Copyright © 2015 by The Learning House.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of The Learning House. For permission requests, write to The Learning House, addressed “Attention: Permissions Coordinator,” at the address below.

The Learning House

427 S. 4th Street #300

Louisville KY 40202

Expressions and Operators in C#

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Define the basic expressions and operators provided by C#

SOFTWARE GUILD

Expressions Defined

- An *expression* is a series of operators and operands.
  - *Operators* are symbols that represent an operation that returns a single result (ex: + - / *)
  - *Operands* are data elements used as input for operators
- Expressions can be combined, using operators, to create complex expressions.
- *Evaluation* is the process of applying operators to operands, in the proper sequence, to produce results.

SOFTWARE GUILD

Literals

- *Literals* are numbers or strings typed directly into the source code. Sometimes they are called “hard-coded” because they do not change at runtime.
- The literal *null* means the variable does not point to data in memory.
- Suffixes can be used to denote a specific type, if necessary.
- Examples of literals:

```
Console.WriteLine("{0}", 1024);   // int literal
Console.WriteLine("{0}", 3.14);   // double literal
Console.WriteLine("{0}", 3.14F);  // float literal
Console.WriteLine("{0}", true);   // boolean literal
Console.WriteLine("{0}", 'a');    // character literal
Console.WriteLine("{0}", "Hi!");  // string literal
```

SOFTWARE GUILD

Special Literal Characters

Some special characters in strings are *escaped,* which means they start with a \.

We saw earlier that \n could embed a new line in string output. To your right, you will see other common literal sequences.

"She said, \"Hello!\""

If you want to actually embed a slash \ in a string verbatim, prefix the string with the @ character:

@"I have a \ character"

Name	Escape Sequence	Hex Encoding
Null	\0	0x0000
Alert	\a	0x0007
Backspace	\b	0x0008
Horizontal tab	\t	0x0009
New line	\n	0x000A
Vertical tab	\v	0x000B
Form feed	\f	0x000C
Carriage return	\r	0x000D
Double quote	\"	0x0022
Single quote	\'	0x0027
Backslash	\\	0x005C

Order of Evaluation

- Expressions can be made up of many nested sub expressions.
- 3 * 5 + 2 = ?
  - If multiplication is first, the answer is 17
  - If addition is first, the answer is 21
- We learned in school that multiplication has a higher precedence so it goes first.
- C# has more than 45 operators and 14 levels of precedence.

SOFTWARE GUILD

Operators in Order of Precedence

Category	Operators
Primary	`a.x`, `f(x)`, `a[x]`, `x++`, `x--`, `new`, `typeof`, `checked`, `unchecked`
Unary	`+`, `-`, `!`, `~`, `++x`, `--x`, `(T)x`
Multiplicative	`*`, `/`, `%`
Additive	`+`, `-`
Shift	`<<`, `>>`
Relational and type	`<`, `>`, `<=`, `>=`, `is`, `as`
Equality	`==`, `!=`
Logical AND	`&`
Logical XOR	`^`
Logical OR	`|`
Conditional AND	`&&`
Conditional OR	`||`
Conditional	`?:`
Assignment	`=`, `*=`, `/=`, `%=`, `+=`, `-=`, `<<=`, `>>=`, `&=`, `^=`, `|=`

SOFTWARE GUILD

Handling Same Level of Precedence

- When sequential operators have the same level of precedence, the order is determined by *operator associativity.*
  - *Left Associative* is evaluated left to right
  - *Right Associative* is evaluated right to left
  - *Binary Operators,* except assignment operators, are left associative
  - *Assignment Operators* are right associative
  - *The Conditional Operator* is right associative

SOFTWARE GUILD

Arithmetic Operators

Operator	Name	Description
+	Addition	Adds the two operands.
-	Subtraction	Subtracts the second operand from the first.
*	Multiplication	Multiplies the two operands.
/	Division	Divides the first operand by the second. Integer division round the result toward 0 to the nearest integer.

