

FUNDAMENTALS OF PROGRAMMING WITH C#

VISUAL STUDIO

Liu Fan

isslf@nus.edu.sg

Total: 53

Learning Objectives

- Understand the purpose of Integrated Development environment (IDE) and the role of Visual Studio in C# application development
- Write and execute a C# console application using Visual Studio
- Perform step-by-step debugging in Visual Studio

Agenda

- Integrated Development Environment
- Elements of Visual Studio
- Creating a Solution and Project
- Write and Execute a simple C# Program
- Running your application
- Debugging your application

Integrated Development Environment (IDE)

- Visual Studio is an IDE
 - Editor
 - Visual Development Tools
 - Compiler/ Builder
 - Debugger
- Purpose
 - Improve developer's productivity by providing various development related features in a single tools
 - Scaffold creation of different types of applications by choosing an appropriate project type
 - Help in packaging the application beyond simple compilation

Visual Studio Screen



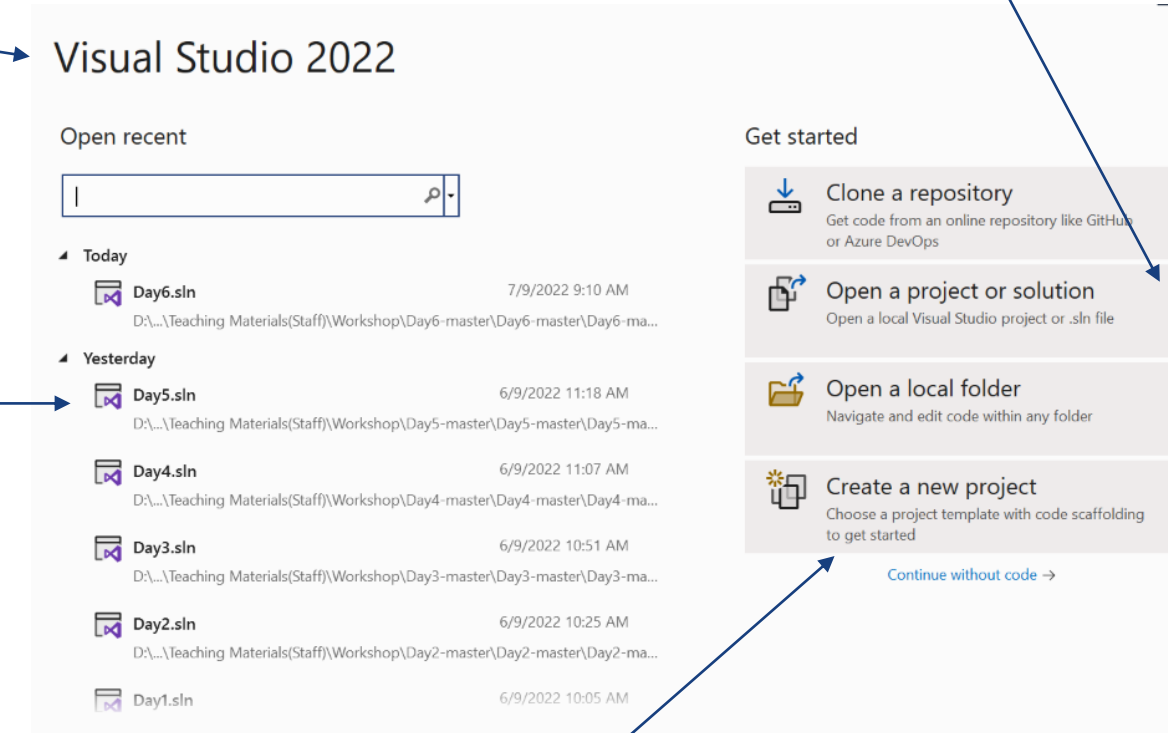
NUS
National University
of Singapore



Shortcut to open a project

Main Window
displaying
Start Page

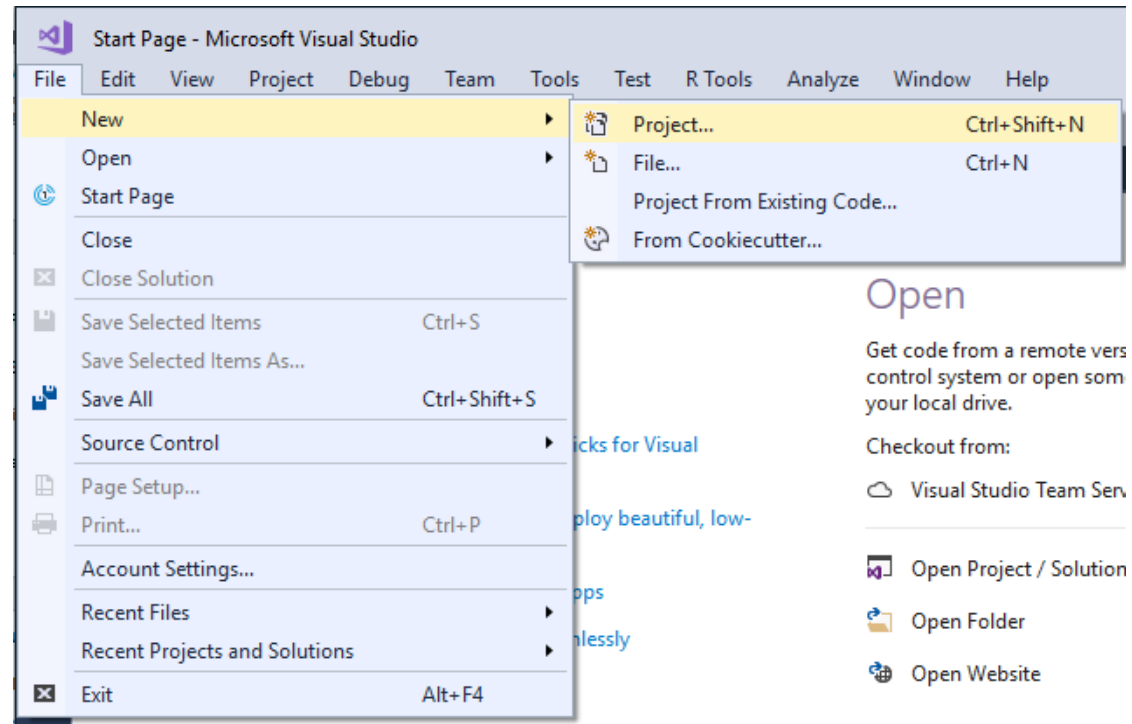
Shortcut to
your recent
projects
(very useful)



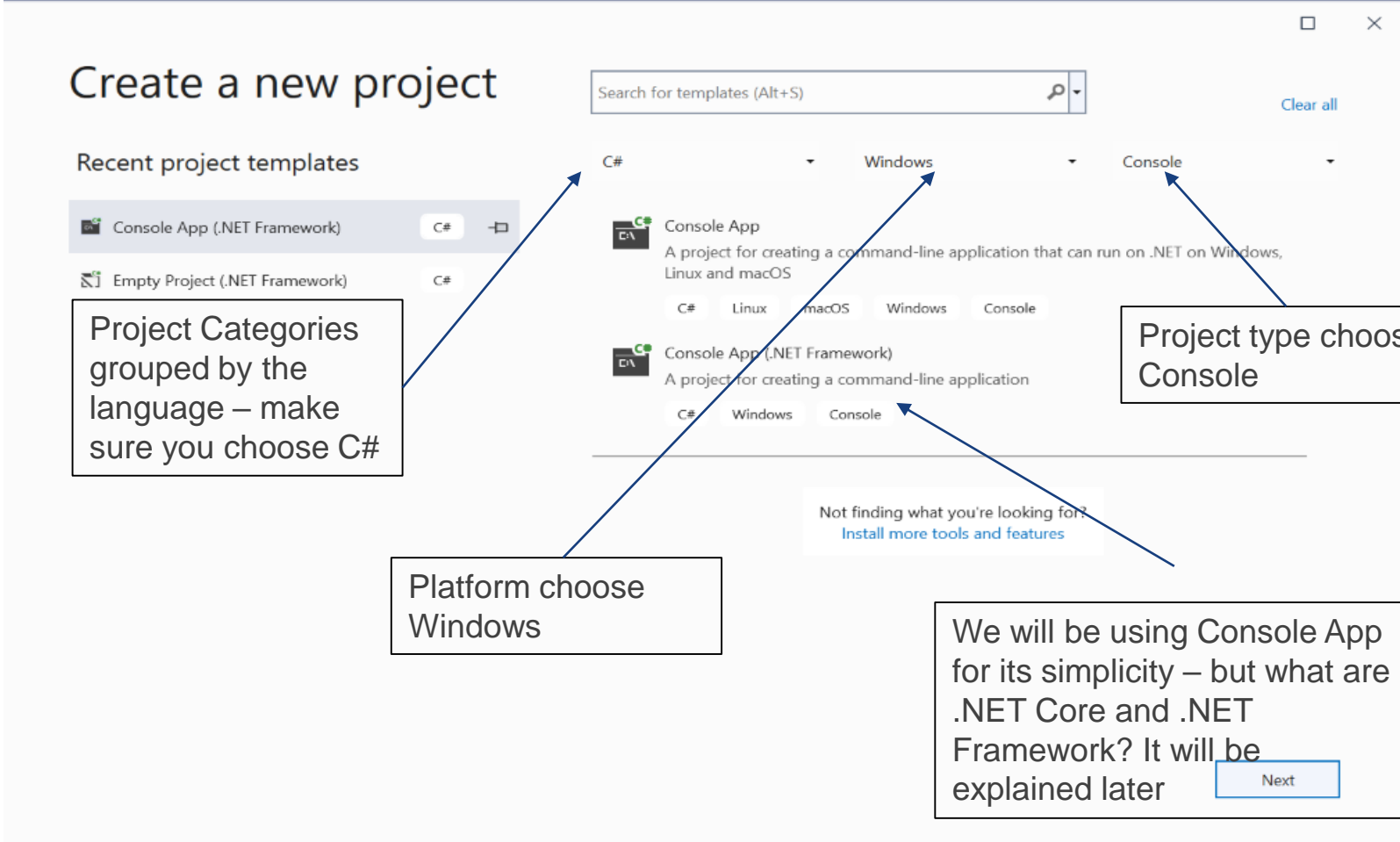
Shortcut for creating
a new project

Creating your first VS solution

- Multiple ways to create a new solution:
 - use the shortcut on the Start Page,
 - click on File > New > Project
 - Press Ctrl+Shift+N



Creating your first VS solution



Create a new project

Search for templates (Alt+S) Clear all

Recent project templates

- Console App (.NET Framework) C#
- Empty Project (.NET Framework) C#

Project Categories grouped by the language – make sure you choose C#

C# Windows Console

Console App
A project for creating a command-line application that can run on .NET on Windows, Linux and macOS
C# Linux macOS Windows Console

Console App (.NET Framework)
A project for creating a command-line application
C# Windows Console

Platform choose Windows

Project type choose Console

Not finding what you're looking for?
[Install more tools and features](#)

We will be using Console App for its simplicity – but what are .NET Core and .NET Framework? It will be explained later

Next

Creating your first VS solution

Configure your new project

Console App (.NET Framework)

Console

C#

Windows

Project name

ConsoleApp4

Location

C:\Users\isslf\source\repos

Solution name ⓘ

ConsoleApp4

☐ Place solution and project in the same directory

Framework

.NET Framework 4.7.2

Framework version –
default should be OK

Name of the solution and project – the
difference will be explain in other slides

Parent folder for the solution
(each solution will have its own
folder within this parent folder)

Back

Create

Solution and Project

- A VS solution can contains multiple projects
- Example:
 - A company wants to have a utility that has two kinds of interface: windows desktop application and console application
 - Since the functionality are similar, we want both application to share as much implementation as possible (this is called as reusability or code reuse)
 - So we can have a VS solution with 3 projects:
 - Windows Forms App – for the windows desktop application
 - Console App – for the console application
 - Class Library – for the reusable functionality shared by both applications
- In VS, you create a solution by creating the first project in the solution.
- Solutions can also contain files that aren't connected to any specific projects.
- Projects hold the items needed to build your app in Visual Studio, such as source code files, bitmaps, icons, and component and service references.

