DEBUGGING

BY ADAM WAGENER AND JUAN IBARA

WHAT DO WE DEBUG?

- There are three major types of errors we run into when programming:
  - Syntax errors: Compile time issues
  - Execution time program crashes
  - Program Logic Errors – program is able to run, but expected results are not returned.

THE DEBUGGER

- The debugger is a set of tools that can be used, but each ide has its own debugger. Most have similar feature, and those are what we will focus on.
  - We can monitor a running program and step through it.
  - We can inspect objects and states during execution.
  - We can follow program flow and view accessible data.

THAT BASICS: BREAKPOINTS

- While debugging we can create breakpoints in order to stop code execution at specific points and view the program state at that location.
- We can make new and remove old breakpoints mid debug and we can even change code to address errors we saw.

THAT BASICS: STEPPING AND BREAKING

- There are many ways to step over, into, and out of our code. Most of us know the basics, but today we are going to explore many of the more complex steps.
- We can even move through code skipping section.
- One of the biggest issues many of us run into are dealing with large loops and long recursion.

LETS SEE IT IN PROGRESS

- There are many more slides here, but I think going over them in the demo does a lot more good than simply reading the information. The next few slides have been included for your future referencing.
- This information is pulled directly from Microsoft, here are some links to debugging tutorials

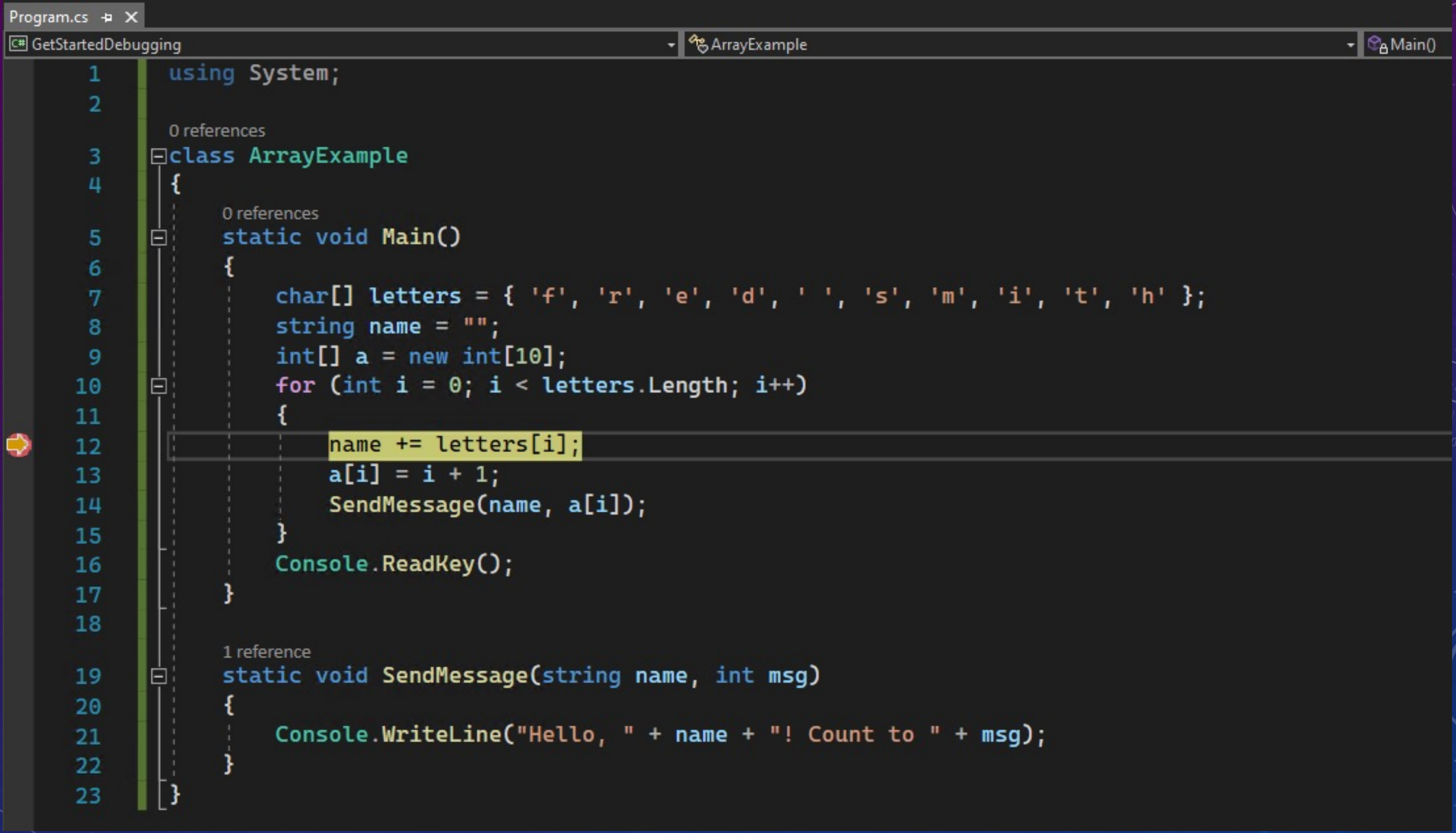
https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-debugger?view=vs-2022

https://learn.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2022

https://learn.microsoft.com/en-us/visualstudio/debugger/using-breakpoints?view=vs-2022