SOFTWARE GUILD

The Remainder (mod) Operator

Operator	Name	Description
%	Remainder	Divides the first operand by the second operand and returns the remainder

- 0 % 3 = 0
- 1 % 3 = 1
- 2 % 3 = 2
- 3 % 3 = 0
- 4 % 3 = 1

SOFTWARE GUILD

Relational and Equality Comparisons

These are binary operators that always return a bool (true/false) and are left associative

Operator	Name	Description
<	Less than	`true` if the first operand is less than the second operand; `false` otherwise
>	Greater than	`true` if the first operand is greater than the second operand; `false` otherwise
<=	Less than or equal to	`true` if the first operand is less than or equal to the second operand `false` otherwise
>=	Greater than or equal to	`true` if the first operand is greater than or equal to the second operand; `false` otherwise
==	Equal to	`true` if the first operand is equal to the second operand; `false` otherwise
!=	Not equal to	`true` if the first operand is not equal to the second operand; `false` otherwise

SOFTWARE GUILD

Comparing Reference Types

- When comparing most reference types for equality, the comparison will be true only if the references point to the same memory location.
  - This is called a shallow comparison
  - Even if two objects have the same values, it will return false if they are different memory locations. (We will show you how to change this later.)
  - Strings are reference types, but when compared for equality, they are compared for the length and case sensitive content. (Deep comparison)

SOFTWARE GUILD

Increment / Decrement Operators

- The increment operator adds 1 and the decrement operator subtracts 1.
- They can be pre- or post-form, which changes the value returned:

	Starting Expression: x = 10	Value Returned to the Expression	Value of Variable Afte Evaluation
Pre-increment	++x	11	11
Post-increment	x++	10	11
Pre-decrement	--x	9	9
Post-decrement	x--	10	9

SOFTWARE GUILD

Conditional Logical Operators

- AND and OR operators evaluate expressions in a binary format and are left associative.

Operator	Name	Description
&&	Logical AND	`true` if both operands are `true`; `false` otherwise
||	Logical OR	`true` if at least one operand is `true`; `false` otherwise
!	Logical NOT	`true` if the operand is false; `false` otherwise

SOFTWARE GUILD

Conditional Operators Short Circuit

These operators *short circuit,* which means they will stop evaluation if, at any point, the result expression can be determined.

```
bool result;
int x = 10;

// 1 == 2 is false, so x == 10 will never be evaluated
result = (1 == 2) && (x == 10);
```

