# Visual Studio

Debugging

Lahiru Dilshan

# QuickStart

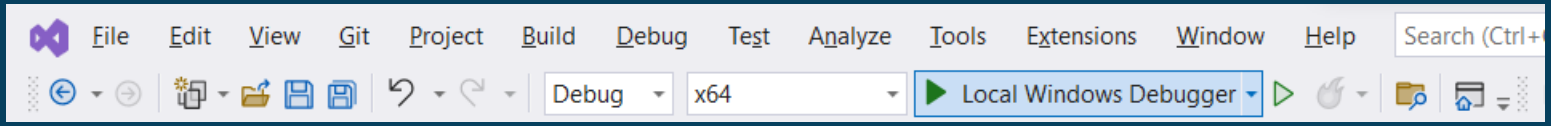
Debugging

# Debugger vs Debugging

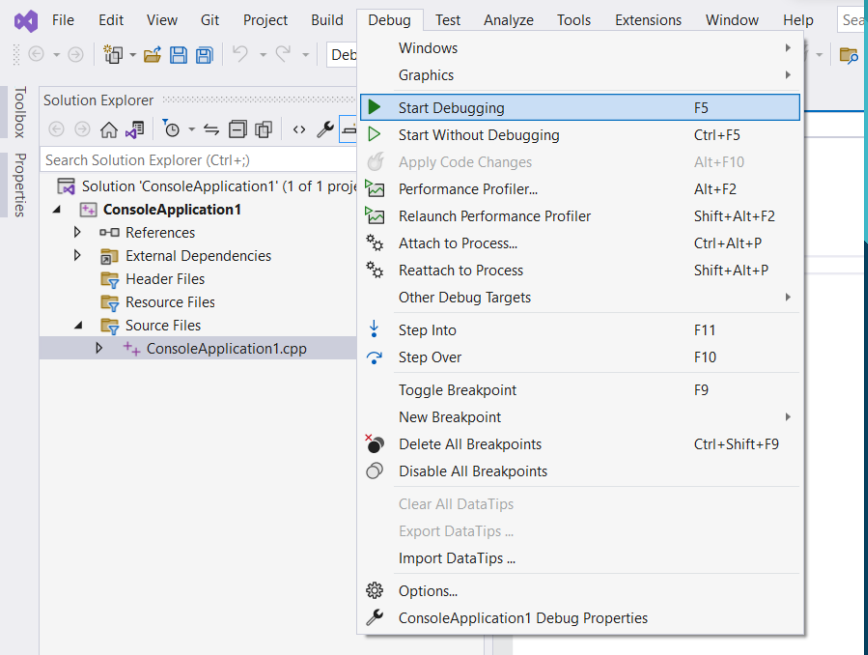
- Debugging – removing bugs from your code
- Ways of debugging
  - Scan your code looking for typos (typographical errors)
  - Use code analyzer (ex: Cppcheck). (A tool that analyzes source code without executing the code)
  - Use a performance profiler (software development tool designed to help you analyze the performance of your applications and improve poorly performing sections of code) (ex: GlowCode)
  - *Debugger*
    - Very specialized developer tool that attaches to your running app and allows you to inspect your code.

# Debug mode vs running your app

- Run your program in debug mode




  - Set the value “Debug” in the drop-down list
  - Press F5 or
  - go to Debug >> Start Debugging or
  - click Local Window Debugger

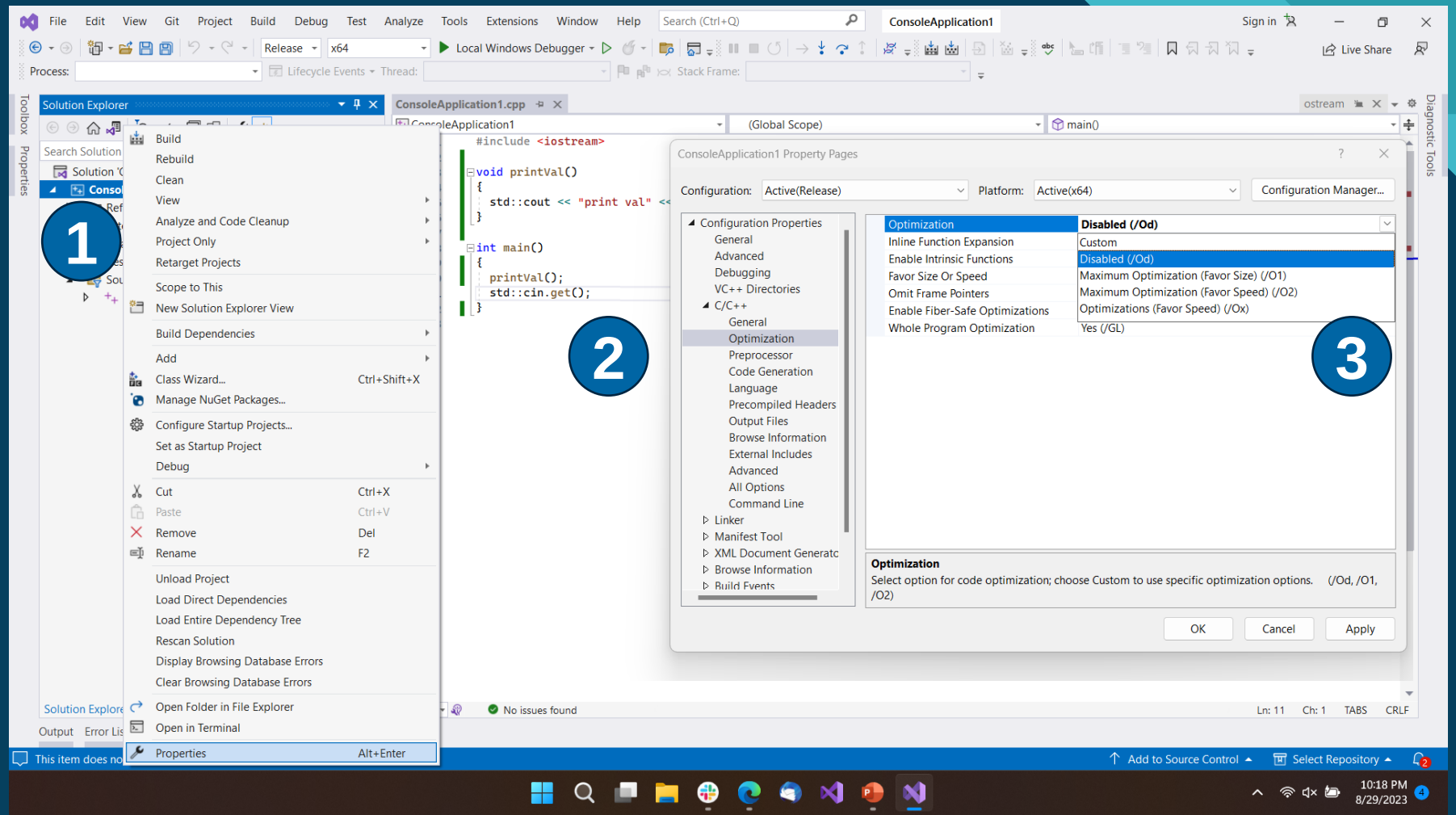


# Break mode

- Examine the values of variables
- Some project types, you can make adjustments to the program
- Most debugger windows are available only when the debugger is attached (Watch)

