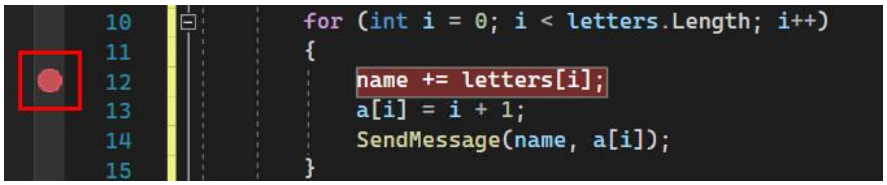
1. Debugging in Visual Studio

1.1 Introduction

- When you *debug your app*, it usually means that you're running your application with the debugger attached. When you do this task, the debugger provides many ways to see what your code is doing while it runs. You can step through your code and look at the values stored in variables, you can set watches on variables to see when values change, you can examine the execution path of your code, see whether a branch of code is running, and so on.
- To start the debugger, select F5, or choose the Debug Target button in the Standard toolbar, or choose the Start Debugging button in the Debug toolbar, or choose Debug > Start Debugging from the menu bar.
- To stop the debugger, select Shift+F5, or choose the Stop Debugging button in the Debug toolbar, or choose Debug > Stop Debugging from the menu bar.

1.2 Debug points

- Breakpoints are an essential feature of reliable debugging. You can set breakpoints where you want Visual Studio to pause your running code so you can look at the values of variables or the behaviour of memory, or know whether or not a branch of code is getting run.
- A red circle appears where you set the breakpoint.



1.3 Different Debug windows

- List of different debug windows are as follows :

Window	Hotkey	See topic
Breakpoints	CTRL+ALT+B	Use Breakpoints
Exception Settings	CTRL+ALT+E	Manage Exceptions with the Debugger
Output	CTRL+ALT+O	Output Window
Watch	CTRL+ALT+W, (1, 2, 3, 4)	Watch and QuickWatch Windows

QuickWatch	SHIFT+F9	Watch and QuickWatch Windows
Autos	CTRL+ALT+V, A	Autos and Locals Windows
Locals	CTRL+ALT+V, L	Autos and Locals Windows
Call Stacks	CTRL+ALT+C	How to: Use the Call Stack Window
Immediate	CTRL+ALT+I	Immediate Window
Parallel Stacks	CTR:+SHIFT+D, S	Using the Parallel Stacks Window
Parallel Watch	CTR:+SHIFT+D, (1, 2, 3, 4)	Get started Debugging Multithreaded Applications
Threads	CTRL+ALT+H	Debug using the Threads Window
Modules	CTRL+ALT+U	How to: Use the Modules Window
GPU Threads	-	How to: Use the GPU Threads Window
Tasks	CTR:+SHIFT+D, K	Using the Tasks Window
Python Debug Interactive	SHIFT+ALT+I	Python Interactive REPL
Live Visual Tree	-	Inspect XAML properties while debugging
Live Property Explorer	-	Inspect XAML properties while debugging
Processes	CTRL+ALT+Z	Debug Threads and Processes
Memory	CTRL+ALT+M, (1, 2, 3, 4)	Memory Windows
Disassembly	CTRL+ALT+D	How to: Use the Disassembly Window
Registers	CTRL+ALT+G	How to: Use the Registers Window

1.4 Editing

- With Hot Reload, or Edit and Continue for C#, you can make changes to your code in break or run mode while debugging. The changes can be applied without having to stop and restart the debugging session.
- To enable or disable Hot Reload:
    - If you're in a debugging session, stop debugging (**Debug** > **Stop Debugging** or **Shift**+**F5**).
    - Open **Tools** > **Options** > **Debugging** > **.NET/C++ Hot Reload**, select or clear the **Enable Hot Reload and Edit and Continue when debugging** check box.
- To use the classic Edit and Continue experience:
    - While debugging, in break mode, make a change to your source code.

    - From the **Debug** menu, click **Continue**, **Step**, or **Set Next Statement**. Debugging continues with the new, compiled code.
- Some types of code changes are not supported by Edit and Continue.

1.5 Conditional break points

- You can control when and where a breakpoint executes by setting conditions. The condition can be any valid expression that the debugger recognizes.
- **To set a breakpoint condition:**
    - Right-click the breakpoint symbol and select **Conditions** (or press **Alt** + **F9**, **C**). Or hover over the breakpoint symbol, select the **Settings** icon, and then select **Conditions** in the **Breakpoint Settings** window.
    - In the dropdown, select **Conditional Expression**, **Hit Count**, or **Filter**, and set the value accordingly.
    - Select **Close** or press **Ctrl**+**Enter** to close the **Breakpoint Settings** window. Or, from the **Breakpoints** window, select **OK** to close the dialog.
- When you select **Conditional Expression**, you can choose between two conditions: **Is true** or **When changed**. Choose **Is true** to break when the expression is satisfied, or **When changed** to break when the value of the expression has changed.
- If you suspect that a loop in your code starts misbehaving after a certain number of iterations, you can set a breakpoint to stop execution after that number of hits, rather than having to repeatedly press **F5** to reach that iteration.
- Under **Conditions** in the **Breakpoint Settings** window, select **Hit Count**, and then specify the number of iterations.
- **To set a dependent breakpoint:**
    - Right-click in the far left margin next to a line of code and select **Insert Dependent Breakpoint** from the context menu.
- Dependent breakpoints don't work if there is only a single breakpoint in your application.

- Dependent breakpoints are converted to normal line breakpoint if the prerequisite breakpoint is deleted.
- **To set a temporary breakpoint:**
    - Right-click in the far left margin next to a line of code and select **Insert Temporary Breakpoint** from the context menu.

1.6 Data inspector

- Data inspection can be done in following ways :
    - Debugger windows
    - Inspect exceptions
    - Inspect variables
    - Set watch on variables
    - View data value in data tip
    - Disassembly
    - Registers
    - Etc

1.7 Conditional compilation

- We can specify the compiler settings for your application in several ways:
    - The property pages
    - The command line
    - #define
- To change compile settings from the property pages dialog box
    - Right-click the project node in Solution Explorer.
    - Choose Properties from the shortcut menu.
    - Click the Build tab in the left pane of the property page, then select the check boxes for the compiler settings you want to enable. Clear the check boxes for settings you want to disable.
- To compile instrumented code using the command line
    - Set a conditional compiler switch on the command line. The compiler will include trace or debug code in the executable.

    - For example, the following compiler instruction entered on the command line would include your tracing code in a compiled executable:
    - csc -r:System.dll -d:TRACE -d:DEBUG=FALSE MyApplication.cs
- The meaning of the conditional-compilation directives used in the above examples is as follows:

Directive	Meaning
vbc	Visual Basic compiler
csc	C# compiler
-r:	References an external assembly (EXE or DLL)
-d:	Defines a conditional compilation symbol

- To perform conditional compilation using #define
    - Type the appropriate statement for your programming language at the top of the source code file.
- Expand table

Statement	Result
#define TRACE	Enables tracing
#undef TRACE	Disables tracing
#define DEBUG	Enables debugging
#undef DEBUG	Disables debugging