SOFTWARE GUILD

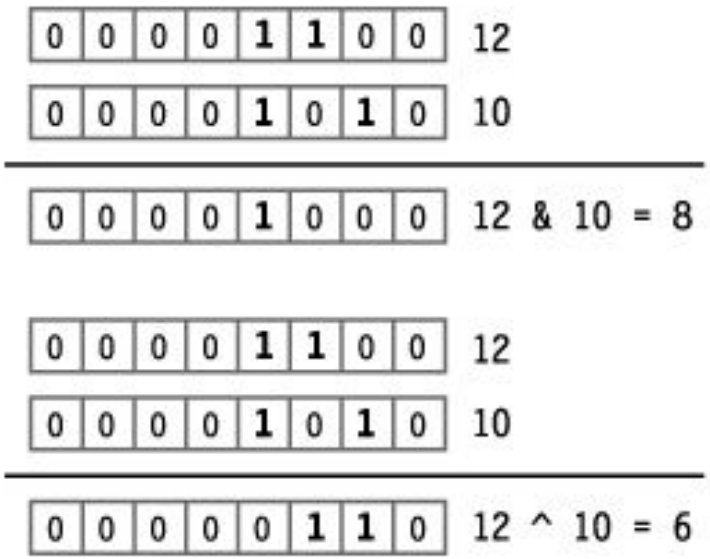
Logical (Bitwise) Operators

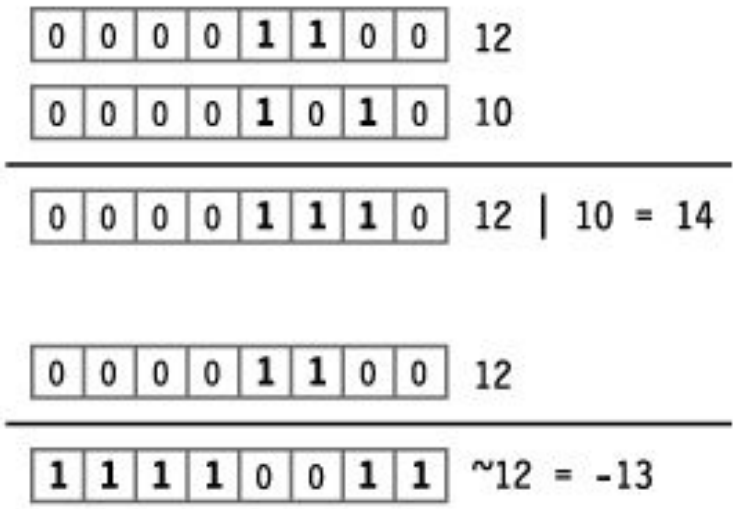
Binary bitwise operators compare bits at each position and set the bit of the return according to the operation.

Operator	Name	Description
&	Bitwise AND	Produces the bitwise AND of the two operands. The resulting bit is 1 only if both operand bits are 1.
|	Bitwise OR	Produces the bitwise OR of the two operands. The resulting bit is 1 if either corresponding operand bit is 1.
^	Bitwise XOR	Produces the bitwise XOR of the two operands. The resulting bit is 1 if one, but not both, of the corresponding operand bits is 1.
~	Bitwise negation	Each bit in the operand is switched to its opposite. This produces the one's complement of the operand. (The *one's complement* of a numbe is the inversion of every bit of its binary representation. That is, every (is switched to 1, and every 1 is switched to 0.)

SOFTWARE GUILD

Bitwise Examples





These are rarely used in normal business programming. Most often, they are seen in embedded software, some games, etc.

SOFTWARE GUILD

Assignment Operators

These operators evaluate the right side and set the value of the variable on the left to the result.

They are binary and right-associative… often used as shortcuts.

x += value

is the same as writing

x = x + value

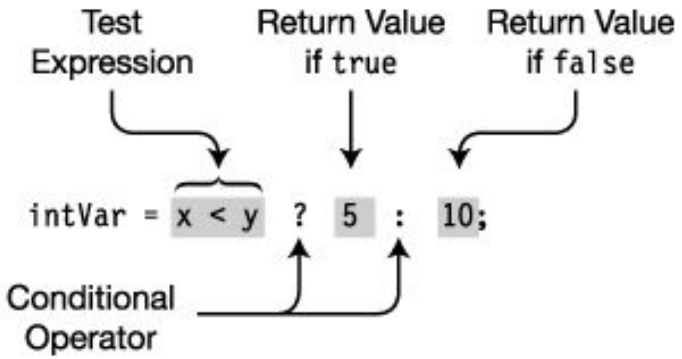
```
int x = 1;

x += 1; // x is now 2
x *= 2; // x is now 4
x -= 1; // x is now 3
x /= 3; // x is now 1
```

Conditional (Ternary) Operator

Operator	Name	Description
?:	Conditional operator	Evaluates an expression and returns one of two values, depending on whether the expression returns `true` or `false`

Shorthand for a simple if/else. Some people like the style… usage is based on team preference.

```
if( x < y )
   intVar = 5;
else
   intVar = 10;
```



SOFTWARE GUILD

New Developer Point of Confusion

Remember! Use = to assign a value and == to compare two values.

- x = 5 sets the variable x equal to 5
- x == 5 returns true/false depending on the value of x.

SOFTWARE GUILD

Conclusion

How you arrange expressions has a significant impact on your program flow and logic.

Although C# can chain many expressions on the same line, it’s best when you are starting out to write one expression per line/statement to keep things simple.

SOFTWARE GUILD