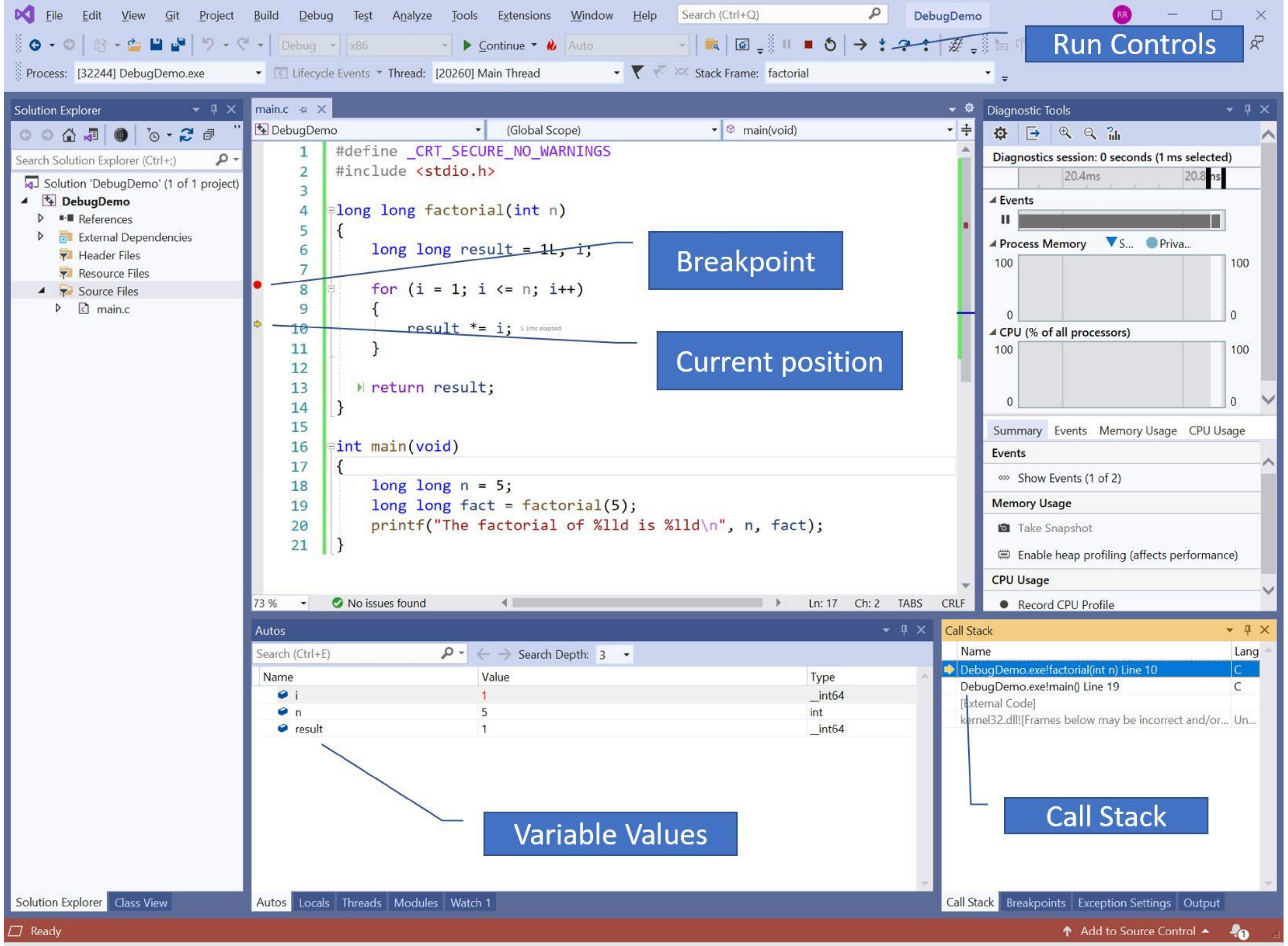
Software Testing

Visual Studio Debugging

Visual Studio Debugger

- Integrated debugger
- Supports
  - Break points,
  - Conditional breakpoints,
  - Stepping,
  - Stack trace,
  - Variable display,
  - Variable watch.



Process: [32244] DebugDemo.exe   Thread: [20260] Main Thread   Stack Frame: factorial

main.c

```c
 1  #define _CRT_SECURE_NO_WARNINGS
 2  #include <stdio.h>
 3
 4  long long factorial(int n)
 5  {
 6      long long result = 1L, i;
 7
 8      for (i = 1; i <= n; i++)
 9      {
10          result *= i;
11      }
12
13      return result;
14  }
15
16  int main(void)
17  {
18      long long n = 5;
19      long long fact = factorial(5);
20      printf("The factorial of %lld is %lld\n", n, fact);
21  }
```

Autos

Name	Value	Type
i	1	__int64
n	5	int
result	1	__int64

Call Stack

Name	Lang
DebugDemo.exe!factorial(int n) Line 10	C
DebugDemo.exe!main() Line 19	C
[External Code]	
kernel32.dll![Frames below may be incorrect and/or...	Un...

