Microsoft® Official Course

Module 1

Review of Visual C# Syntax

Microsoft®

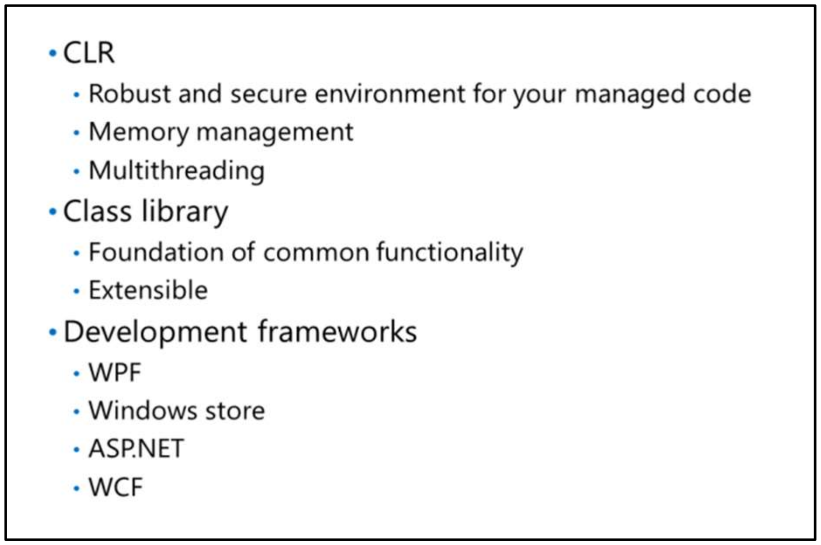
Module Overview

1. Overview of Writing Application by Using Visual C#
2. Data Types, Operators, and Expressions
3. Visual C# Programming Language Constructs

Lesson 1:
Overview of Writing Application by Using Visual C#

What Is the .NET Framework?

- The .NET Framework 4.5 provides a comprehensive development platform that offers a fast and efficient way to build applications and services.
- By using Visual Studio 2012, you can use the .NET Framework 4.5 to create a wide range of solutions that operate across a broad range of computing devices.
- The .NET Framework 4.5 provides three principal elements:
  - The Common Language Runtime (CLR).
  - The .NET Framework class library.
  - A collection of development frameworks.



- CLR
  - Robust and secure environment for your managed code
  - Memory management
  - Multithreading
- Class library
  - Foundation of common functionality
  - Extensible
- Development frameworks
  - WPF
  - Windows store
  - ASP.NET
  - WCF

What Is the .NET Framework?

- **The Common Language Runtime**
  - manages the execution of code and simplifies the development process by providing a robust and highly secure execution environment that includes:
    - Memory management.
    - Transactions.
    - Multithreading.

- **The .NET Framework Class Library**
  - a library of reusable classes that you can use to build applications.
  - The classes provide a foundation of common functionality and constructs that help to simplify application development by, in part, eliminating the need to constantly reinvent logic.
    - For example, the System.IO.File class contains functionality that enables you to manipulate files on the Windows file system.
    - In addition to using the classes in the .NET Framework class library, you can extend these classes by creating your own libraries of classes.

- **Development Frameworks**
  - use to build common application types, including:
    - Desktop client applications, by using Windows Presentation Foundation (WPF).
    - Windows 8 desktop applications, by using XAML.
    - Server-side web applications, by using Active Server Pages (ASP.NET) Web Forms or ASP.NET MVC.
    - Service-oriented web applications, by using Windows Communication Foundation (WCF).
    - Long-running applications, by using Windows services.
  - Each framework provides the necessary components and infrastructure to get you started.

Key Features of Visual Studio 2012

- Visual Studio 2012 provides a single development environment that enables you to rapidly design, implement, build, test, and deploy various types of applications and components by using a range of programming languages.

- Some of the key features of Visual Studio 2012 are
  - Intuitive IDE
  - Rapid application development
    - design view (for graphical components), code editor, wizards
  - Server and data access
    - Server Explorer enables you to log on to servers and explore their databases and system services.
  - Internet Information Services (IIS) Express
    - lightweight version of IIS as the default web server for debugging your web applications
  - Debugging features
    - debugger that enables you to step through local or remote code, pause at breakpoints, and follow execution paths
  - Error handling
    - **Error List** window
  - Help and documentation
    - code snippets, and the integrated help system, which contains documentation and samples.

Templates in Visual Studio 2012

- Visual Studio 2012 supports the development of different types of applications such as Windows-based client applications, web-based applications, services, and libraries.

- To help you get started, Visual Studio 2012 provides application templates that provide a structure for the different types of applications.

- These templates:
  - Provide starter code that you can build on to quickly create functioning applications.
  - Include supporting components and controls that are relevant to the project type.
  - Configure the Visual Studio 2012 IDE to the type of application that you are developing.
  - Add references to any initial assemblies that this type of application usually requires.