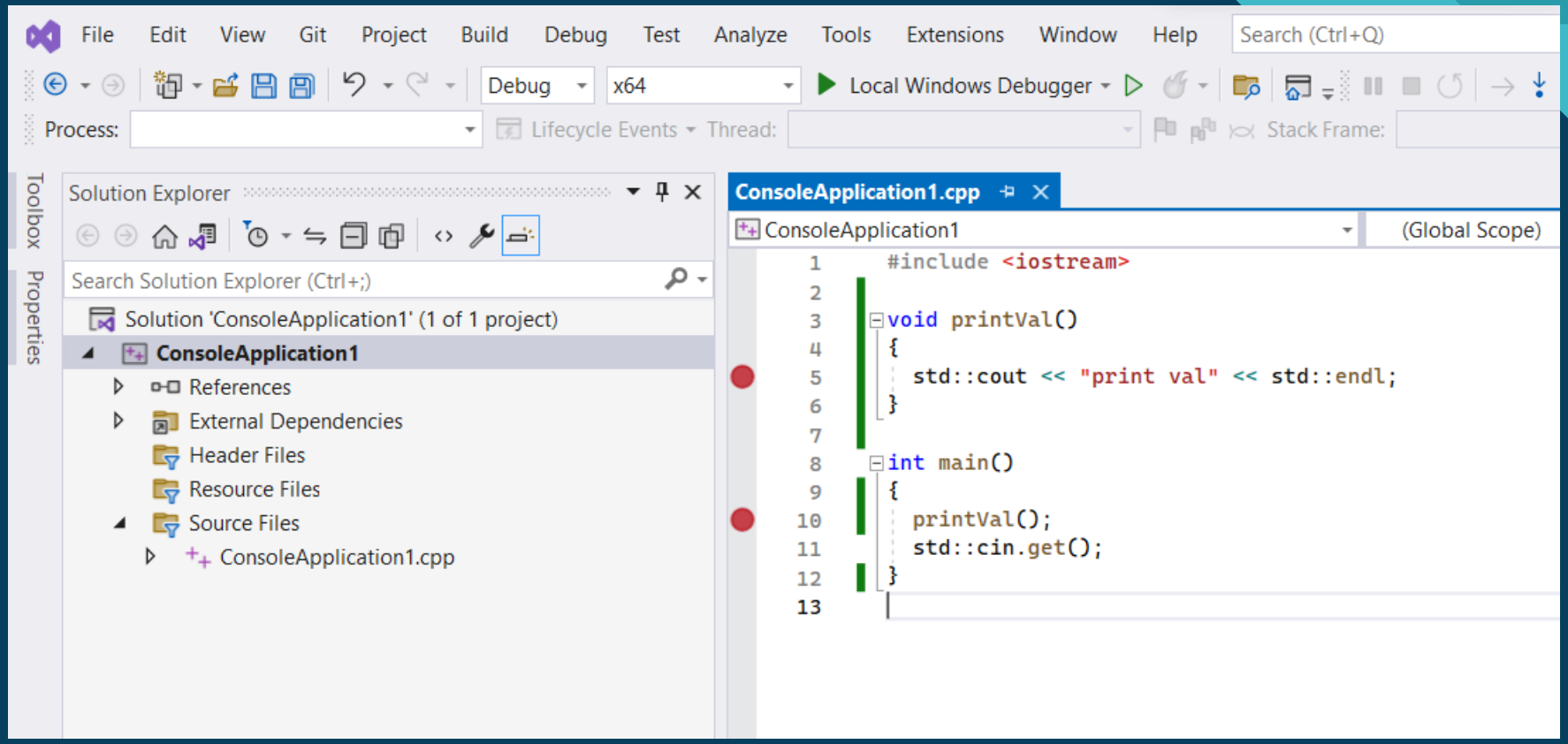
# Debug mode vs running your app

- Remove optimization in the Release mode



# Debug mode vs Release mode

- Example
  - What will happen when this code runs in the debug mode and the release mode?




Example 1

# When to use the Debugger

- The main goal is to quickly eliminate bugs and errors
- Debugger is not *the* only option. It is *an* option.
- Better to use good coding practices to eliminate bugs.

# Debugger provides…

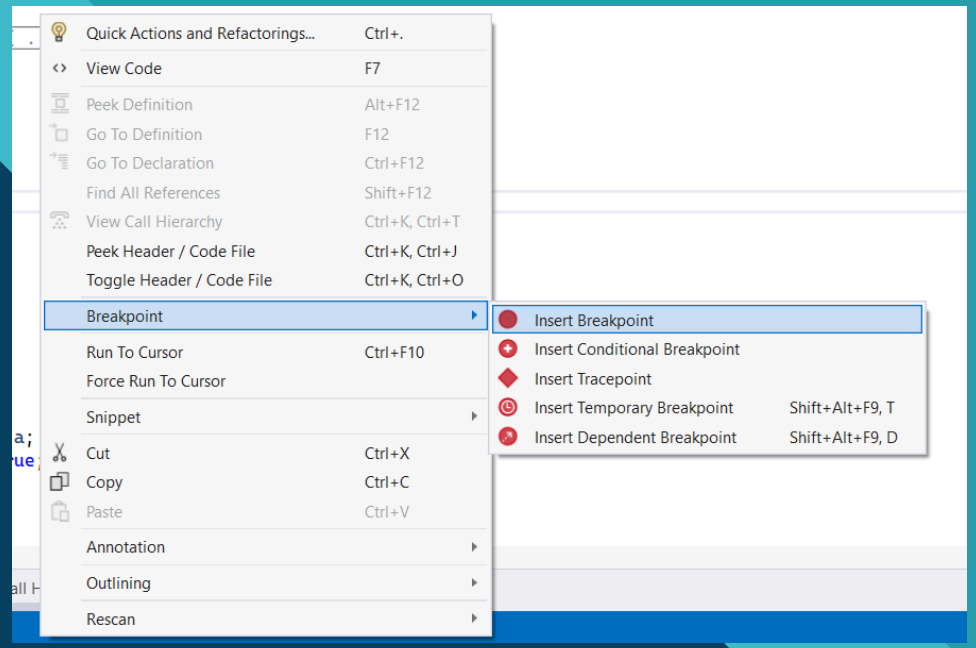
- You can step through your code and look at the values stored in variables
- You can set watches on variables to see when values change
- You can examine the execution path
- ……

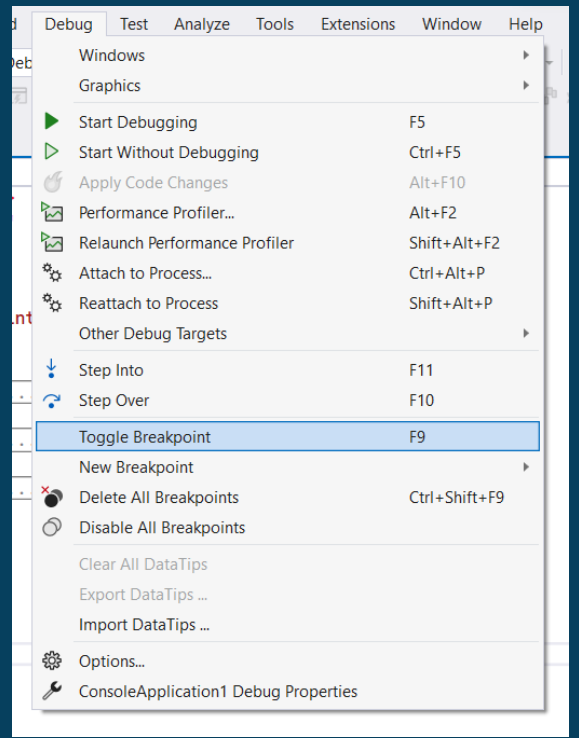
# Breakpoint

- A breakpoint indicates where Visual Studio should suspend your running code
- So, you can take a look at the values of,
  - Variables
  - Behavior of memory
  - Whether or not a branch of code is getting run

# Add Breakpoints

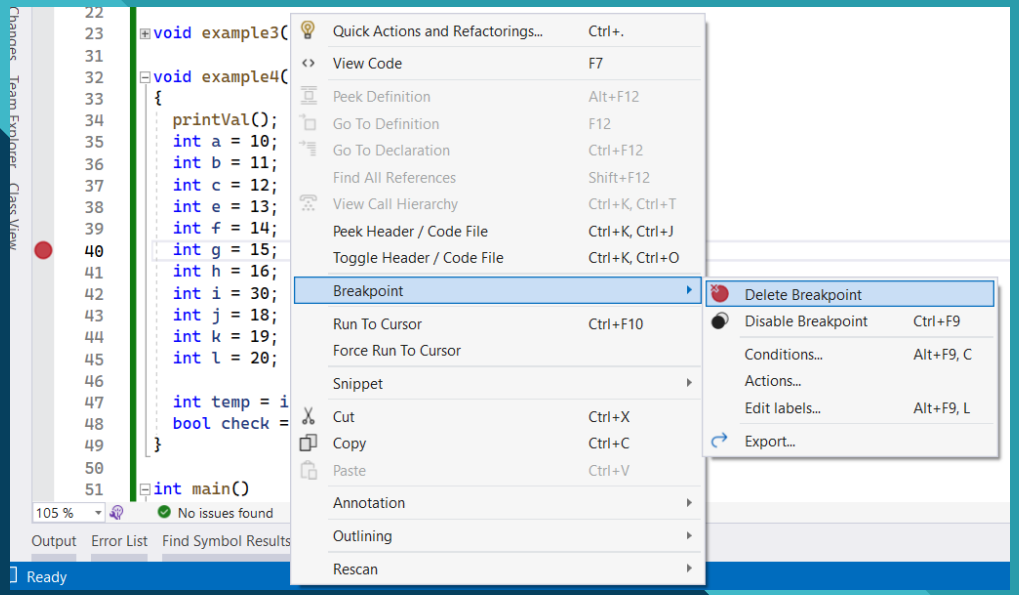
- Click on the left margin column
- Right click >> Breakpoint >> Insert Breakpoint
- Debug >> Toggle Breakpoint
- F9

