



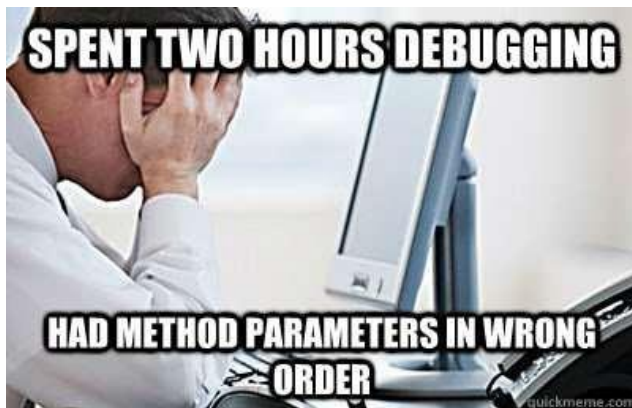
DEBUGGING IN VISUAL STUDIO (BASIC LEVEL)

Arctech Info

AGENDA

- ▣ Break Point & Trace Point
- ▣ Code Stepping
- ▣ Edit variable value
- ▣ Debug with Object ID
- ▣ Set the Next Statement
- ▣ Edit and continue
- ▣ Q & A

A Slide on Debugging



Start with the Basics?



Ido Flatow

@IdoFlatow

 Follow

That moment when a .NET developer with over 5 years of experience asks you what is that “Immediate” window in Visual Studio you just used...

9:11 AM - 7 Aug 2016

  1  3



Uri Goldstein

@urig

 Follow

I've found out about it earlier this year >15 years into using Visual Studio. You learn something every day ;)

[twitter.com/IdoFlatow/stat...](https://twitter.com/IdoFlatow/status/724111111111111111)

9:53 AM - 7 Aug 2016

   2



Debug Windows (1)

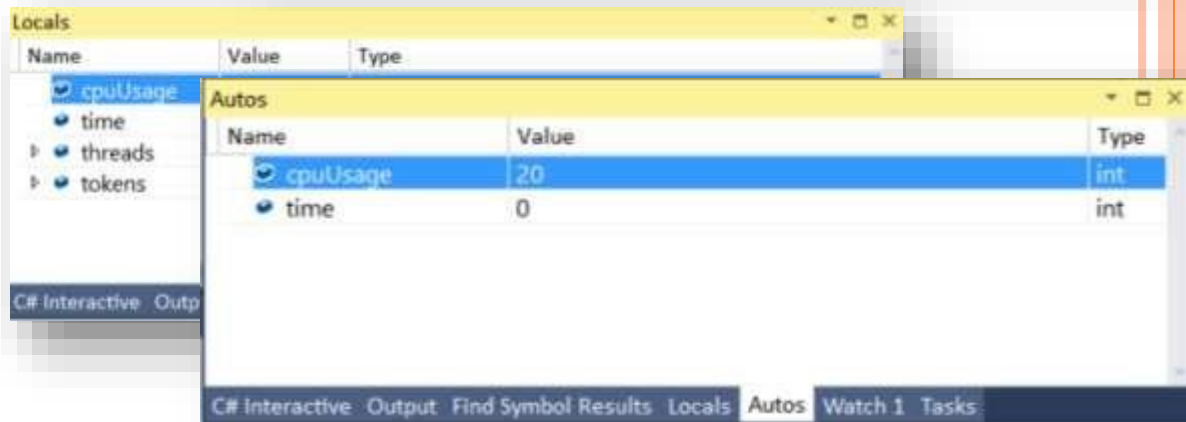
- Data Tips



- Locals / Auto

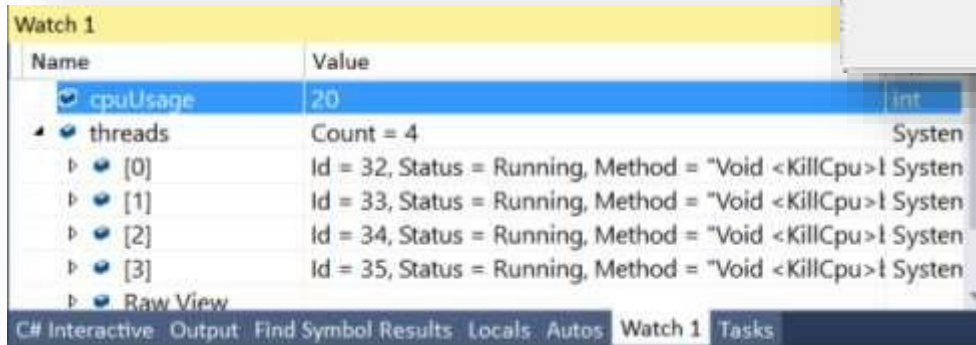
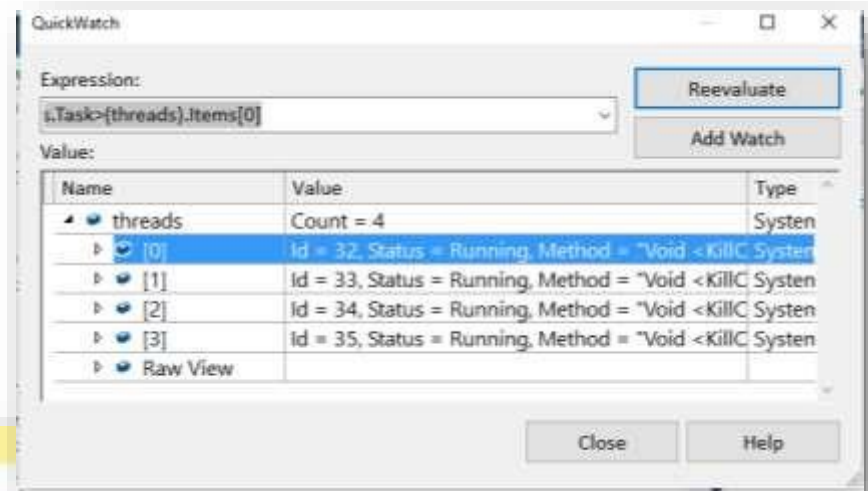
The **Autos** window shows variables used around the current breakpoint.

The **Locals** window shows variables defined in the local scope, which is usually the current function or method.



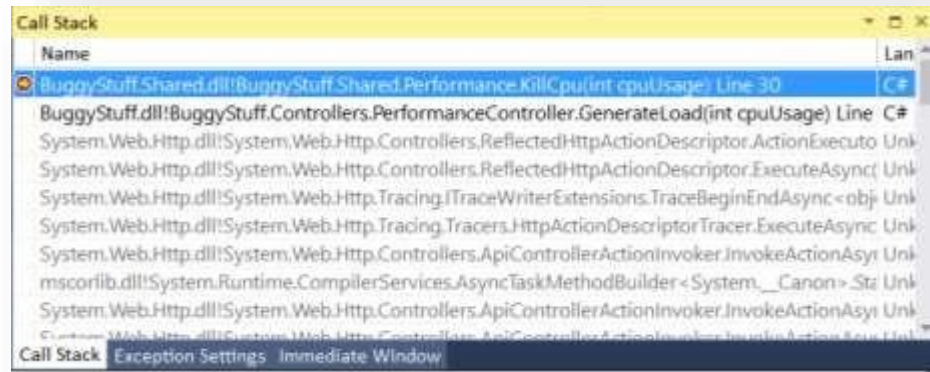
Debug Windows (2)

- Quick Watch
- Watch



Debug Windows (3)

- Call Stack
- Immediate





















BREAK POINT & TRACE POINT

3

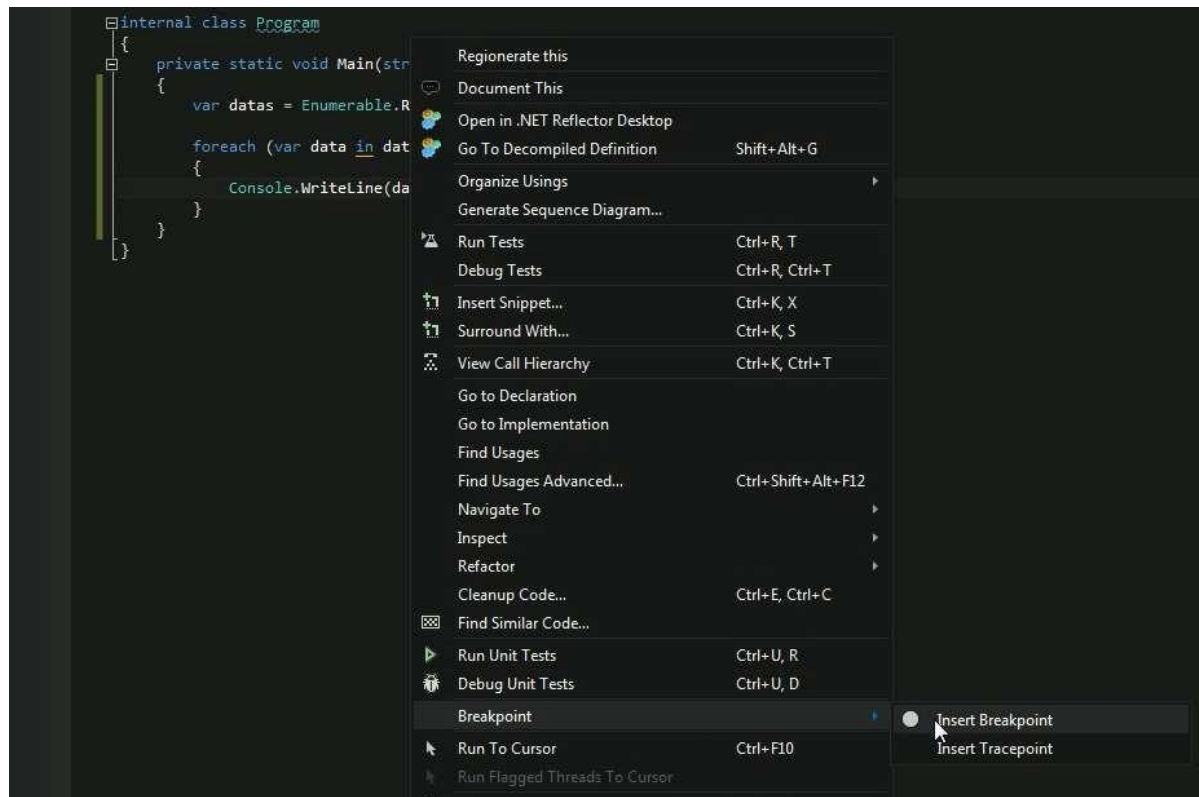
BREAKPOINT GLYPHS

Glyph	Description
 	Normal breakpoint. The solid glyph indicates that the breakpoint is enabled. The hollow glyph indicates that it is disabled.
 	Advanced breakpoint. Active/disabled. The + sign indicates that the breakpoint has at least one advanced feature, such as condition, hit count, or filter, attached to it.
 	Mapped breakpoint. Active/disabled. The breakpoint is set in ASP/ASP.NET code and mapped to a breakpoint in the corresponding HTML page or set in a server-side script file and mapped to the corresponding client-side script file.
 	Tracepoint. Active/disabled. Hitting this point performs a specified action but does not break program execution.
 	Advanced tracepoint. Active/disabled. The + sign indicates that the tracepoint has at least one advanced feature, such as condition, hit count, or filter, attached to it.
 	Mapped tracepoint. Active/disabled. The tracepoint is set in ASP/ASP.NET code and mapped to a tracepoint in the corresponding HTML page.
 	Breakpoint or tracepoint error. The X indicates that the breakpoint or tracepoint could not be set because of an error condition.
 	Breakpoint or tracepoint warning. The exclamation mark indicates that a breakpoint or tracepoint could not be set because of a temporary condition. Usually, this means the code at the breakpoint or tracepoint location has not been loaded. It can also be seen if you attach to a process and the symbols for the process are not loaded. When the code or symbols are loaded, the breakpoint will be enabled and the glyph will change.

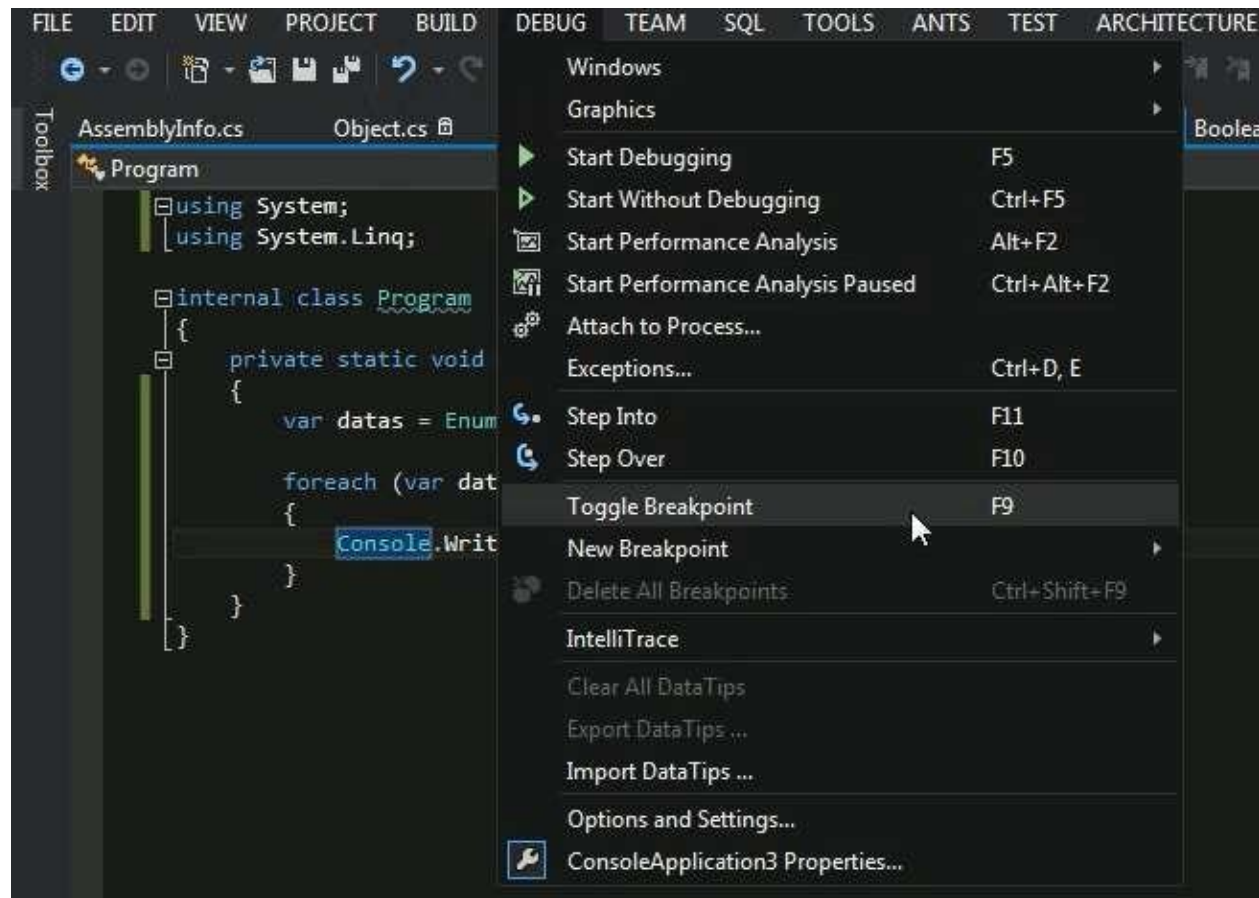
SET A SIMPLE BREAKPOINT

1. **[Debug | Toggle Breakpoint]**
2. **[Breakpoint | Insert Breakpoint]** from context menu in source window.
3. Click on left margin
4. Shortcut
 - F9