Templates in Visual Studio 2012

- Console Application
    - to develop an application that runs in a command-line interface.
    - is considered lightweight because there is no graphical user interface.
- Windows Forms Application
    - to build a graphical Windows Forms application
- WPF Application ←Windows Presentation Foundation
    - to build a rich graphical Windows application.
    - enables you to create Windows applications, with much more control over user interface design.
- Windows Store
    - to build a rich graphical application targeted at the Windows 8 operating system.
- Class Library
    - to build a .dll assembly.
- ASP.NET Web Application
    - to create a server-side, compiled ASP.NET web application
- ASP.NET MVC 4 Application
    - to create a Model-View-Controller (MVC) Web application.
    - ASP.NET MVC web app. architecture helps you separate the presentation layer, business logic layer, and data access layer.
- WCF Service Application ←Windows Communication Foundation
    - to build Service Orientated Architecture (SOA) services.

Creating a .NET Framework Application

1. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, choose a template, location, name, and then click **OK**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
   class Program
   {
       static void Main(string[] args) { }
   }
}
```

Overview of XAML

- Extensible Application Markup Language (XAML) is an XML-based language that you can use to define your .NET application UIs.
  - By declaring your UI in XAML as opposed to writing it in code makes your UI more portable and separates your UI from your application logic.

- XAML uses **elements** and **attributes** to define controls and their properties in XML syntax.
  - you can use the **toolbox** and **properties pane** in Visual Studio to visually create the UI, you can use the **XAML pane** to declaratively create the UI, you can use Microsoft Expression Blend, or you can use other third-party tools.
  - Using the XAML pane gives you finer grained control than dragging controls from the toolbox to the window.

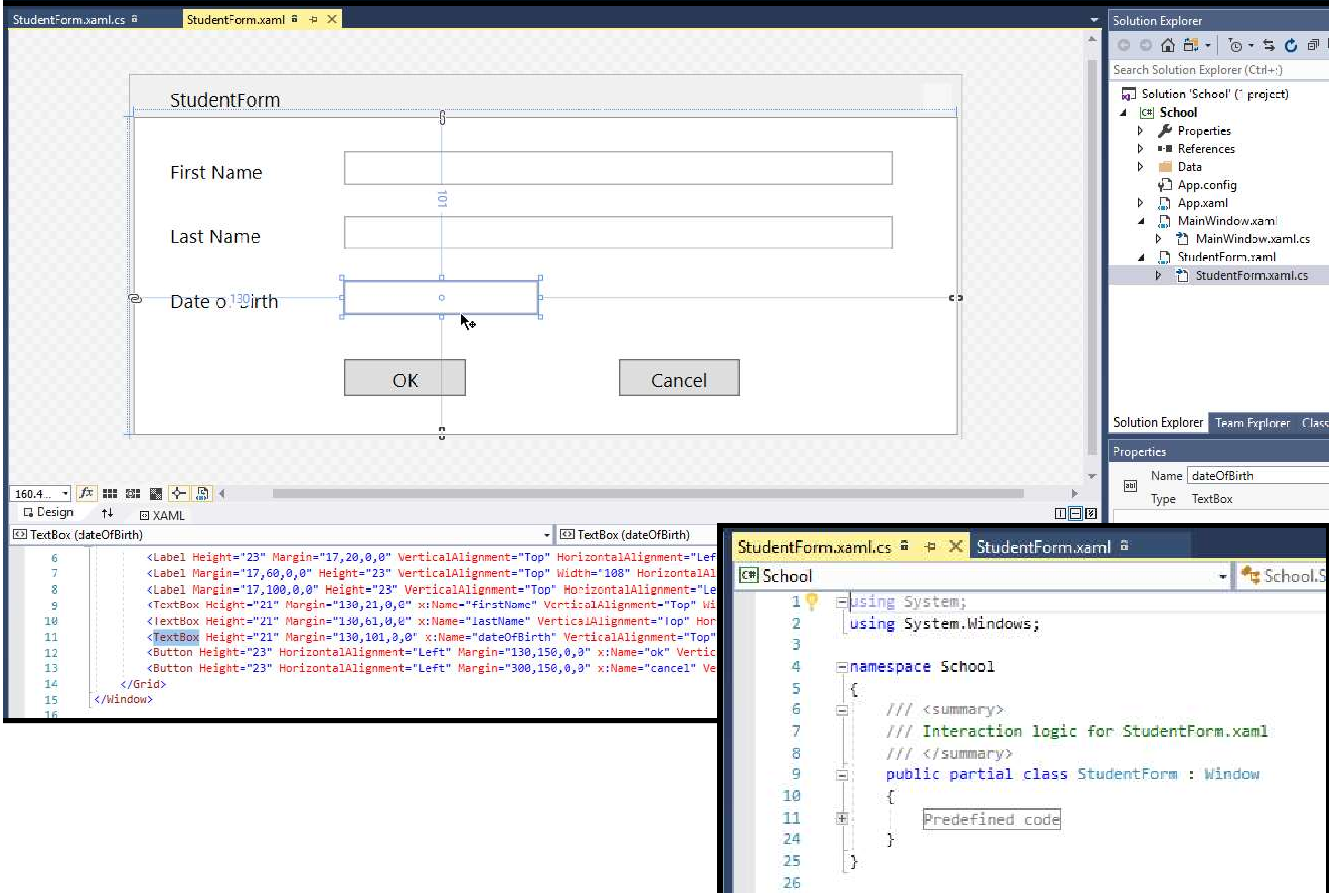
- The following example shows the XAML declaration for a **label**, **textbox**, and **button**:
  - **Elements**: Label, TextBox, Button,
  - **Attributes**: Width, HorizontalAlignment, Margin
  - We'll see more in module 9: "Designing the User Interface for a Graphical Application."