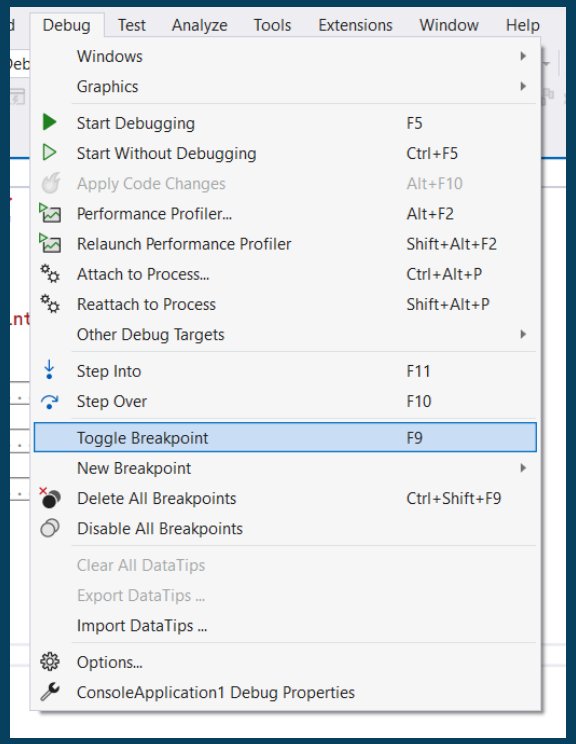


# Remove Breakpoints

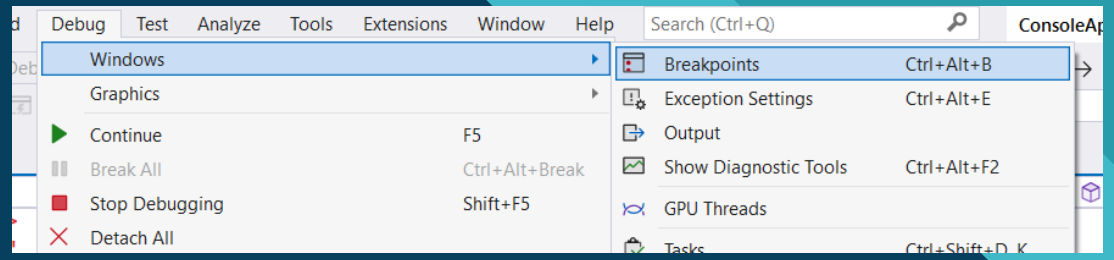
- Click on the breakpoint circle in the left margin
- Right click >> Breakpoint >> Delete Breakpoint
- Debug >> Toggle Breakpoint
- F9

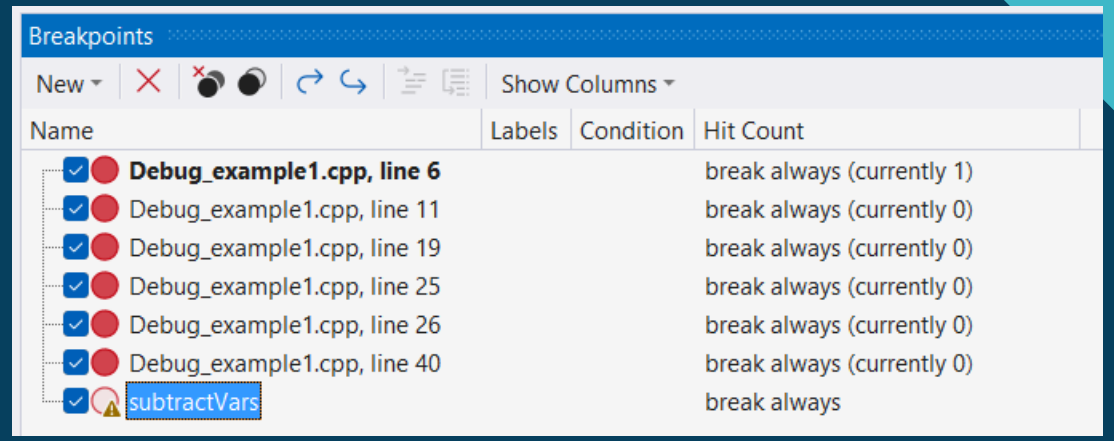


# Inspect Breakpoint

- You can show all the breakpoints in Breakpoint Window
- All breakpoint options can be modified
  - Disable
  - Delete
  - …..



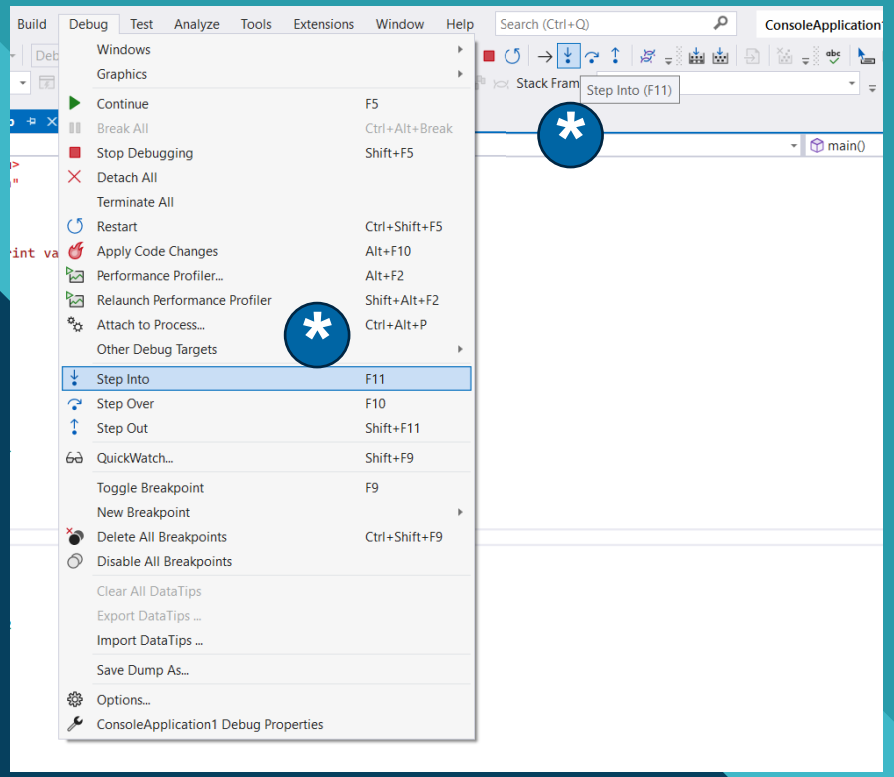
# Navigation

Debugging

# Navigate code in the debugger

- **Step Into** command.
  - Advances the app execution one statement at a time
  - **F11** or **Debug >> Step Into** or **Toolbar button**

```
20  int main()
21  {
22      int a = 10;
23      int b = 20;   ≤ 1ms elapsed
24      int c = a + b;
25      std::cin.get();
26  }
```




- The yellow arrow represents the statement on which the debugger paused.
  This suspends the app execution.
  This statement has not yet been executed.

# Navigate code in the debugger

1. **Step Over (F10 or Debug >> Step Over)**
   - Advances the debugger without stepping into functions or methods
   - You can skip over code that you're not interested in.
2. **Step out (Shift + F11 or Debug >> Step Out)**
   - Advance the debugger all the way through the current function

# Navigate code in the debugger

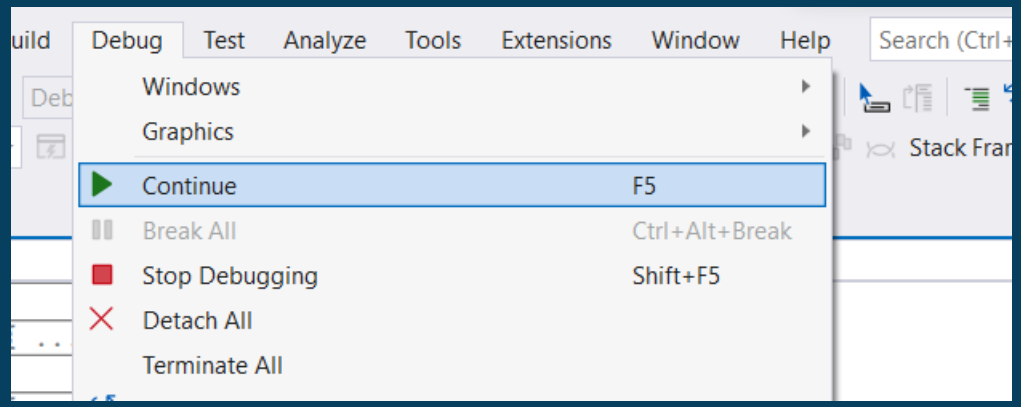
1. **Debugger steps through code statements, not physical lines (code statement – a single line of code that performs a specific task)**

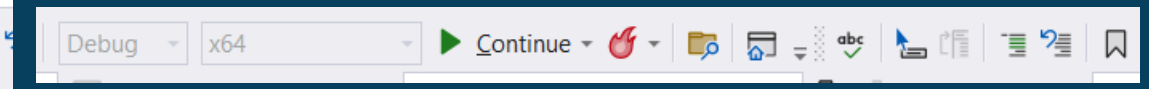
```
74  void example5()
75  {
76    int a = 10;
77    int b = 20;
78    int c = 30;
79    if (a > 0) std::cout << a << std::endl;
80    if (b < 0)
81      std::cout << b << std::endl;
82    if (c > 0)
83      std::cout << c << std::endl;
84    std::cin.get();
85  }
```