Writing your codes



NUS
National University
of Singapore

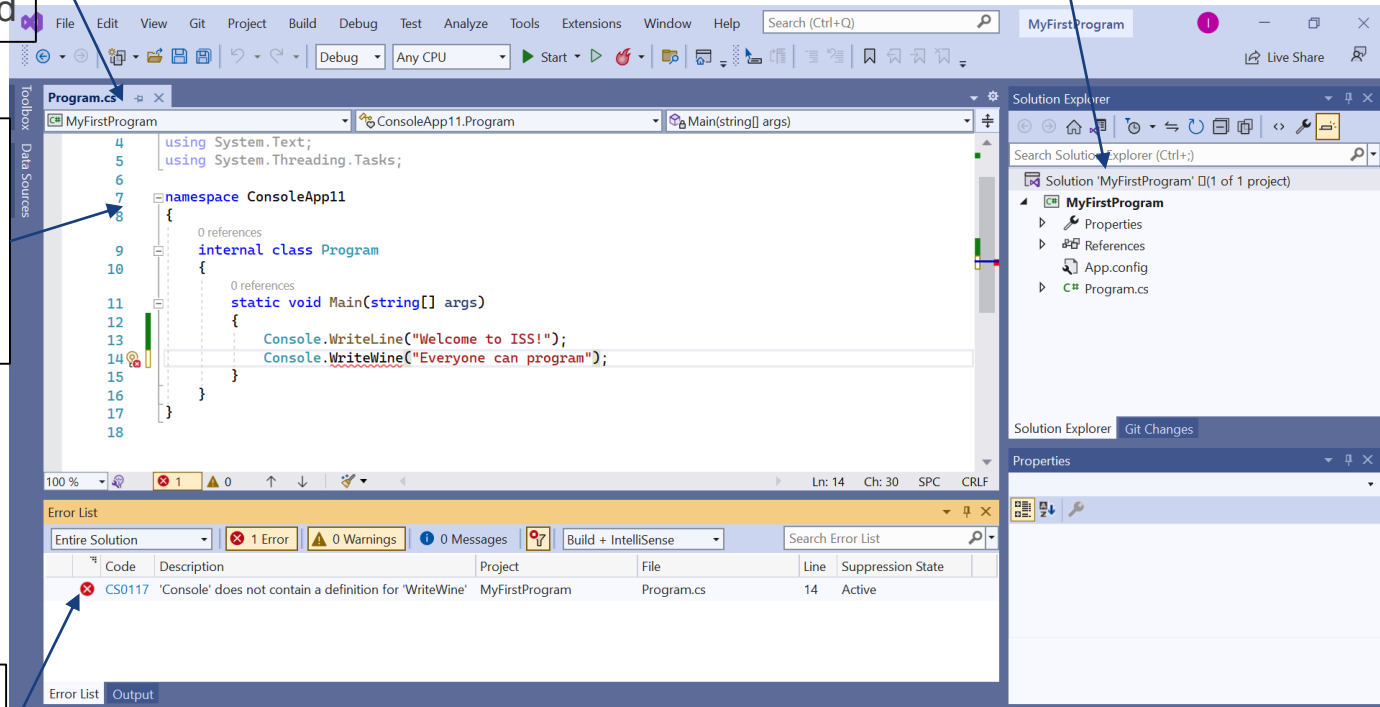


Program.cs is automatically created for you based on the template. The file is safe to be deleted

Solution Explorer – browse through the projects, folders and files in your solution

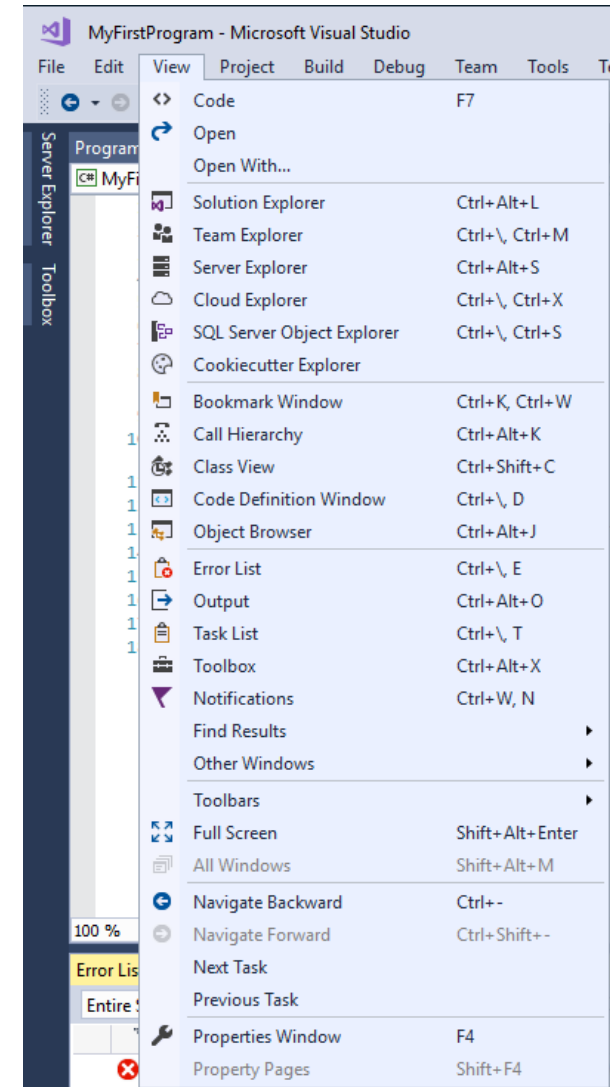
Main Editor Window – if you open multiple files, they will be shown as tabs inside this window

List of syntax errors – click on the row to go to the location shown



Windows in VS

- Each of the windows have its own purpose
- You can close or hide windows that you don't use
- You can re-open the windows using the View menu items
 - It's useful to pay attention to the names of windows so that you can re-open them



First C# Program

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */
    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

First C# Program - Main

FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

Program execution starts here.

- Main method is the entry point of a C# application
 - .NET will look for this method within a class and execute the method when application starts
 - Main must be static
 - This means that the function is called without creating an instance of the class. It is initialized automatically the first time it is accessed.
 - Main can return void or int
 - Void does not return values, int returns an integer value.
 - No difference for our purpose in this module
 - Can be declared with or without string[] parameter.
 - The function receives or not receives parameters.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/main-and-command-args/>

First C# Program - Main

- How to declare a Main() function

Method 1. The function does not return values and does not receive parameters. The general form of the `Main()` function in this case is as follows:

```
static void Main()  
{  
    // actions, operators // ...  
}
```

Method 2. The function returns an integer value and does not receive parameters. In this case, the general form of the function is as follows

```
static int Main()  
{  
    // actions, operators // ...  
    return value;  
}
```

Here *value* is some integer value, which is the result of returning from the program. Other processes running the current program may use this result. For example, if the `Main()` function returns -1, then this may indicate an internal error. And conversely, if the function returns 0, then this may mean the correct execution of the program.

Method 3. The function does not return a value, but receives parameters. The parameters of the `Main()` function can be an array of strings. In this case, the general form of the function is as follows:

```
static void Main(string[] args)  
{  
    // actions, operators // ...  
}
```

First C# Program - Statements



FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

We write our instructions as statements within the methods

- A method contains a series of statements (instructions) wrapped in a pair of curly braces { }
- The instruction will be executed one-by-one from top to bottom.
- Every statement ends with a semi-colon ;
 - C# will raise an error if you forget your semi-colon

First C# Program - Class

FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

Methods have to be written inside a class.

- A class is a template for an object (cover in next module)
- A class can contain multiple methods
- The contents of a class must be wrapped inside its curly braces

First C# Program - Comments



FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

Comments should be added to make your program more readable

- Comments are useful for improving readability and documentation
- Single comments are placed after **//** symbol
- Multi-line comments are enclosed between **/*** and ***/**

First C# Program - Namespace

FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

Namespace is useful to avoid classes can be referred to uniquely

- To prevent class name duplication, we can put classes within namespaces
- Namespaces are optional
- Namespaces can be nested

Namespace	Class	Full class name
	Program	Program
ISS	Program	ISS.Program
NUS.ISS	Program	NUS.ISS.Program

First C# Program - Using

FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

We want to be able to use classes in **System** namespace without having to qualify or specify the namespace

e.g. we can refer to **System.Console** class as just **Console**

- The scope of a using directive is limited to the file in which it appears.
- The using directive can appear:
 - Using directive can appear at the beginning of a source file, before any namespace or type definition
 - In any namespace but before any namespace or types declared in this namespace

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-directive>

First C# Program - Using

The using directive has three uses:

- To allow the use of types in a namespace so that you do not have to qualify the use of a type in that namespace:

```
using System.Text;
```

- To allow you to access static members and nested types of a type without having to qualify the access with the type name.

```
using static System.Math;
```

- To create an alias for a namespace or a type. This is called a *using alias directive*.

```
using Project = PC.MyCompany.Project;
```

In any *using directive*, the fully-qualified namespace or type must be used regardless of the using directives that come before it.

First C# Program

FirstProgram.cs

```
using System;

namespace FirstProject
{
    /* This program when executed will print
       Welcome to ISS and
       Everyone can program. */

    // An example program for FOPCS
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to ISS!");
            Console.WriteLine("Everyone can program");
        }
    }
}
```

We call a `WriteLine` method to print a text to the screen and then terminate the line

Console.WriteLine Method

Namespace: `System`

Assemblies: `System.Console.dll, mscorlib.dll, netstandard.dll`

Writes the specified data, followed by the current line terminator, to the standard output stream.

