Operators and Expressions

Operators and Expressions

Performing Simple Calculations with C#

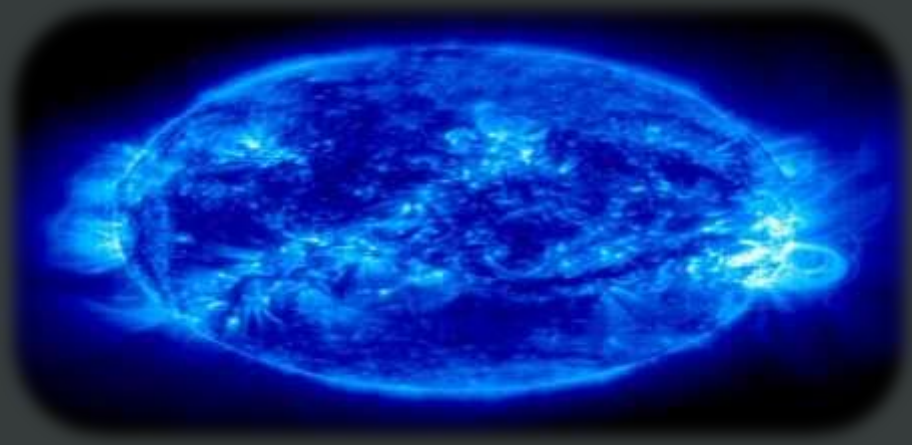


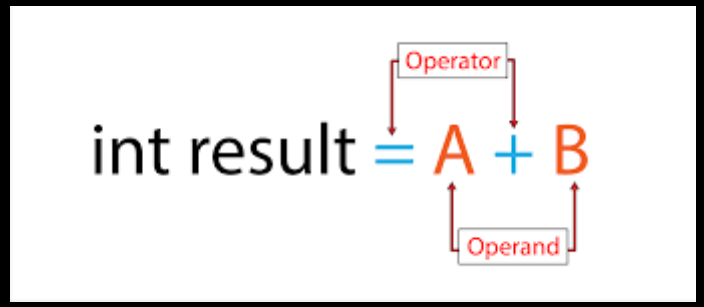
TABLE OF CONTENTS

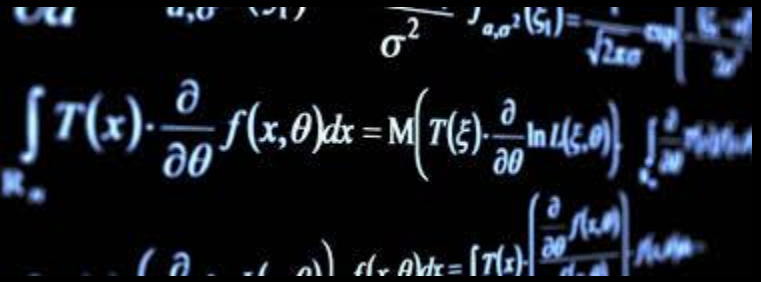
Operators in C#

ARITHMETIC, LOGICAL, COMPARISON, ASSIGNMENT, ETC.

WHAT IS AN OPERATOR?

- Operator is an operation performed over data at runtime
  - Takes one or more arguments (operands)
  - Produces a new value
- Operators have precedence
  - Precedence defines which will be evaluated first
  - int data = 10 + 5 * 5
- Expressions are sequences of operators and operands that are evaluated to a single value





OPERATORS IN C#

- Operators in C# :
  - Unary – take one operand
  - Binary – take two operands
  - Ternary (?:) – takes three operands
- Except for the assignment operators, all binary operators are left-associative
- The assignment operators and the conditional operator (?:) are right-associative

	Operator	Type
Binary Operator	+, -, *, /, %	Arithmetic Operators
	<,<=, >, >=, ==, !=	Relational Operators
	&&, ||, !	Logical Operators
	&, |, <<, >>, ~, ^	Bitwise Operators
	=, +=, -=,*=, /=, %=	Assignment Operators
Unary Operator →	++, --	Unary Operator
Ternary Operator →	?:	Ternary or Conditional Operator