```
<Label Content="Name:" HorizontalAlignment="Left" Margin="72,43,0,0" VerticalAlignment="Top" />

<TextBox HorizontalAlignment="Left" Height="23" Margin="141,43,0,0" Text=""
VerticalAlignment="Top" Width="120" />

<Button Content="Click Me!" HorizontalAlignment="Left" Margin="119,84,0,0"
VerticalAlignment="Top" Width="75" />
```

Overview of XAML



```
        <Label Height="23" Margin="17,20,0,0" VerticalAlignment="Top" HorizontalAlignment="Lef
        <Label Margin="17,60,0,0" Height="23" VerticalAlignment="Top" Width="108" HorizontalAl
        <Label Margin="17,100,0,0" Height="23" VerticalAlignment="Top" HorizontalAlignment="Le
        <TextBox Height="21" Margin="130,21,0,0" x:Name="firstName" VerticalAlignment="Top" Wi
        <TextBox Height="21" Margin="130,61,0,0" x:Name="lastName" VerticalAlignment="Top" Hor
        <TextBox Height="21" Margin="130,101,0,0" x:Name="dateOfBirth" VerticalAlignment="Top"
        <Button Height="23" HorizontalAlignment="Left" Margin="130,150,0,0" x:Name="ok" Vertic
        <Button Height="23" HorizontalAlignment="Left" Margin="300,150,0,0" x:Name="cancel" Ve
    </Grid>
</Window>
```

```
using System;
using System.Windows;