Overloads

<code>WriteLine(String, Object, Object)</code>	Writes the text representation of the specified objects, followed by the current standard output stream using the specified format information.
<code>WriteLine(String)</code>	Writes the specified string value, followed by the current line terminator, to the standard output stream.
<code>WriteLine(Char[], Int32, Int32)</code>	Writes the specified subarray of Unicode characters, followed by the current line terminator, to the standard output stream.

- In our program, we can use methods in .NET Framework built-in libraries or any additional third-party libraries
- We must refer to the documentation on the method usage

<https://docs.microsoft.com/en-us/dotnet/api/system.console.writeline?view=netframework-4.7.2>

WriteLine is a method that belongs to System.Console class

System.Console.WriteLine("Welcome to ISS!");

System.Console class is another way to say that there's a Console class in a System namespace

We can refer to the class just by typing Console as long as we have import System class with "using System;" directive

Method can have zero to multiple arguments

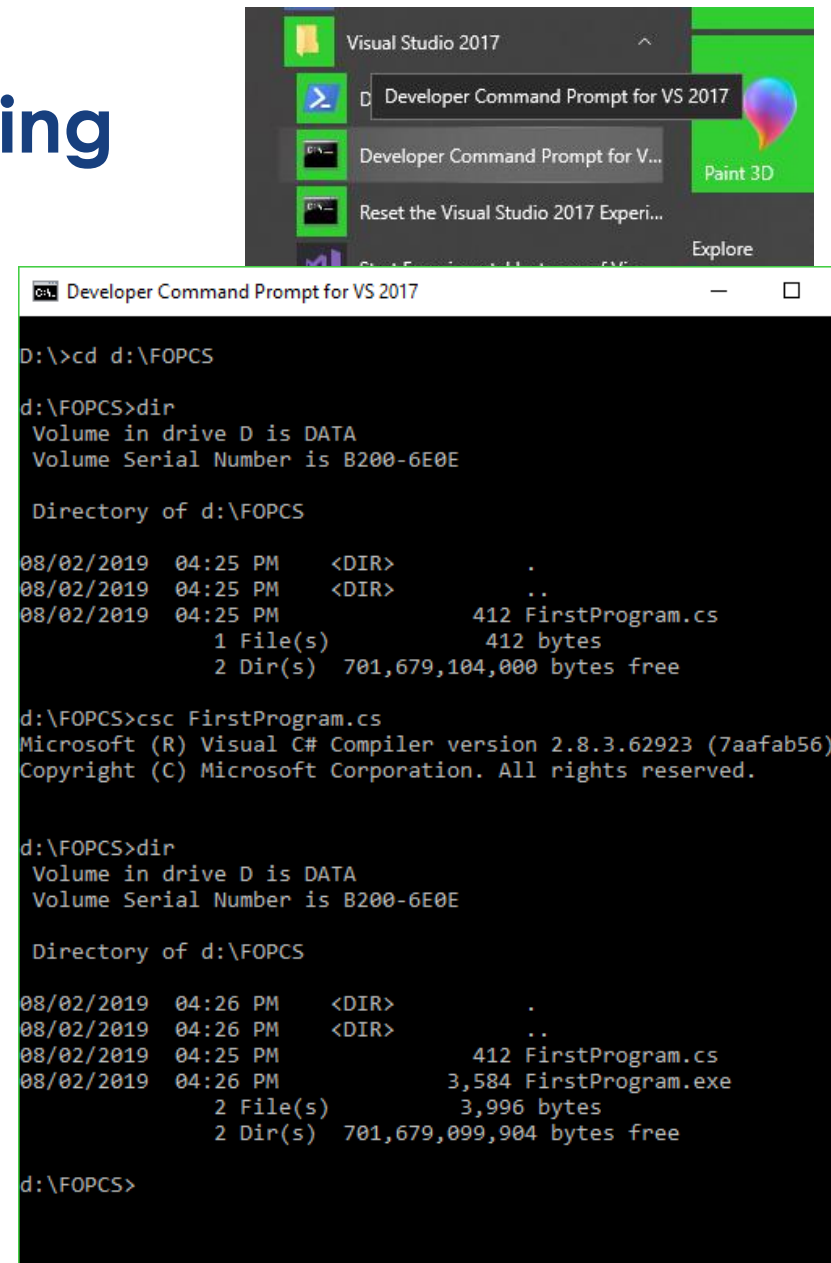
Arguments are specified within the bracket separated with commas

In this example, there's only one argument which contains the text that we want to print out.

We need to enclose text (a.k.a. string) value with double quote in C#

Execute C# Code using Command Line - Compiling

- C# compiler transform C# source code into CLI (common language infrastructure) code that can be executed by .NET Framework.
- Run Developer Command Prompt from Start Menu
- Go to the folder where we put our source code
- Run csc (C Sharp Compiler) and pass our source code file
- This is probably the first and last time we use csc.exe manually in this module



The image shows a Windows Start menu with the Visual Studio 2017 icon. A search bar is active, showing results for "Developer Command Prompt for VS 2017". Below the Start menu, a Developer Command Prompt for VS 2017 window is open. The command prompt shows the following commands and output:

```
D:\>cd d:\FOPCS

d:\FOPCS>dir
Volume in drive D is DATA
Volume Serial Number is B200-6E0E

Directory of d:\FOPCS

08/02/2019  04:25 PM    <DIR>          .
08/02/2019  04:25 PM    <DIR>          ..
08/02/2019  04:25 PM                412 FirstProgram.cs
               1 File(s)                412 bytes
               2 Dir(s)  701,679,104,000 bytes free

d:\FOPCS>csc FirstProgram.cs
Microsoft (R) Visual C# Compiler version 2.8.3.62923 (7aafab56)
Copyright (C) Microsoft Corporation. All rights reserved.

d:\FOPCS>dir
Volume in drive D is DATA
Volume Serial Number is B200-6E0E

Directory of d:\FOPCS

08/02/2019  04:26 PM    <DIR>          .
08/02/2019  04:26 PM    <DIR>          ..
08/02/2019  04:25 PM                412 FirstProgram.cs
08/02/2019  04:26 PM            3,584 FirstProgram.exe
               2 File(s)                3,996 bytes
               2 Dir(s)  701,679,099,904 bytes free

d:\FOPCS>
```

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-options/command-line-building-with-csc-exe>

Execute C# Code using Command Line - Running

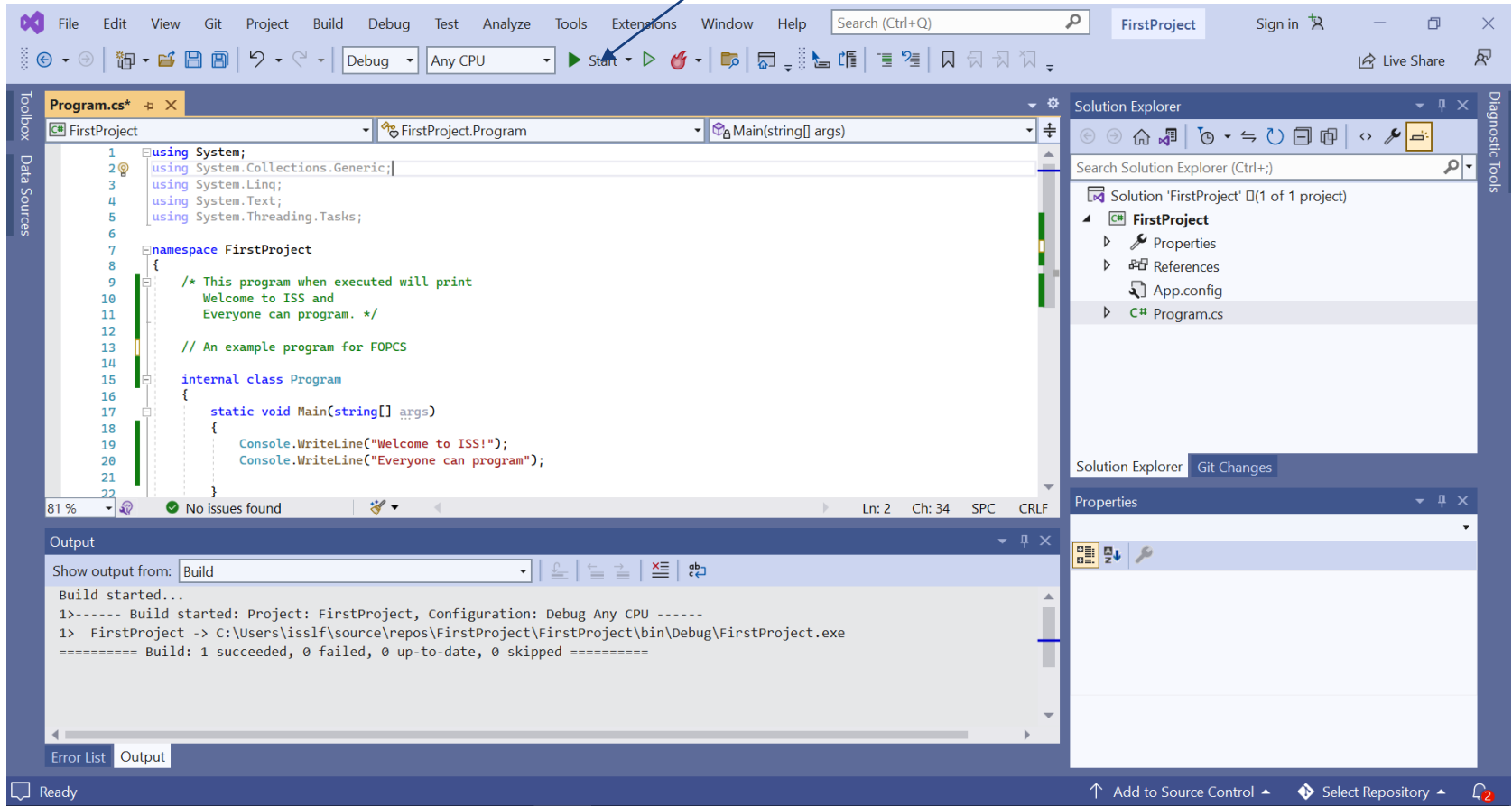


- The compilation process will produce an executable file – FirstProgram.exe
- We can call this executable file from the command prompt to run our program
- We should see the text that we put in our program printed on the screen.

```
Developer Command Prompt for VS 2017
d:\FOPCS>FirstProgram
Welcome to ISS!
Everyone can program
d:\FOPCS>
```


Running your program

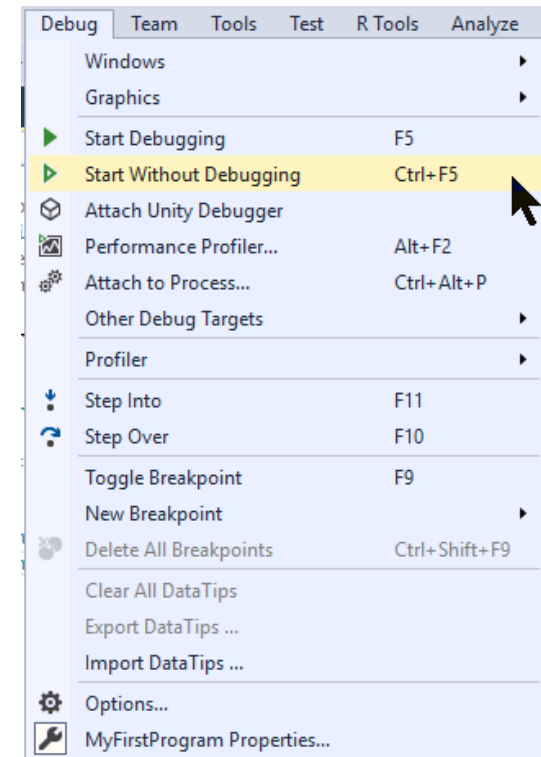
Once you have no syntax error, you can click on Start button to run your program – but you probably don't see the expected output – Why?