OPERATOR PRECEDENCE

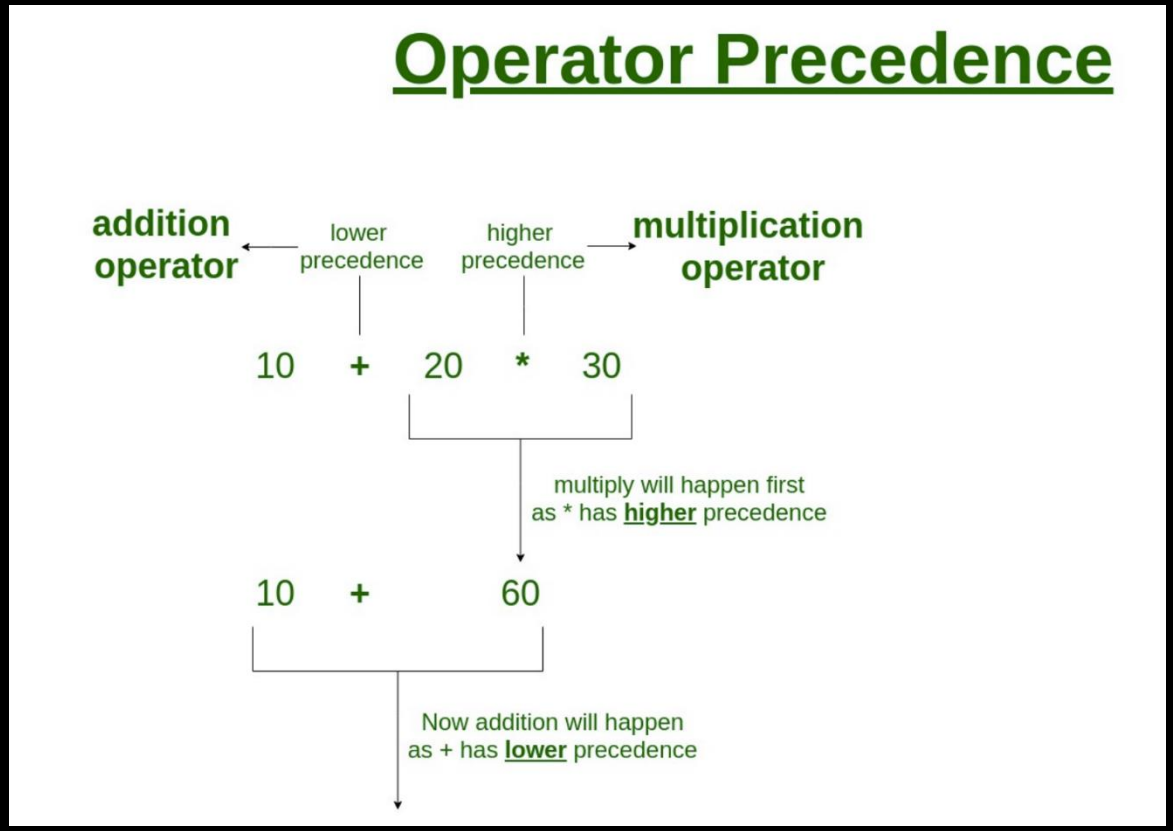
Operators	Category or name
x.y, f(x), a[i], x?.y, x?[y], x++, x--, x!, new, typeof, checked, unchecked, default, nameof, delegate, sizeof, stackalloc, x->y	Primary
+x, -x, !x, ~x, ++x, --x, ^x, (T)x, await, &x, *x, true and false	Unary
x..y	Range
switch, with	`switch` and `with` expressions
x * y, x / y, x % y	Multiplicative
x + y, x – y	Additive
x << y, x >> y	Shift
x < y, x > y, x <= y, x >= y, is, as	Relational and type-testing
x == y, x != y	Equality

Operators	Category or name
`x & y`	Boolean logical AND or bitwise logical AND
`x ^ y`	Boolean logical XOR or bitwise logical XOR
`x | y`	Boolean logical OR or bitwise logical OR
x && y	Conditional AND
x || y	Conditional OR
x ?? y	Null-coalescing operator
c ? t : f	Conditional operator
x = y, x += y, x -= y, x *= y, x /= y, x %= y, x &= y, x |= y, x ^= y, x <<= y, x >>= y, x ??= y,	Assignment and lambda declaration

OPERATORS PRECEDENCE

determines which operator is performed first in an expression with more than one operators with different precedence.