SET A SIMPLE BREAKPOINT

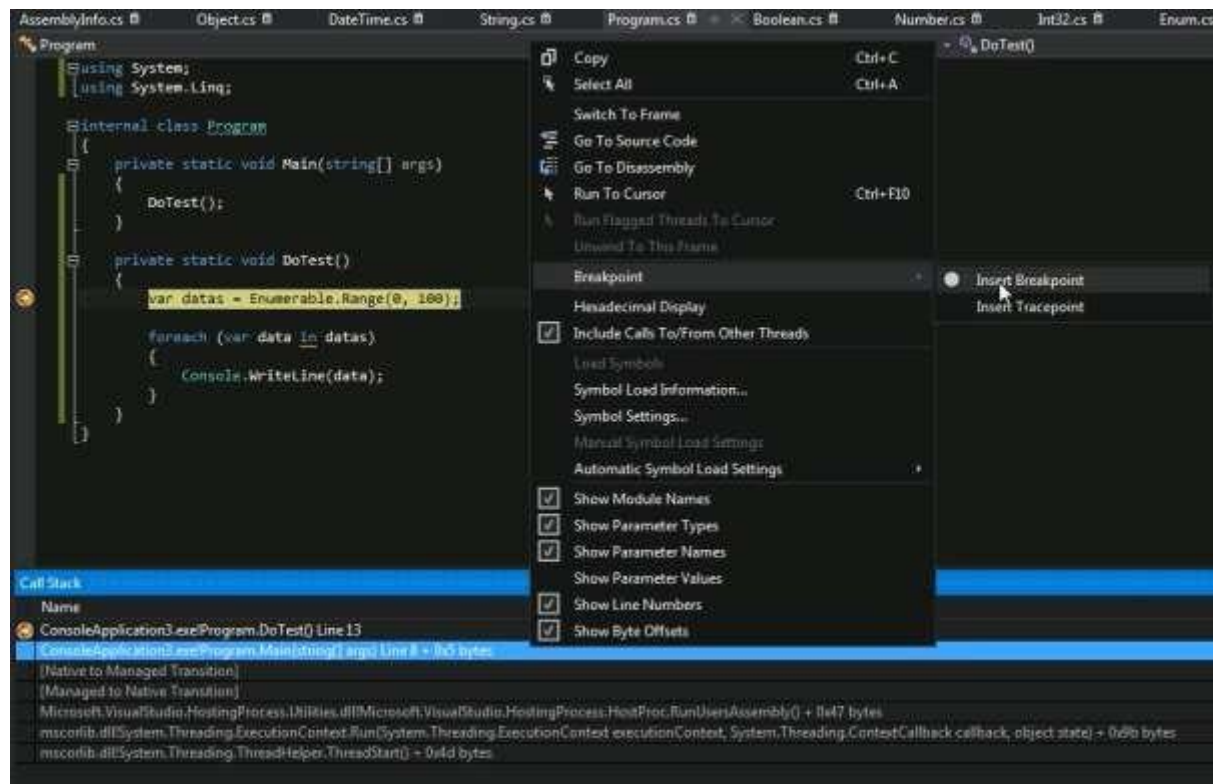


SET A SIMPLE BREAKPOINT



SET A BREAKPOINT ON A FUNCTION CALL FROM THE CALL STACK WINDOW

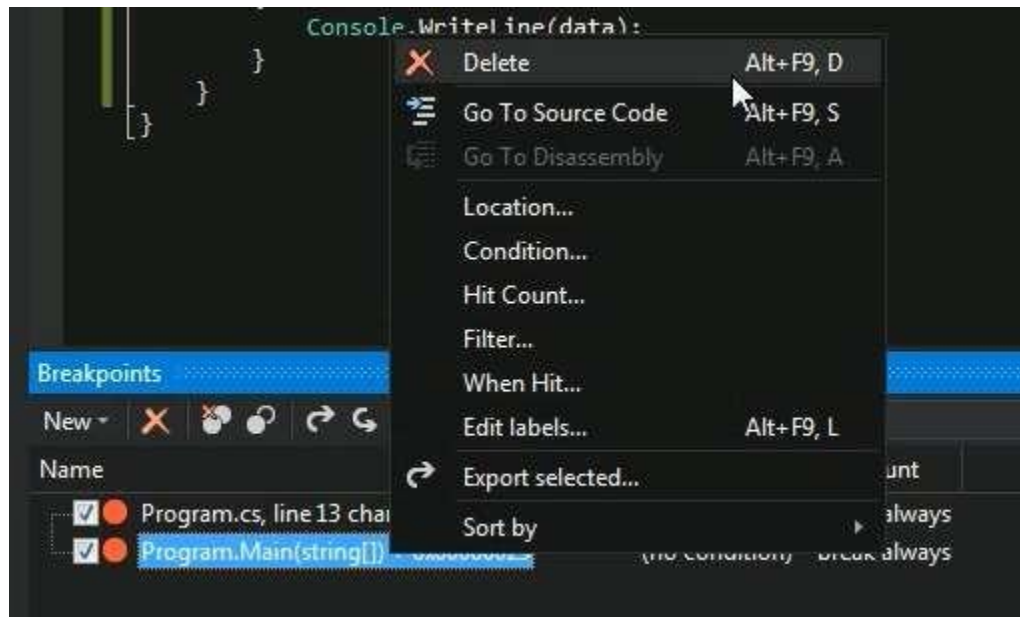
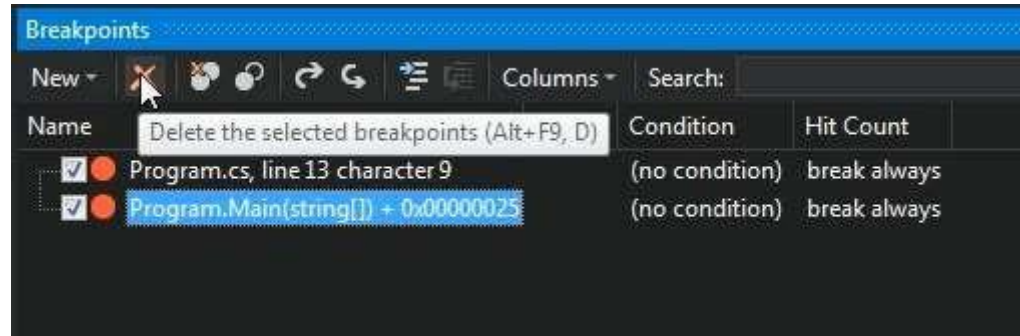
- **[Breakpoint | Insert Breakpoint]** from context menu in Call Stack window.



DELETE A BREAKPOINT

- In the **Breakpoints** window, choose a breakpoint, and click **Delete** button from the toolbar
- In the **Breakpoints** window, right-click on a breakpoint, and choose [**Delete**] from context menu.
- In a source window or Disassembly window, click on the breakpoint glyph.
- Shortcut
 - F9

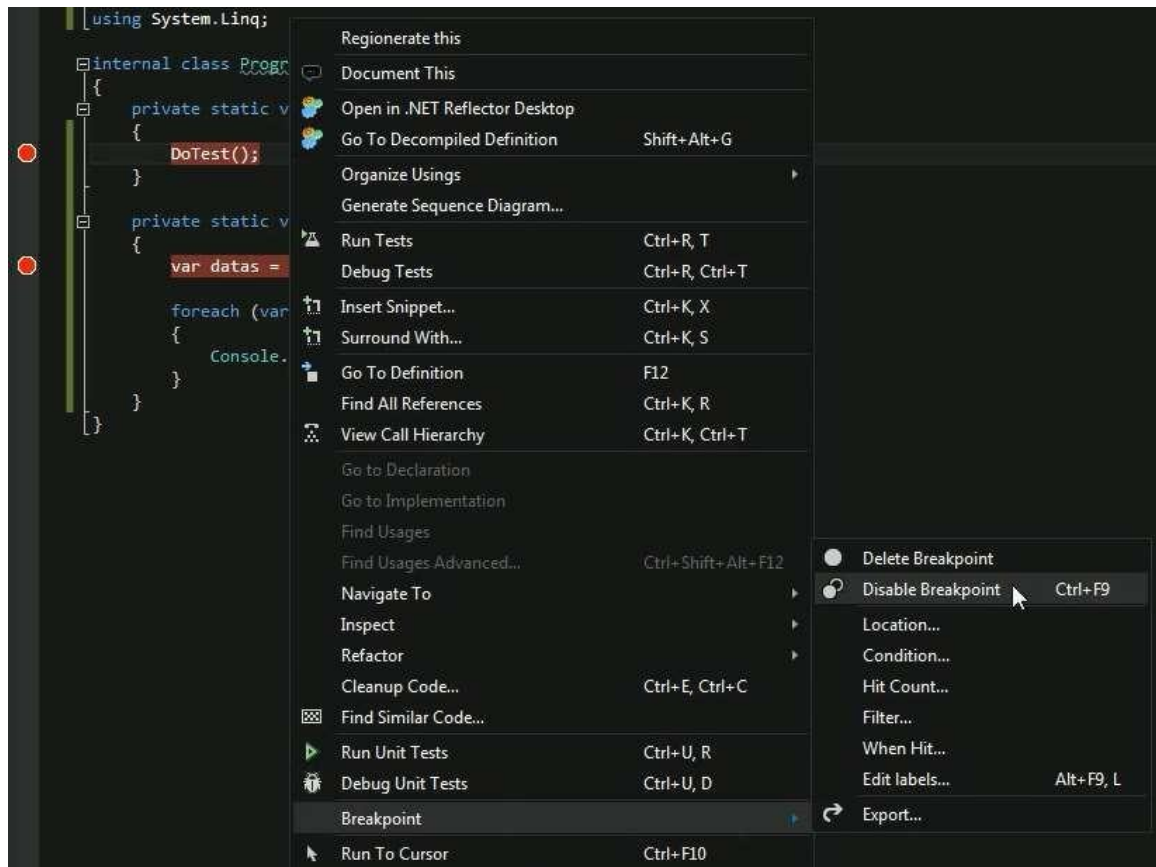
DELETE A BREAKPOINT



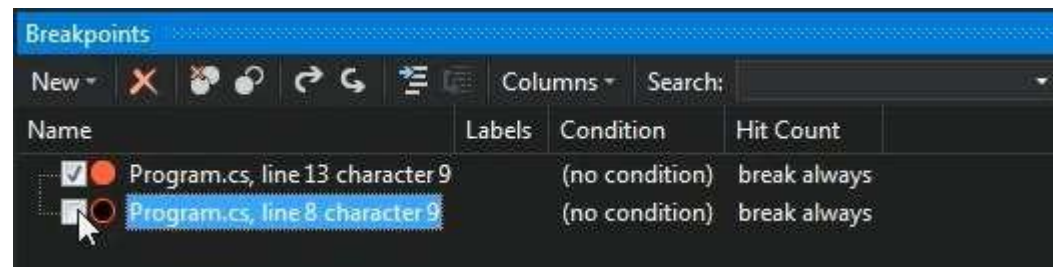
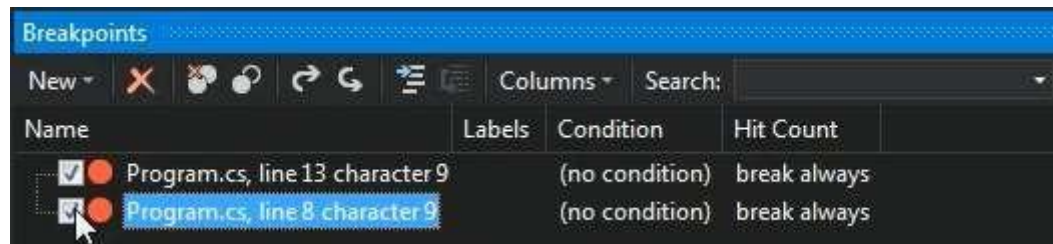
DISABLE A BREAKPOINT

- In a source, **Disassembly**, or **Call Stack** window, right-click on a line containing an enabled breakpoint glyph and choose **[Breakpoint | Disable Breakpoint]** from the shortcut menu.
- In the **Breakpoints** window, clear the checkbox next to an enabled breakpoint.
- Shortcut
 - Ctrl + F9

DISABLE A BREAKPOINT



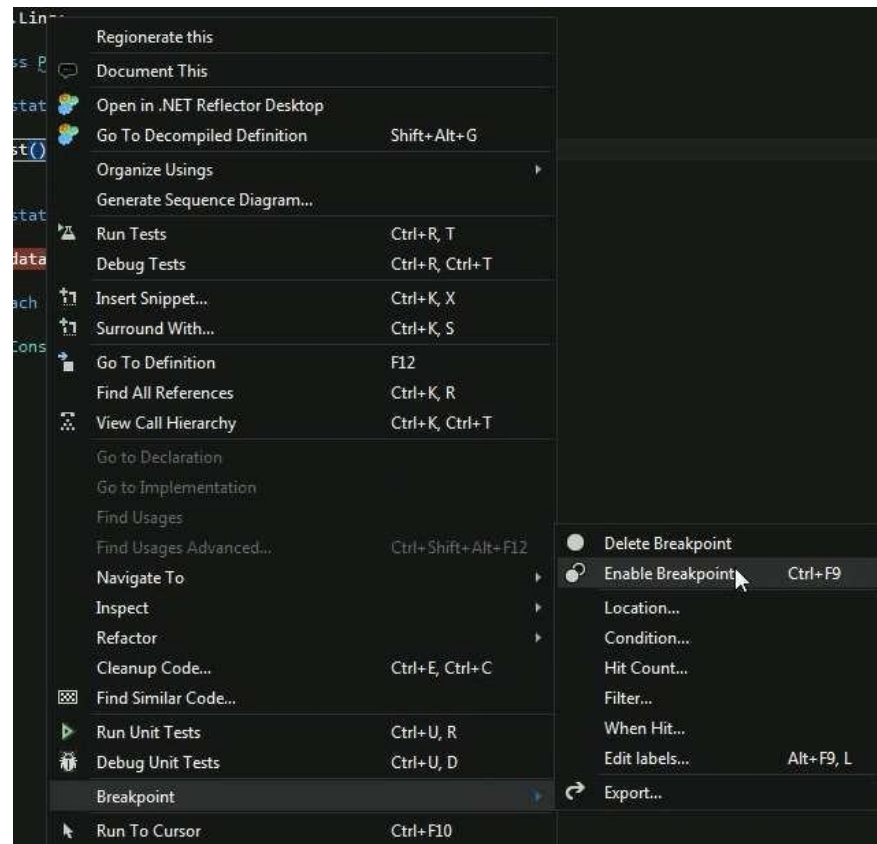
DISABLE A BREAKPOINT



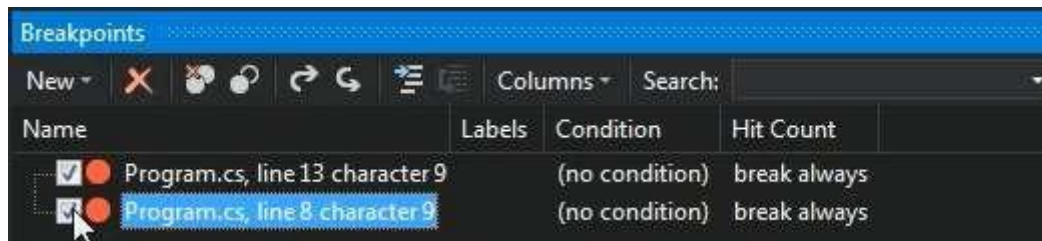
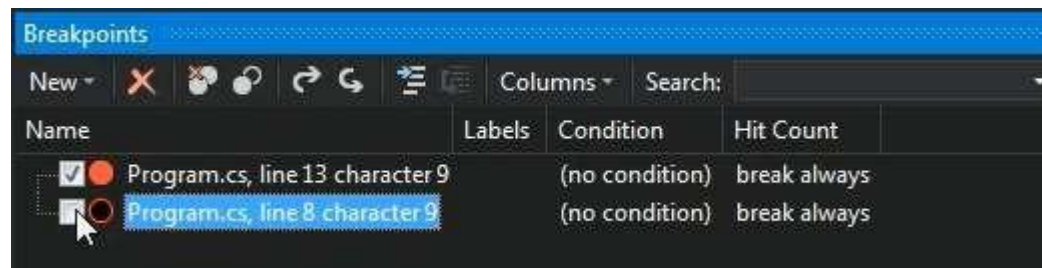
ENABLE A BREAKPOINT