Running your program

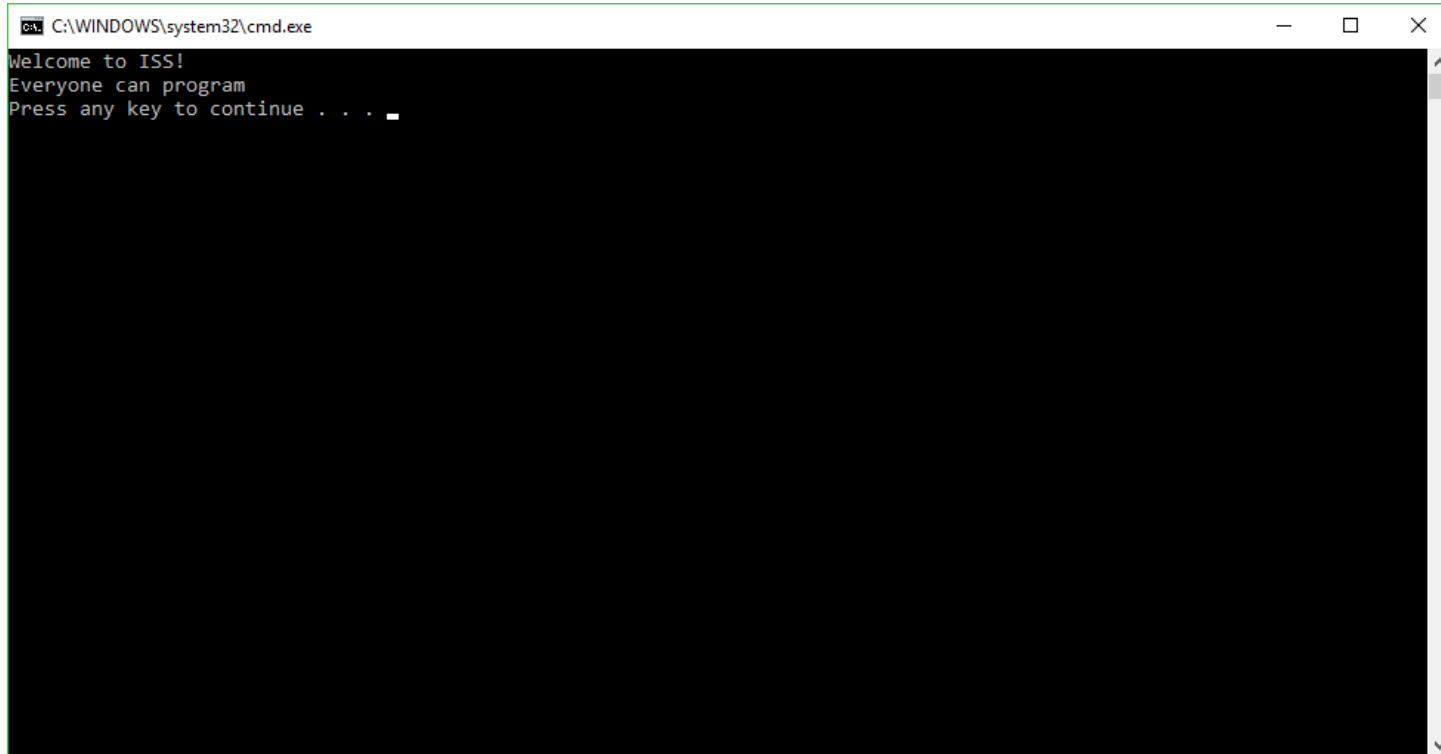


- There are two ways to run your program
 - Start (with) Debugging
 - This is what happened when you click the Start button
 - The program will stop when a breakpoint is reached (but we have none so far)
 - At the end of the program, the console window will be closed automatically
 - This is why we can't see our output
 - Start without Debugging (Ctrl+F5)
 - Breakpoints will be ignored
 - The console window will not be closed automatically
 - This is perfect for us.



Running our program

Choose Start without Debugging or press Ctrl-F5 and we will see the output of our program



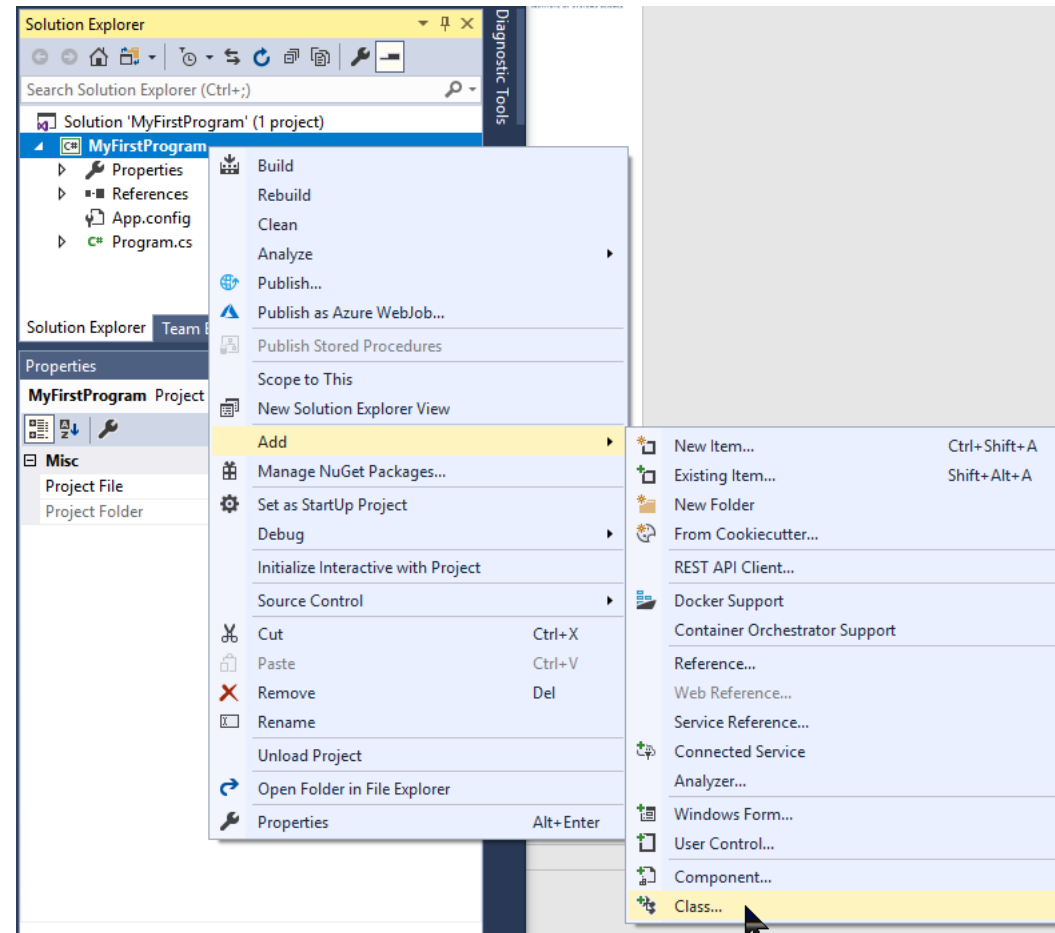
```
C:\WINDOWS\system32\cmd.exe
Welcome to ISS!
Everyone can program
Press any key to continue . . .
```

Writing another program

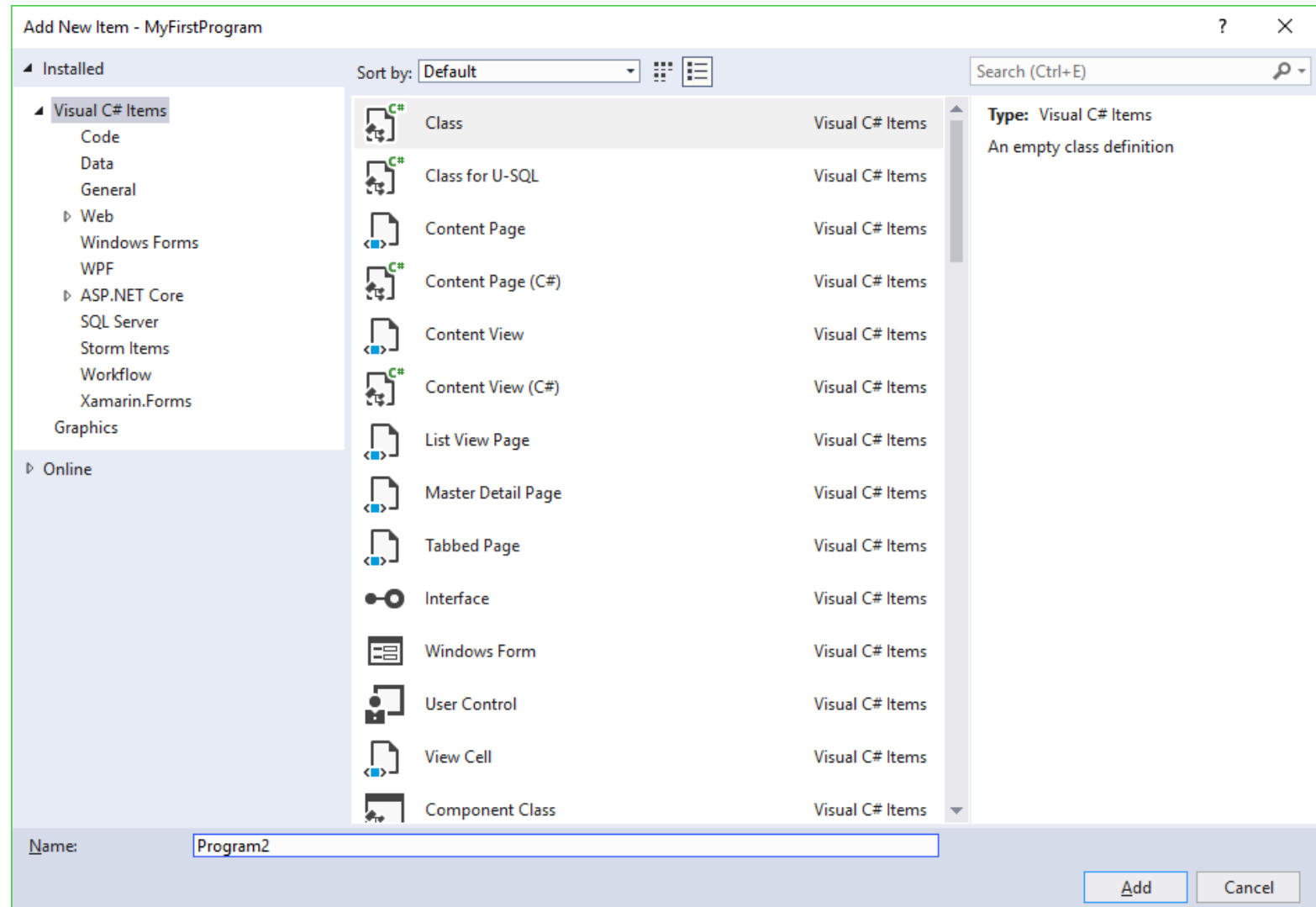
- If we want to write a second program, we have at least two options:
 - Create a new solution, a new project and a new class with a `Main` method for our second program
 - Preferred approach when we are writing a totally different program and the program is relatively complex
 - Write the new program in the same project
 - Challenge: a program can have one entry point but both program have its own entry points
 - we'll learn how to solve this
 - This is probably preferred for practice so that you won't have too many folders and source files created on your hard disk.

