number1 <= number2 )
            Console.WriteLine( number1 + " <= " + number2 );

         if ( number1 >= number2 )
            Console.WriteLine( number1 + " >= " + number2 );

      } // end method Main

   } // end class Comparison
```

If number1 is less than or equal to

nu ed

Lastly if number1 is greater than or equal to number2 then this code will be executed

Program Output

```
Please enter first integer: 2000

Please enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

```
Please enter first integer: 1000

Please enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Please enter first integer: 1000

Please enter second integer: 1000
1000 == 1000
1000 <= 1000
1000 >= 1000
```

Condional Operators

- The character pair ? : is ternary operator available in c#
- exp1 ? exp2 : exp3

e.g:

```
  a=10;
  b=15;
x=(a>b) ?  a : b;
```

This is same as

```
If(a>b)
x=a;
Else
x=b;
```

Comparing Characters

- We can also use the relational operators on character data
- The results are based on the Unicode character set
- The following condition is true because the character ’+’ comes before the character ’J’ in Unicode:

```
if ('+' < 'J')
   Console.WriteLine("+ is less than J");
```

- The uppercase alphabet (A–Z) and the lowercase alphabet (a–z) both appear in alphabetical order in Unicode

Logical Operators

- Boolean expressions can also use the following *logical and conditional operators*:

！	Logical NOT
＆	LogicalAND
｜	Logical OR
＾	Logical exclusive OR (XOR)
＆＆	ConditionalAND
｜｜	Conditional OR

- They all take boolean operands and produce boolean results

Logical and Conditional Operators

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Truth table for the && (logical AND) operator.

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

Truth table for the || (logical OR) operator.

Logical and Conditional Operators

expression1	expression2	expression1 ^ expression2
false	**false**	**false**
false	**true**	**true**
true	**false**	**true**
true	**true**	**false**

Truth table for the logical exclusive OR (**^**) operator.

expression	**!**expression
false	**true**
True	**false**

Truth table for operator **!** (logical NOT).

Program: LogicalOperators.cs

```
// Fig. 5.20: LogicalOperators.cs
// Demonstrating the logical operators.
using System;

class LogicalOperators
{
   // main entry point for application
   static void Main( string[] args )
   {
      // testing the conditional AND operator (&&)
      Console.WriteLine( "Conditional AND (&&)" +
         "\nfalse && false: " + ( false && false ) +
         "\nfalse && true:  " + ( false && true ) +
         "\ntrue && false:  " + ( true && false ) +
         "\ntrue && true:   " + ( true && true ) );

      // testing the conditional OR operator (||)
      Console.WriteLine( "\n\nConditional OR (||)" +
         "\nfalse || false: " + ( false || false ) +
         "\nfalse || true:  " + ( false || true ) +
         "\ntrue || false:  " + ( true || false ) +
         "\ntrue || true:   " + ( true || true ) );

      // testing the logical AND operator (&)
      Console.WriteLine( "\n\nLogical AND (&)" +
         "\nfalse & false: " + ( false & false ) +
         "\nfalse & true:  " + ( false & true ) +
         "\ntrue & false:  " + ( true & false ) +
         "\ntrue & true:   " + ( true & true ) );

```

Outputs a truth table for the conditional AND operator (&&)

Only true if both inputs are true

Outputs a truth table for the conditional OR operator (||)

Only false if both inputs are false

Outputs a truth table for the logical AND operator (&)

The result is only true if both are true

LogicalOperators.cs

```
         // testing the logical OR operator (|)
         Console.WriteLine( "\n\nLogical OR (|)" +
            "\nfalse | false: " + ( false | false ) +
            "\nfalse | true:  " + ( false | true ) +
            "\ntrue | false:  " + ( true | false ) +
            "\ntrue | true:   " + ( true | true ) );

         // testing the logical exclusive OR operator (^)
         Console.WriteLine( "\n\nLogical exclusive OR (^)"
            "\nfalse ^ false: " + ( false ^ false ) +
            "\nfalse ^ true:  " + ( false ^ true ) +
            "\ntrue ^ false:  " + ( true ^ false ) +
            "\ntrue ^ true:   " + ( true ^ true ) );

         // testing the logical NOT operator (!)
         Console.WriteLine( "\n\nLogical NOT (!)" +
            "\n!false: " + ( !false ) +
            "\n!true:  " + ( !true ) );
      }
   }
```

Outputs a truth table for the logical OR operator (||)

If one is true the result is true

Outputs a truth table for the logical exclusive OR operator (||)

conditionals are the same

Outputs a truth table for the logical NOT operator (!)