- In a source, **Disassembly**, or **Call Stack** window, right-click on a line containing a disabled breakpoint glyph and choose **[Breakpoint | Enable Breakpoint]** from the shortcut menu.
- In the **Breakpoints** window, set the checkbox next to a disabled breakpoint.
- Shortcut
 - Ctrl + F9

ENABLE A BREAKPOINT



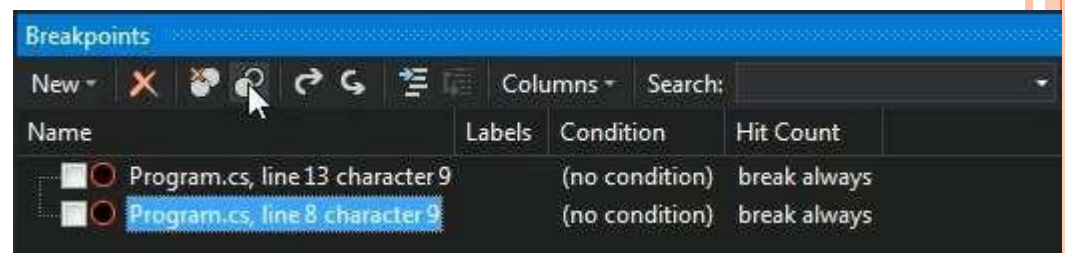
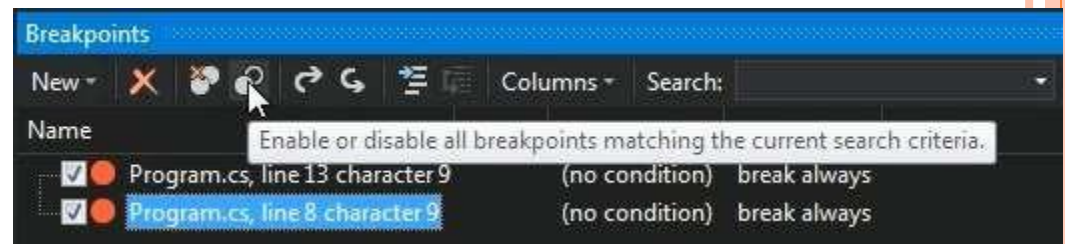
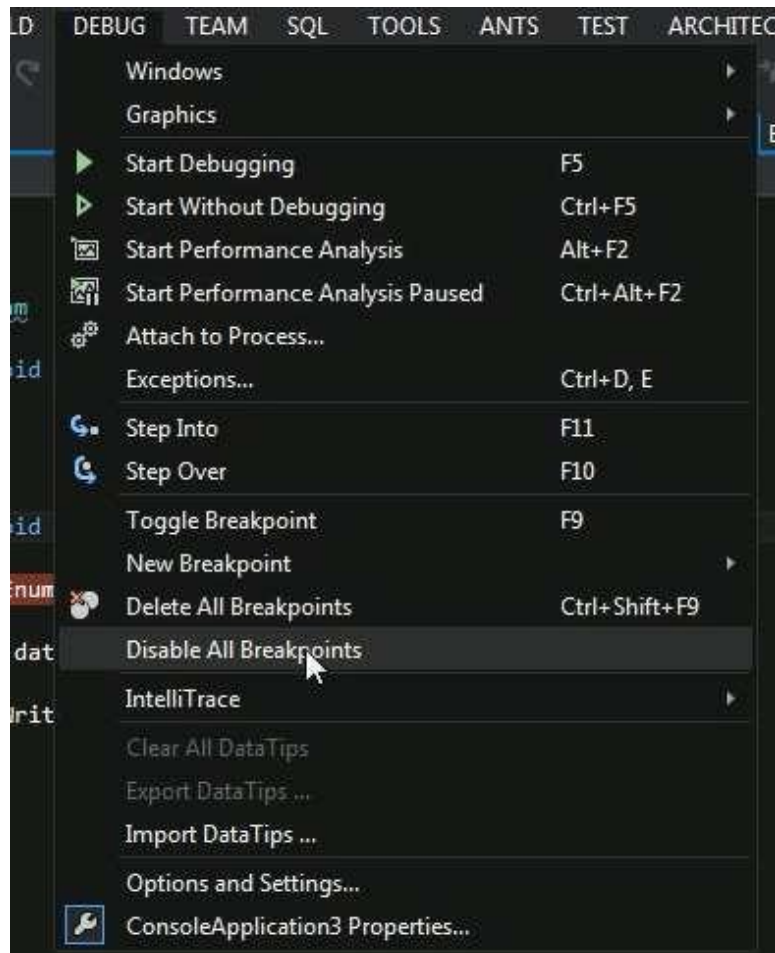
ENABLE A BREAKPOINT



DISABLE ALL BREAKPOINTS

- [**Debug | Disable All Breakpoints**]
- In the **Breakpoints** window, click disable-all button in toolbar.

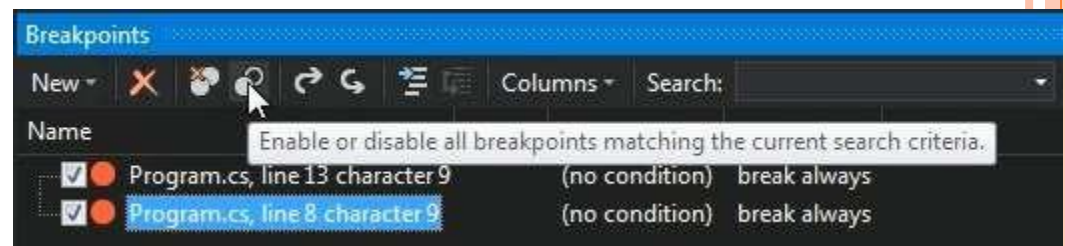
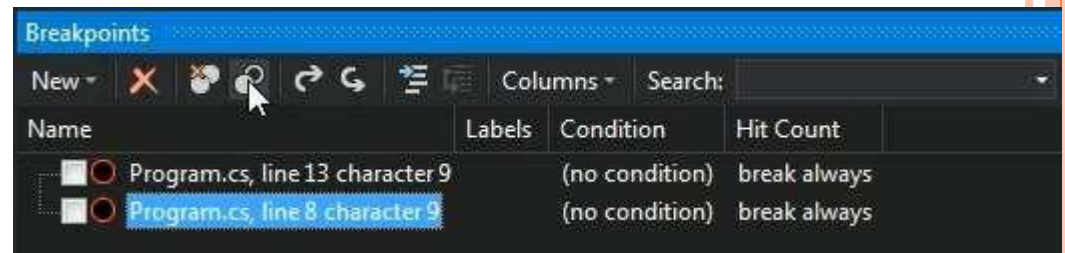
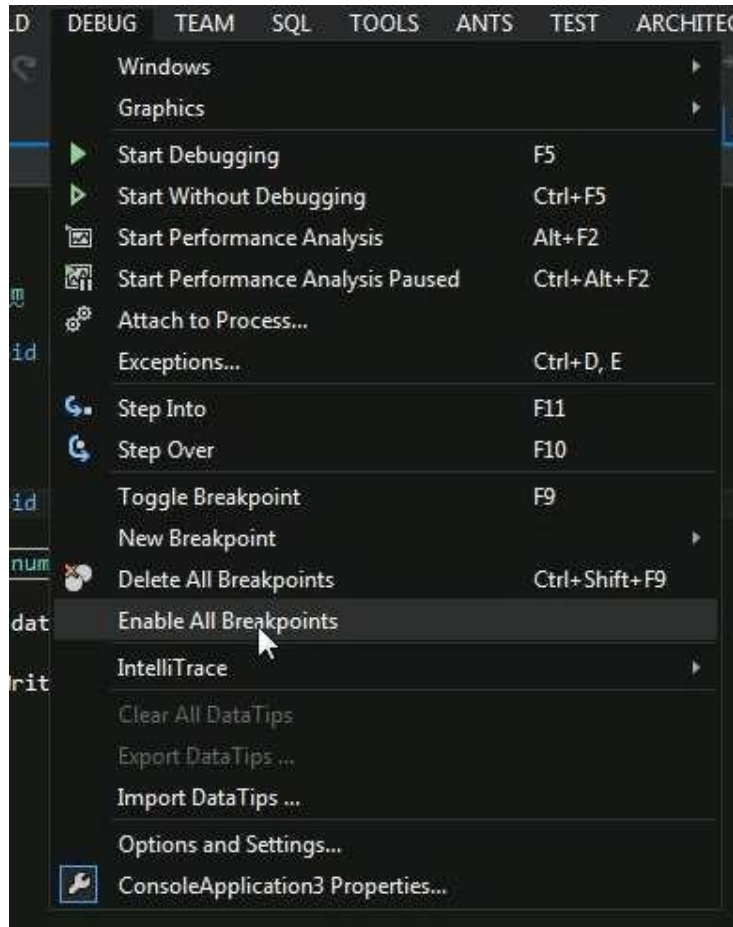
DISABLE ALL BREAKPOINTS



ENABLE ALL BREAKPOINTS

- [Debug | Enable All Breakpoints]
- In the **Breakpoints** window, click enable-all button in toolbar.

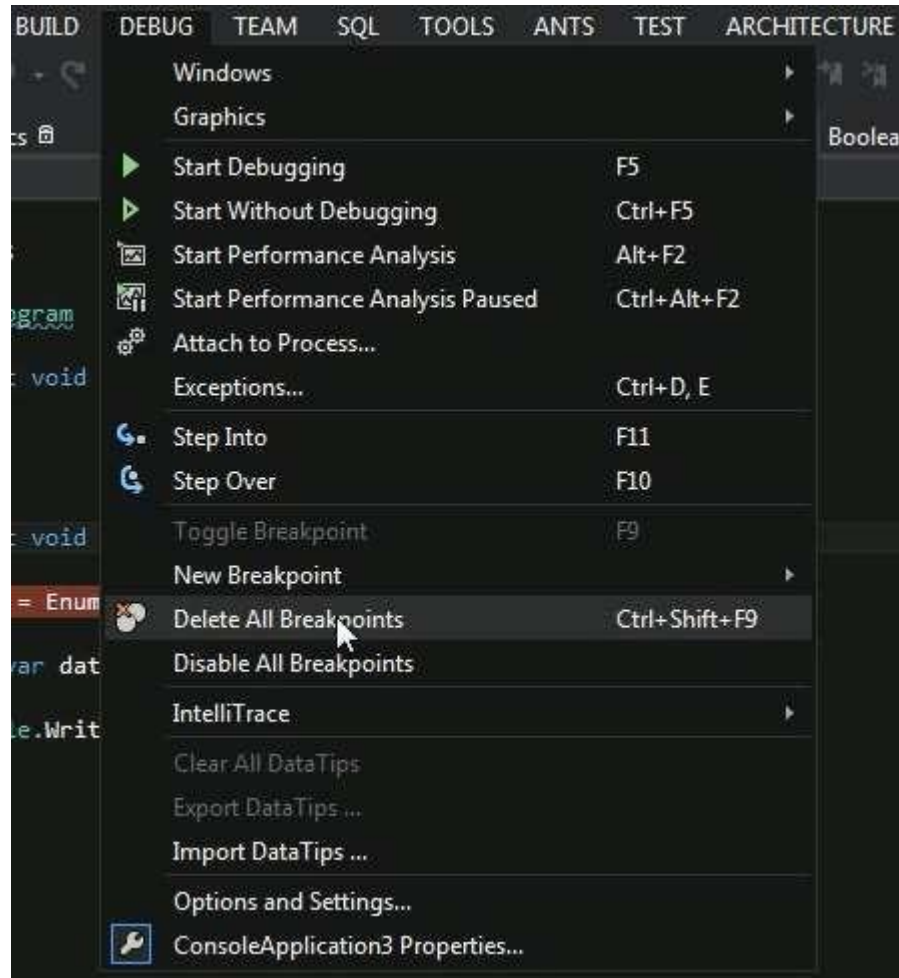
ENABLE ALL BREAKPOINTS



DELETE ALL BREAKPOINTS

- **[Debug | Delete All Breakpoints]**

DELETE ALL BREAKPOINTS



BREAKPOINT FILTER

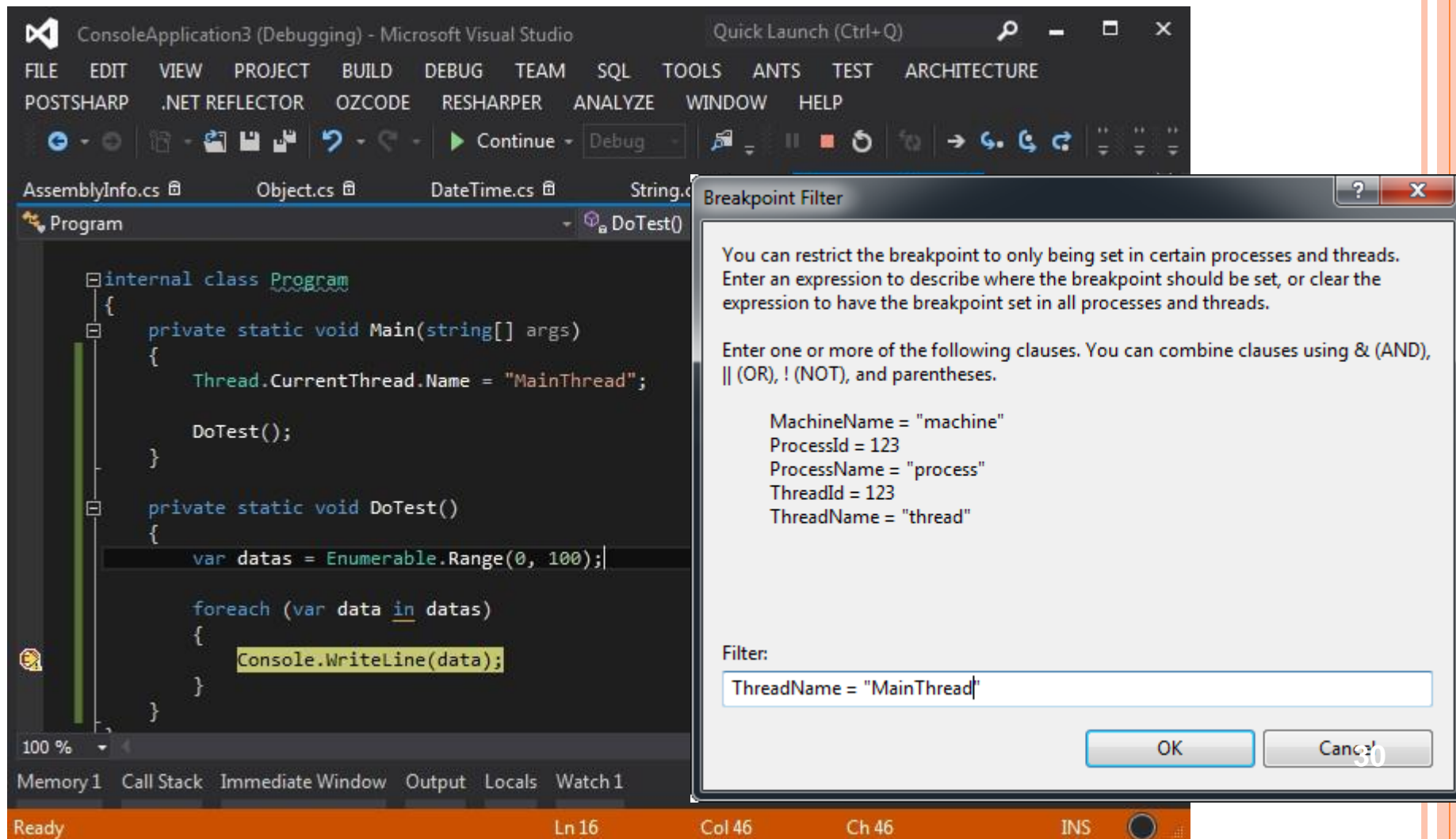
- Function
 - Limit the behavior of a breakpoint to specified machines, processes, and threads.
- **[Filter]** from the context menu in the Breakpoints window.
- In a source, Disassembly, or Call Stack, right-click a line containing a breakpoint glyph and choose **Filter** from **Breakpoints** in the shortcut menu.
- **[Filter]** from context menu in breakpoint glyph.