Example 5

# Navigate code in the debugger

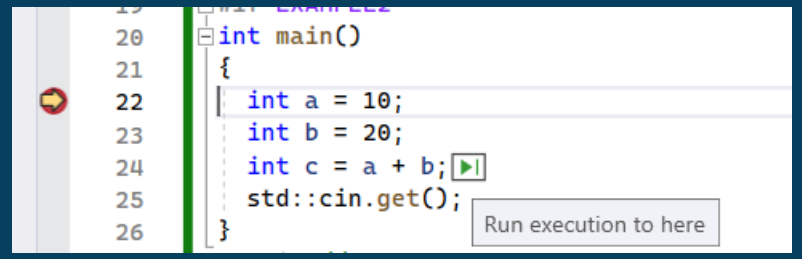
1. **Continue**
   - Go to the next breakpoint
   - You should be in the debug mode
   - Click *Continue* button or Debug >> Continue or F5

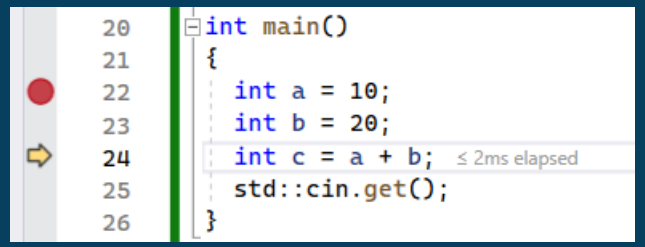


# Navigate code in the debugger

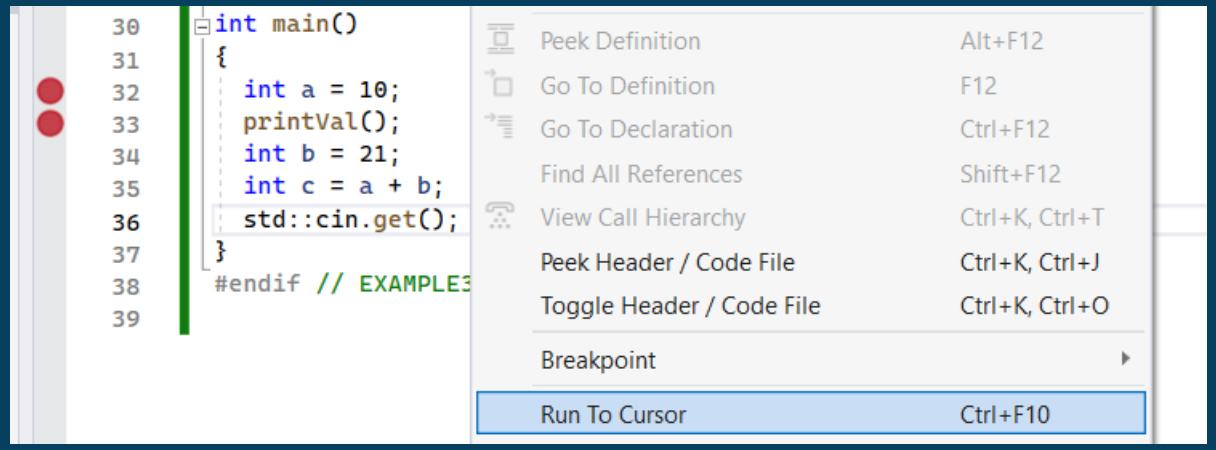
- **Run to click**
  - Similar to setting a temporary breakpoint
  - Hover over the line of code until the “Run to click” button appeared.



# Navigate code in the debugger

- **Run to cursor**
  - When you are editing the code and not in the debug mode, and if you want to debug into that line use this command.
  - This will create a temporary breakpoint and start debugging at the same time
  - ***Right click a line of code >> Run to Cursor*** or ***Ctrl+F10***
  - If there are a breakpoints previous to the select line, the debugger pauses on the first breakpoint.




Example 3

# Navigate code in the debugger

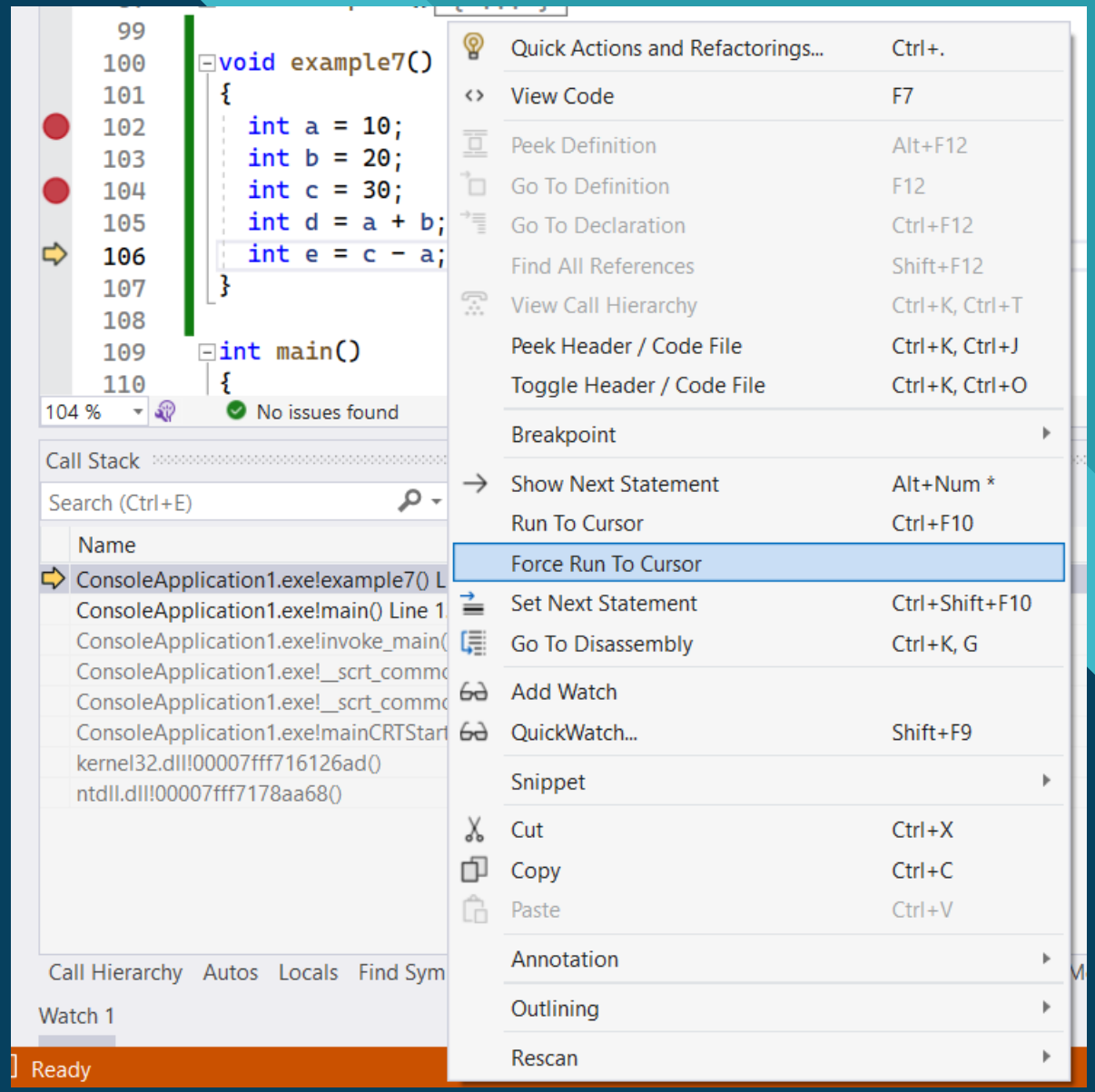
- **Force run to click**
  - Application attaches the Debugger and pauses at the cursor location.
  - Any breakpoints and first-chance exceptions found during execution are temporarily disabled.
  - Shift + Click double green arrow

```
100  void example7()
101  {
102      int a = 10;
103      int b = 20;
104      int c = 30;
105      int d = a + b;
106      int e = c - a;
107  }
108
```

Force run execution to here

# Navigate code in the debugger

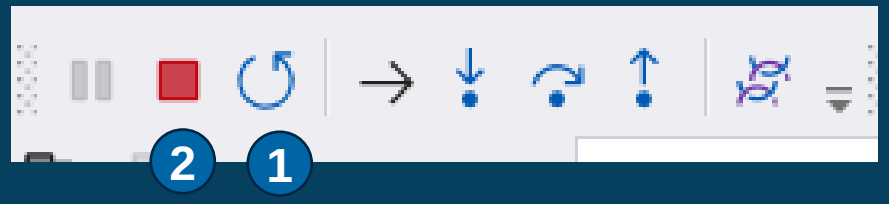
- **Force run to cursor**
  - Skip any breakpoints and first-chance exceptions until the debugger reaches the line of code where the cursor is located.
  - Use Call Stack or right click option