HOW TO DEBUG

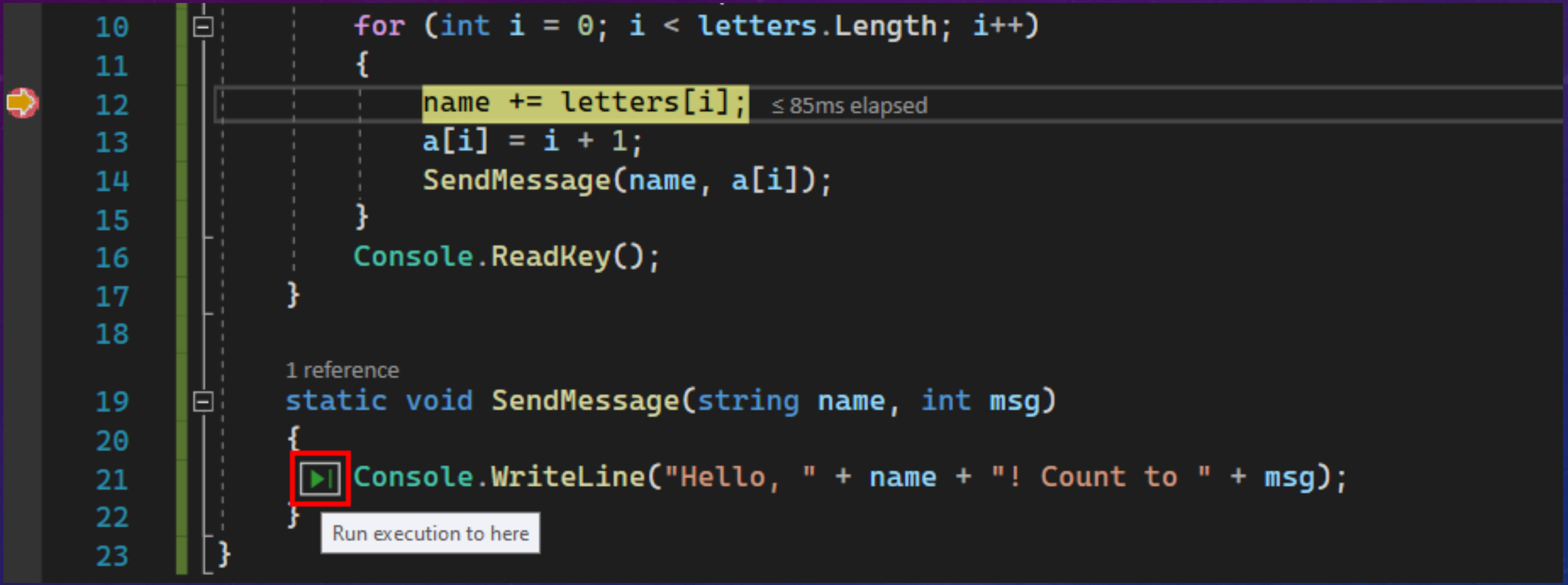
- To start debugging, select **F5**, or choose the **Debug Target** button in the Standard toolbar, or choose the **Start Debugging** button in the Debug toolbar, or choose **Debug** > **Start Debugging** from the menu bar. The app starts and the debugger runs to the line of code where you set the breakpoint.



```
using System;

0 references
class ArrayExample
{
    0 references
    static void Main()
    {
        char[] letters = { 'f', 'r', 'e', 'd', ' ', 's', 'm', 'i', 't', 'h' };
        string name = "";
        int[] a = new int[10];
        for (int i = 0; i < letters.Length; i++)
        {
            name += letters[i];
            a[i] = i + 1;
            SendMessage(name, a[i]);
        }
        Console.ReadKey();
    }

    1 reference
    static void SendMessage(string name, int msg)
    {
        Console.WriteLine("Hello, " + name + "! Count to " + msg);
    }
}
```

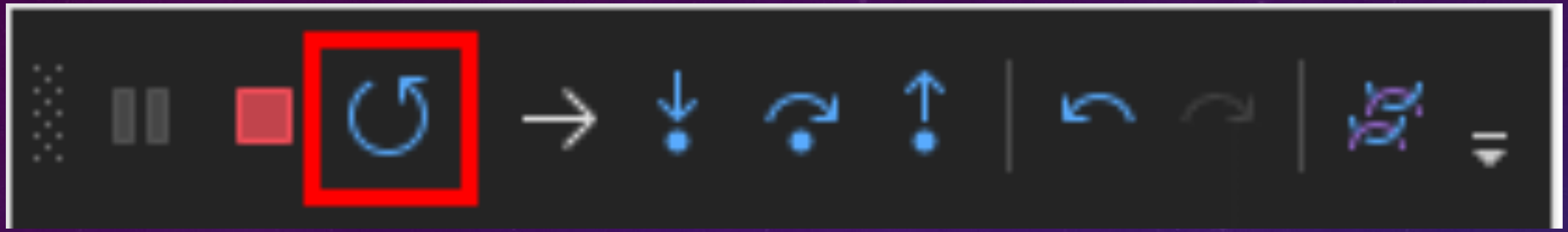
- The yellow arrow points to the statement on which the debugger paused.