Breakpoints

- when the code is being debugged and just before it hits the line on which a breakpoint has been set execution will be paused.
- While paused:
  - you can examine the state of all the variables,
  - you can use the run controls at the top of the window to step the program ahead

Run Controls

▶ Continue ▾	continue to execute until it either hits the next breakpoint or the program completes.
↻	stop the current execution of the program and restart it
→	position the cursor on the next statement that is going to be executed
■	terminate the current debugging session
↓	resume execution until it enters the next function call.
↷	execute the next statement
↑	continue to the end of a function and return to the after the function call

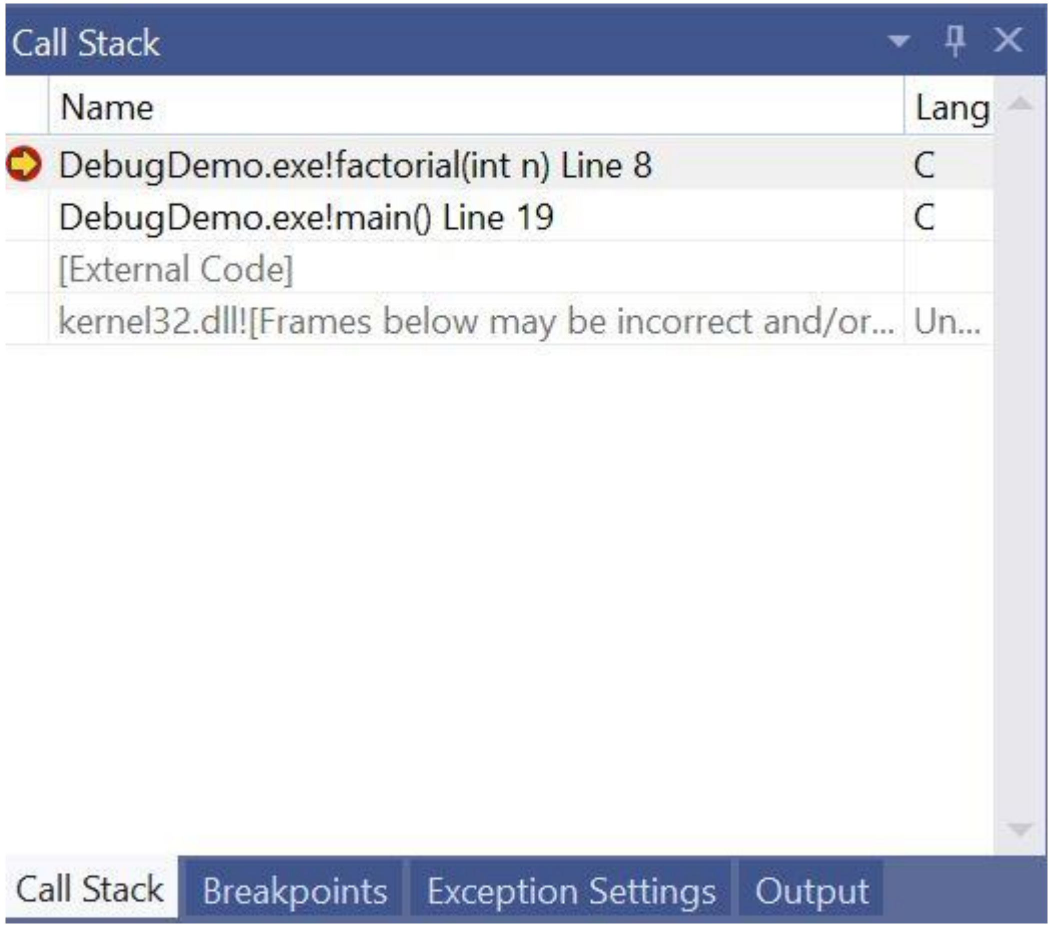
Variables

- The pane at the bottom has several tabs to show different information.
- The Autos and Locals tabs show the values of automatic and local variables.
- What is shown will change based on the scope in which the debugger is paused.
- If the variable is an array, struct or class,
  - it will allow you to expand it and see the internal structure of the variable.

Watches

- The watch tab is beside the autos and locals tab
- allows you to watch specific variables and expressions.
- You can either
  - type the name of the variable into the window or
  - right click on the variable name in the code and select "Add watch" from the menu that appears.
- You can enter simple expressions into the window to have them calculated and displayed.