BREAKPOINT FILTER

The image illustrates the process of applying a breakpoint filter in Visual Studio. It is composed of three panels:

- Top Left:** A screenshot of the Visual Studio IDE showing a C# code file. The 'Breakpoints' menu is open, displaying options like 'New', 'Delete', 'Go To Source Code', 'Go To Disassembly', 'Location...', 'Condition...', 'Hit Count...', 'Filter...', 'When Hit...', 'Edit labels...', 'Export selected...', and 'Sort by'. The 'Filter...' option is highlighted.
- Top Right:** A screenshot of the 'Breakpoints' menu with the 'Filter...' option selected. The 'Filter...' dialog box is open, showing a list of breakpoints. The breakpoint 'Program.cs, line 8 character 9' is selected, and the 'Filter...' option is highlighted in the dialog.
- Bottom:** A screenshot of the 'Filter...' dialog box. The 'Filter...' option is selected, and the 'Filter...' dialog box is open, showing a list of breakpoints. The breakpoint 'Program.cs, line 8 character 9' is selected, and the 'Filter...' option is highlighted in the dialog.

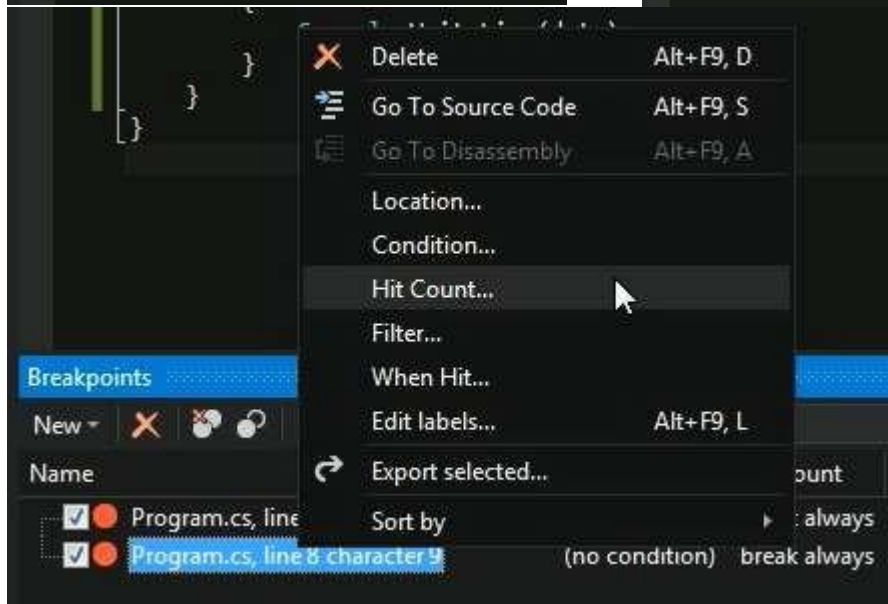
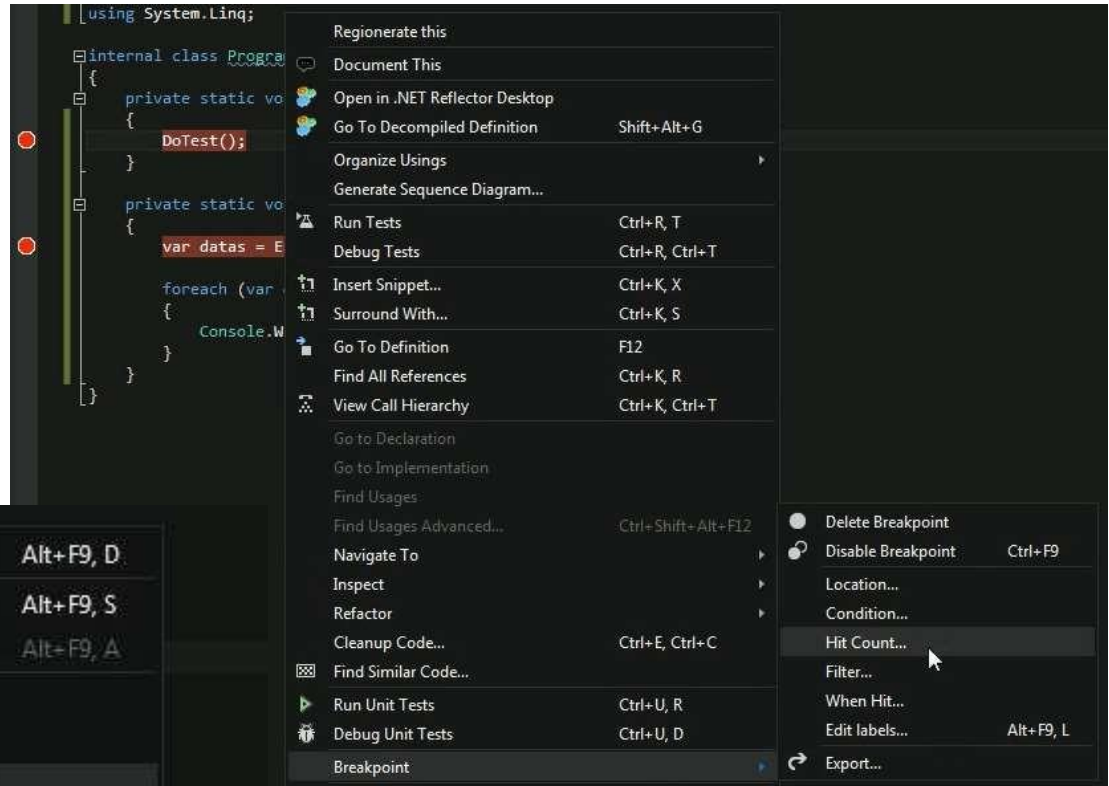
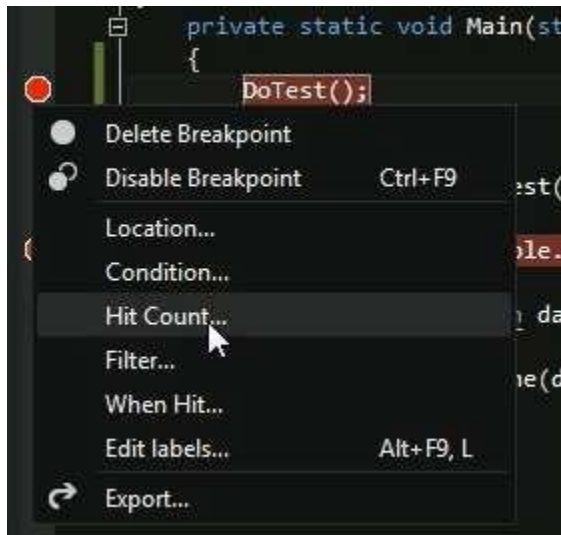
BREAKPOINT FILTER



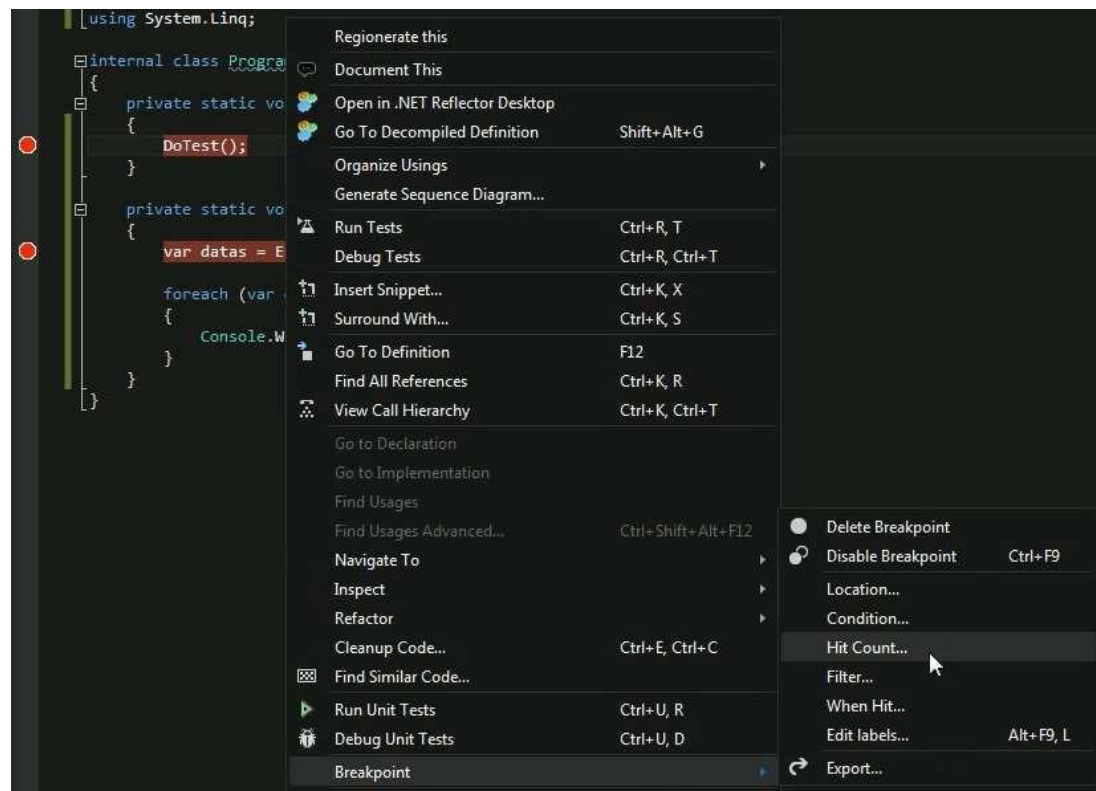
BREAKPOINT HIT COUNT

- Function
 - Keep track of how many times a breakpoint is hit.
- In the Breakpoints window, right-click on a breakpoint and choose **Hit Count** from the shortcut menu.
- **[Hit Count...]** from context menu in breakpoint glyph.
- In a source, Disassembly, or Call Stack window, right-click on a line containing a breakpoint and choose **[Breakpoints | Hit Count]** in the shortcut menu.

BREAKPOINT HIT COUNT



BREAKPOINT HIT COUNT



BREAKPOINT HIT COUNT

The screenshot shows the Visual Studio IDE with a console application named 'ConsoleApplication3 (Debugging)'. The 'Program.cs' file is open, showing the following code:

```
internal class Program
{
    private static void Main(string[] args)
    {
        DoTest();
    }

    private static void DoTest()
    {
        var datas = Enumerable.Range(0, 100);

        foreach (var data in datas)
        {
            Console.WriteLine(data);
        }
    }
}
```

A breakpoint is set on the line `Console.WriteLine(data);`. The 'Breakpoint Hit Count' dialog box is open, displaying the following information:

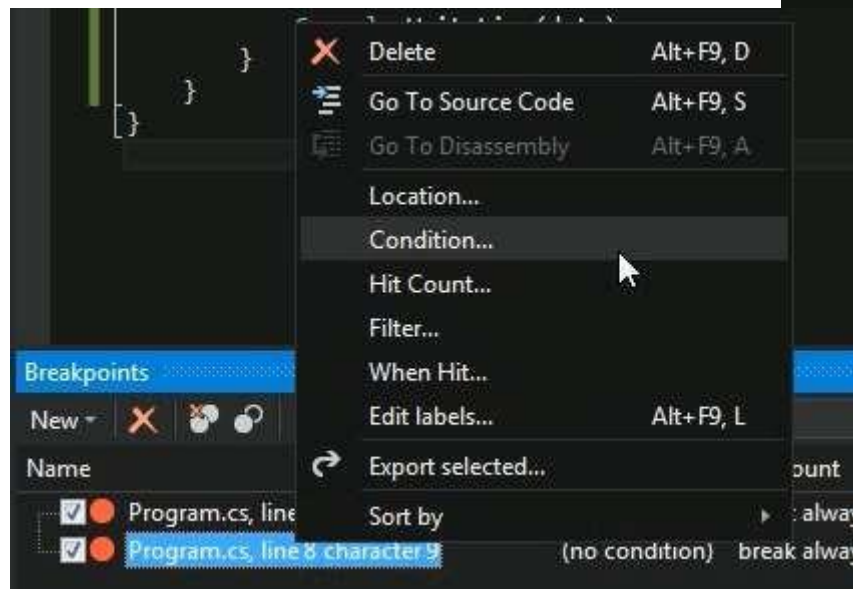
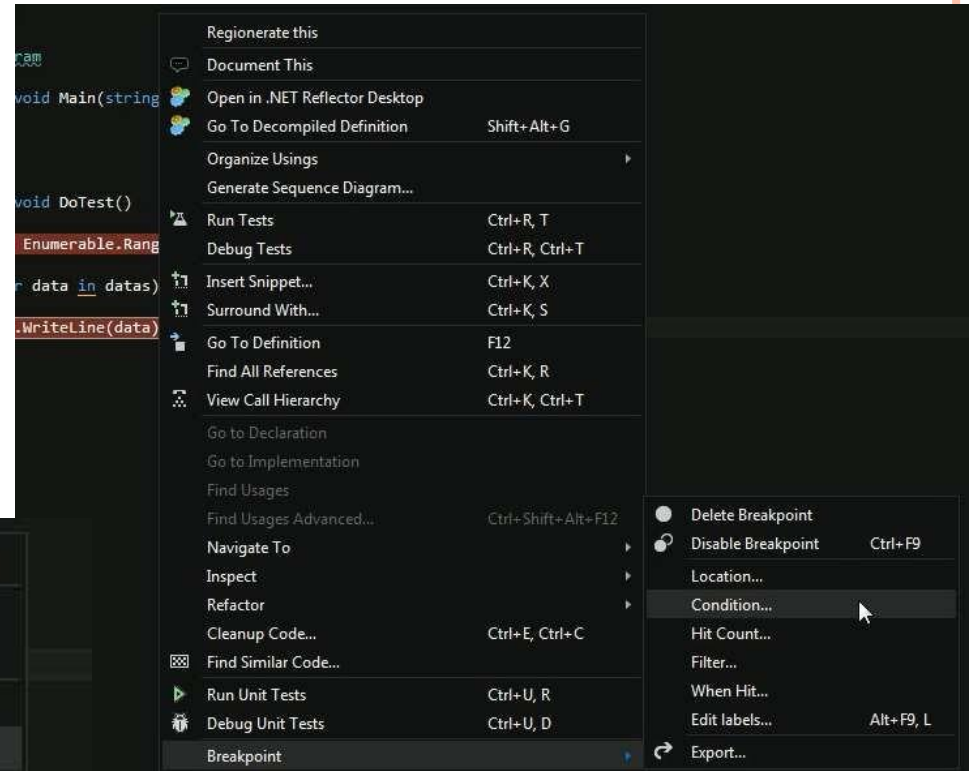
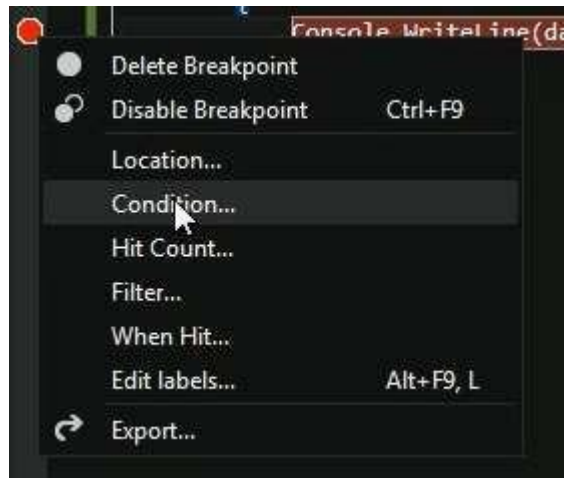
- Breakpoint Hit Count**
- A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.
- When the breakpoint is hit:
break when the hit count is greater than or equal to
- Current hit count: 0
- Buttons: Reset, OK, Cancel

The status bar at the bottom indicates 'Ready', 'Ln 18', 'Col 13', 'Ch 13', and 'INS'.