Writing another program

- Create a new class
- Right-click on the project in Solution Explorer, choose Add > Class
- Name the class

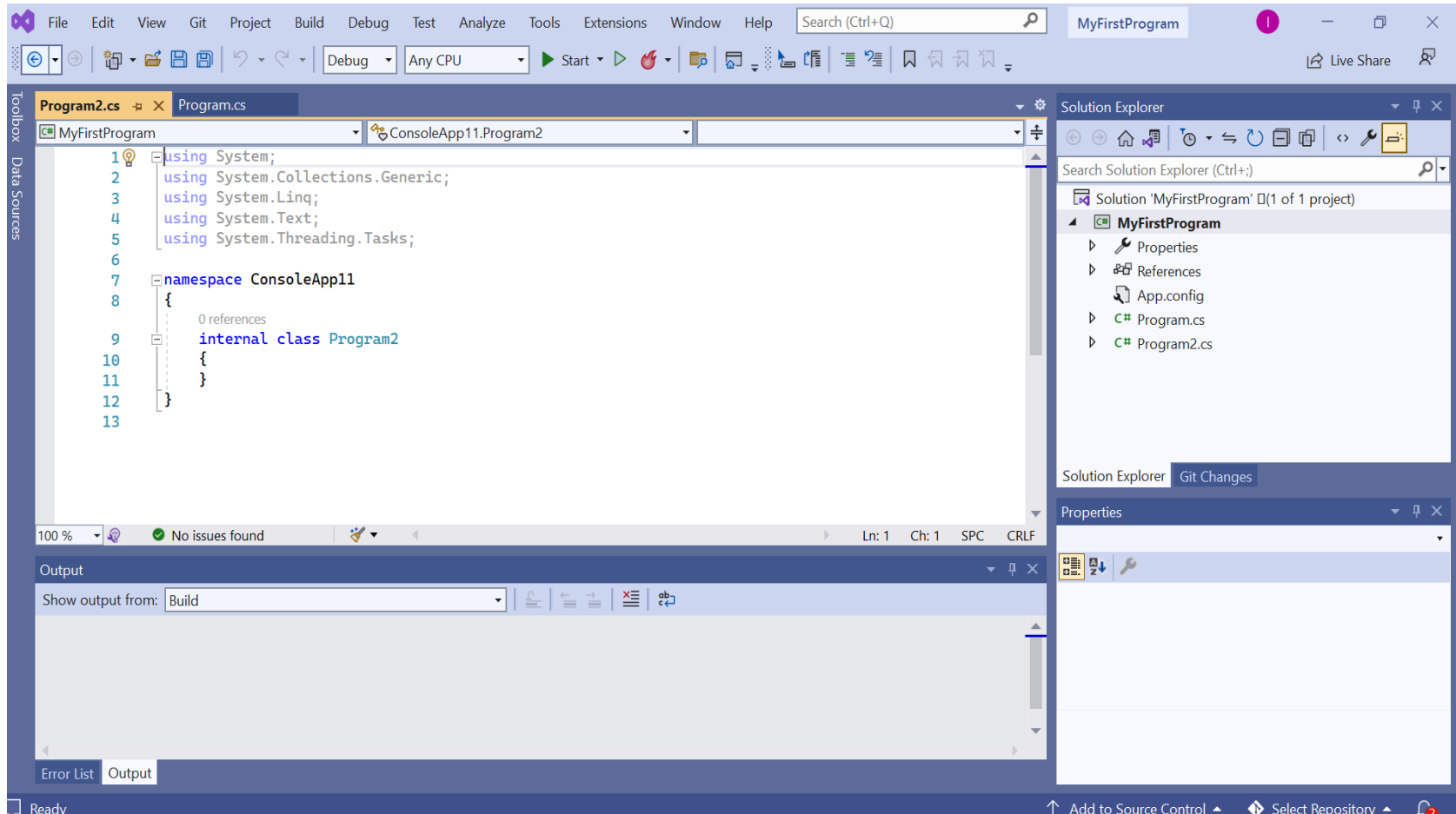


Writing another program



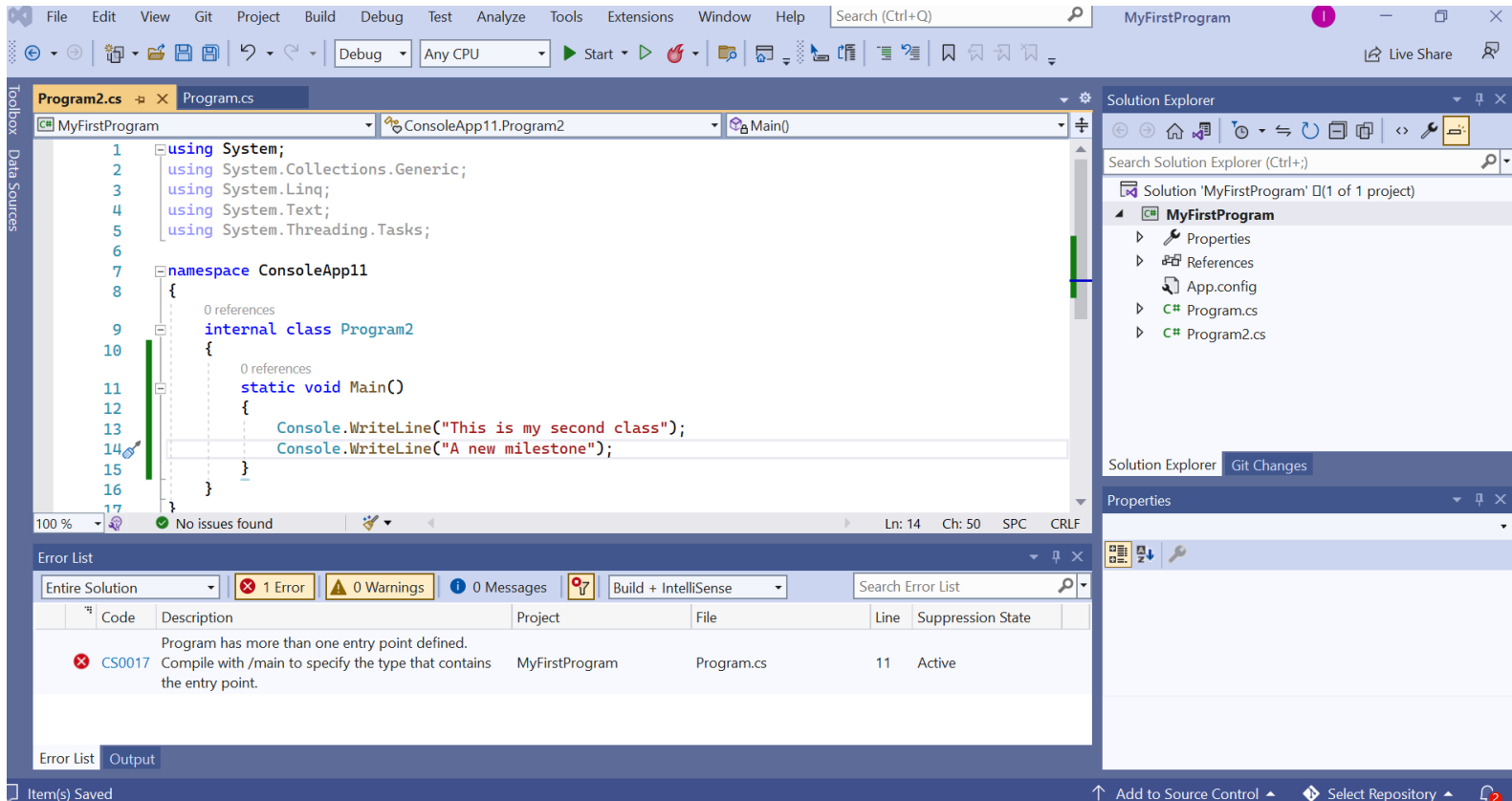
Writing another program

The second class does not automatically have the Main method defined because normally one project will only have one Main method. So we have to write our own Main method.