Example 7

# Others

1. **Restart app**
   - *Click the button* or *Ctrl* + *Shift* + *F5* or *Debug >> Restart*
   - This saves time versus stopping the app and restarting the debugger
   - The debugger pauses at the first breakpoint that is hit by executing code.
   - This will remove “Run to Cursor” options which already set
2. **Stop**
   - *Click the button* or *Shift* + *F5* or *Debug >> Stop Debugging*
   - Stop the debugger and get back into the code editor



# Inspect variables

- **You can inspect variables in your program while paused in the debugger**
- **Data tips**
  - Hover over an object with the mouse
  - If the variable has properties, you can expand the object to see all its properties

1. **Autos**
   - Shows all variables used on the current line or the preceding line (in C++, the window shows variables in the preceding three lines of code)
2. **Locals**
   - Shows the variables that are currently in scope

# Inspect variables

```
32  void example4()
33  {
34      int a = 10;
35      int b = 11;
36      int c = 12;
37      int e = 13;
38      int f = 14;
39      int g = 15;
40      int h = 16;
41      int i = 17;
42      int j = 18;
43      int k = 19;
44      int l = 20;
45  }
46
```

Autos

Search (Ctrl+E) Search Depth: 3

| Name | Value | Type |
|---|---|---|
| c | 12 | int |
| e | 13 | int |
| f | 14 | int |
| g | 32759 | int |

```
32  void example4()
33  {
34      int a = 10;
35      int b = 11;
36      int c = 12;
37      int e = 13;
38      int f = 14;
39      int g = 15;
40      int h = 16;
41      int i = 17;
42      int j = 18;
43      int k = 19;
44      int l = 20;
45  }
46
```

Locals

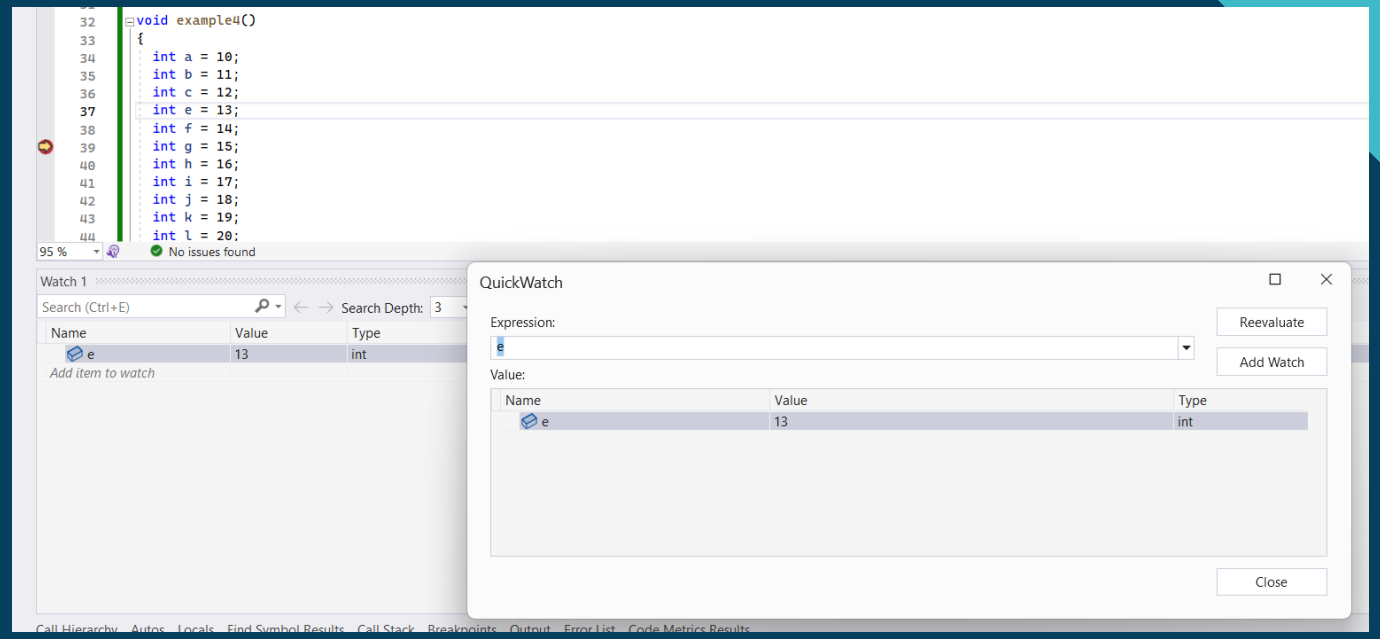
Search (Ctrl+E) Search Depth: 3

| Name | Value | Type |
|---|---|---|
| a | 10 | int |
| b | 11 | int |
| c | 12 | int |
| e | 13 | int |
| f | 14 | int |
| g | 32759 | int |
| h | 40 | int |
| i | 7012467 | int |
| j | 4390979 | int |
| k | 7602273 | int |
| l | 4391004 | int |

Example 4

# Inspect variables

- **Set a watch**
  - Specify a variable or an expression that you want to keep an eye on
  - Always show the variables that you are watching
  - They are greyed out when out of scope




Example 4

# Call stack

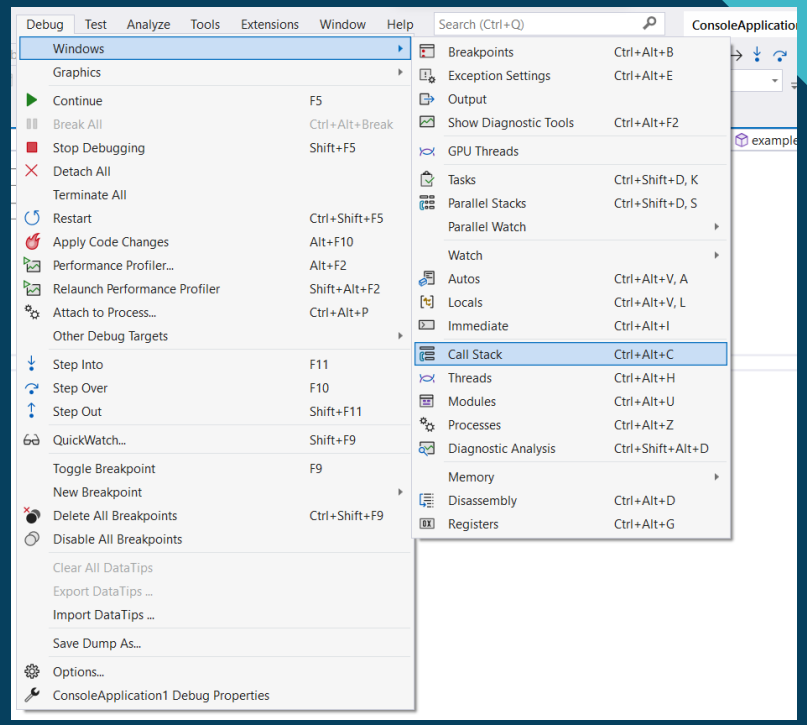
- Shows the order in which methods and functions are getting called
- Top line shows the current function.
  The second line shows the function or property it was called from, and so on.

```
void printVal()
{
    std::cout << "print val" << std::endl;
}

void example1() { ... }

void example2() { ... }

void example3() { ... }

void example4()
{
    printVal();
    int a = 10;
    int b = 11;
    int c = 12;
```

Call Stack

Search (Ctrl+E) | View all Threads | Show External Code

| Name |
|---|
| ConsoleApplication1.exe!printVal() Line 6 |
| ConsoleApplication1.exe!example4() Line 35 |
| ConsoleApplication1.exe!main() Line 60 |




Example 4

# Edit and continue

- If you identify change that you want to test in your code in the middle of a debugging session,…….
  - Change the variable
  - Press **F10 (Debug >> Step Over)** few times to advance the debugger

```
32  void example4()
33  {
34      printVal();
35      int a = 10;
36      int b = 11;
37      int c = 12;
38      int e = 13;
39      int f = 14;
40      int g = 15;
41      int h = 16;
42      int i = 17;
43      int j = 18;
44      int k = 19;
45      int l = 20;
46
47      int temp = i + a;
48      bool check = true;
49  }
```

```
32  void example4()
33  {
34      printVal();
35      int a = 10;
36      int b = 11;
37      int c = 12;
38      int e = 13;
39      int f = 14;
40      int g = 15;
41      int h = 16;        i 30
42      int i = 30;
43      int j = 18;        ≤ 1ms elapsed
44      int k = 19;
45      int l = 20;
46
47      int temp = i + a;
48      bool check = true;
49  }
```