RUN TO CLICK



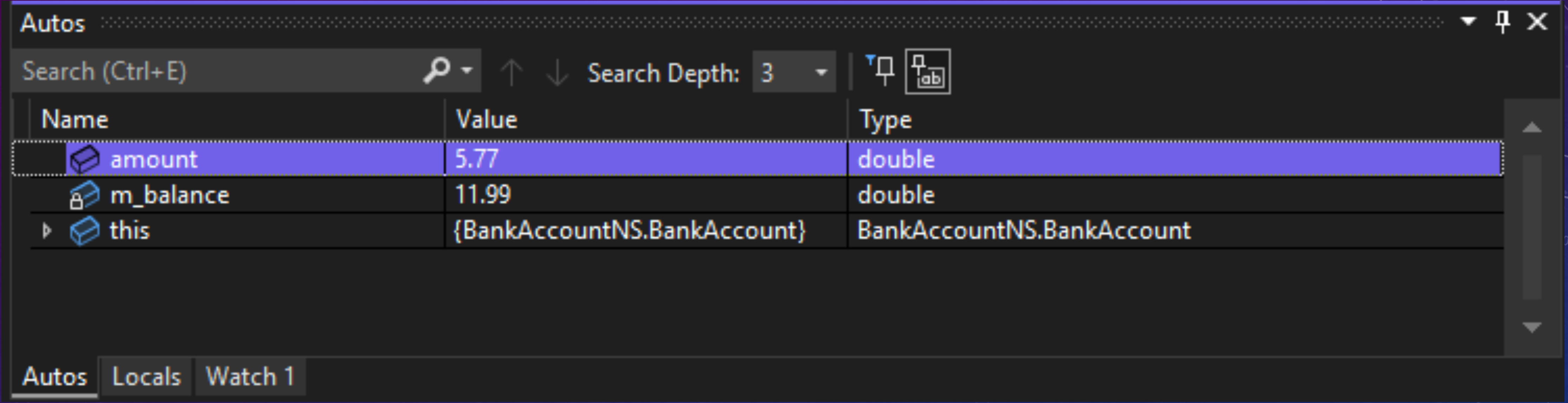
```
        for (int i = 0; i < letters.Length; i++)
        {
            name += letters[i];  ≤ 85ms elapsed
            a[i] = i + 1;
            SendMessage(name, a[i]);
        }
        Console.ReadKey();
    }

    1 reference
    static void SendMessage(string name, int msg)
    {
        Console.WriteLine("Hello, " + name + "! Count to " + msg);
    }
}
```

Using the **Run to Click** button is similar to setting a temporary breakpoint, and very useful for skipping over long loops and sections of code you are uninterested in viewing step by step.



- Pause
- Stop
- Restart
- Continue to next break
- Step into function
- Step over line
- Step out of function