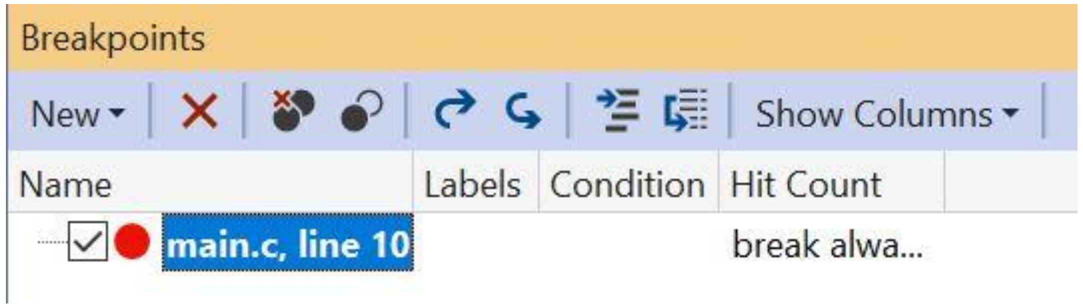
The Call Stack


Call Stack

Name	Lang
DebugDemo.exe!factorial(int n) Line 8	C
DebugDemo.exe!main() Line 19	C
[External Code]	
kernel32.dll![Frames below may be incorrect and/or...	Un...

Shows the line paused at and
All calls paused before that
Point.

The Breakpoint Pane



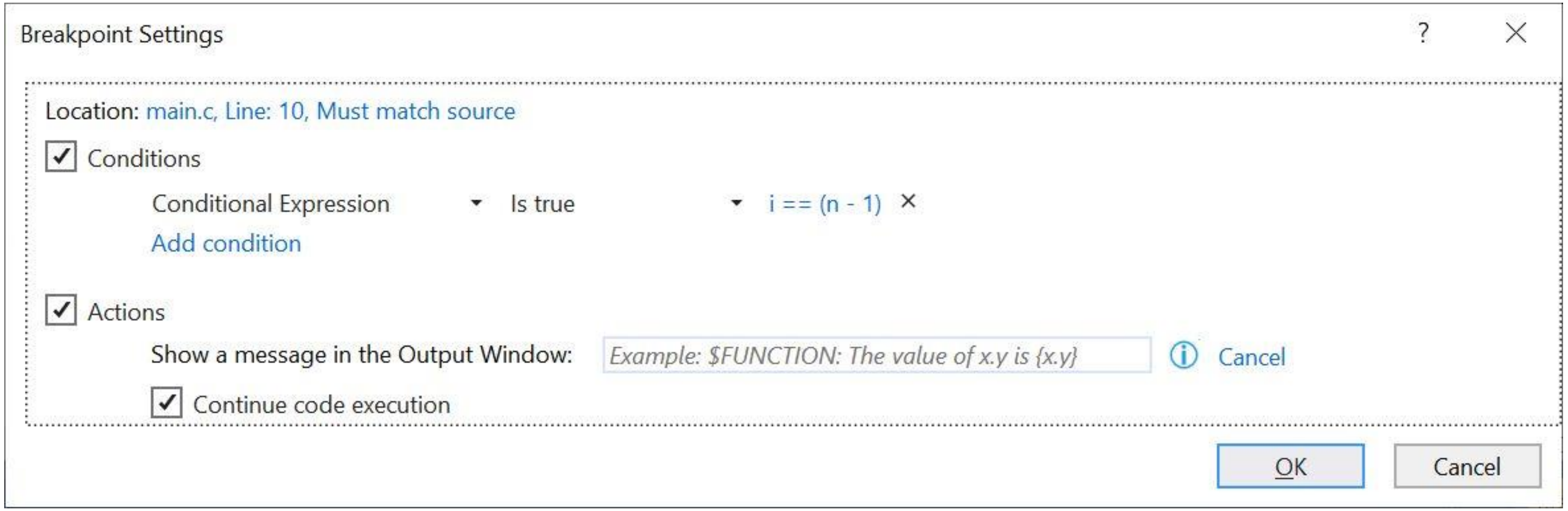
Name	Labels	Condition	Hit Count
✓ ● main.c, line 10			break alwa...

- Shows a list of breakpoints,
- Can be unchecked to temporarily disable
- Can be removed with X at top

Breakpoint Types

- Using the **New** button at the top of the breakpoint pane can create:
  - **A function breakpoint**
    - This is what we have been using and stops when a function is called
  - **A data breakpoint**
    - Allows you to set the address of a variable and a number of bytes after it
    - Triggered when that memory is changed.

Breakpoint Settings



Breakpoint Settings

Location: main.c, Line: 10, Must match source

✓ Conditions

Conditional Expression: Is true: i == (n - 1)

Add condition

✓ Actions

Show a message in the Output Window: *Example: $FUNCTION: The value of x.y is {x.y}*

✓ Continue code execution

- Set a condition under which breakpoint will trigger (eg. x == 5)
- Create an action to print a message when triggered

Action Macros

- $CALLER - The name of the function which called this function.
- $CALLSTACK - Displays the call stack showing how this function was called.
- $FILEPOS - The name of thecurrent file and the line of the breakpoint
- $FUNCTION - The name of the function containing the breakpoint
- $TICK - The number of milliseconds since the computer was started

```
In $FUNCTION called from $CALLER: i = {i}
```

Exporting and Importing Breakpoints

- Breakpoints can be exported from the breakpoints pane using the curved arrow buttons at the top of the pane.
- This lets you create a set of breakpoints to debug a certain problem and save them to an XML file.
- Since you might not have completely solved the problem, you can save the breakpoints and then reload them if the same problem occurs again.