Example 4

# Breakpoints

Debugging

# Breakpoint

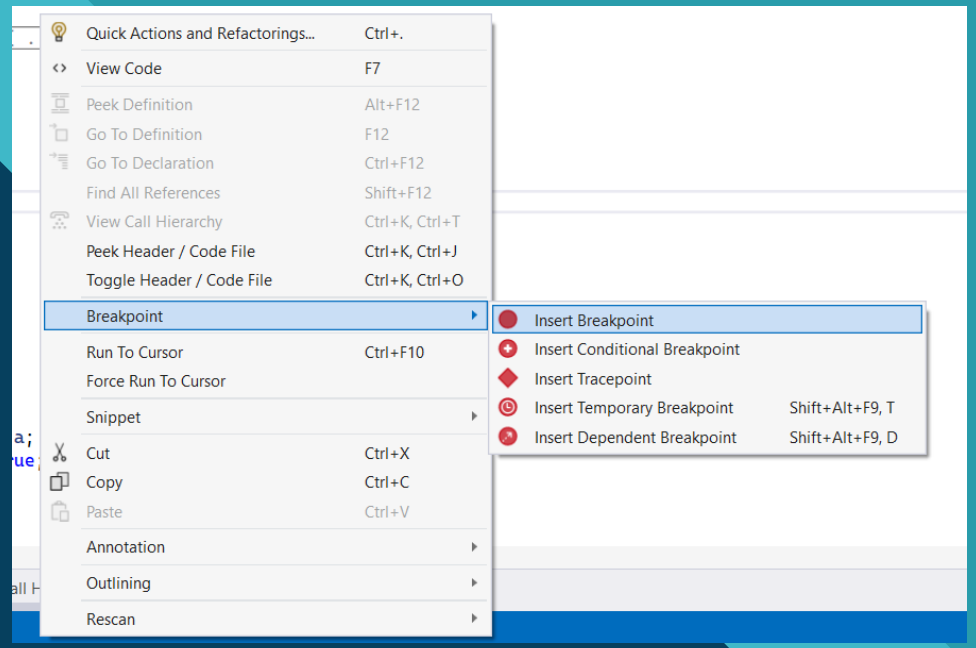
- One of the most important debugging techniques
- Set breakpoint wherever you want to pause the debugger execution
- You can’t set a breakpoint
  - On a method signature
  - Declarations for a namespace or class
  - Variable declaration if there’s no assignment and no getter/setter

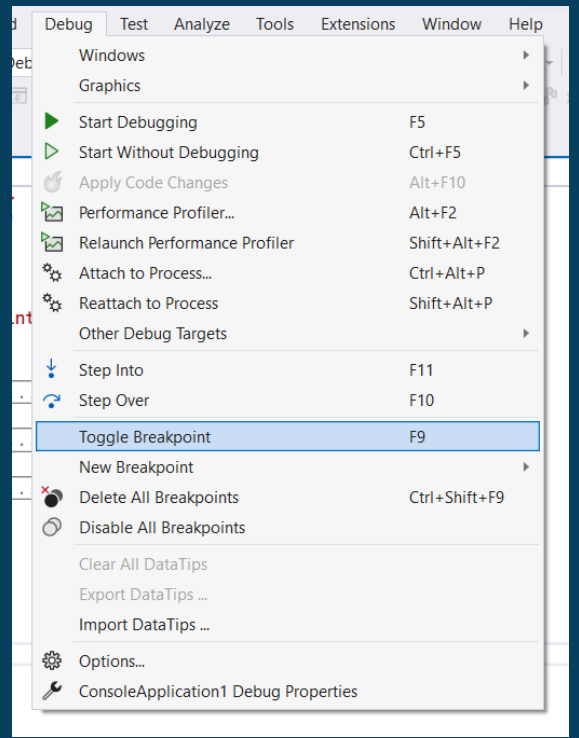
# Breakpoint

- One of the most important debugging techniques
- Set breakpoint wherever you want to pause the debugger execution
- You can’t set a breakpoint
  - On a method signature
  - Declarations for a namespace or class
  - Variable declaration if there’s no assignment and no getter/setter

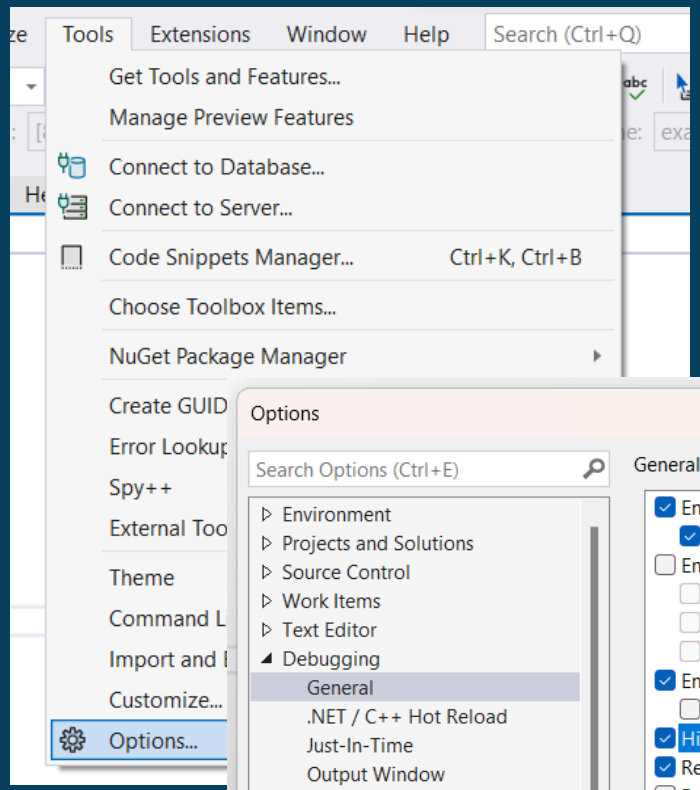
# Add Breakpoints

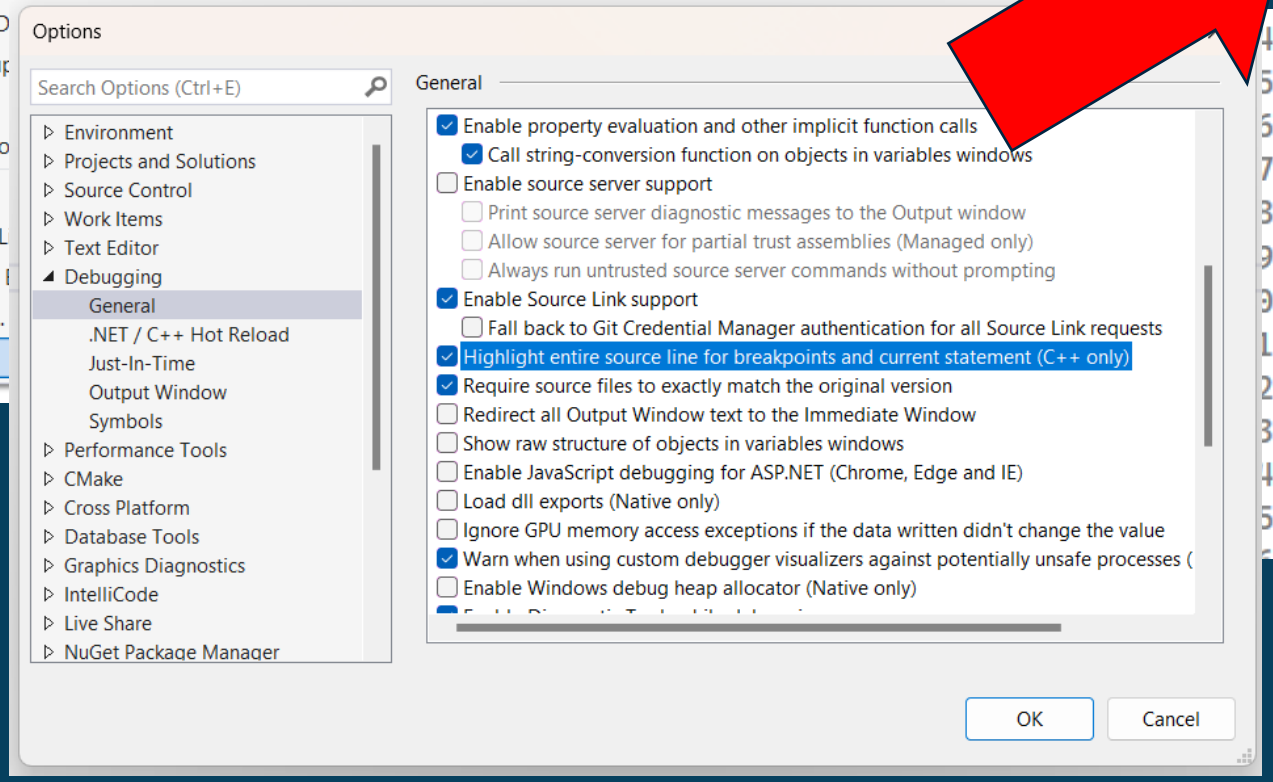
- Click on the left margin column
- Right click >> Breakpoint >> Insert Breakpoint
- Debug >> Toggle Breakpoint
- F9



# Highlight breakpoint




```
100  void example7()
101  {
102      int temp = getValue();
103      int a = 10;
104      int b = 20;   ≤ 1ms elapsed
105      int c = 30;
106      int d = a + b;
107      int e = c - a;
108  }
```