Returns the opposite as the input

Program Output

```
Conditional AND (&&)
false && false: False
false && true:  False
true && false:  False
true && true:   True


Conditional OR (||)
false || false: False
false || true:  True
true || false:  True
true || true:   True
```

Outline

LogicalOperators.cs
Program Output

```
Logical AND (&)
false & false: False
false & true:  False
true & false:  False
true & true:   True


Logical OR (|)
false | false: False
false | true:  True
true | false:  True
true | true:   True


Logical exclusive OR (^)
false ^ false: False
false ^ true:  True
true ^ false:  True
true ^ true:   False


Logical NOT (!)
!false: True
!true:  False
```

Comparison: Logical and Conditional Operators

- LogicalAND (&) and Logical OR (|)
  - Always evaluate both conditions

- ConditionalAND (&&) and Conditional OR (||)
  - Would not evaluate the second condition if the result of the first condition would already decide the final outcome.
  - Ex 1: `false && (x++ > 10)` --- no need to evaluate the 2nd condition
  - Ex 2:

```
if (count != 0 && total /count)
{

        …
 }
```

Program: LogicalVsConditional.cs

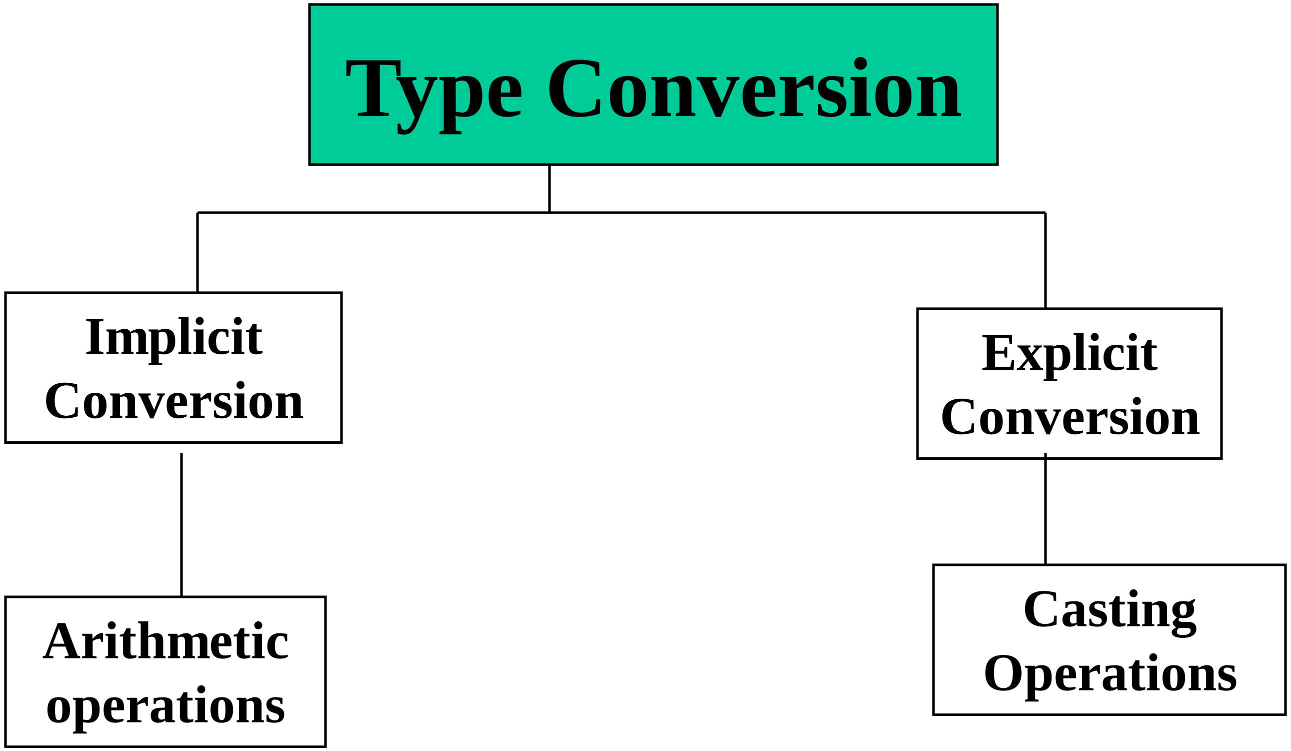
Precedence andAssociativity

high

low

Operators	Associativity	Type
( ) **++ --**	**left to right** **right to left**	**parentheses** **unary postfix**
++ -- + - (*type*)	**right to left**	**unary prefix**
*** / %**	**left to right**	**multiplicative**
+ -	**left to right**	**additive**
< <= > >=	left to right	relational
== !=	left to right	equality
?:	right to left	conditional
= += -= *= /= %=	**right to left**	**assignment**

Type Conversions



Type Conversion

Implicit Conversion

Arithmetic operations

Explicit Conversion

Casting Operations

Implicit conversion

An implicit conversion is also known as automatic coversion.

For e,g

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

The compiler will give an error message:
“cannot implicitly convert type **int** to type **byte**”

Implicit Conversion

We should write this

```
int b3=b1+b2;                    // no error
```

```
short b=75;
int  a=b                  // implicit conversion
```

This coversion is possible in following casses:

- ❒ From **byte** to **decimal**
- ❒ From **int** to **double**
- ❒ From **short** to **long**

Explicit conversion

The error will appear if you write this

```
int m=10;
 byte n=m;
```

We can carry out such conversion using the ‘cast’ operator The process is known as casting and is done as follows:

```
type variable1=(type) variable2;
```

e.g;

```
int m=50;
byte n=(byte) m;
```