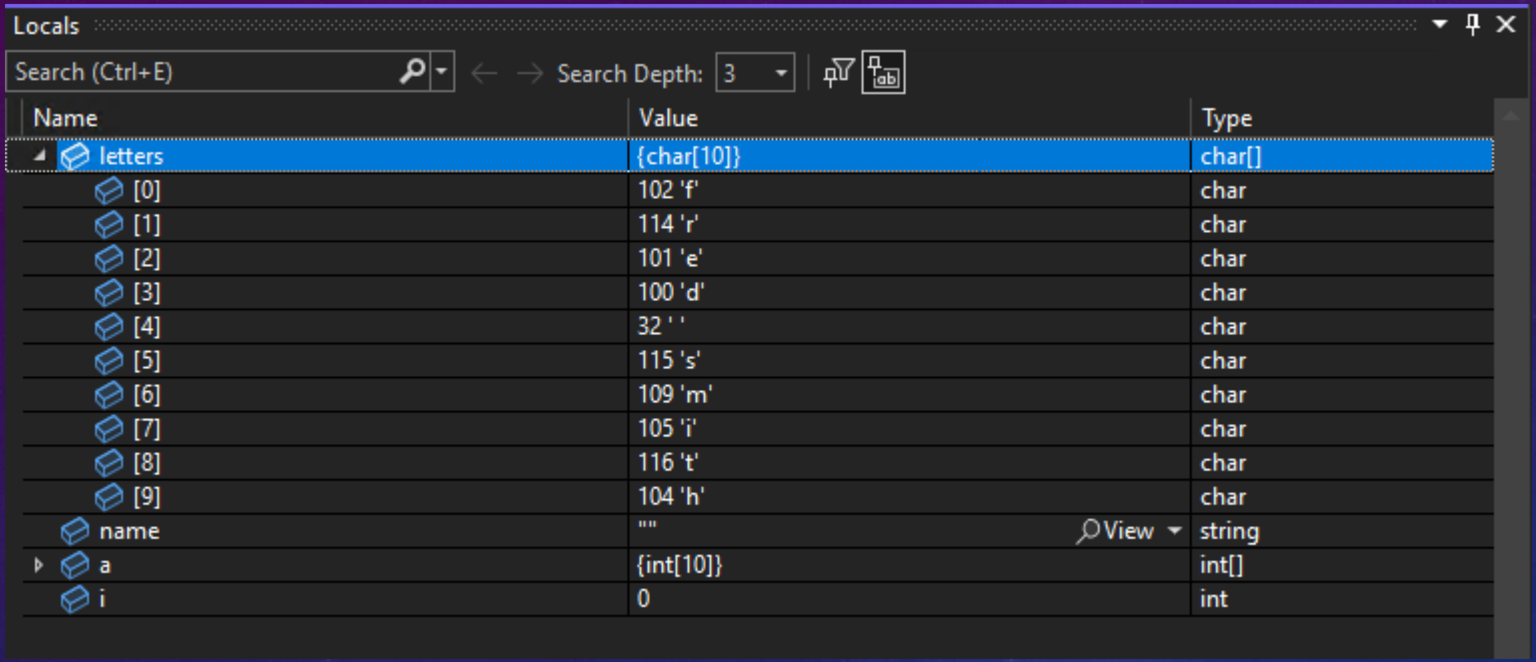
AUTOS WINDOW



Name	Value	Type
amount	5.77	double
m_balance	11.99	double
this	{BankAccountNS.BankAccount}	BankAccountNS.BankAccount

- This is the Autos window. Most of us are familiar with this window as it is the default window when debugging. This window does its best to inspect only those things that are changing in the immediate region or are being used, but it is not perfect. Often we will not see something we want here. There are many other windows and views we will see next.

LOCALS WINDOW



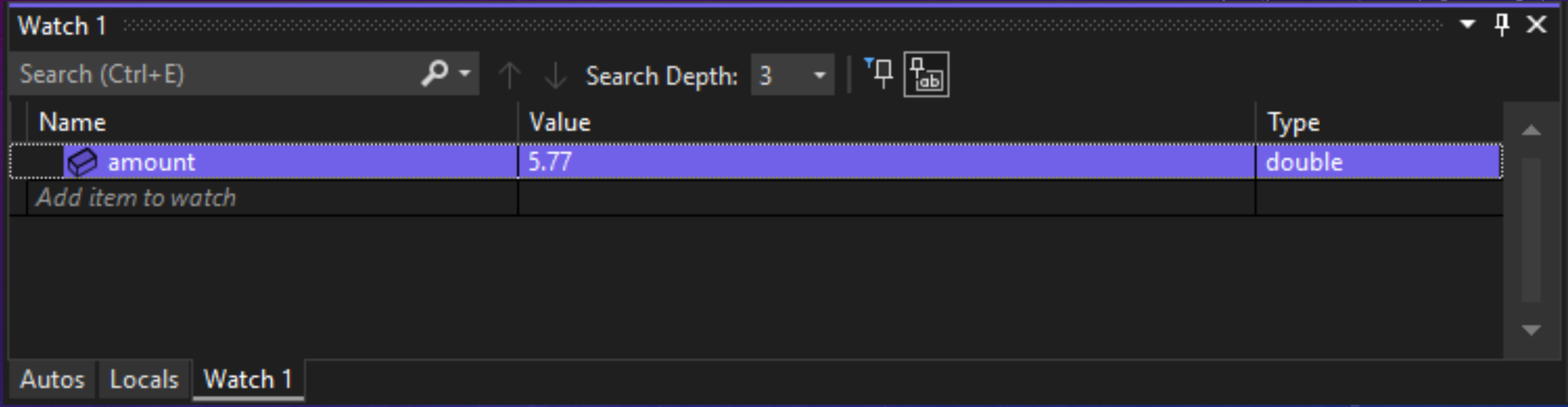
Locals

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
letters	{char[10]}	char[]
[0]	102 'f'	char
[1]	114 'r'	char
[2]	101 'e'	char
[3]	100 'd'	char
[4]	32 ' '	char
[5]	115 's'	char
[6]	109 'm'	char
[7]	105 'i'	char
[8]	116 't'	char
[9]	104 'h'	char
name	"" View	string
a	{int[10]}	int[]
i	0	int

- This window shows you everything in the local access at the paused point of the program. Here we can view many of the items we want to see a it includes all those things that are in scope.