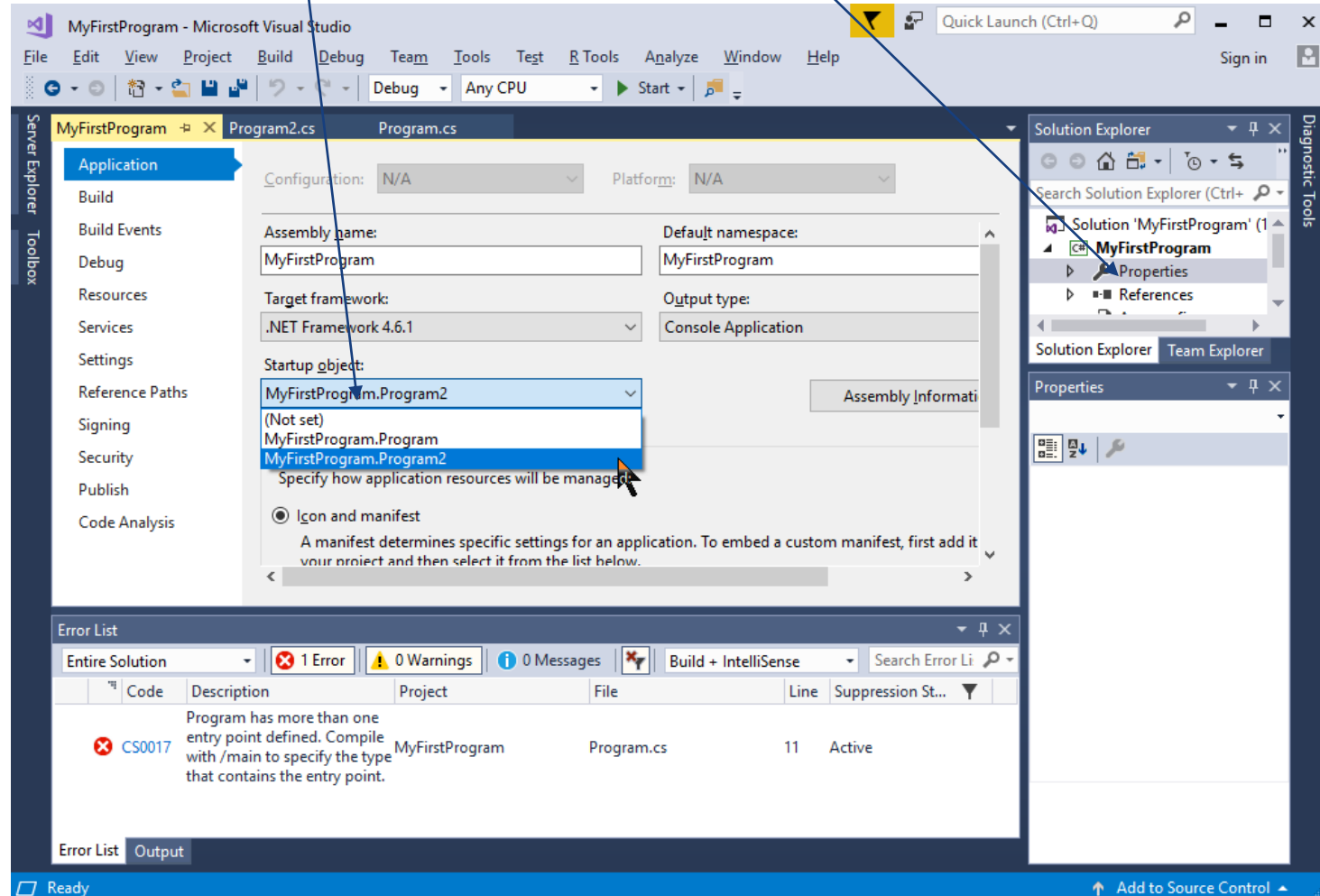
Writing another program

If we run our program, we will get this error because we have more than one possible entry points in our program. VS expects us to explicitly specify which one to use.



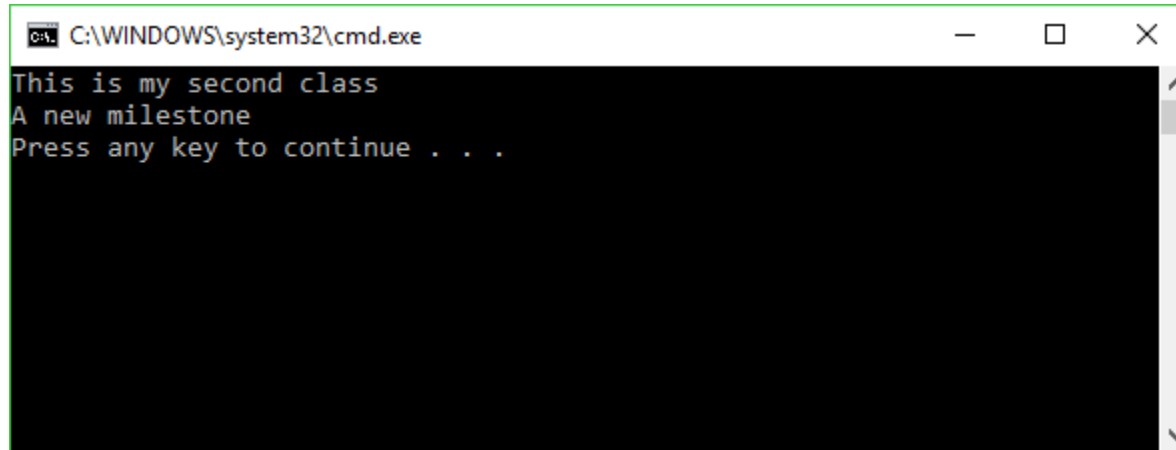
Writing another program

Go to project properties page
and nominate a class to be our startup object.
If your new class doesn't appear, check for typo and re-build



Writing another program

At this point we should be able to run our second program.



```
C:\WINDOWS\system32\cmd.exe
This is my second class
A new milestone
Press any key to continue . . .
```

Do we have to keep changing the startup objects?
Yes, for now.

Once we learn conditionals, we can run different programs
based on user's input.