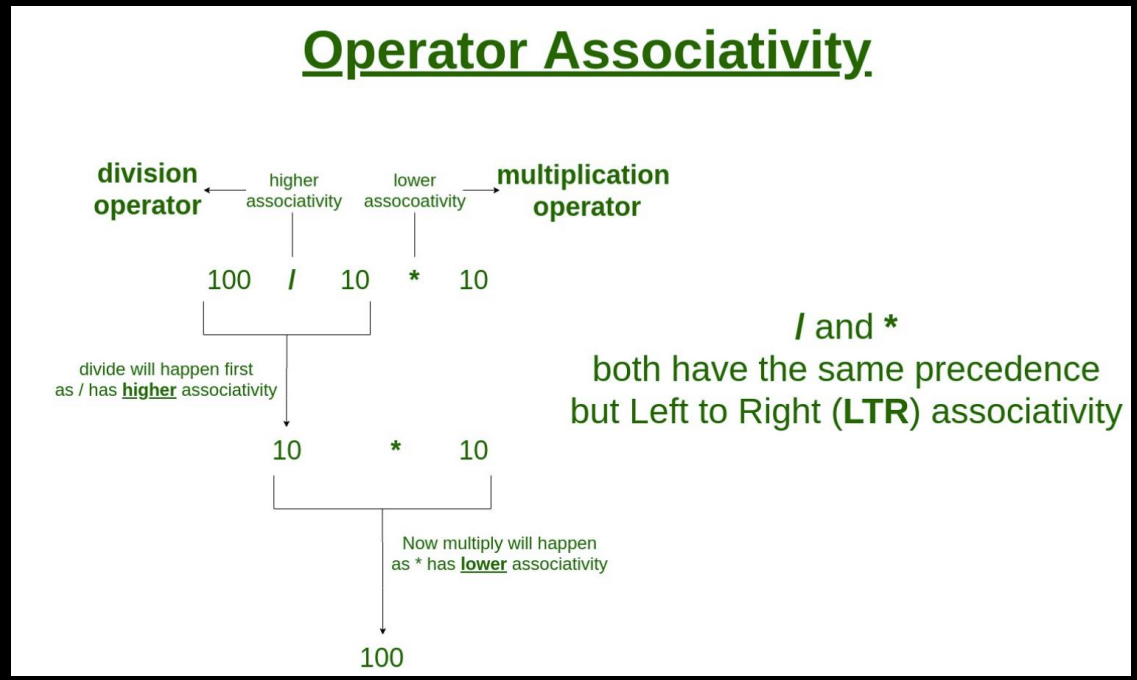
- The operator precedence affect the grouping and evaluation of operands in expressions.
- Precedence is meaningful only if other operators with higher or lower precedence are present.
- Expressions with higher-precedence operators are evaluated first.
- Precedence can also be described by the word "binding.“
- Operators with a higher precedence are said to have tighter binding.



OPERATORS ASSOCIATIVITY

is used when two operators of same precedence appear in an expression. Associativity can be either Left to Right or Right to Left.
For example: '*' and '/' have same precedence and their associativity is Left to Right, so the expression "100 / 10 * 10" is treated as "(100 / 10) * 10".

Category (By Precedence)	Operator(s)	Associativity
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to Left
Additive	+ -	Left to Right
Multiplicative	% / *	Left to Right
Relational	< > <= >=	Left to Right
Shift	<< >>	Left to Right
Equality	== !=	Right to Left
Logical AND	&	Left to Right
Logical OR	|	Left to Right
Logical XOR	^	Left to Right
Conditional OR	||	Left to Right
Conditional AND	&&	Left to Right
Null Coalescing	??	Left to Right
Ternary	?:	Right to Left
Assignment	= *= /= %= += - = <<= >>= &= ^= |= =>	Right to Left



OPERATORS PRECEDENCE AND ASSOCIATIVITY

- two characteristics of operators that determine the evaluation order of sub-expressions in absence of brackets
- Solve
  - Precedence 10 + 20 * 30
  - Associativity 100 + 200 / 10 - 3 * 10
- Associativity is only used when there are two or more operators of same precedence
- All operators with the same precedence have same associativity
- Precedence and associativity of postfix ++ and prefix ++ are different