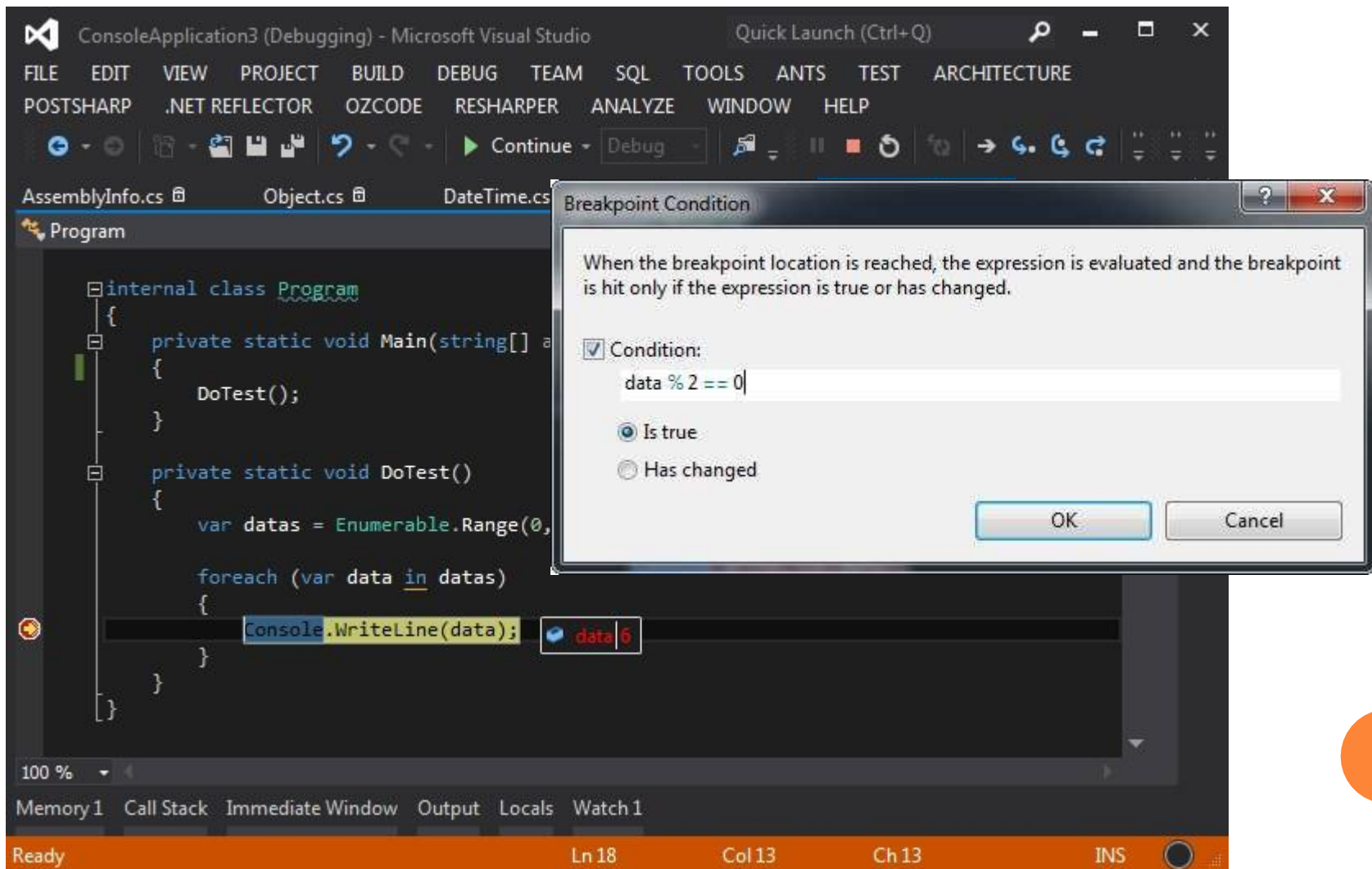
BREAKPOINT CONDITION

- Function
 - An expression that the debugger evaluates when a breakpoint is reached.
- In the **Breakpoints** window, right-click the line containing a breakpoint glyph and choose **Condition** from the shortcut menu.
- In a source, Disassembly, or Call Stack, right-click a line containing a breakpoint glyph and choose **Condition** from **Breakpoints** in the shortcut menu.
- **[Condition...]** from context menu in breakpoint glyph.

BREAKPOINT CONDITION



BREAKPOINT CONDITION



TRACE POINT

- Function
 - A tracepoint is a breakpoint with a custom action associated with it.
- In a source window, click a line where you want to set a tracepoint and choose **Insert Tracepoint** from **Breakpoints** in the shortcut menu.
- In a source, Disassembly, or Call Stack window, right-click a breakpoint glyph and choose **When Hit**.
- In the Breakpoints window, right-click a breakpoint glyph and choose **When Hit**.

TRACE POINT

The screenshot illustrates setting a trace point in Visual Studio. A breakpoint is set on the line `data.WriteLine(data);` within a `foreach` loop. The context menu for the breakpoint is open, with the 'When Hit...' option selected. The 'When Breakpoint Is Hit' dialog is displayed, allowing configuration of actions when the breakpoint is hit. The dialog shows that 'Print a message' and 'Continue execution' are both checked. The message field contains `data: {data}`. Below the message field, there is explanatory text about including variable values in curly braces and a list of special keywords that can be used in the message. The 'Output' window at the bottom shows the program's execution trace, including the exit of various threads and the program itself.

When Breakpoint Is Hit

Specify what to do when the breakpoint is hit.

☒ Print a message:

data: {data}

☒ Continue execution

You can include the value of a variable or other expression in the message by placing it in curly braces, such as "The value of x is {x}." To insert a curly brace, use `\{`. To insert a backslash, use `\\`.

The following special keywords will be replaced with their current values:

SADDRESS - Current Instruction, SCALLER - Previous Function Name,
SCALLSTACK - Call Stack, SFUNCTION - Current Function Name,
SPID - Process Id, SPNAME - Process Name
STID - Thread Id, STNAME - Thread Name

OK Cancel

Output

Show output from: Debug

```
data: 80
data: 82
data: 84
data: 86
data: 88
data: 90
data: 92
data: 94
data: 96
data: 98
The thread 'vshost.RunParkingWindow' (0x2ce8) has exited with code 0 (0x0).
The thread '<No Name>' (0x57e0) has exited with code 0 (0x0).
The program '[15300] ConsoleApplication3.vshost.exe: Managed (v2.0.50727)' has exited with code 0 (0x0).
The program '[15300] ConsoleApplication3.vshost.exe: Program Trace' has exited with code 0 (0x0).
```



CODE STEPPING

35

STEP INTO

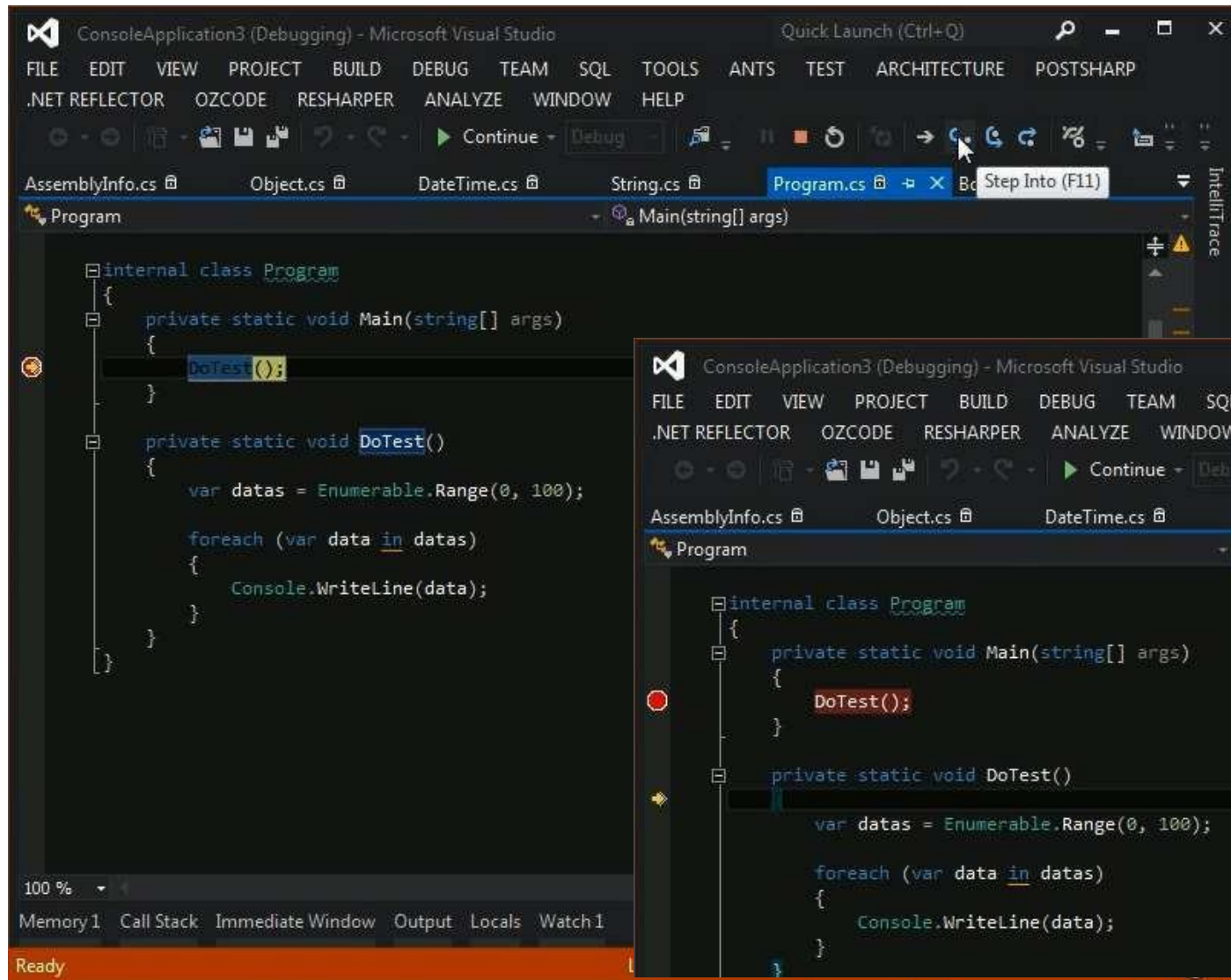
- Function

- If the line contains a function call, **Step Into** executes only the call itself, then halts at the first line of code inside the function. Otherwise, **Step Into** executes the next statement.

- ShortCut

- F11

STEP INTO



STEP OVER

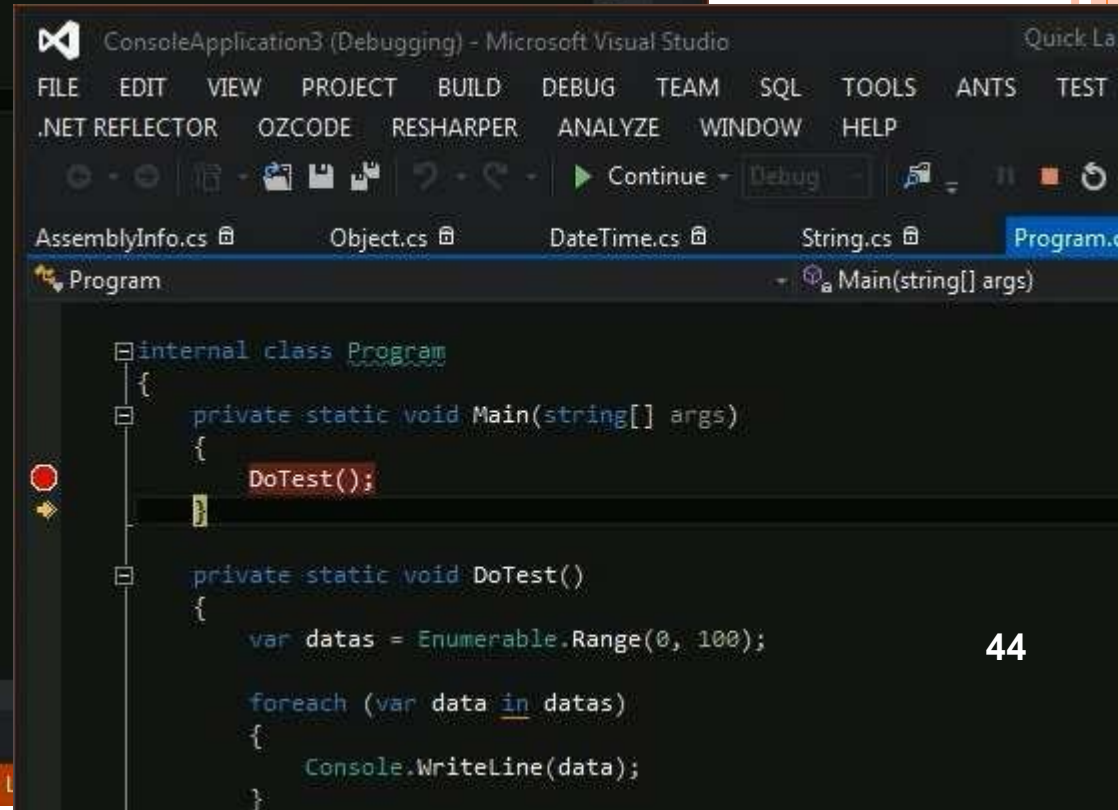
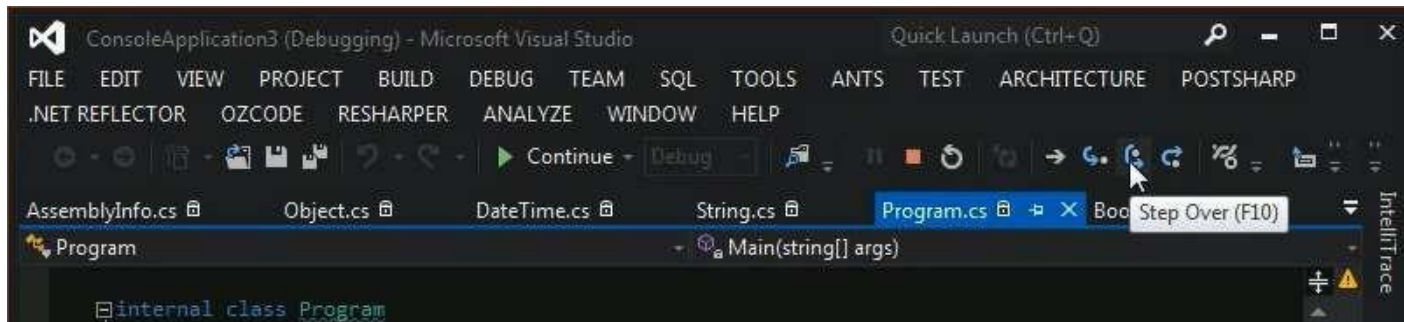
- Function

- If the line contains a function call, **Step Over** executes the called function, then halts at the first line of code inside the calling function. Otherwise, **Step Into** executes the next statement.