namespace School
{
    /// <summary>
    /// Interaction logic for StudentForm.xaml
    /// </summary>
    public partial class StudentForm : Window
    {
        Predefined code
    }
}
```

Lesson 2:

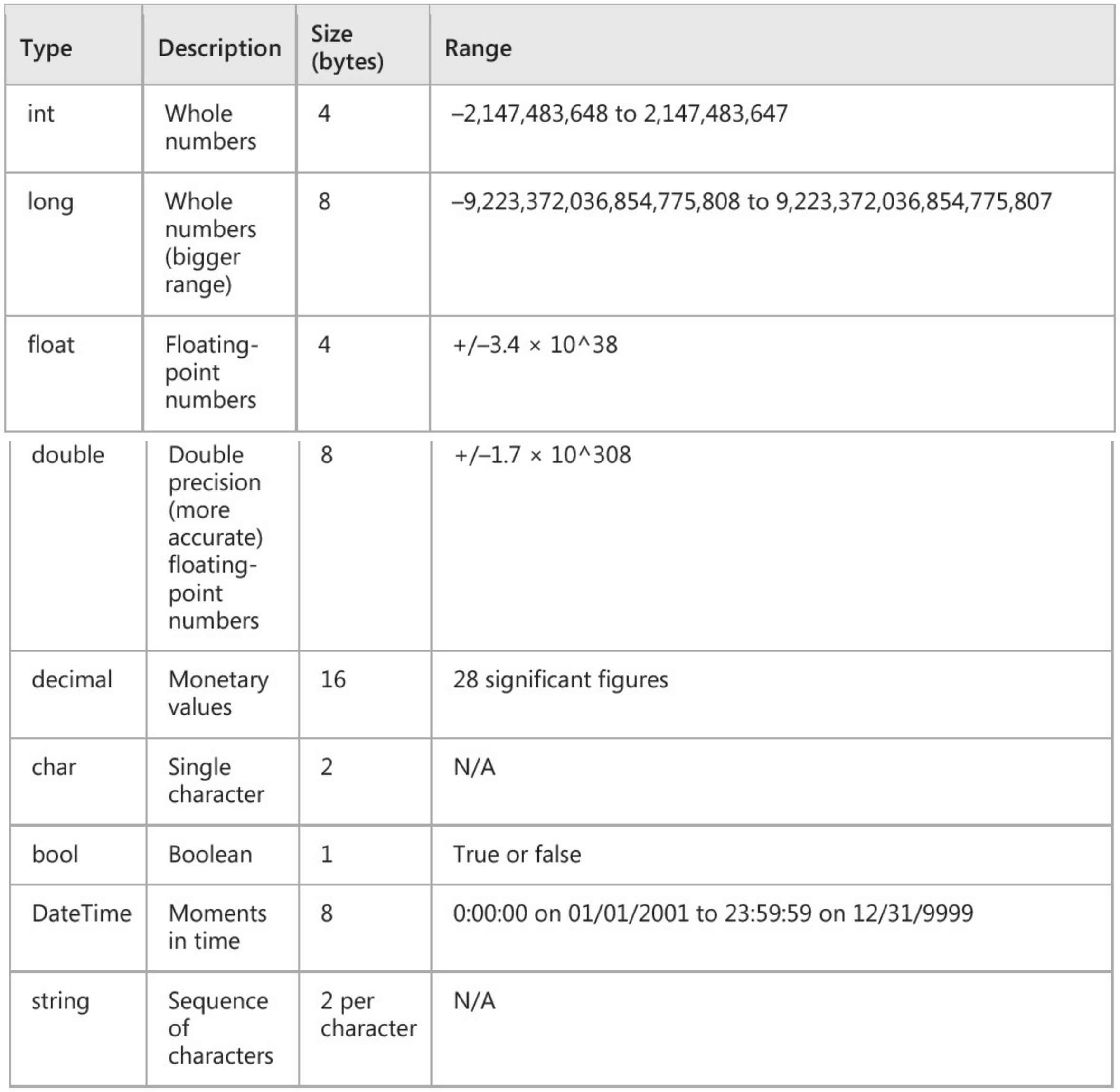
Data Types, Operators, and Expressions

What are Data Types?

- A **variable** holds data of a specific type. When you **declare** a variable to store data in an application, you need to choose an appropriate data type for that data.
- Visual C# is a type-safe language, which means that the compiler guarantees that values stored in variables are always of the appropriate type.

  - **int** – whole numbers
  - **long** – whole numbers (bigger range)
  - **float** – floating-point numbers
  - **double** - double precision
  - **decimal** - monetary values
  - **char** - single character
  - **bool** - Boolean
  - **DateTime** - moments in time
  - **string** - sequence of characters

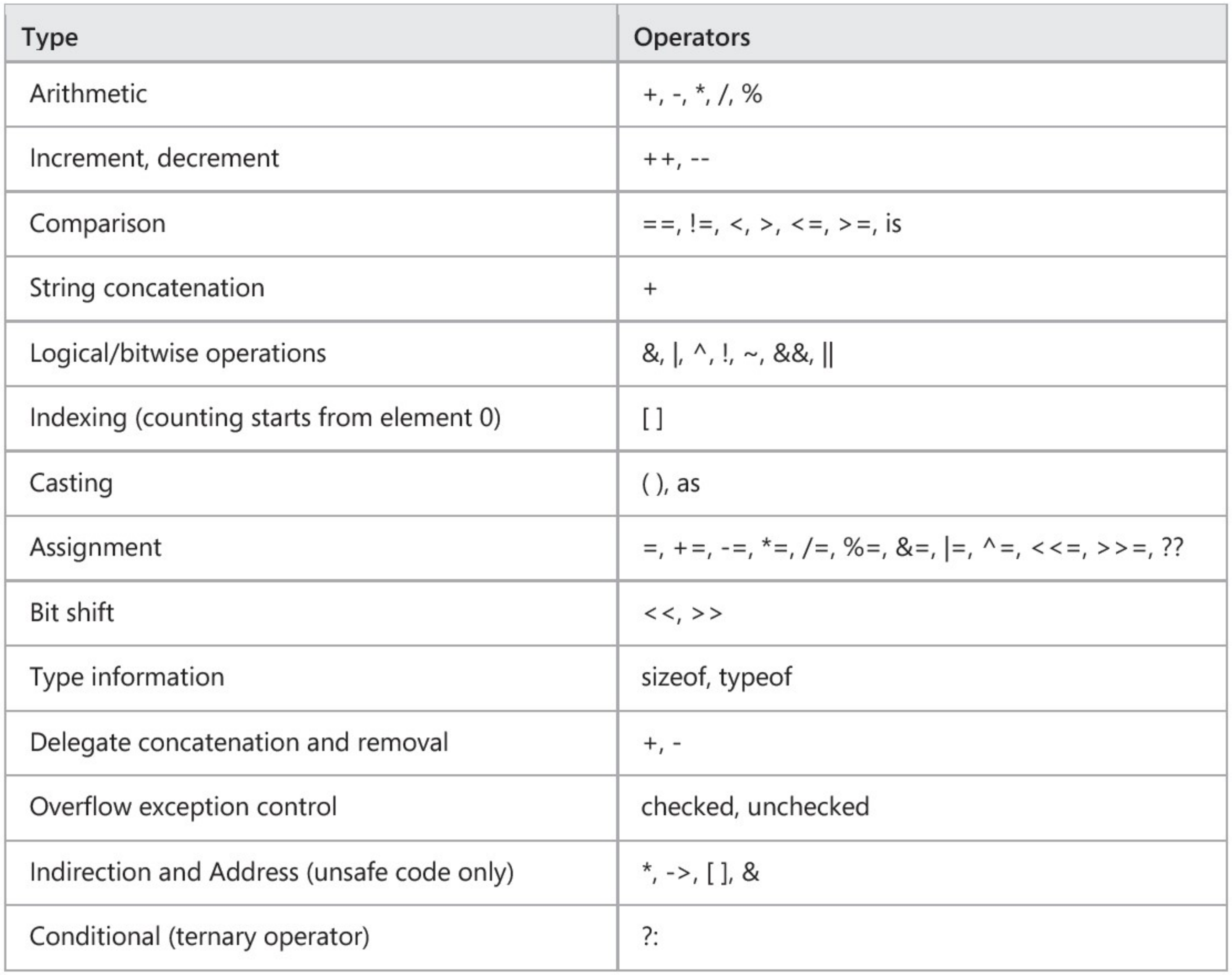
Commonly Used Data Types



Type	Description	Size (bytes)	Range
int	Whole numbers	4	–2,147,483,648 to 2,147,483,647
long	Whole numbers (bigger range)	8	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Floating-point numbers	4	+/–3.4 × 10^38
double	Double precision (more accurate) floating-point numbers	8	+/–1.7 × 10^308
decimal	Monetary values	16	28 significant figures
char	Single character	2	N/A
bool	Boolean	1	True or false
DateTime	Moments in time	8	0:00:00 on 01/01/2001 to 23:59:59 on 12/31/9999
string	Sequence of characters	2 per character	N/A

Expressions and Operators in Visual C#

- Operators fall into the following three categories:
  - Unary
  - Binary
  - Ternary



Type	Operators
Arithmetic	+, -, *, /, %
Increment, decrement	++, --
Comparison	==, !=, <, >, <=, >=, is
String concatenation	+
Logical/bitwise operations	&, |, ^, !, ~, &&, ||
Indexing (counting starts from element 0)	[ ]
Casting	( ), as
Assignment	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, ??
Bit shift	<<, >>
Type information	sizeof, typeof
Delegate concatenation and removal	+, -
Overflow exception control	checked, unchecked
Indirection and Address (unsafe code only)	*, ->, [ ], &
Conditional (ternary operator)	?:

Expressions and Operators in Visual C#

Example expressions:

- + operator