ARITHMETIC OPERATORS

- Arithmetic operators +, -, * are the same as in math
- Division operator / if used on integers returns integer (without rounding) or exception
- Division operator / if used on real numbers returns real number or Infinity
- or NaN
- Remainder operator % returns the remainder from division of integers
- The special addition operator ++ increments a variable

ARITHMETIC OPERATORS – EXAMPLE

```
int squarePerimeter   = 17;
double squareSide   = squarePerimeter/4.0;
double squareArea   = squareSide*squareSide;
Console.WriteLine(squareSide);      //  4.25
Console.WriteLine(squareArea);      //  18.0625

int a = 5;
int b = 4;
Console.WriteLine(    a + b );//  9
Console.WriteLine(    a + b++ );// 9
Console.WriteLine(    a + b );// 10
Console.Wri teLine(    a + (++b) );// 11
Console.WriteLine(    a + b );// 11

Console.WriteLine(11    /  3);  //  3
Console.WriteLine(11    % 3);  //  2
Console.WriteLine(12    /  3);  //  4
```

Arithmetic Operators

Live Demo

LOGICAL OPERATORS

LOGICAL OPERATORS

- Logical operators take boolean operands and return boolean result
- Operator ! turns true to false and false
- to true
- Behavior of the operators &&,|| and ^
- (1 == true, 0 == false) :

Operation	||	||	||	||	&&	&&	&&	&&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

