CS 112 Introduction to Programming

Lecture #6:

Arithmetic
and Assignment Operations

http://flint.cs.yale.edu/cs112/

Outline

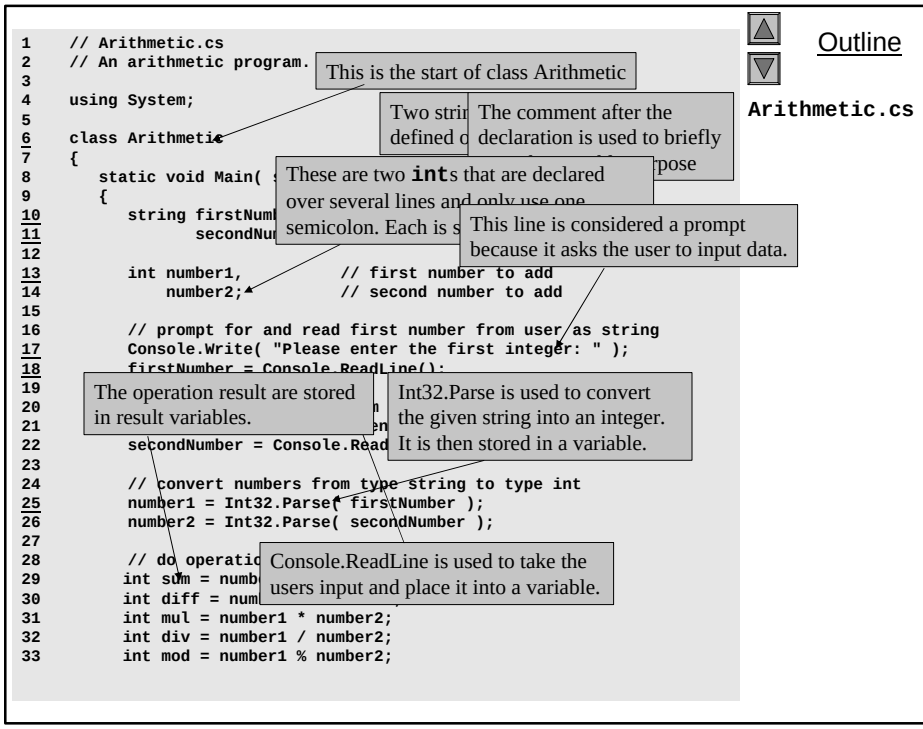
- Admin. and review
- *Arithmetic operations*
- Assignment operations

Arithmetic Expressions

- An *expression* is a combination of operators and operands
- *Arithmetic expressions* (we will see logical expressions later) compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- There are no exponents




```
1   // Arithmetic.cs
2   // An arithmetic program.
3
4   using System;
5
6   class Arithmetic
7   {
8      static void Main( s
9      {
10        string firstNumb
11               secondNu
12
13        int number1,          // first number to add
14            number2;          // second number to add
15
16        // prompt for and read first number from user as string
17        Console.Write( "Please enter the first integer: " );
18        firstNumber = Console.ReadLine();
19
20
21
22        secondNumber = Console.Read
23
24        // convert numbers from type string to type int
25        number1 = Int32.Parse( firstNumber );
26        number2 = Int32.Parse( secondNumber );
27
28        // do operatio
29        int sum = numb
30        int diff = num
31        int mul = number1 * number2;
32        int div = number1 / number2;
33        int mod = number1 % number2;
```

```
32          // display results
33          Console.WriteLine( "\n{0} + {1} = {2}.", number1, number2, sum );
34          Console.WriteLine( "\n{0} - {1} = {2}.", number1, number2, diff );
35          Console.WriteLine( "\n{0} * {1} = {2}.", number1, number2, mul );
36          Console.WriteLine( "\n{0} / {2} = {2}.", number1, number2, div );
37          Console.WriteLine( "\n{0} % {1} = {2}.", number1, number2, mod );
34
35       } // end method Main
36
37    } // end class Arithmetic
```

Outline

Arithmetic.cs

Putting a variable out through Console.WriteLine is done by placing the variable after the text while using a marked place to show where the variable should be placed.

Division and Remainder

- If both operands to the division operator (`/`) are integers, the result is an integer (the fractional part is discarded)

`14 / 3`	**equals?**	**`4`**
`8 / 12`	**equals?**	**`0`**

- The remainder operator (%) returns the remainder after dividing the second operand into the first

`14 % 3`	**equals?**	**`2`**
`8 % 12`	**equals?**	**`8`**

6

Operator Precedence

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated

- Precedence rules
  - Parenthesis are done first
  - Division, multiplication and modulus are done second
    - Left to right if same precedence (this is called associativity)
  - Addition and subtraction are done last
    - Left to right if same precedence

7

Precedence of Arithmetic Operations

Operator(s)	Operation	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*****, **/** or **%**	Multiplication Division Modulus	Evaluated second. If there are several such operators, they are evaluated left to right.
+ or **-**	Addition Subtraction	Evaluated last. If there are several such operators, they are evaluated left to right.