```
a + 1
```

- / operator

```
7 / 2                    vs                    7.0/2
```

- + and – operators

```
a + b - 2
```

- + operator (string concatenation)

```
"ApplicationName: " + appName.ToString()
```

Declaring and Assigning Variables

- Before you can use a variable, you must declare it so that you can specify its name and characteristics.
  - When you declare a variable, you reserve some storage space for that variable in memory and the type of data that it will hold.

- The name of a variable is referred to as an identifier. Visual C# has specific rules concerning the identifiers that you can use:
  - An identifier can only contain letters, digits, and underscore characters.
  - An identifier must start with a letter or an underscore.
  - An identifier for a variable should not be one of the keywords that Visual C# reserves for its own use

- Visual C# is case sensitive.
  - When declaring variables you should use meaningful names for your variables

- Visual C# does not allow you to use an unassigned variable.
  - You must assign a value to a variable before you can use it; otherwise, your program might not compile.

Declaring and Assigning Variables

- Declaring variables:

```
int price;
// OR
int price, tax;
```

- Assigning variables:

```
price = 10;
// OR
int price = 10;
```

- Implicitly typed variables:

```
var price = 20;
```

- Instantiating object variables by using the **new** operator

```
ServiceConfiguration config = new ServiceConfiguration();
```

  - The **new** operator causes the CLR to allocate memory for your object, and it then invokes a constructor to initialize the fields in that object.

Accessing Type Members

- Invoke instance members (dot notation)

```
<instanceName>.<memberName>
```

- Example (method vs property access ...):

```
var config = new ServiceConfiguration();

// Invoke the LoadConfiguration method.
config.LoadConfiguration();

// Get the value from the ApplicationName property.
var applicationName = config.ApplicationName;

// Set the .DatabaseServerName property.
config.DatabaseServerName = "78.45.81.23";

// Invoke the SaveConfiguration method.
config.SaveConfiguration();
```

Casting Between Data Types

- The process of converting a value of one data type to another type is called type conversion or casting.

- Implicit conversion (when a value is converted automatically):

```
int a = 4;
long b = 5;
b = a;              //implicit conversion – in here it is widening
```

- Explicit conversion:

```
int a = (int) b;    //explicit conversion
```

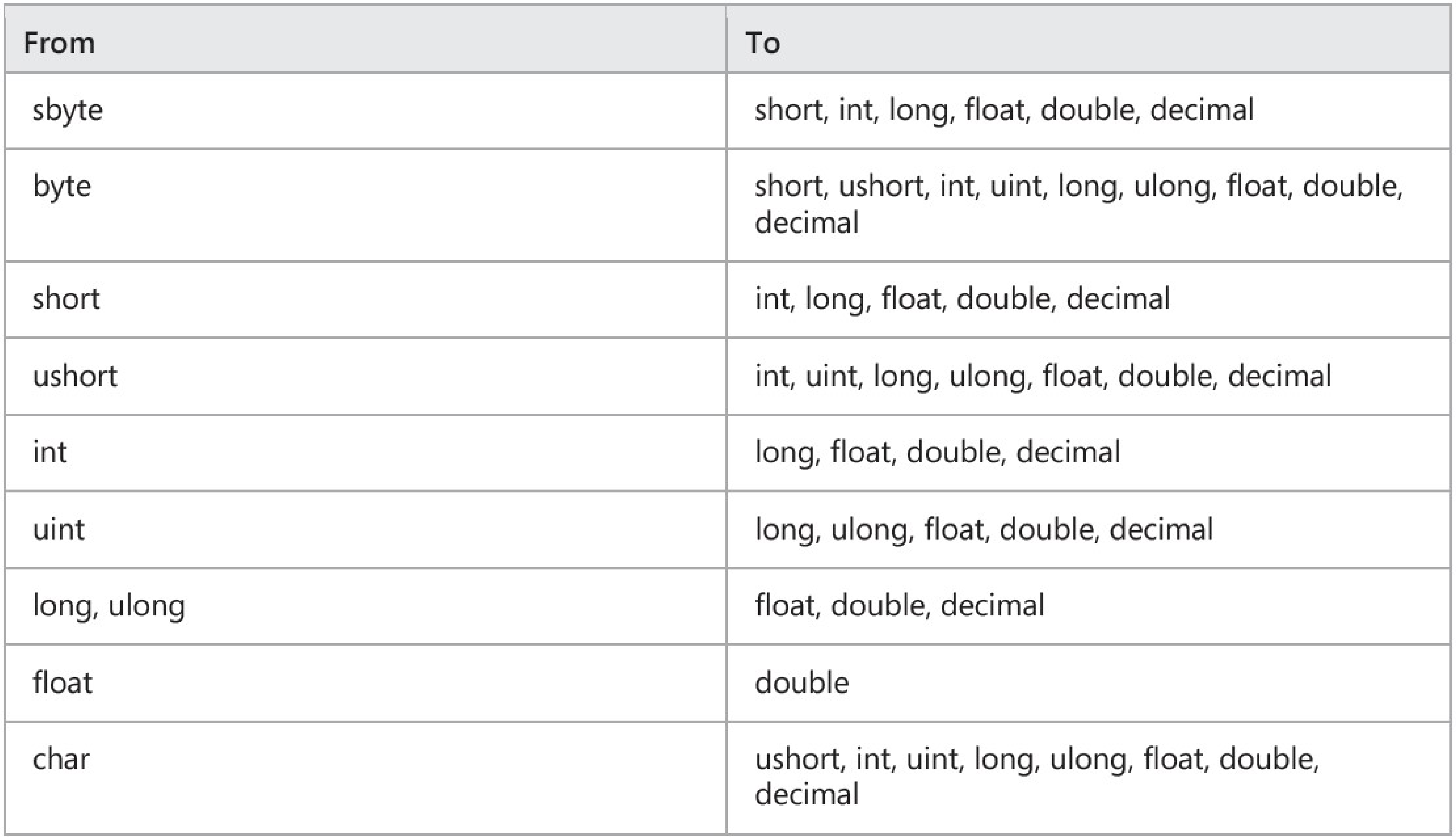
- System.Convert conversion:

```
string possibleInt = "1234";
int count = Convert.ToInt32(possibleInt);
```

  - **System.Convert** class provides methods that can convert a base data type to another base data type. **int.TryParse**(numberString, out number) … tries before

Casting Between Data Types

- table that shows the implicit type conversions that are supported in Visual C#.



From	To
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long, ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal

Manipulating Strings

- Concatenating multiple strings in Visual C# is simple to achieve by using the + operator.
  - For example “Saint”+”Martin” will give you “SaintMartin”

- However, this is considered bad coding practice because strings are immutable.
  - This means that every time you concatenate a string, you create a new string in memory and the old string is discarded.

- **StringBuilder** class enables you to build a string dynamically and much more efficiently.

Manipulating Strings

- Concatenating strings

```
StringBuilder address = new StringBuilder();
address.Append("23");
address.Append(", Main Street");
address.Append(", Buffalo");
string concatenatedAddress = address.ToString();
```

- Validating strings (**Regular expressions** … mechanism to validate input)