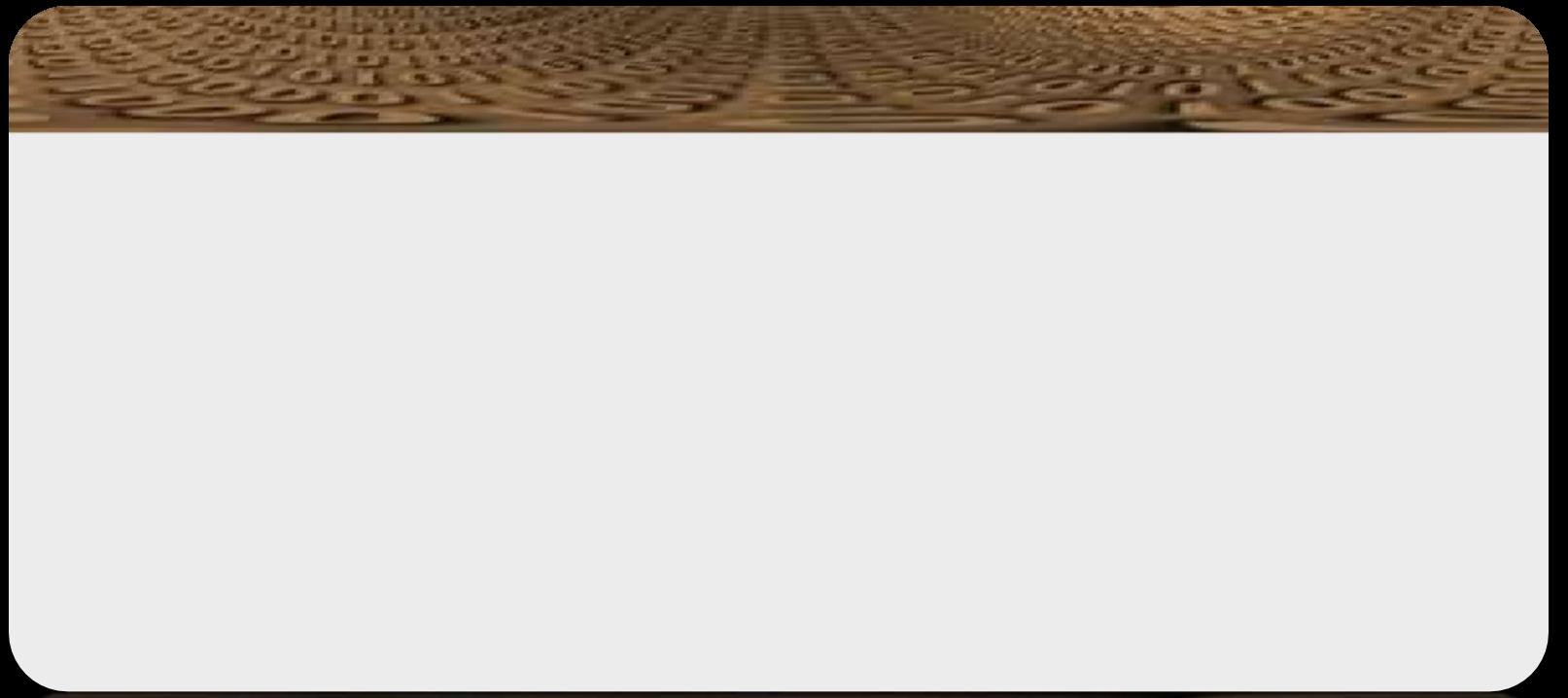
LOGICAL OPERATORS – EXAMPLE

- Using the logical operators:

```
bool a = true;
bool b = false;
Console.WriteLine(a && b); //False
Console.WriteLine(a || b); //True
Console.WriteLine(a ^ b); //  True
Console.WriteLine(!b);//   True
Console.WriteLine(b || true); //  True
Console.WriteLine(b && true); //  False
Console.WriteLine(a || true); //  True
Console.WriteLine(a && true); //  True
Console.WriteLine(!a);//   False
Console.WriteLine((5>7) ^(a==b)); // False
```

Logical Operators

Live Demo



BITWISE OPERATORS

BITWISE OPERATORS

- Bitwise operator ~ turns all 0 to 1 and all 1 to 0
  - Like ! for boolean expressions but bit by bit
- The operators |, & and ^ behave like ||, && and ^ for boolean expressions but bit by bit
- The << and >> move the bits (left or right)
- Behavior of the operators|, &and^:

Operation	|	|	|	|	&	&	&	&	^	^	^	^
Operand1	0	0	1	1	0	0	1	1	0	0	1	1
Operand2	0	1	0	1	0	1	0	1	0	1	0	1
Result	0	1	1	1	0	0	0	1	0	1	1	0

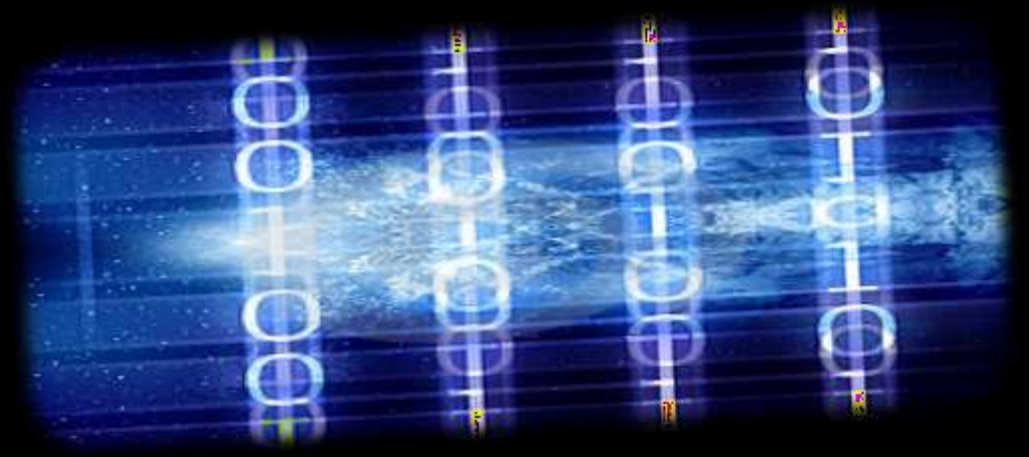
BITWISE OPERATORS (2)

- Bitwise operators are used on integer numbers (byte, sbyte, int, uint, long,
- ulong)
- Bitwise operators are applied bit by bit
- Examples:

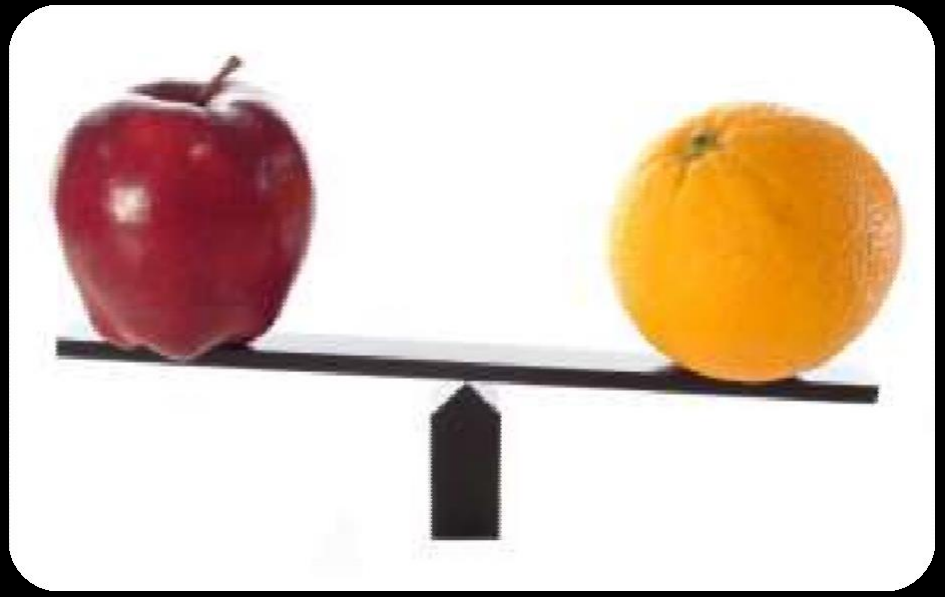
```
ushort a = 3;                  // 00000000 00000011
ushort b = 5;                  // 00000000 00000101
Console.WriteLine( a | b); // 00000000 00000111
Console.WriteLine( a & b); // 00000000 00000001
Console.WriteLine( a ^ b); // 00000000 00000110
Console.WriteLine(~a & b); // 00000000 00000100
Console.WriteLine( a<<1 ); // 00000000 00000110
Console.WriteLine( a>>1 ); // 00000000 00000001
```

Bitwise Operators

Live Demo



COMPARISON AND Assignment Operators



COMPARISON OPERATORS

- Comparison operators are used to compare variables
  - ==, <, >, >=, <=, !=
- Comparison operators example:

```
int a = 5;
int b = 4;
Console.WriteLine(a >=b);   // True
Console.WriteLine(a != b); // True
Console.WriteLine(a ==b);   // False
Console.WriteLine(a ==a);   // True
Console.Wri teLine(a !=++b);// False
Console.WriteLine(a > b); // False
```

ASSIGNMENT OPERATORS

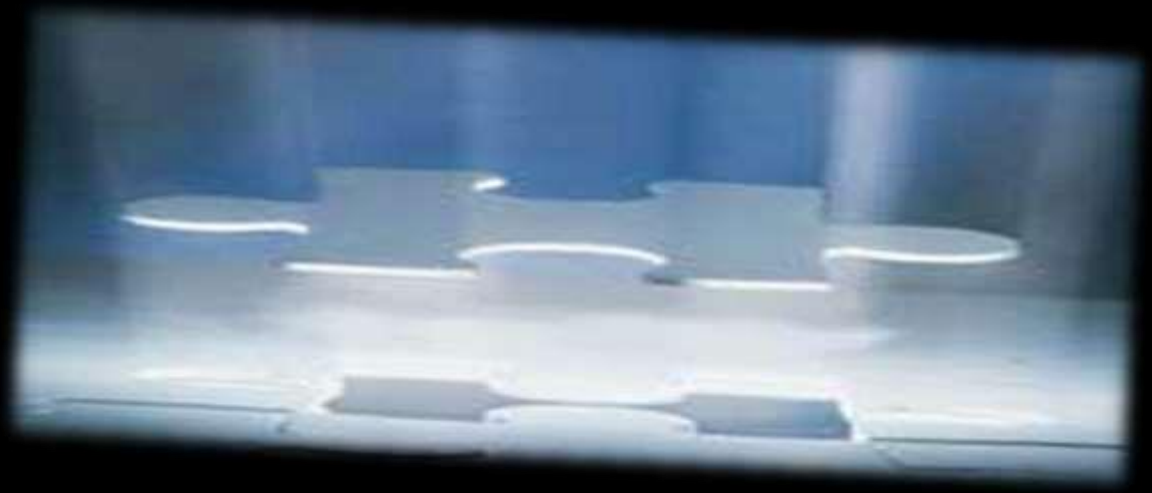
- Assignment operators are used to assign a value to a variable ,
  - =, +=, -=, |=, …
- Assignment operators example:

```
int x = 6;
int y = 4;
Console.WriteLine(y *= 2); // 8
int z = y = 3;             // y=3 and z=3
Console.WriteLine(z);      // 3
```