SET A WATCH

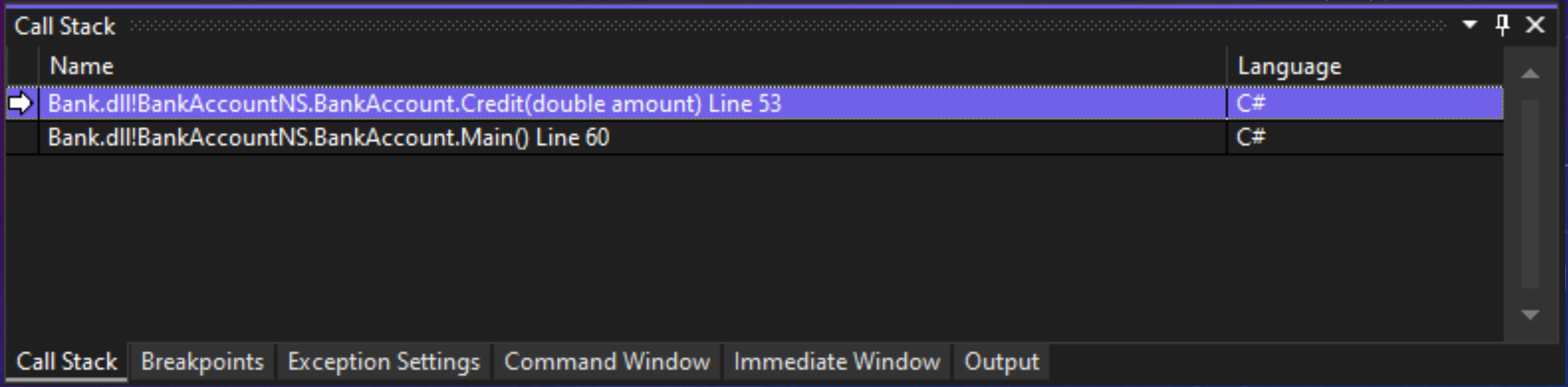

Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
amount	5.77	double
Add item to watch		

- Watch – select subtab watch and set variables to watch.
  This can be very useful when local has a lot of data or you are looking at a single deeply nested value.
- You can set multiple watch variables that can come in and out of scope, which can be very useful to see when they are in scope.

VIEW THE CALL STACK



	Name	Language
➡	Bank.dll!BankAccountNS.BankAccount.Credit(double amount) Line 53	C#
	Bank.dll!BankAccountNS.BankAccount.Main() Line 60	C#

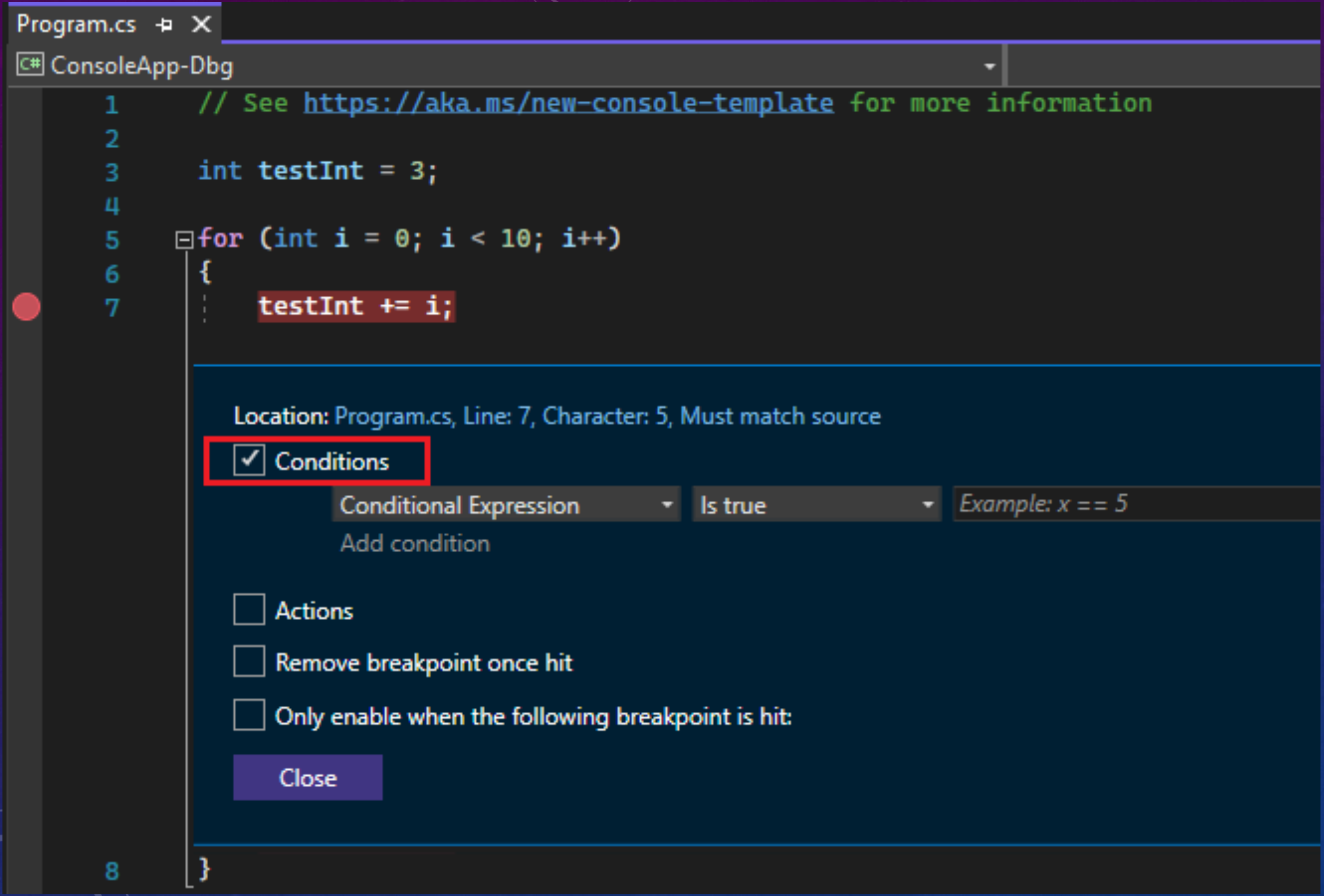
- Call Stack – you can view the call stack to inspect loops and recursion as they process. This is a huge help for recursive functions.