```
var textToTest = "hell0 w0rld";
var regularExpression = "\\d";          //\d matches any numeric data

var result = Regex.IsMatch(textToTest, regularExpression, RegexOptions.None);

if (result)
{
   // Text matched expression.
}
```

Lesson 3:
Visual C# Programming Language Constructs

Implementing Conditional Logic

- if statements (if … if vs if … else if …)

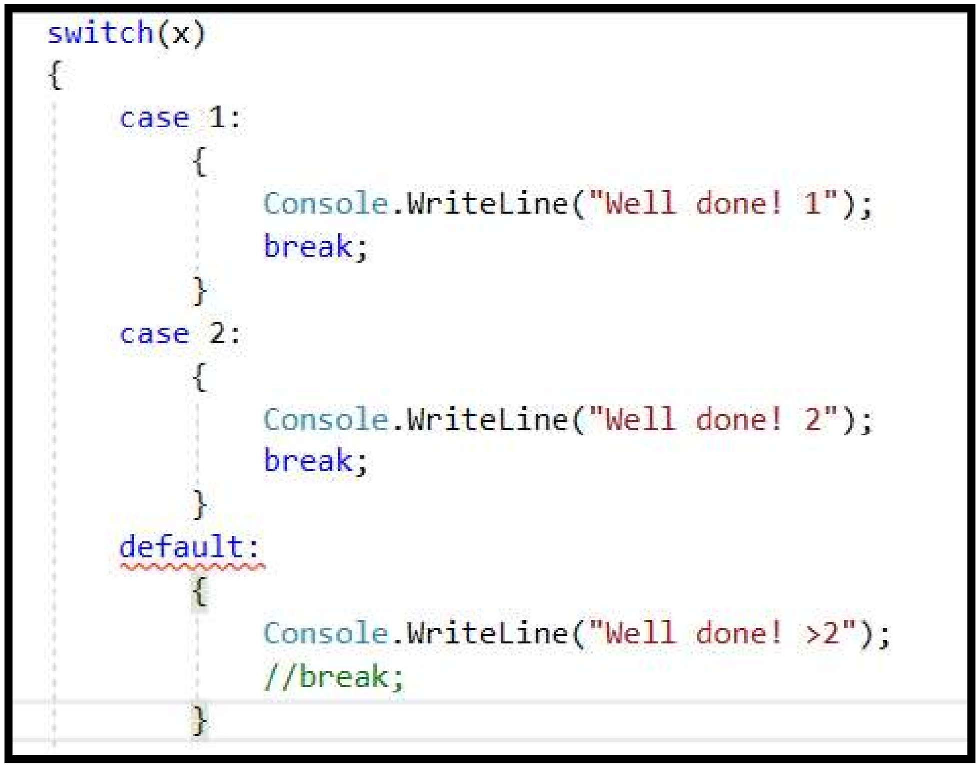
```
if (response == "connection_failed") {. . .}
else if (response == "connection_error") {. . .}
else { }
```

- select statements

```
switch (response)
{
   case "connection_failed":
      . . .
      break;
   case "connection_success":
      . . .
      break;
   default:
      . . .
      break;
}
```

Implementing Conditional Logic

- In each case statement, notice the **break** keyword.
  - This causes control to jump to the end of the switch after processing the block of code.
  - If you omit the break keyword, your code will not compile.
    - This is different than in other languages!



```
switch(x)
{
    case 1:
        {
            Console.WriteLine("Well done! 1");
            break;
        }
    case 2:
        {
            Console.WriteLine("Well done! 2");
            break;
        }
    default:
        {
            Console.WriteLine("Well done! >2");
            //break;
        }
```

Implementing Iteration Logic

- for loop ←repeat until a condition becomes false

```
for (int i = 0 ; i < 10; i++) { ... }
```

- foreach loop

```
string[] names = new string[10];
foreach (string name in names) { ... }
```

- while loop ←repeat while a condition remains true

```
bool dataToEnter = CheckIfUserWantsToEnterData();
while (dataToEnter)
{
    ...
    dataToEnter = CheckIfUserHasMoreData();
}
```

- do loop ←similar to while, but the body executes at least once

```
.. do
{
    ...
    moreDataToEnter = CheckIfUserHasMoreData();
} while (moreDataToEnter);
```

Creating and Using Arrays

- An array is a set of objects that are grouped together and managed as a unit.
- You can think of an array as a sequence of elements, all of which are the same type.
- Arrays in Visual C#:
  - Every element in the array contains a value.
  - Arrays are zero-indexed, that is, the first item in the array is element 0.
  - The size of an array is the total number of elements that it can contain.
  - Arrays can be single-dimensional, multidimensional, or jagged.
  - The rank of an array is the number of dimensions in the array.
- **Note**: If you need to manipulate a set of unlike objects or value types, consider using one of the collection types that are defined in the **System.Collections** namespace

Creating and Using Arrays

- C# supports:
  - Single-dimensional arrays ← int[] myArr1D = new int[20];
  - Multidimensional arrays ← int[,] myArr2D = new int[10,20];
  - Jagged arrays ← see next slide ...

- Creating an array