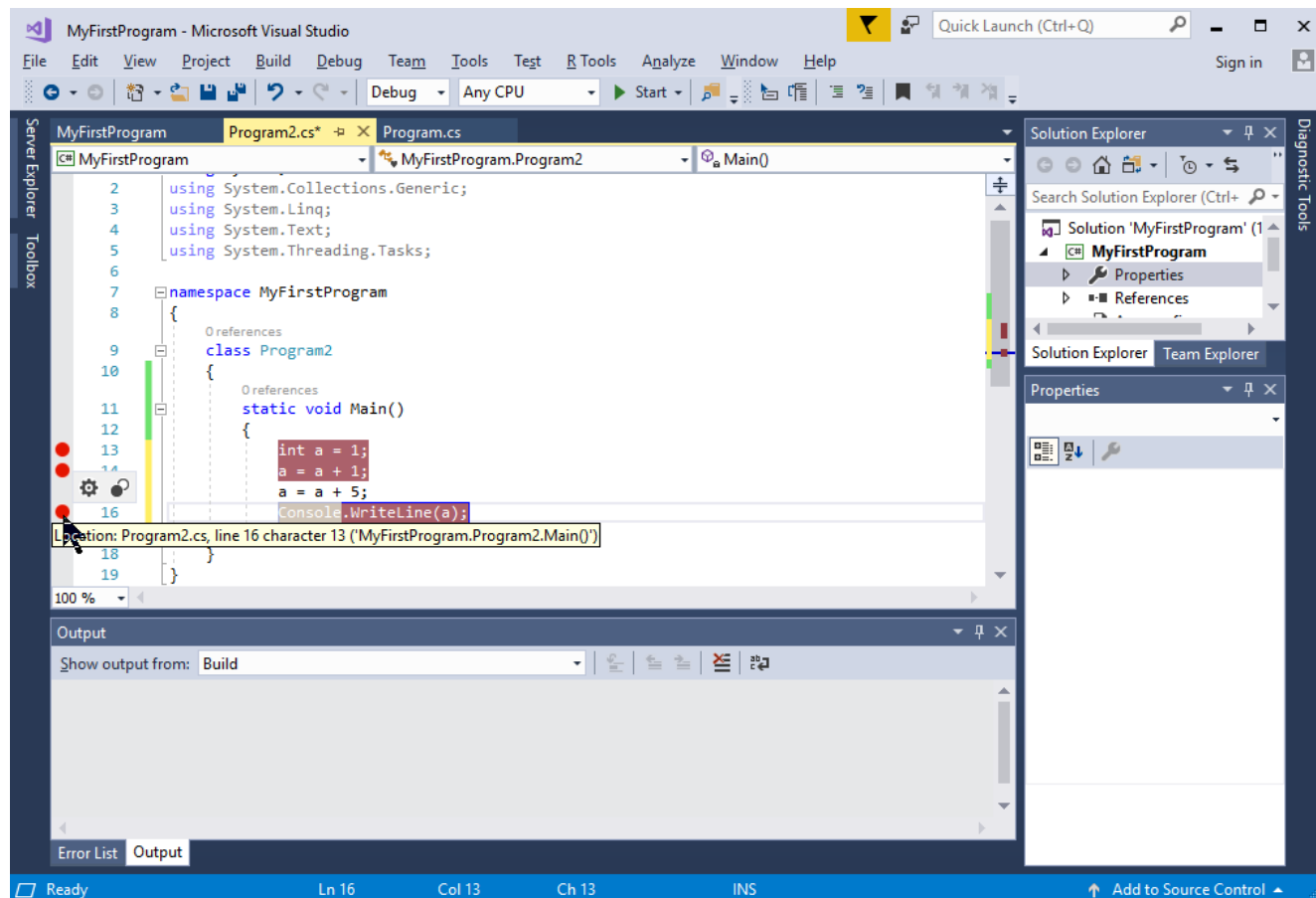
Debugging Features

- Breakpoint
- Watch Window
- Immediate Window
- Auto Window

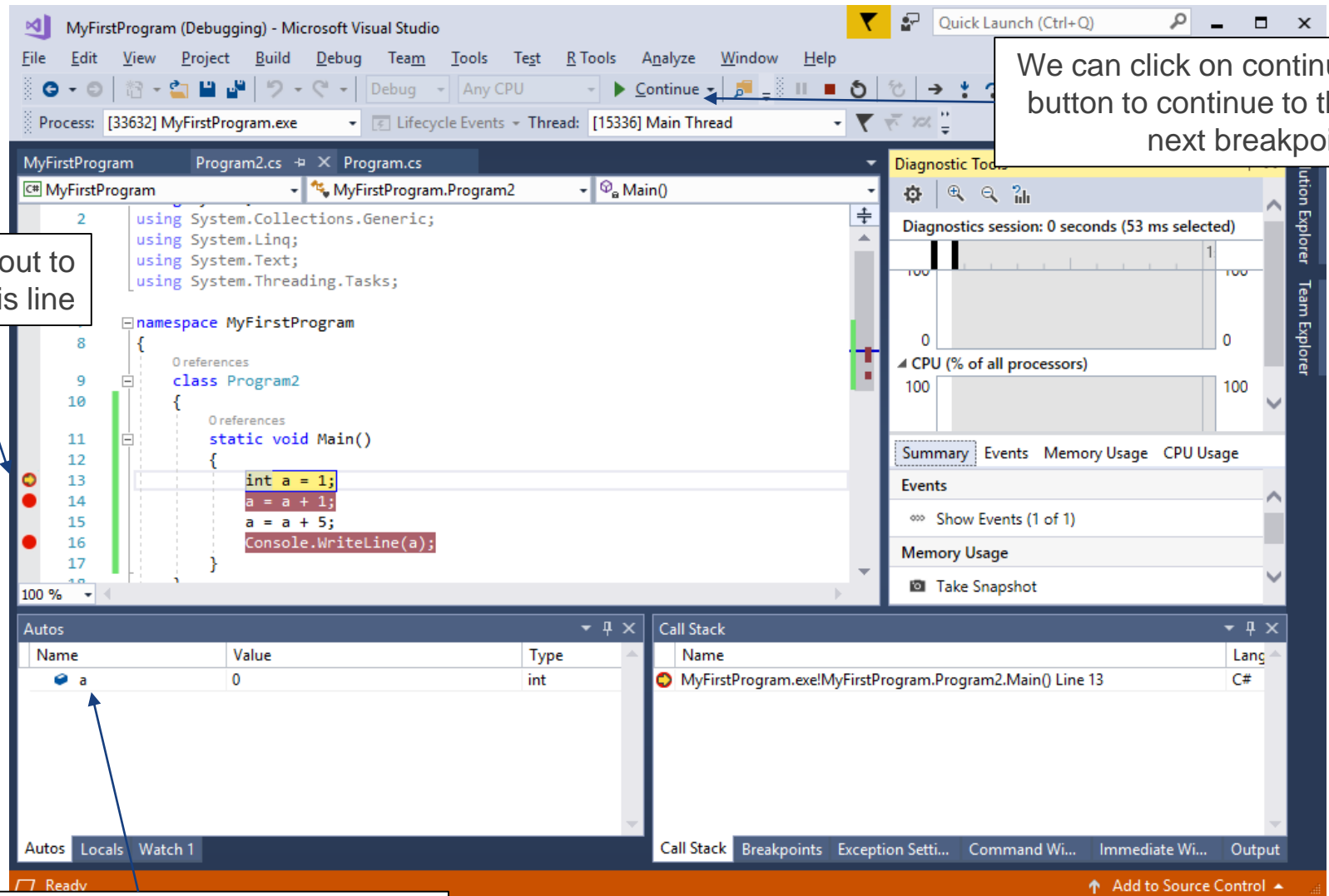
Breakpoints

We can set breakpoints by clicking on the grey left border in our editor windows. Breakpoints are marked with the red circles.

Setting breakpoint means that we want our program to pause at that point (“take a break”) so that we can inspect the state of our program.



Debugging with breakpoints



MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test R Tools Analyze Window Help

Process: [33632] MyFirstProgram.exe Thread: [15336] Main Thread

MyFirstProgram Program2.cs Program.cs

MyFirstProgram Program2 Main()

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

namespace MyFirstProgram

{

0 references

class Program2

{

0 references

static void Main()

{

int a = 1;
a = a + 1;
a = a + 5;
Console.WriteLine(a);

}

100 %

Autos

Name	Value	Type
a	0	int

Call Stack

Name	Lang
MyFirstProgram.exe!MyFirstProgram.Program2.Main() Line 13	C#

Summary Events Memory Usage CPU Usage

Events

Show Events (1 of 1)

Memory Usage

Take Snapshot

Call Stack Breakpoints Exception Setti... Command Wi... Immediate Wi... Output

Ready

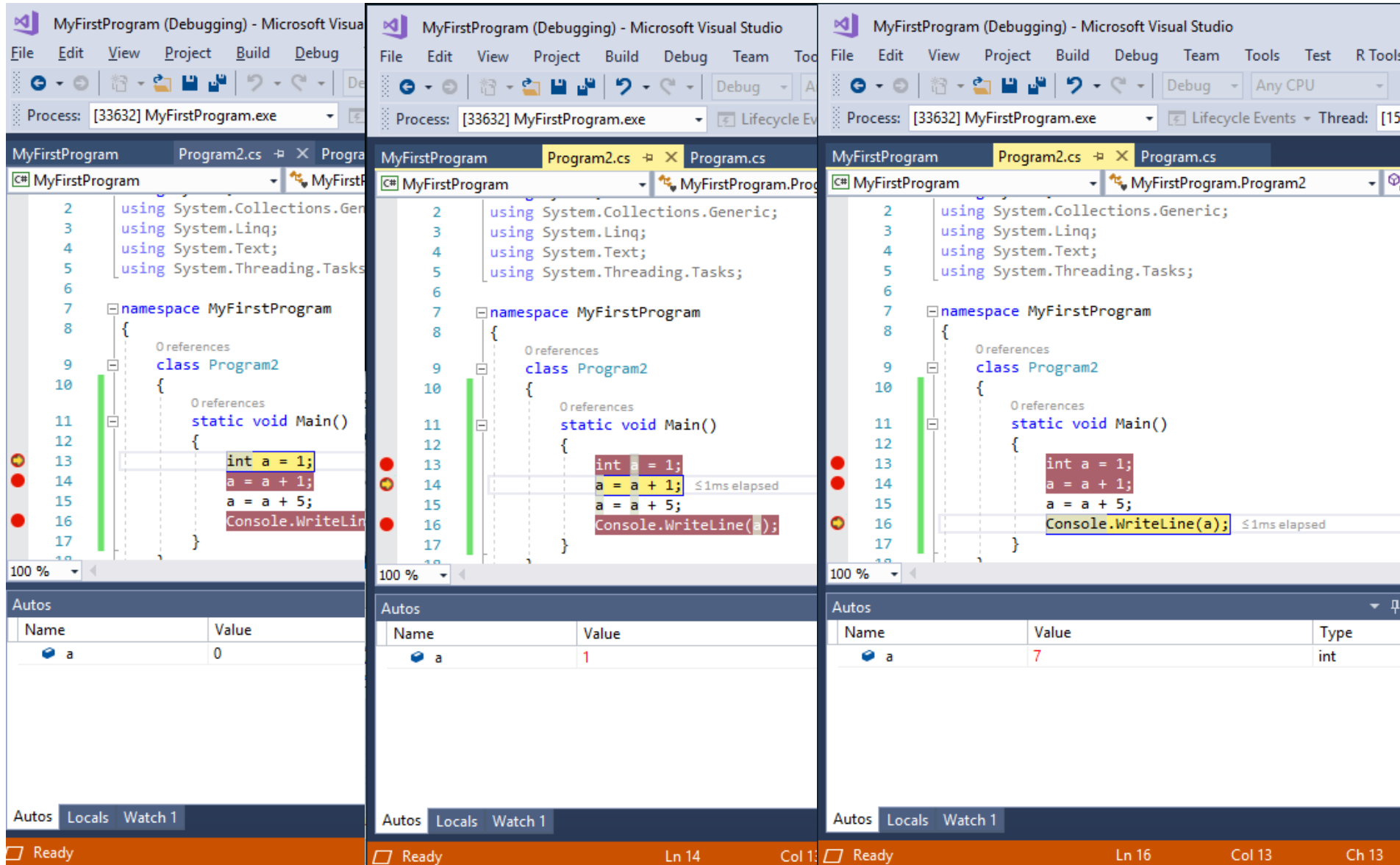
Add to Source Control

We are about to execute this line

We can click on continue button to continue to the next breakpoint

Auto window shows the state of variables that are related to our current line of code

Debugging with breakpoints



MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test R Tools

Process: [33632] MyFirstProgram.exe

MyFirstProgram Program2.cs Program.cs

```
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     class Program2
10     {
11         static void Main()
12         {
13             int a = 1;
14             a = a + 1;
15             a = a + 5;
16             Console.WriteLine(a);
17         }
18     }
19 }
```

Autos

Name	Value
a	0

MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test R Tools

Process: [33632] MyFirstProgram.exe

MyFirstProgram Program2.cs Program.cs

```
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     class Program2
10     {
11         static void Main()
12         {
13             int a = 1;
14             a = a + 1;
15             a = a + 5;
16             Console.WriteLine(a);
17         }
18     }
19 }
```

Autos

Name	Value
a	1

MyFirstProgram (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test R Tools

Process: [33632] MyFirstProgram.exe

MyFirstProgram Program2.cs Program.cs

```
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MyFirstProgram
8 {
9     class Program2
10     {
11         static void Main()
12         {
13             int a = 1;
14             a = a + 1;
15             a = a + 5;
16             Console.WriteLine(a);
17         }
18     }
19 }
```

Autos

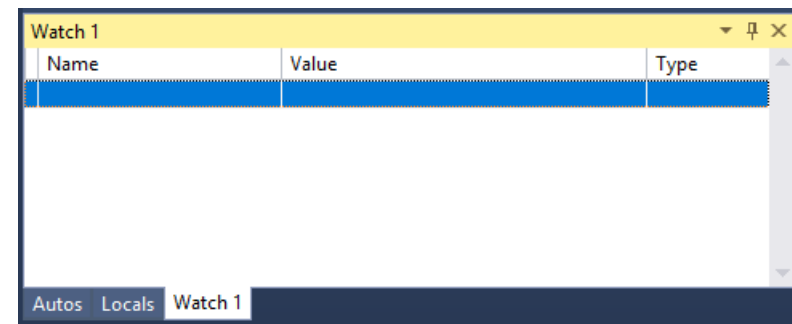
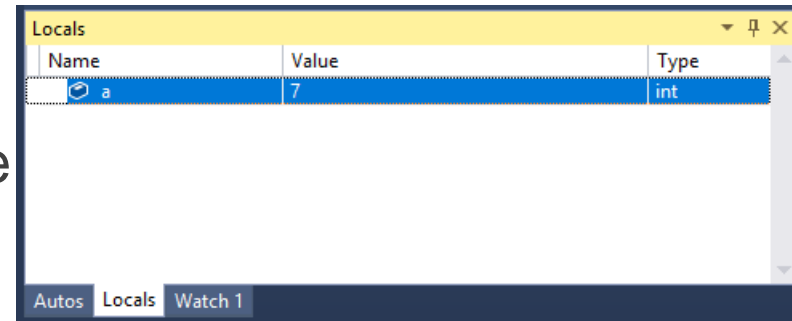
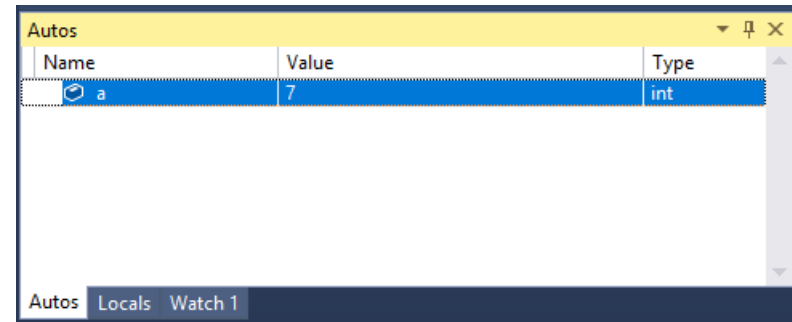
Name	Value	Type
a	7	int

Options on a breakpoint

- Continue to the next breakpoint
- Continue to the next line (Debug > Step Over)
- Continue into the implementation of the method on our current line (Debug > Step Into)
 - Useful when we have learned about methods
- Continue to the next line outside of our method implementation (Debug > Step Out)
 - Step Into followed with Step Out = Step Over

Monitoring Program State

- With IDE, we can check the value of our variables
- Auto window
 - Display the state of variables related to the current line of code
- Watch window
 - Allow user to manually enter the variables to be monitored
- Local window
 - The state of all local variables



Immediate Window

References
static void Main()
{
int a = 1;
a = a + 1;
a = a + 5;
Console.WriteLine(a);
}

100 %

Autos

Name	Value	Type
a	0	int

Summary Events Memory Usage CPU Usage

Events

Show Events (1 of 1)

Memory Usage

Take Snapshot

Allow you to perform C# statements, including changing the variable value during a break

Autos

Name	Value	Type
a	2	int

Immediate Window

```
a  
0  
a=2  
2  
|
```

Autos Locals Watch 1

Call Stack Breakpoints Exception Setti... Command Wi... Immediate Wi...