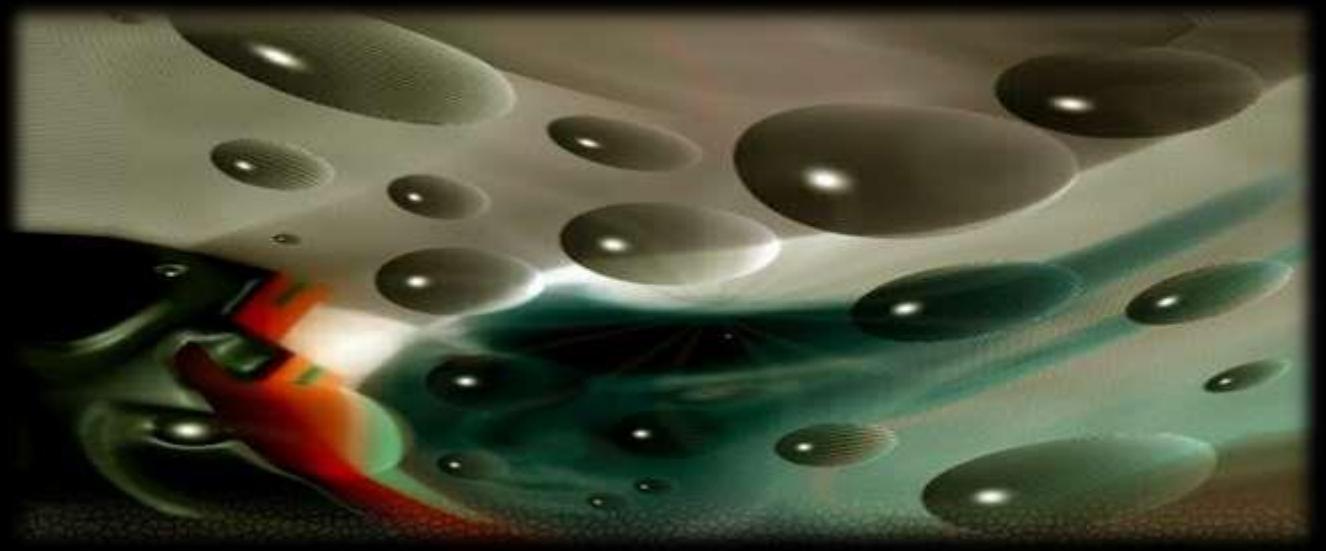
```
Console.WriteLine(x |= 1); // 7
Console.WriteLine(x += 3); // 10
Console.WriteLine(x /= 2); // 5
```

Assignment Operators

Live Demo



OTHER OPERATORS



OTHER OPERATORS

- String concatenation operator + is used to concatenate strings
- If the second operand is not a string, it is converted to string automatically

```
string first = "First";
string second = "Second";
Console.WriteLine(first + second);
// FirstSecond
string output = "The number is : ";
int number = 5;
Console.WriteLine(output + number);
// The number is : 5
```

OTHER OPERATORS (2)

- Member access operator `.` is used to access object members
- Square brackets `[]` are used with arrays indexers and attributes
- Parentheses `()` are used to override the default operator precedence
- Class cast operator `(type)` is used to cast one compatible type to another

OTHER OPERATORS (3)

- Conditional operator ?: has the form
- b ? x : y
- (if b is true then the result is x else the result is y)
- The `new` operator is used to create new objects
- The `typeof` operator returns `System.Type` object (the reflection of a type)
- The `is` operator checks if an object is compatible with given type

OTHER OPERATORS – EXAMPLE

- Using some other operators:

```
int a = 6;
int b = 4;
Console.WriteLine(a > b ? "a>b" : "b>=a"); // a>b
Console.WriteLine((long)a);  // 6

int c = b = 3; // b=3; followed by  c=3;
Console.WriteLine(c); // 3
Console.WriteLine(a isint);  //True
Console.WriteLine((a+b)/2); // 4
Console.WriteLine(typeof(int)); // System.Int32

int d = new int();
Console.WriteLine(d); // 0
```