- ShortCut

- F10

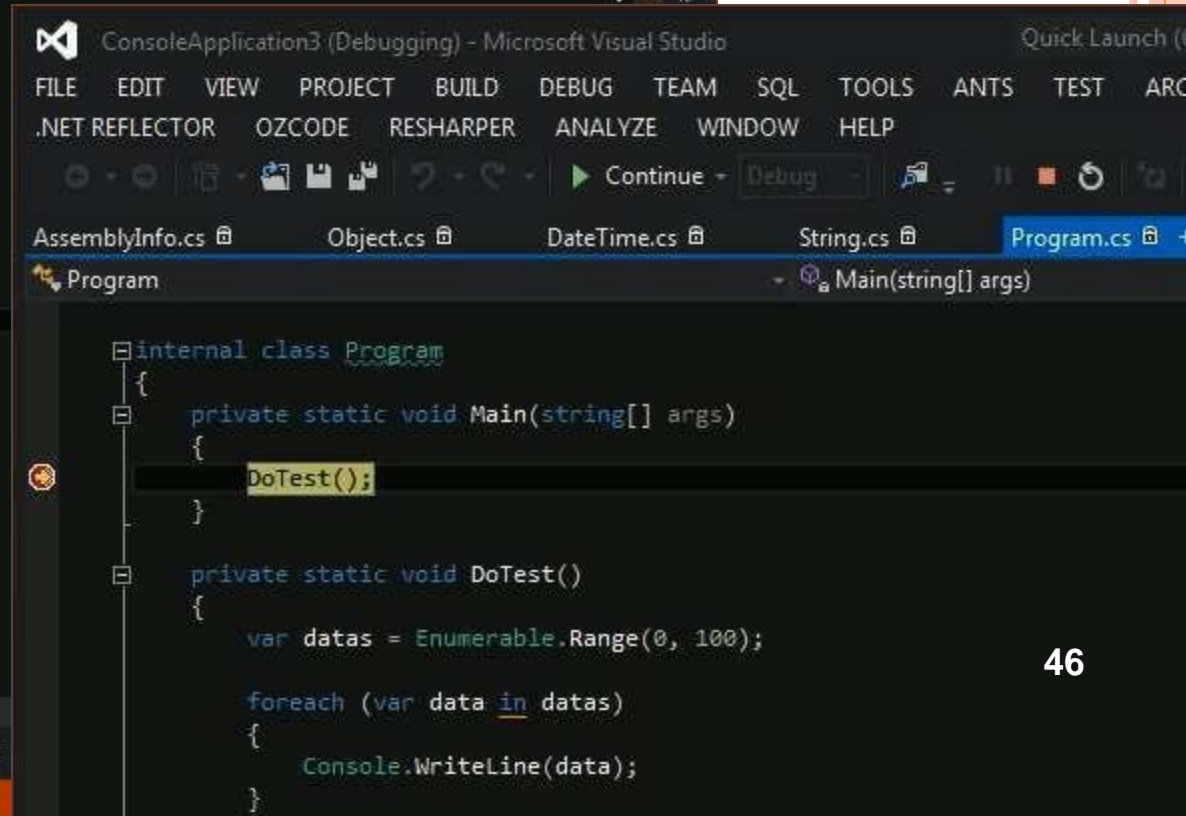
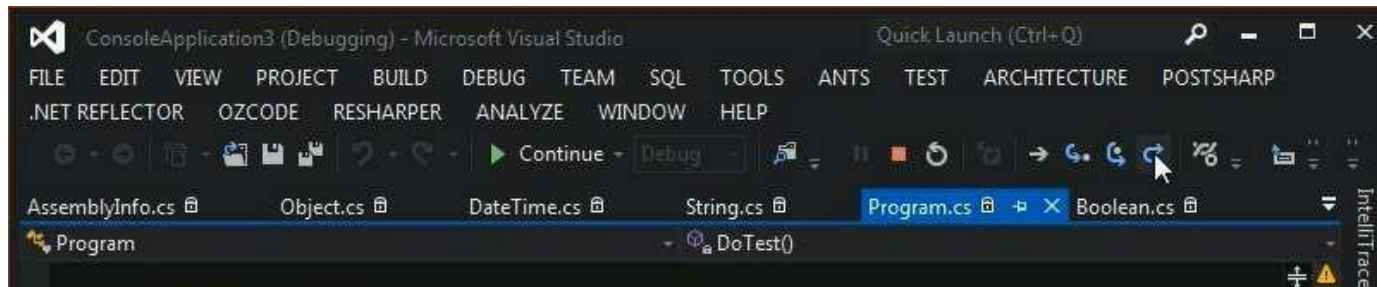
STEP OVER



STEP OUT

- Function
 - **Step Out** resumes execution of your code until the function returns, then breaks at the return point in the calling function.
- ShortCut
 - Shift + F11