CHANGE EXECUTION FLOW

- This advanced technique can get you in trouble, but is very useful for skipping over or repeating code that may have issues in it without fully restarting or commenting out section. It also can allow us to enter areas without first running into an issue.
- To do this we will use the yellow arrow we see at the paused point and drag it to a section of code we wish to execute. This means we can skip over or go back to a prior statement.
- Note, this does not revert the program state, nor advance it. So if you go back, variables changes made will be retained and if you go forward, lines will be skipped.
- Be careful using this feature.

ADVANCED BREAKPOINTS

- So we know how to set breakpoints to stop the code running, but did you know they can also set conditions and other very useful information?



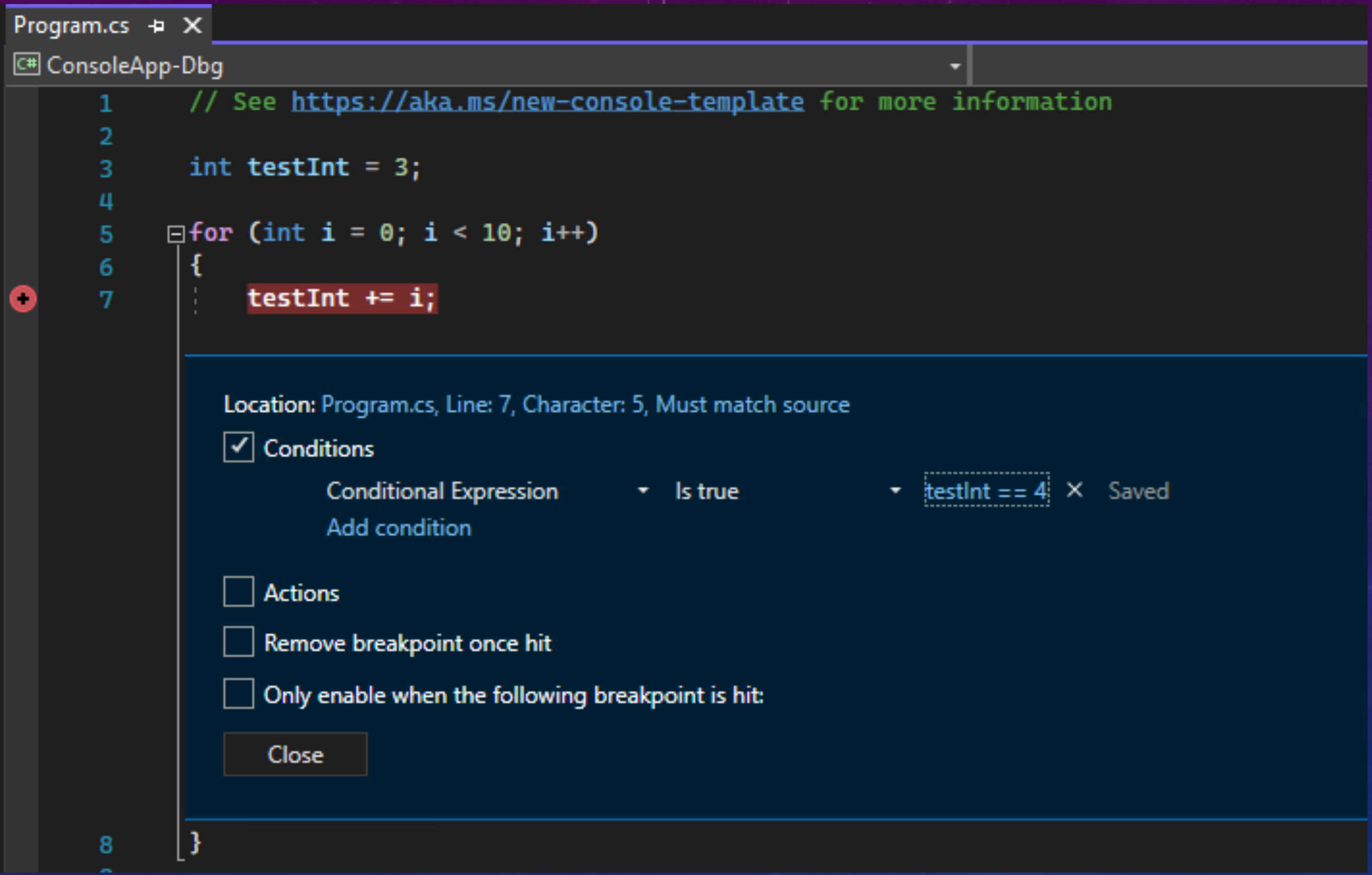
```
// See https://aka.ms/new-console-template for more information

int testInt = 3;

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

Location: Program.cs, Line: 7, Character: 5, Must match source

- [x] Conditions
  - Conditional Expression | Is true | Example: x == 5
  - Add condition
- [ ] Actions
- [ ] Remove breakpoint once hit
- [ ] Only enable when the following breakpoint is hit:

```
}
```

1. 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.



```
// See https://aka.ms/new-console-template for more information

int testInt = 3;

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

Location: Program.cs, Line: 7, Character: 5, Must match source

[✓] Conditions

Conditional Expression | Is true | testInt == 4 | Saved

Add condition

[ ] Actions

[ ] Remove breakpoint once hit

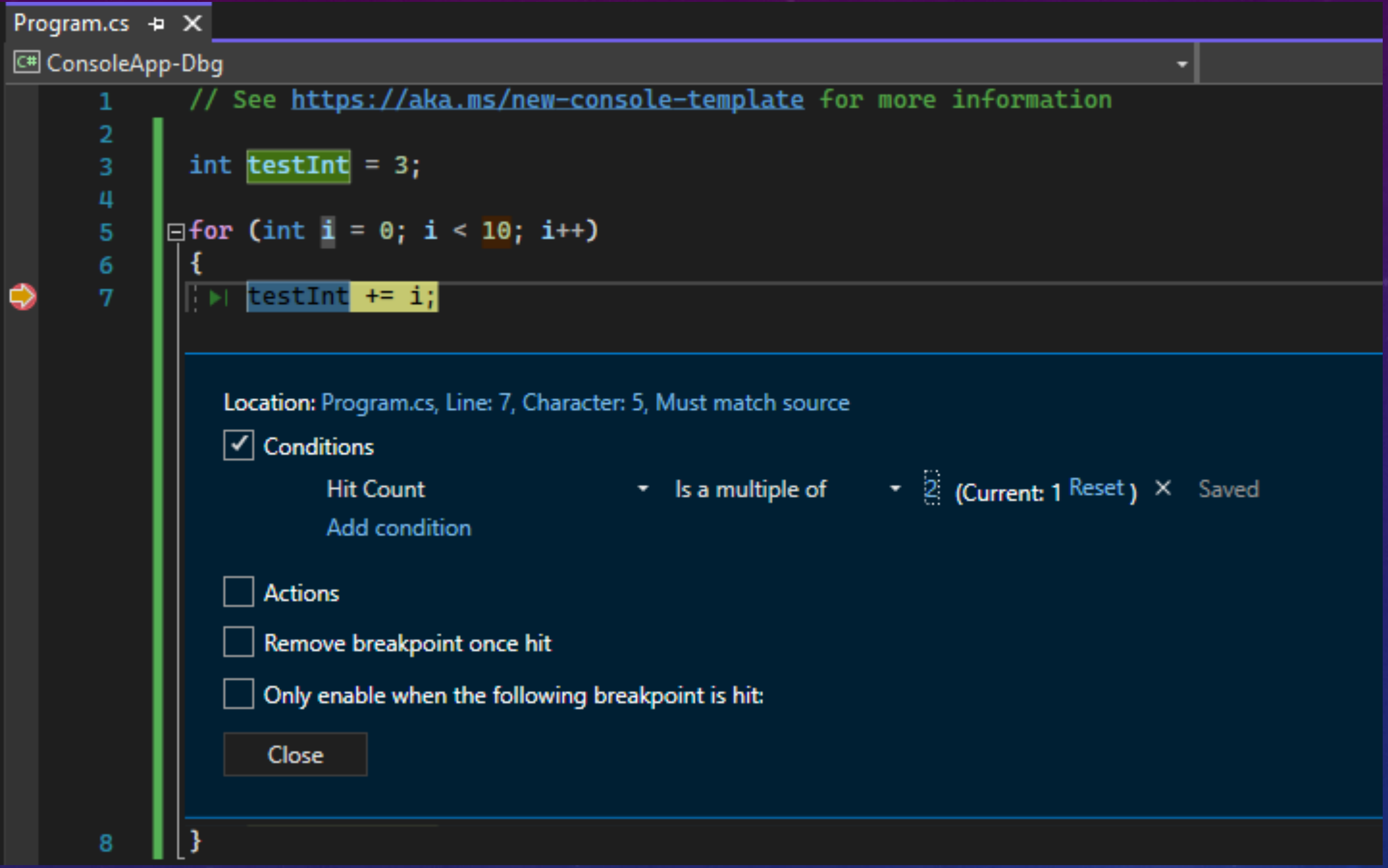
[ ] Only enable when the following breakpoint is hit:

Close

```
}
```

2. In the dropdown, select **Conditional Expression**, **Hit Count**, or **Filter**, and set the value accordingly.

3. Select **Close** or press **Ctrl+Enter** to close the **Breakpoint Settings** window. Or, from the **Breakpoints** window, select **OK** to close the dialog.