Other Operators

Live Demo

IMPLICIT AND EXPLICIT TYPE

Conversions

IMPLICIT TYPE CONVERSION

- Implicit Type Conversion
  - Automatic conversion of value of one data type to value of another data type
  - Allowed when no loss of data is possible
    - "Larger" types can implicitly take values of smaller "types"
  - Example:

```
int i = 5;
longl = i;
```

EXPLICIT TYPE CONVERSION

- Explicit type conversion
  - Manual conversion of a value of one data type to a value of another data type
  - Allowed only explicitly by `(type)` operator
  - Required when there is a possibility of loss of data or precision
  - Example:

```
long l = 5;
int i = (int) l;
```

TYPE CONVERSIONS – EXAMPLE

- Example of implicit and explicit conversions:

```
float heightInMeters = 1.74f; // Explicit conversion
double maxHeight = heightInMeters; // Implicit

doubleminHeight  = (double) heightInMeters;//  Explicit

float actualHeight = (float)maxHeight;  // Explicit

float maxHeightFloat = maxHeight; // Compilation error!
```

- Note: Explicit conversion may be used even if not required by the compiler

Type Conversions

Live Demo

EXPRESSIONS

EXPRESSIONS

- Expressions are sequences of operators, literals and variables that are evaluated to some value
- Examples:

```
int r = (150-20) / 2 + 5; // r=70

// Expression for calculation of circle area
double surface = Math.PI * r * r;

// Expression forcalculation of circle perimeter
double perimeter = 2 * Math.PI*r;
```

EXPRESSIONS (2)

- Expressions has:
  - Type (integer, real, boolean, ...)
  - Value
- Examples:

```
int a = 2 + 3; // a = 5
int b = (a+3)*(a-4) + (2*a+ 7)/ 4; // b = 12
bool greater =(a> b)|| ((a== 0)&& (b== 0));
```

Expression of type int. Calculated at compile time.

Expression of type int. Calculated at runtime.

Expression of type bool. Calculated at runtime.

Expressions

Live Demo

SUMMARY

- We discussed the operators in C#:
  - Arithmetic, logical, bitwise, comparison, assignment and others
  - Operator precedence
- We learned when to use implicit and explicit type conversions
- We learned how to use expressions

OPERATORS AND EXPRESSIONS

Questions?

http://academy.telerik.com

EXERCISES

1. Write an expression that checks if given integer is odd or even.
2. Write a boolean expression that checks for given integer if it can be divided (without remainder) by 7 and 5 in the same time.
3. Write an expression that calculates rectangle's area by given `width` and `height`.
4. Write an expression that checks for given integer if its third digit (right-to-left) is 7. E. g. 1732 → `true`.
5. Write a boolean expression for finding if the bit 3 (counting from 0) of a given integer is 1 or 0.
6. Write an expression that checks if given point (`x`, `y`) is within a circle K(O, 5).

EXERCISES (2)

7. Write an expression that checks if given positive integer number n (n ≤ 100) is prime. E.g. 37 is prime.
8. Write an expression that calculates trapezoid's area by given sides a and b and height h.
9. Write an expression that checks for given point (x, y) if it is within the circle K((1,1), 3) and out of the rectangle R(top=1, left=-1, width=6, height=2).
10. Write a boolean expression that returns if the bit at position p (counting from 0) in a given integer number v has value of 1. Example: v=5; p=1 → false.

EXERCISES (3)

11. Write an expression that extracts from a given integer `i` the value of a given bit number `b`. Example: i=5; b=2 → value=1.

12. We are given integer number `n`, value `v` (`v`=0 or 1) and a position `p`. Write a sequence of operators that modifies `n` to hold the value `v` at the position `p` from the binary representation of `n`.

    Example: n = 5 (00000101), p=3, v=1 → 13 (00001101)

    n = 5 (00000101), p=2, v=0 → 1 (00000001)

EXERCISES (4)

13. Write a program that exchanges bits 3, 4 and 5 with bits 24,25 and 26 of given 32-bit unsigned integer.
14. * Write a program that exchanges bits {p, p+1, …, p+k-1) with bits {q, q+1, q+k-1} of given 32-bit unsigned integer.