STEP OUT

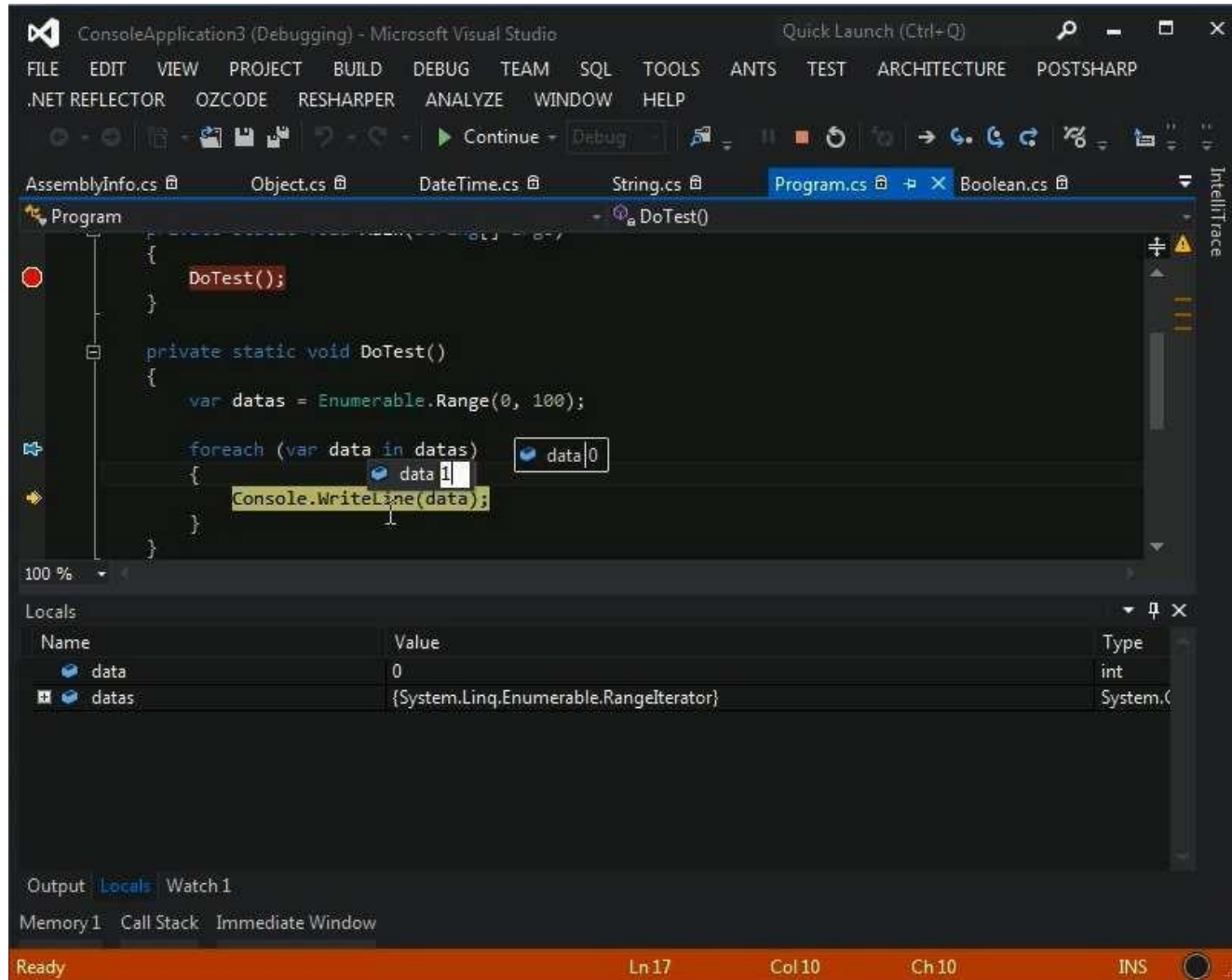




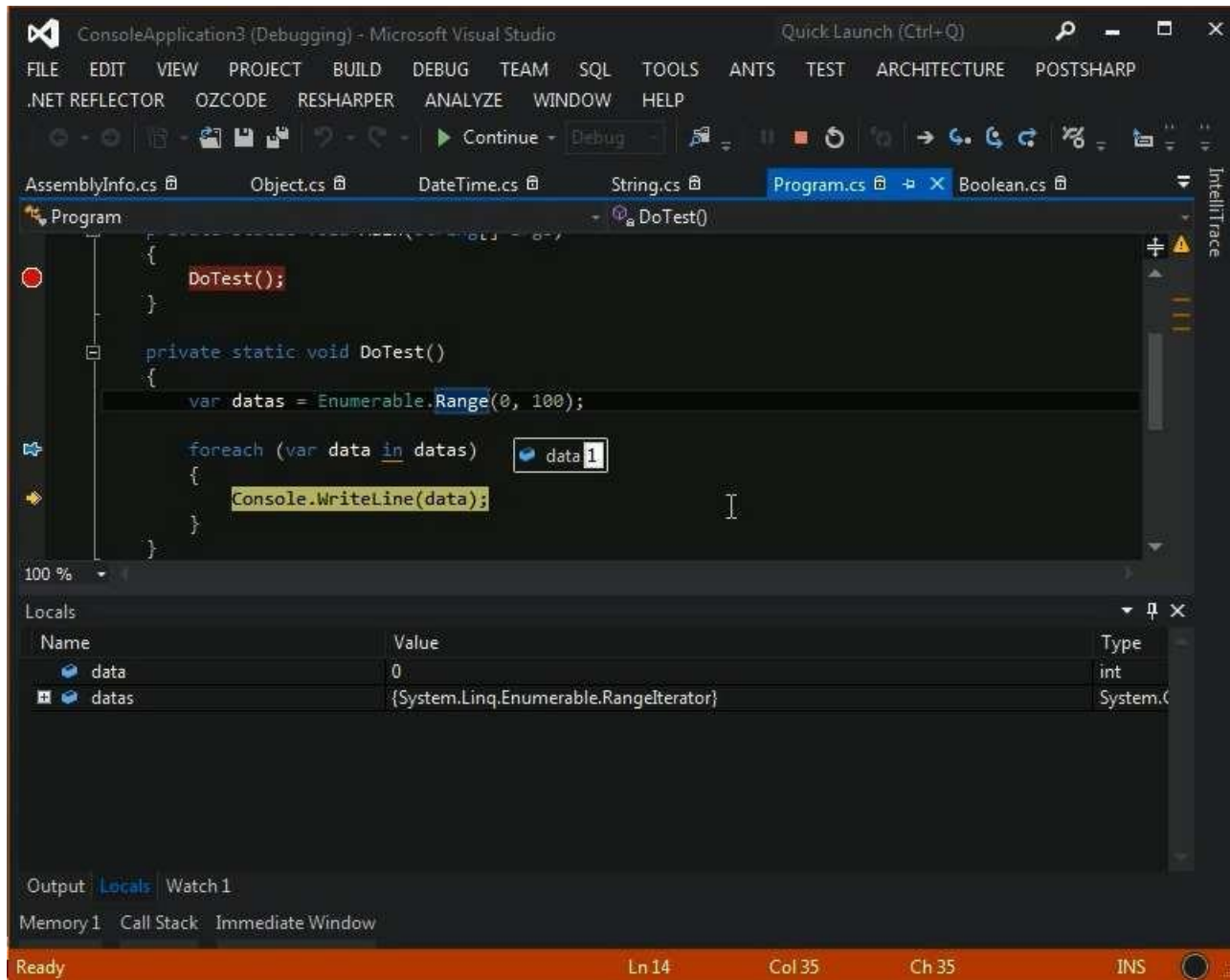
EDIT VARIABLE VALUE

42

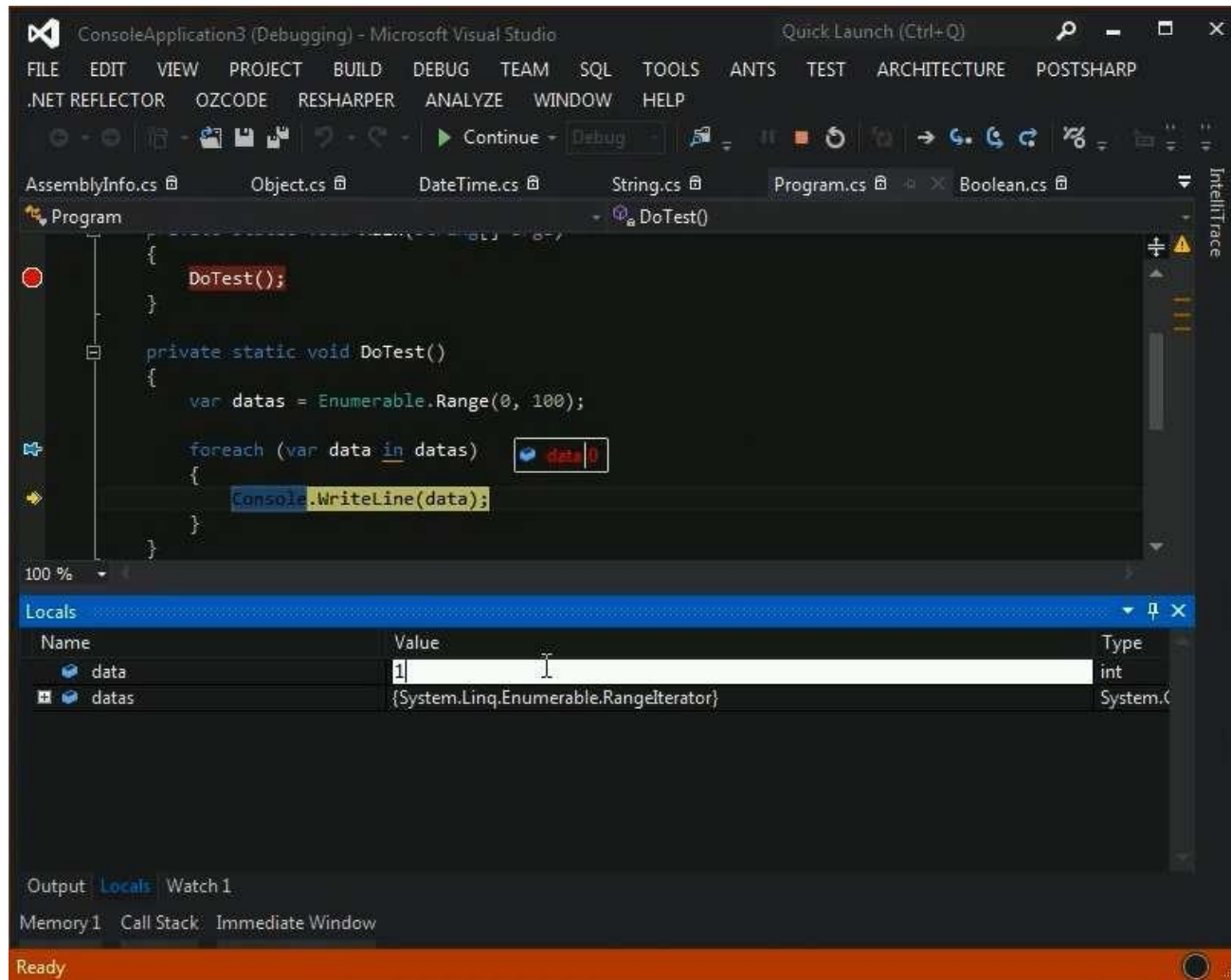
EDIT VARIABLE VALUE



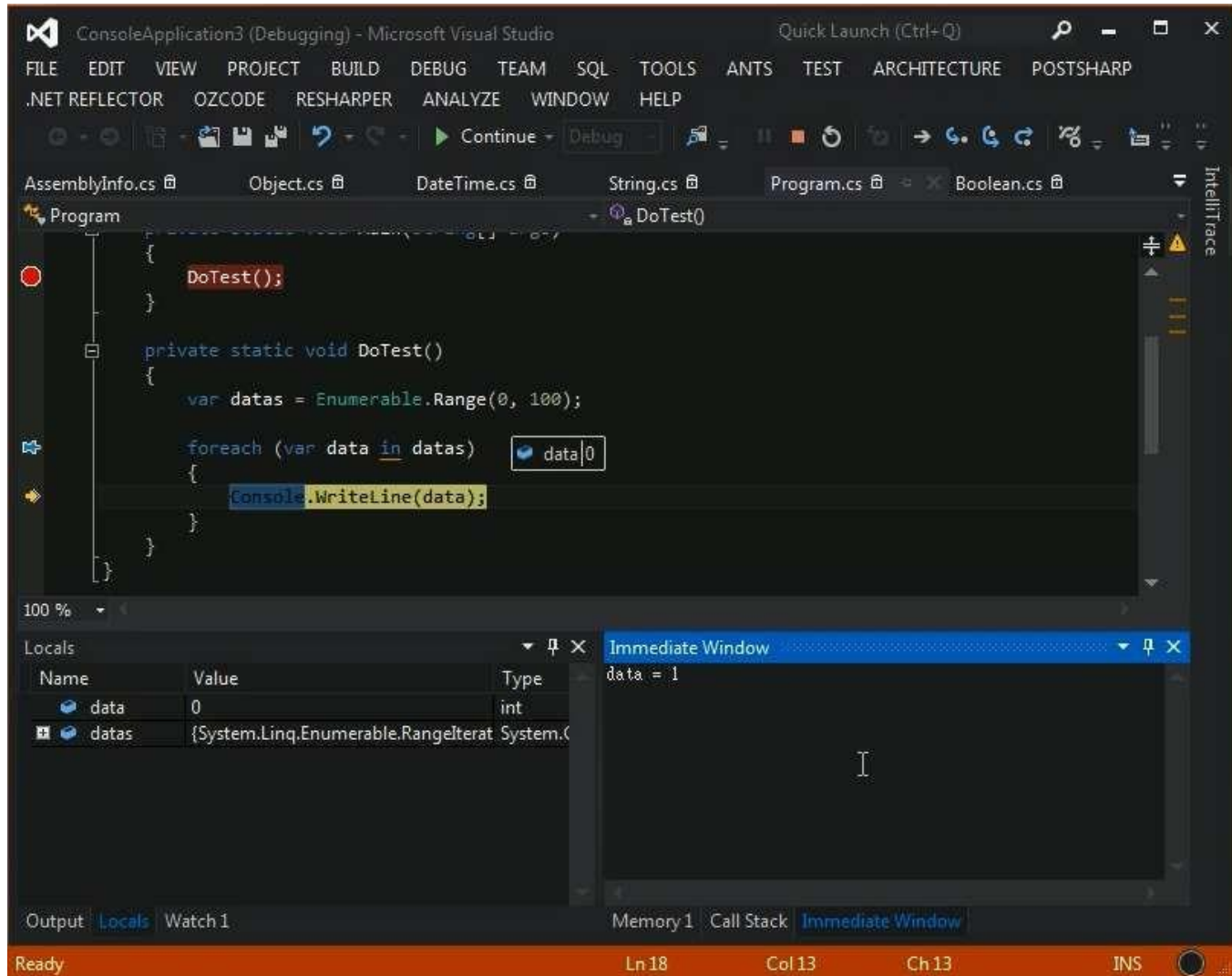
EDIT VARIABLE VALUE



EDIT VARIABLE VALUE



EDIT VARIABLE VALUE





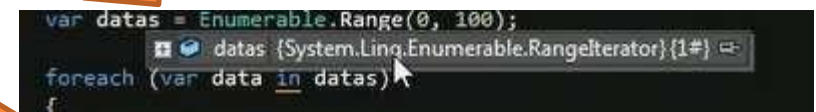
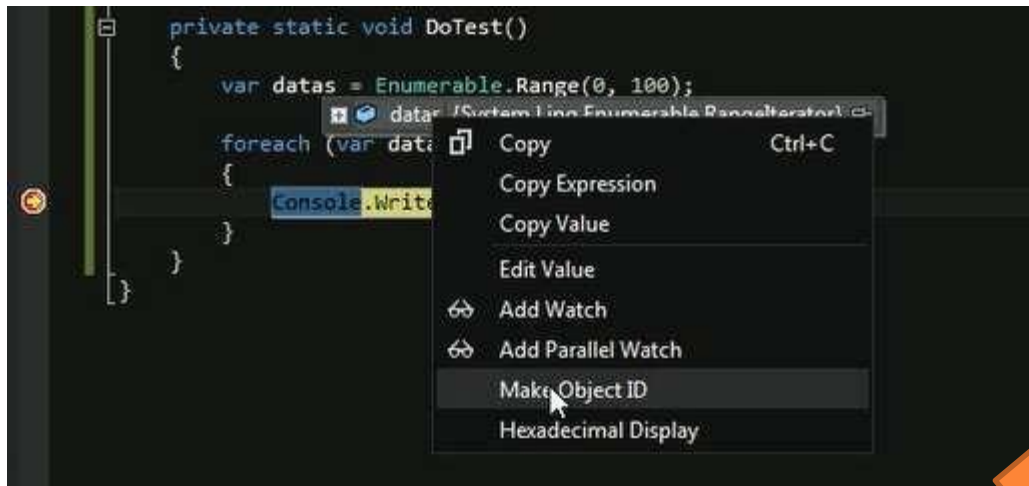
DEBUG WITH OBJECT ID

47

DEBUG WITH OBJECT ID

- Function
 - Identify debugging object

DEBUG WITH OBJECT ID



Watch 1	
Name	Value
1#	{System.Linq.Enumerable.RangeIterator} {1#}
count	100
start	0
System.Collections.Generic.IEnumerator<System.Int32>.Current	0
System.Collections.IEnumerator.Current	0
Results View	Expanding the Results View will enumerate the IEnumerable

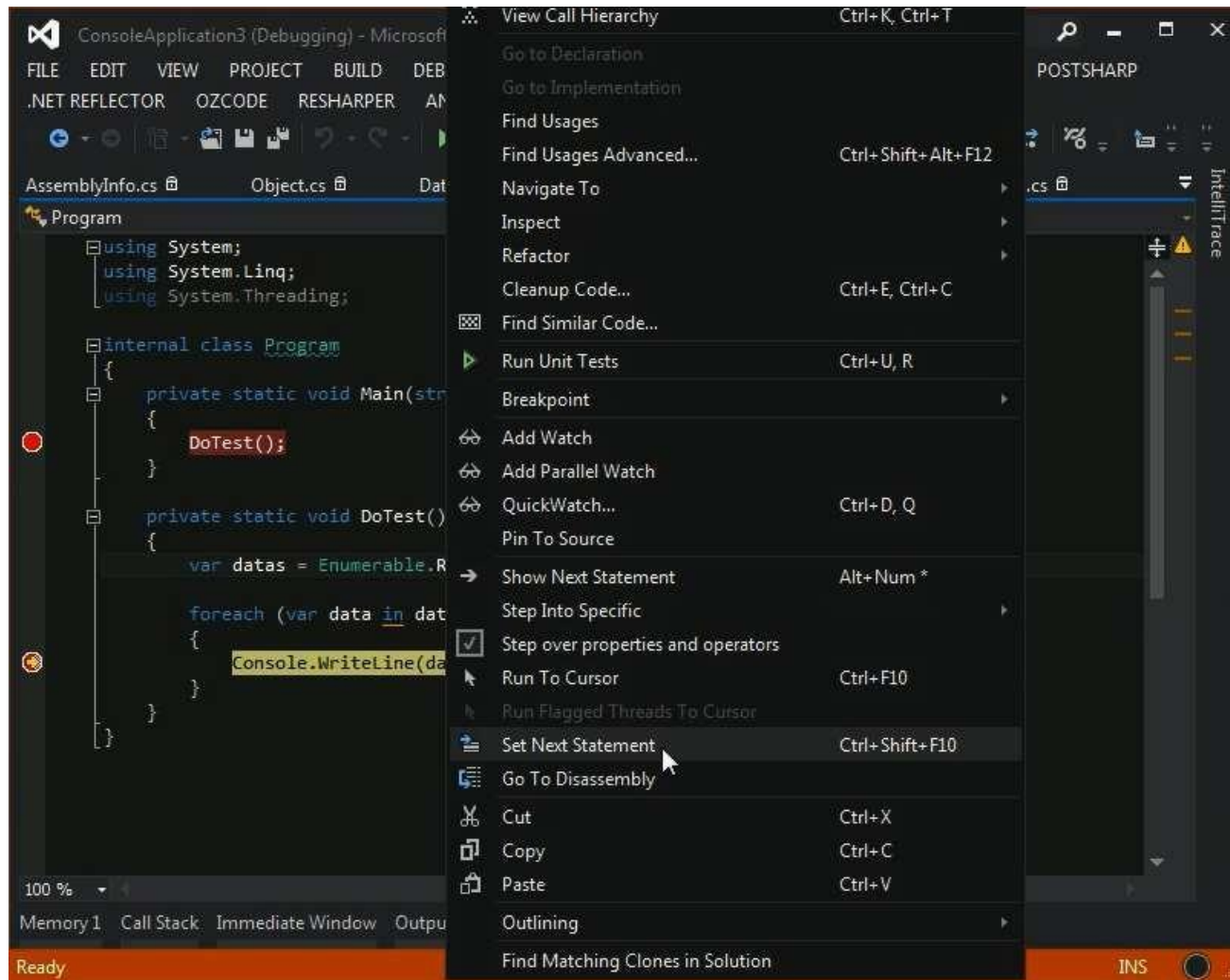


SET THE NEXT STATEMENT

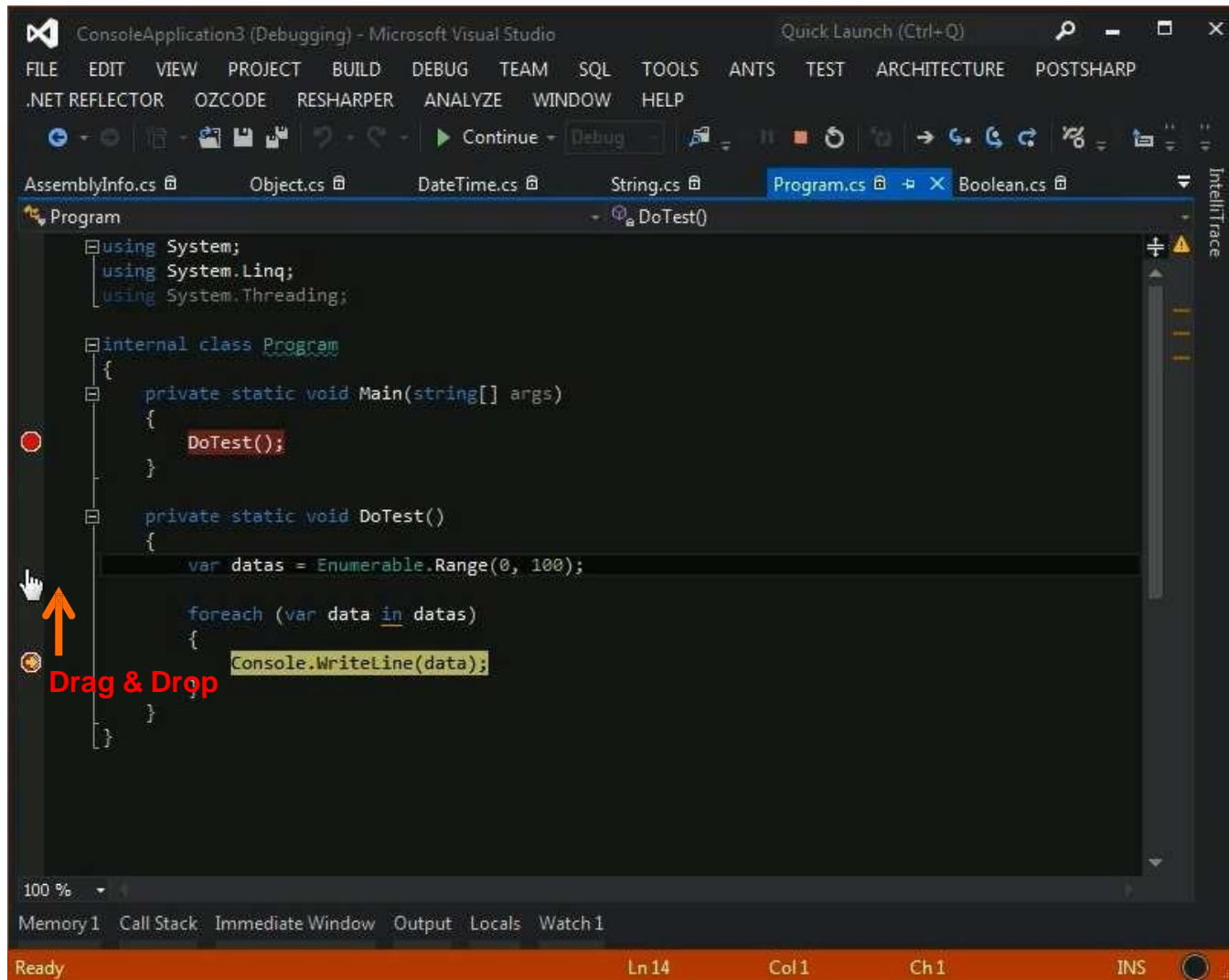
SET THE NEXT STATEMENT

- Function
 - Move the execution point to set the next statement of code to be executed
- In a source window, click the yellow arrowhead and drag it to a location where you want to set the next statement (in the same source file), or
- In a source window, right-click on the statement you want to execute next and choose **Set Next Statement** from the shortcut menu.
- In the **Disassembly** window, right-click the assembly-language instruction that you want to execute next and choose **Set Next Statement** from the shortcut menu.