Program.cs

ConsoleApp-Dbg

```
// See https://aka.ms/new-console-template for more information

int testInt = 3;

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

Location: Program.cs, Line: 7, Character: 5, Must match source

[✓] Conditions

Hit Count — Is a multiple of — 2 (Current: 1 Reset) × Saved

Add condition

[ ] Actions

[ ] Remove breakpoint once hit

[ ] Only enable when the following breakpoint is hit:

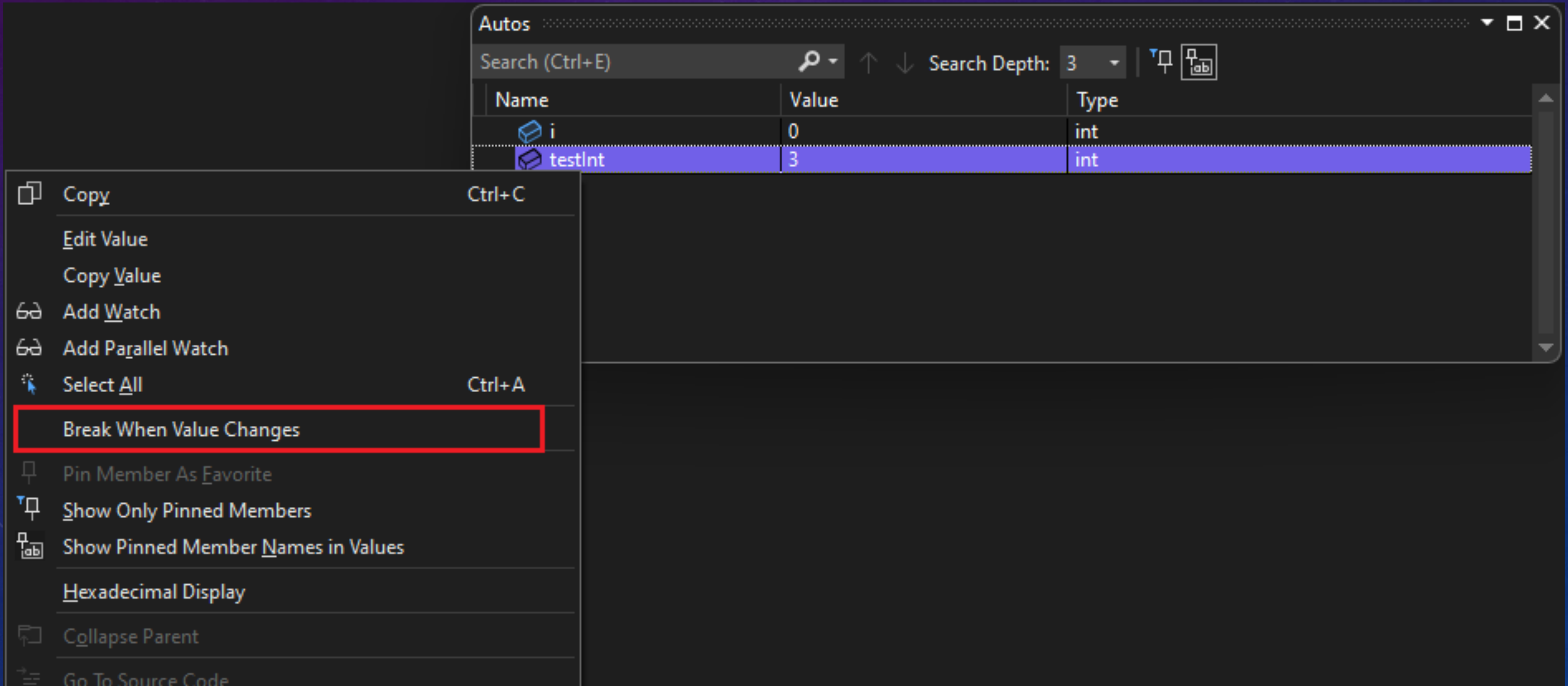
Close

```
}
```

- 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. In the following example, the breakpoint is set to hit on every other iteration:

SOME DEBUGGING IS UNIQUE TO LANGUAGE

- For example, in C++ you can set a function breakpoint using a memory address.
- Or in .Net core 3.x or .NET 5+ you can set a data breakpoint



Autos

Search (Ctrl+E)   Search Depth: 3

Name	Value	Type
i	0	int
testInt	3	int