Debugging a Program

- When program doesn't run as expected (a program bug), we need to investigate the problem (debug).
- Many approach to debugging:
 - Write (log) variable values to screens or files
 - Use breakpoint and step through your program
 - Comment out part of your codes and check how your program runs without those part
 - Try to isolate the bug

- What is Intellisense?
 - It is a editor feature, that helps the developer with prompts, lookup and other visual features.
 - It improves the programmers editing productivity
- What are the features?
 - List Members
 - Parameter Info
 - Quick Info
 - Complete Word
 - Automatic Brace Matching

<https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019>

```
// Place the frame in the current Window
Window::Current->Content = rootFrame;
```

Window::C

```
// Ensure
Window::
```

- Close
- Closed
- Content
- CoreWindow
- Current

public : void Windows::UI::Xaml::IWindow::Close()
File: Windows.winmd
+ 1 overload

```
InitializeComponent();
string s = "hello";
bool b = s.EndsWith("o", |)
```

▲ 2 of 3 ▼ **bool string.EndsWith(string value, StringComparison comparisonType)**
Determines whether the end of this string instance matches the specified string when compared using the specified comparison option.
comparisonType: One of the enumeration values that determines how this string and value are compared.

```
InitializedComponent();
string s = "hello";
bool b = s.EndsWith(
```

bool string.EndsWith(string value) (+ 2 overloads)
Determines whether the end of this string instance matches the specified string.
No overload for method 'EndsWith' takes 0 arguments

Syntax Errors

- Syntax errors happens when the compiler cannot correctly understand our program e.g. we mistype something
- Let's try to put in some typo error.

```
static void Main(string[] args)
```



```
static void main(string[] args)
```

Developer Command Prompt for VS 2017

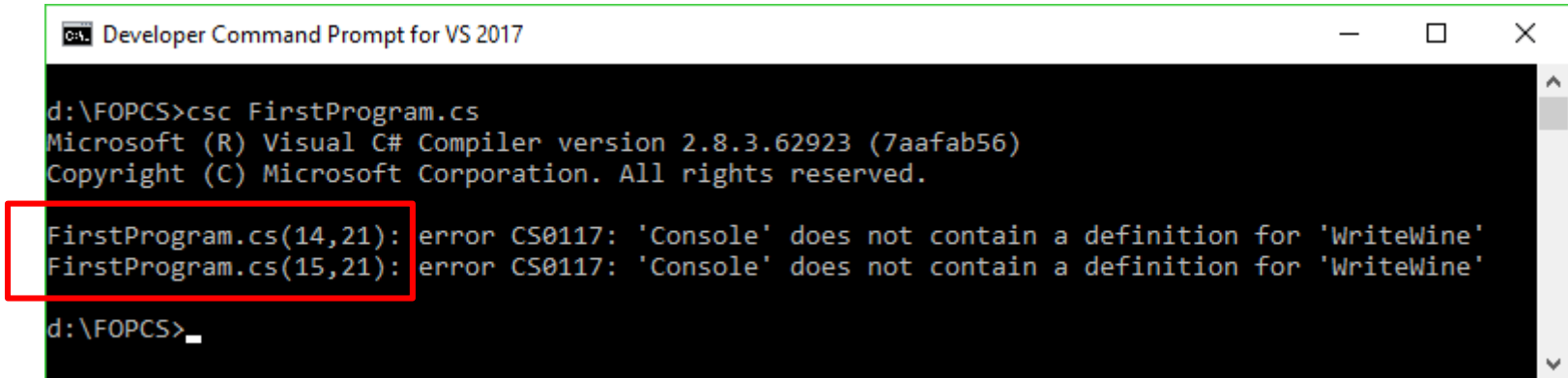
```
d:\FOPCS>csc FirstProgram.cs
Microsoft (R) Visual C# Compiler version 2.8.3.62923 (7aafab56)
Copyright (C) Microsoft Corporation. All rights reserved.

error CS5001: Program does not contain a static 'Main' method suitable for an entry point

d:\FOPCS>_
```

Syntax Error

- If we replace WriteLine to WriteWine



```
Developer Command Prompt for VS 2017

d:\FOPCS>csc FirstProgram.cs
Microsoft (R) Visual C# Compiler version 2.8.3.62923 (7aafab56)
Copyright (C) Microsoft Corporation. All rights reserved.

FirstProgram.cs(14,21): error CS0117: 'Console' does not contain a definition for 'WriteWine'
FirstProgram.cs(15,21): error CS0117: 'Console' does not contain a definition for 'WriteWine'

d:\FOPCS>_
```

- The compiler will try to infer the location of the syntax error
 - We use this information to find the error and fix it
 - Sometimes the location inferred is not accurate – we need to analyse our own source code to find the root problem

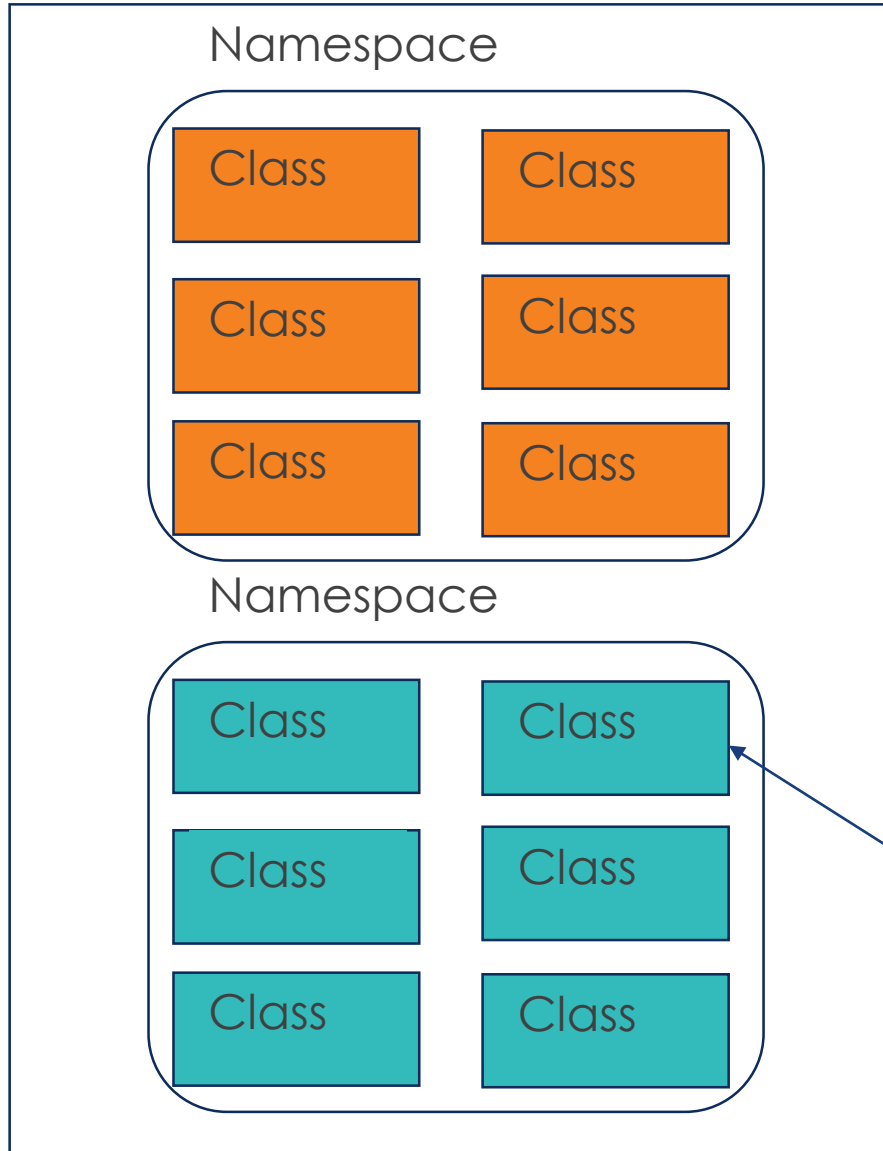
Common Beginner's Syntax Errors

- Using a wrong case e.g. Main become main
- Forget to terminate a statement with semi-colon
- Typo on class names or method names
- Mismatch curly braces, brackets or quotes
- Use the short name of a class without importing the namespace with “using” directive

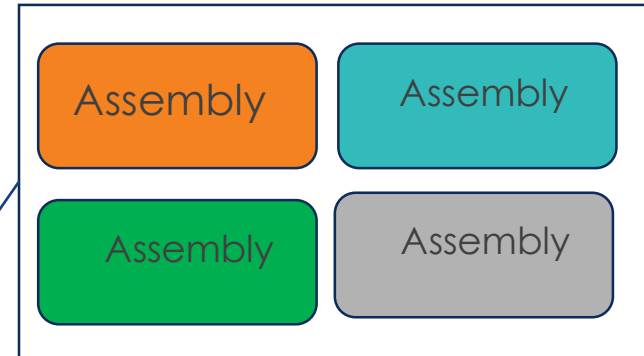
Summary

- IDE is a very useful productivity tools for programmers/ developers
 - Provide templates
 - Editing features
 - Automate application building
 - Debugging features
- Not discussed
 - Source control features
 - Deployment features
 - Can install plugins to allow for additional features

APPENDIX



Application



Class	Car
Data	Engine Model Color
Methods	Start() Move()

- Application
- Assembly(DLL or EXE)
 - Containers for related namespaces
- Namespace
 - Containers for related classes. E.g. namespace for working
- Class
 - Data
 - Methods

Example for Main method

Example for “string[] args” in Main method

The parameter of the Main method is a String array that represents the command-line arguments

So, if I had a program (MyApp.exe) like this:

```
class Program
{
    static void Main(string[] args)
    {
        foreach(var arg in args)
        {
            Console.WriteLine(arg);
        }
    }
}
```

- That I started at the command line like this:
MyApp.exe Arg1 Arg2 Arg3
- The Main method would be passed an array that contained three strings: "Arg1", "Arg2", "Arg3".

Method

- A metaphor for a method is like a machine that can take multiple input and produce an output
- Arguments are the input to the machine
- The output is the return value
 - Some methods like WriteLine doesn't have a return value