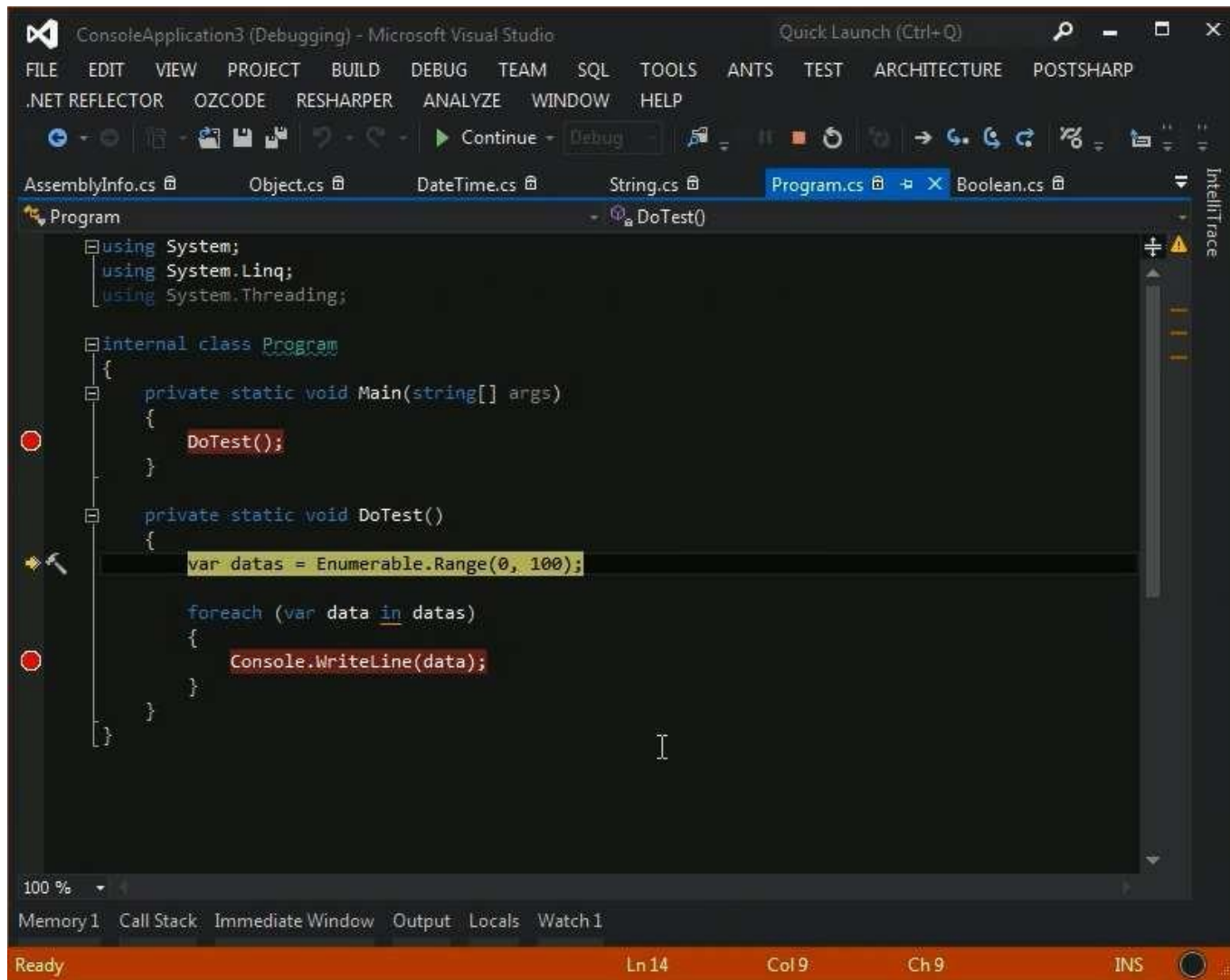
SET THE NEXT STATEMENT



SET THE NEXT STATEMENT



SET THE NEXT STATEMENT





EDIT AND CONTINUE

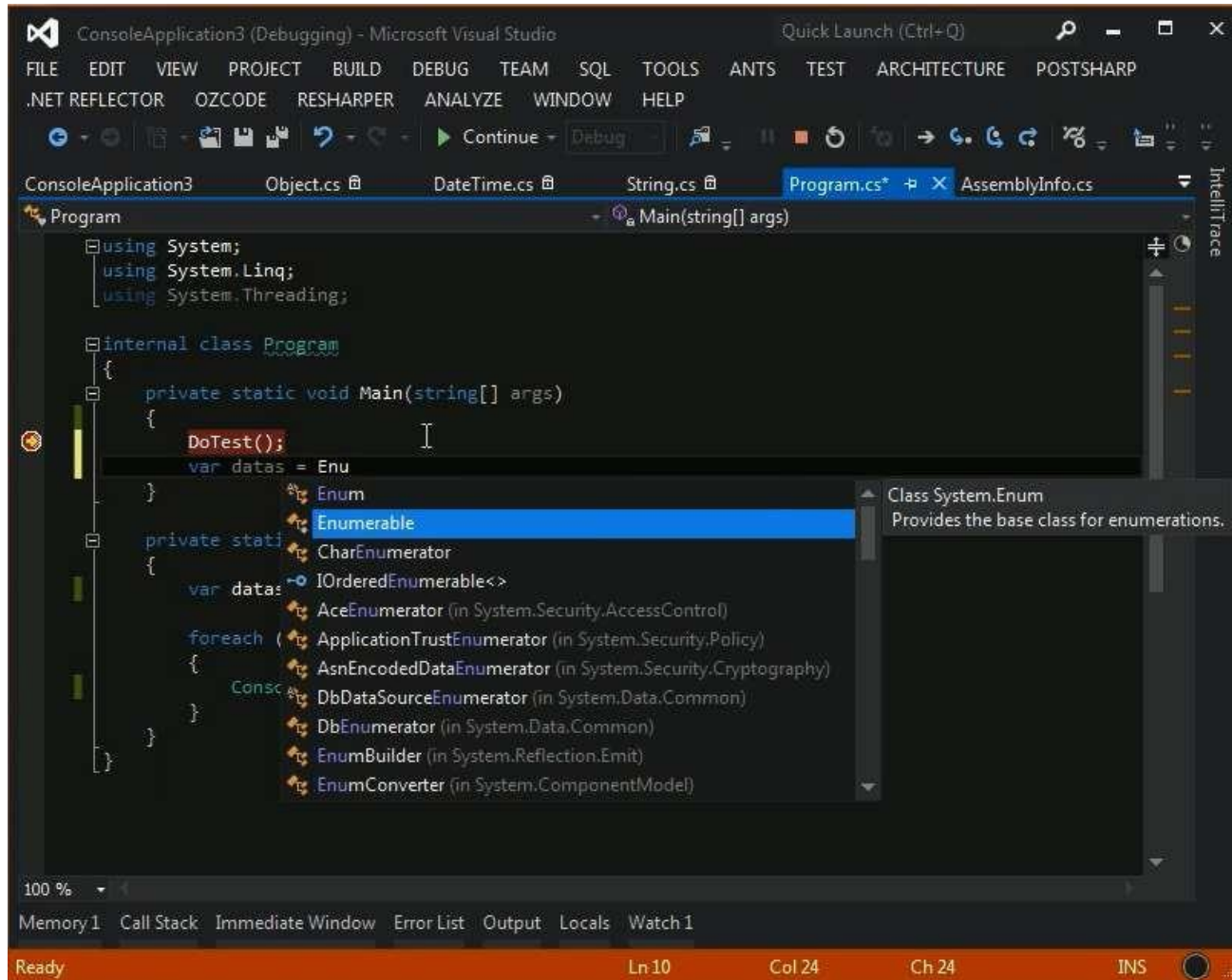
55

EDIT AND CONTINUE

- Function

- A time-saving feature that enables you to make changes to your source code while your program is in break mode

EDIT AND CONTINUE

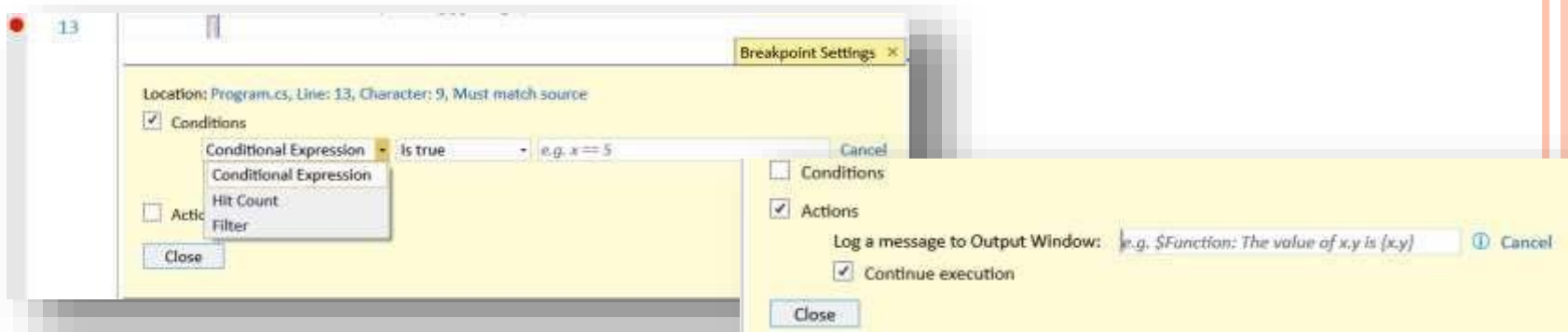


Breakpoint Settings Window

- Two new icons appear with breakpoint

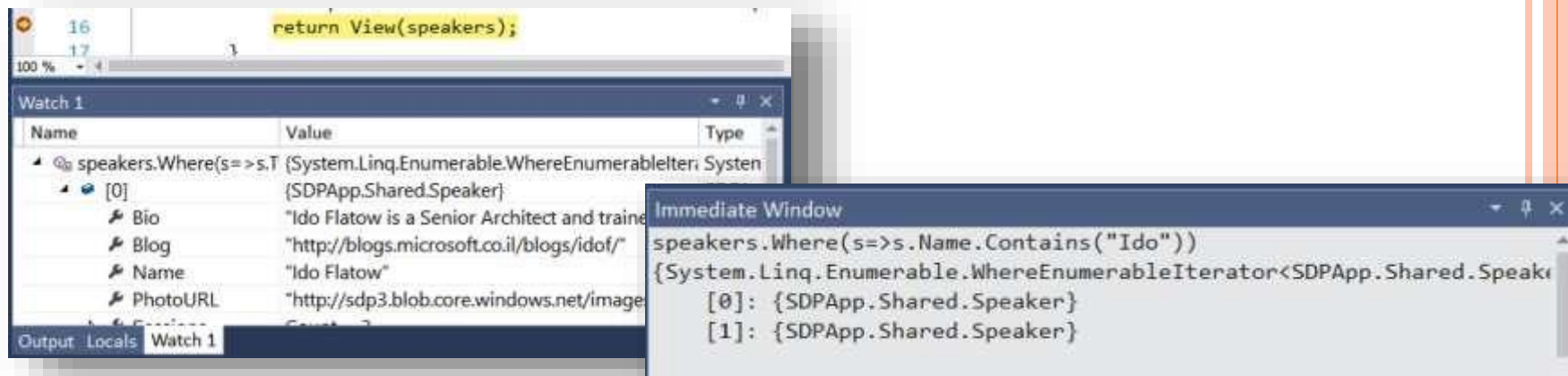


- Settings opens a Peek window



Lambda in Debug Windows

- Watch & Immediate window support lambda expressions
- Rebuilt on top of the “Roslyn” compilers



PerfTips

- Performance Information at-a-glance
- Excludes major debugger related overhead
 - Time stopped under debugger
 - Symbol loading
- Suitable for order of magnitude measurements

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultC

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

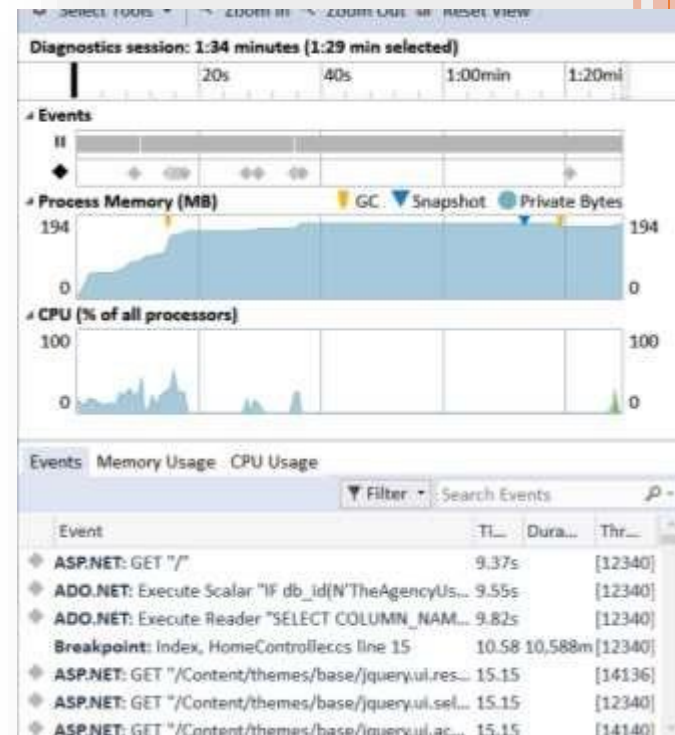
    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}
```

); 329ms elapsed



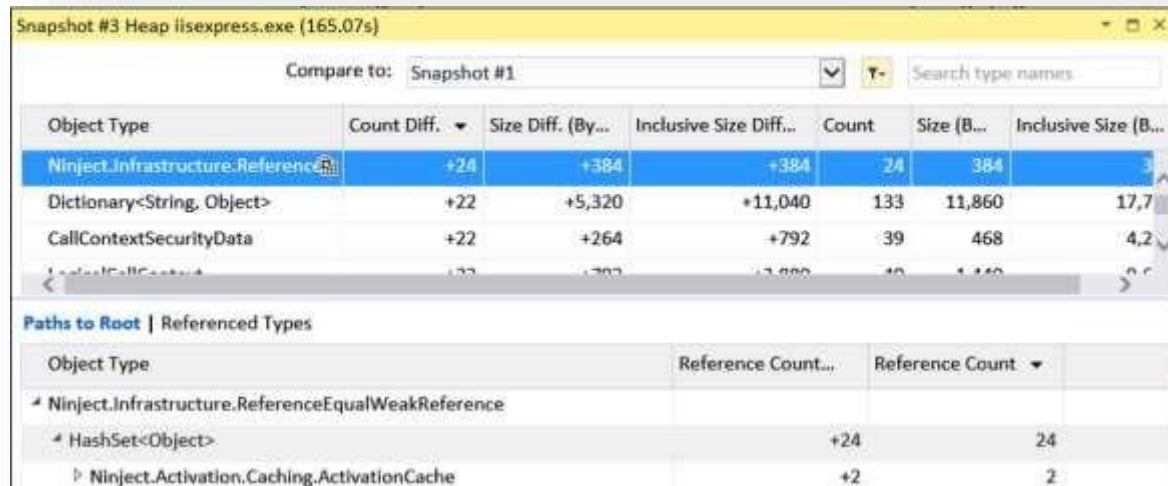
Diagnostics Tool Window

- Just hit F5
- Memory and CPU usage graphs
- IntelliTrace UI is now part of the window
- Take memory snapshots and profile CPU usage
- Time sections of code with PerfTips



Diagnostics Tool Window Memory Usage – Cntd.

- Use memory snapshots to inspect the heap and to find memory leaks
 - Watch which objects were added, and their size
 - Track object paths to its root



Snapshot #3 Heap ilisexpress.exe (165.07s)

Compare to: Snapshot #1

Object Type	Count Diff.	Size Diff. (By...	Inclusive Size Diff...	Count	Size (B...	Inclusive Size (B...
Ninject.Infrastructure.ReferenceEqualWeakReference	+24	+384	+384	24	384	384
Dictionary<String, Object>	+22	+5,320	+11,040	133	11,860	17,7
CallContextSecurityData	+22	+264	+792	39	468	4,2
LocalCallContext	+22	+302	+1,000	40	1,160	1,160

Paths to Root | Referenced Types

Object Type	Reference Count...	Reference Count
Ninject.Infrastructure.ReferenceEqualWeakReference		
HashSet<Object>	+24	24
Ninject.Activation.Caching.ActivationCache	+2	2



58



REFERENCE

REFERENCE

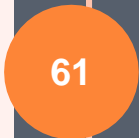
- Breakpoints and Tracepoints
 - [https://msdn.microsoft.com/en-us/library/ktf38f66\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/ktf38f66(v=vs.90).aspx)
- Edit and Continue
 - <https://msdn.microsoft.com/en-us/library/bcew296c.aspx>
- Debugger Tips, Tricks and Tools #6 - Gooney Bugs - Site Home - MSDN Blogs
 - <http://blogs.msdn.com/b/jimgries/archive/2005/11/16/493431.aspx>
- [C#][Visual Studio]Debug With Object ID - Level Up- 點部落
 - <http://www.dotblogs.com.tw/larrynung/archive/2011/05/05/24296.aspx>

REFERENCE

- [Visual Studio]追蹤點(Tracepoint)的使用 - Level Up-點部落
 - <http://www.dotblogs.com.tw/larrynung/archive/2009/11/04/11399.aspx>
- How to: Set the Next Statement
 - <https://msdn.microsoft.com/en-us/library/09yze4a9%28VS.80%29.aspx>
- Start, Break, Step, Run through Code, and Stop Debugging in Visual Studio
 - <https://msdn.microsoft.com/en-us/library/y740d9d3.aspx>



Q & A



61



QUESTION & ANSWER