```
int[] arrayName = new int[10];
```

- Accessing data in an array:
  - By index

```
int result = arrayName[2];  //← this is the third element
```

  - In a loop

```
for (int i = 0; i < arrayName.Length; i++) //←use Length property
{
    int result = arrayName[i];
}
```

Creating and Using Arrays

- Declaring an array brings the array into scope, but does not actually allocate any memory for it.
  - int[] myArr;
  - int[] myArr2 = new int[2];

- The CLR physically creates the array when you use the **new** keyword

- A jagged array is simply an array of arrays, and the size of each array can vary. Jagged arrays are useful for modeling **sparse** data structures.
  - Note that you must specify the size of the first array, but you must not specify the size of the arrays that are contained within this array. You allocate memory to each array within a jagged array separately, by using the **new** keyword.

```
int[][] jaggedArray = new int[10][];
jaggedArray[0] = new Type[5]; // Can specify different sizes.
jaggedArray[1] = new Type[7];
...
jaggedArray[9] = new Type[21];
```

- If you attempt to access an element outside this range, the CLR throws an **IndexOutOfRangeException** exception.

Referencing Namespaces

- Use namespaces to <u>organize classes</u> into a logically related hierarchy
- .NET Class Library includes:

System.Windows	Provides the classes that are useful for building WPF applications.
System.IO	Provides classes for reading and writing data to files.
System.Data	Provides classes for data access.
System.Web	Provides classes that are useful for building web applications.

- Define your own namespaces:

```
namespace FourthCoffee.Console
{
   class Program {. . .}
}
```

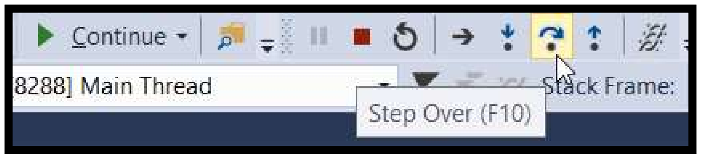
- Use namespaces: //do console application example if time …
  - Add reference to containing library
  - Add **using** directive to code file

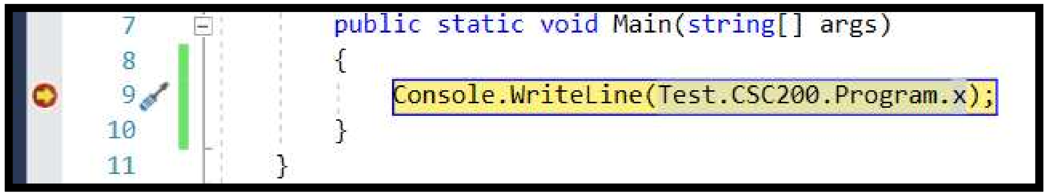
Using Breakpoints in Visual Studio 2012

- Breakpoints enable you to view and modify the contents of variables:
  - Immediate Window
  - Autos, Locals, and Watch panes

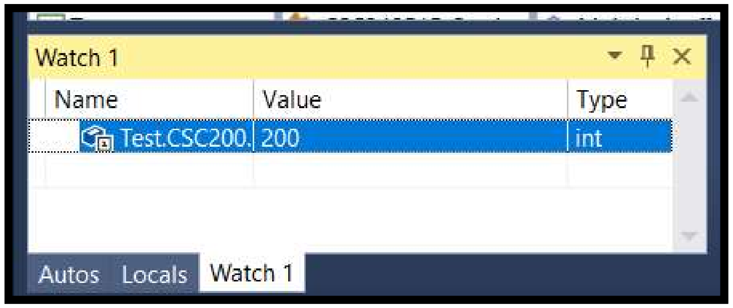
- Debug menu and toolbar functions enable you to:
  - Start and stop debugging
  - Enter break mode
  - Restart the application
  - Step through code

- Example:...





```
    public static void Main(string[] args)
    {
        Console.WriteLine(Test.CSC200.Program.x);
    }
}
```



Watch 1

Name	Value	Type
Test.CSC200.	200	int

Module Review and Takeaways

- **Question:** What Visual Studio template would you use to create a .dll?
  - (   )Option 1: Console application
  - (   )Option 2: Windows Forms application
  - (   )Option 3: WPF application
  - (   )Option 4: Class library
  - (   )Option 5: WCF Service application

- **Question:** Given the following for loop statement, what is the value of the count variable once the loop has finished executing?

```
var count = 0;
for (int i = 5; i < 12; i++)
{
  count++;
}
```

  - (   )Option 1: 3
  - (   )Option 2: 5
  - (   )Option 3: 7
  - (   )Option 4: 9
  - (   )Option 5: 11