Precedence of arithmetic operators.

8

Operator Precedence: Examples

- What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4

a + b * c - d / e
  3   1   4   2

a / (b + c) - d % e
  2   1   4   3

a / (b * (c + (d - e)))
  4   3   2   1
```

Example: TemperatureConverter.cs

9

Data Conversions

- Sometimes it is convenient to convert data from one type to another
  - For example, we may want to treat an integer as a floating point value during a computation
- Conversions must be handled carefully to avoid losing information
- *Two types of conversions*
  - *Widening conversions* are generally safe because they tend to go from a small data type to a larger one (such as a `short` to an `int`)
    - Q: how about int to long?
  - *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an `int` to a `short`)

10

Data Conversions

- In C#, data conversions can occur in three ways:
  - Assignment conversion
    - occurs automatically when a value of one type is assigned to a variable of another
    - only widening conversions can happen via assignment
    - Example: aFloatVar = anIntVar
  - Arithmetic promotion
    - happens automatically when operators in expressions convert their operands
    - Example: aFloatVar / anIntVar
  - Casting

11

Data Conversions: Casting

- *Casting* is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted
- For example, if `total` and `count` are integers, but we want a floating point result when dividing them, we can cast `total`:

```
result = (float) total / count;
```

Example: DataConversion.cs

12

Outline

- Admin. and review
- Arithmetic operations
- *Assignment operations*

13

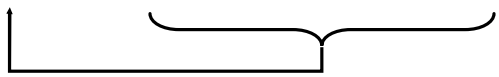
Assignment Revisited

- 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;
```

4 1 3 2



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

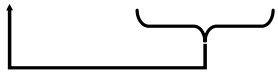
14

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)

15

Assignment Operators

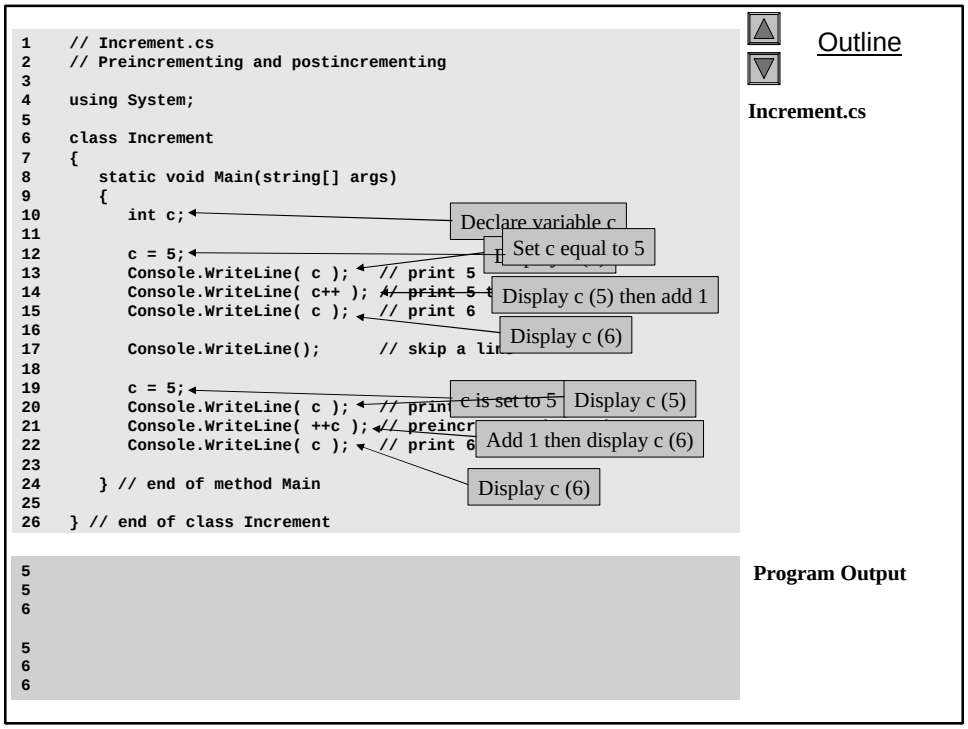
Assignment operator	Sample expression	Explanation
`+=`	`c += 7`	`c = c + 7`
`-=`	`d -= 4`	`d = d - 4`
`*=`	`e *= 5`	`e = e * 5`
`/=`	`f /= 3`	`f = f / 3`
`%=`	`g %= 2`	`g = g % 2`

16

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.

The increment and decrement operators.



Backup Slides

Precedence and Associativity

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**