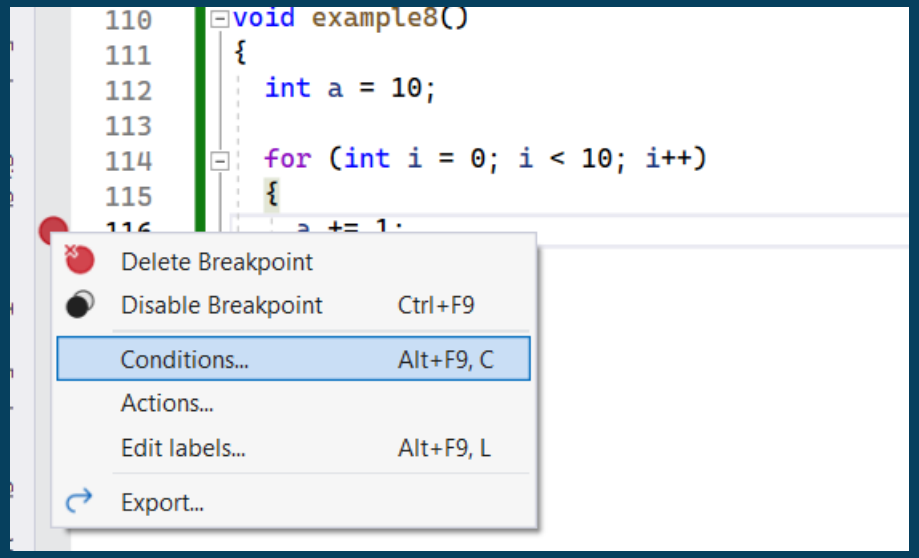
```
void example5()
{
    int a = 10;
    int b = 20;
    int c = 30;
    if (a > 0) std::cout << a << std::endl;
    if (b < 0)
        std::cout << b << std::endl;
    if (c > 0)
        std::cout << c << std::endl;
    std::cin.get();
}
```

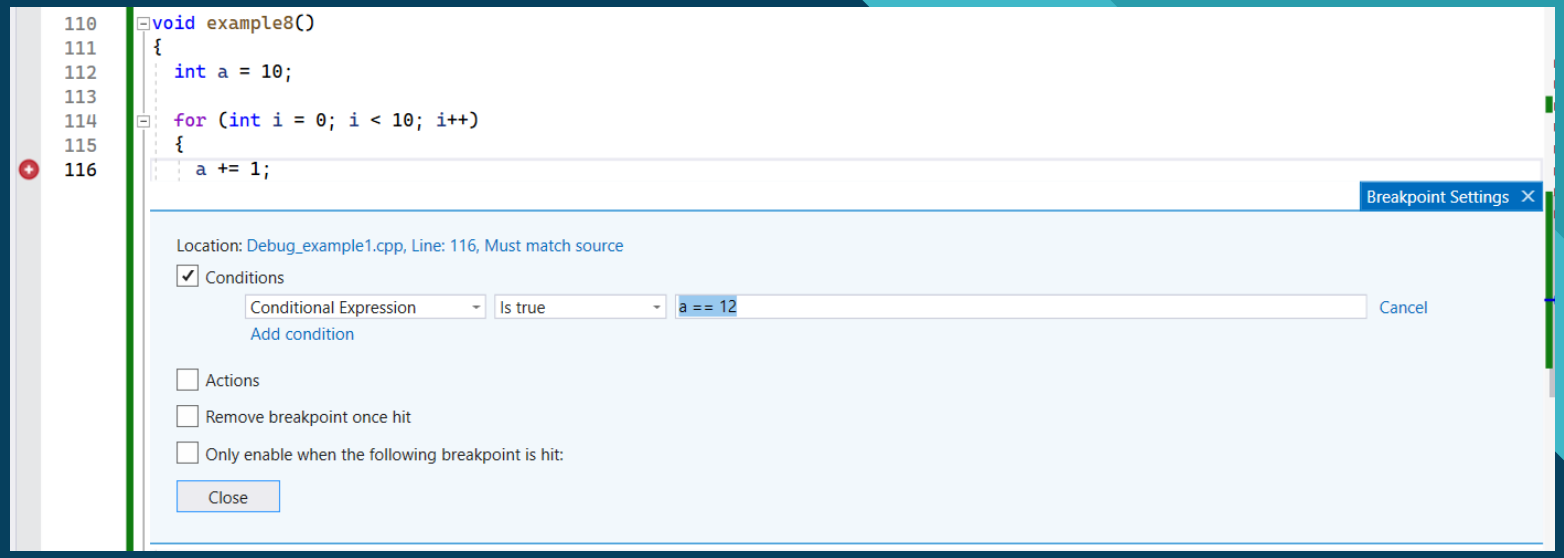
# Breakpoint conditions

- You can control when and where a breakpoint executes by setting conditions
- A condition can be any valid expression that the debugger recognizes.
- To set a breakpoint condition
  - Right lick the breakpoint symbol >> Conditions
  - Hover the breakpoint symbol >> Settings >> Conditions
  - Right lick the far left margin next to the line of code >> Insert Conditional Breakpoint
  - In breakpoint window, Right click >> Settings >> Conditions

# Breakpoint conditions

- Hit when the value is equals to 12




- You can add multiple conditions

# Breakpoint conditions

- If the breakpoint condition has invalid syntax, a warning message appears
- If the breakpoint condition has a valid syntax but invalid semantics, a warning message appears the first time the breakpoint hits.
- In either case, the debugger breaks when it hits the invalid breakpoint.
- The breakpoint is skipped only if the condition is valid and evaluates to “false”

**Example 8**

# Breakpoint conditions

```
126 void example9()
127 {
128   int a = 10;
129   for (int i = 0; i < 10000; i++)
130   {
131     a++;
```

Breakpoint Settings

Location: Debug_example1.cpp, Line: 131, Must match source

☑ Conditions

Hit Count | = | 345 | Cancel

Add condition

☐ Actions

☐ Remove breakpoint once hit

☐ Only enable when the following breakpoint is hit:

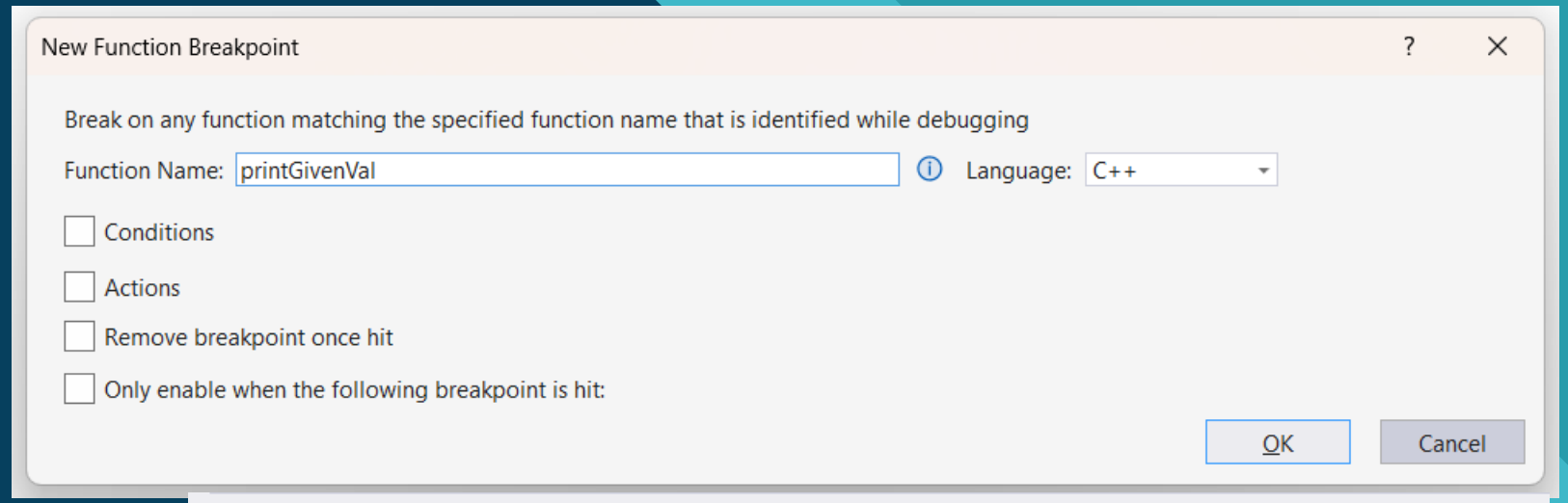
Close

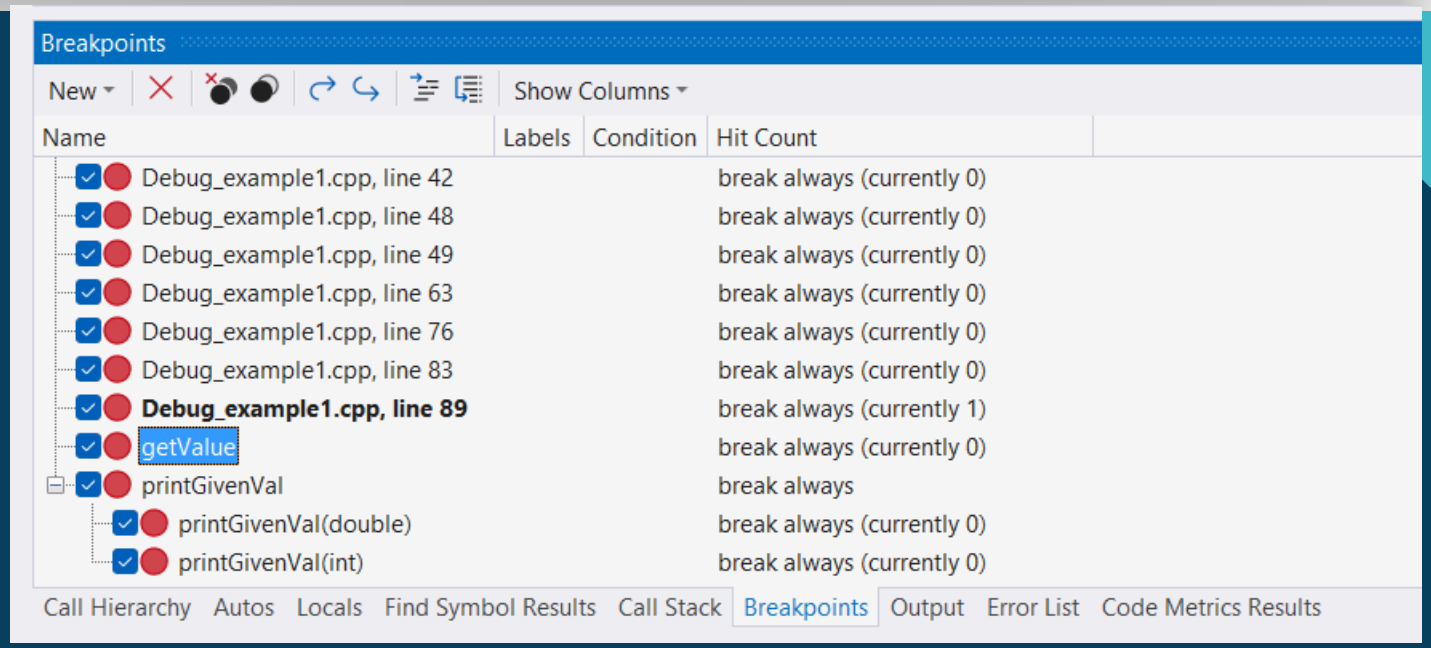
Example 9

# Run to a function breakpoint

- You can set the debugger to run until it reaches a specific function.
- Can specify the function by name
- **Debug >> New Breakpoint >> Function Breakpoint**
- In the dialog, enter the name of the function
- If the function is overloaded or in more than one namespace, you can choose the one you want

# Run to a function breakpoint

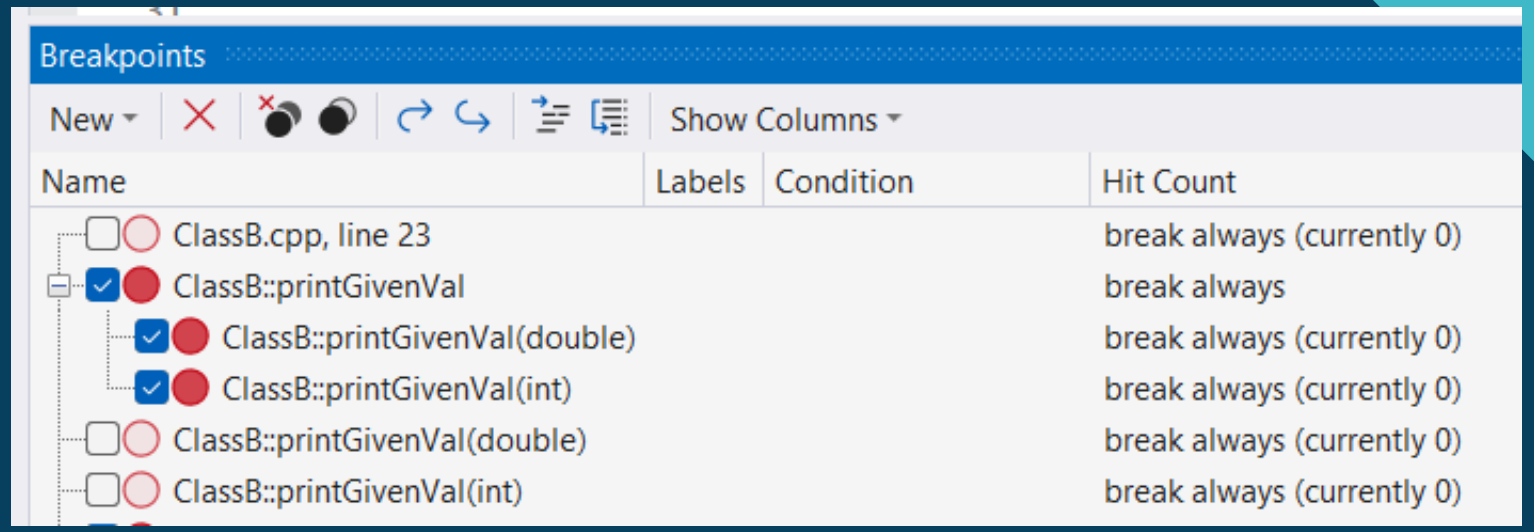



Example 6

# Function breakpoints

- To narrow the function specification
  - Use the fully qualified function name *Namespace1::ClassB::MethodA()*
  - Add parameters types of an overloaded function *MethodA(int,string)*




Example 6

# Dependent breakpoint

- Dependent breakpoints break execution only if another breakpoint is first hit.
- Hover the breakpoint >> Setting >> Only enable when the following breakpoint is hit
- 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

```
136  void example10(int a)
137  {
138      int c = 50;
139      if (a == 10)
140      {
141          c = 100;   ≤ 2ms elapsed
142      }
143      else if (a == 20)
144      {
145          c = 200;
146      }
147      else if (a == 30)
148      {
149          c = 300;
150      }
151
152      bool checkConditionA = true; // when hit c = 100
153      bool checkConditionC = true; // when hit c = 200
154      bool checkConditionB = true; // when hit c = 300
155  }
```

# Dependent breakpoint

```
141        c = 100;
142    }
143    else if (a == 20)
144    {
145        c = 200;
146    }
147    else if (a == 30)
148    {
149        c = 300;
150    }
151
152    bool checkConditionA = true; // when hit c = 100
153    bool checkConditionC = true; // when hit c = 200
154    bool checkConditionB = true; // when hit c = 300
155    }
156
157    int main() { ... }
```

Breakpoint Settings

Location: Debug_example1.cpp, Line: 154, Must match source

- [ ] Conditions
- [ ] Actions
- [ ] Remove breakpoint once hit
- [x] Only enable when the following breakpoint is hit: Debug_example1.cpp, line 149 ('example10(int a)')

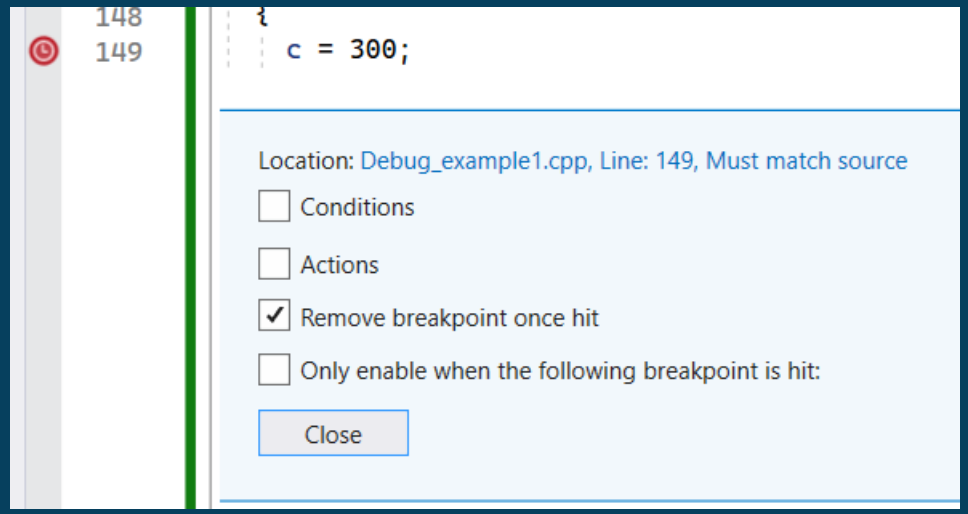
Close

Debug_example1.cpp, line 116 ('example8()')
Debug_example1.cpp, line 120 ('example8()')
Debug_example1.cpp, line 123 ('example8()')
Debug_example1.cpp, line 131 ('example9()')
Debug_example1.cpp, line 138 ('example10(int a)')
Debug_example1.cpp, line 141 ('example10(int a)')
Debug_example1.cpp, line 145 ('example10(int a)')
Debug_example1.cpp, line 149 ('example10(int a)')
Debug_example1.cpp, line 152 ('example10(int a)')

Example 10

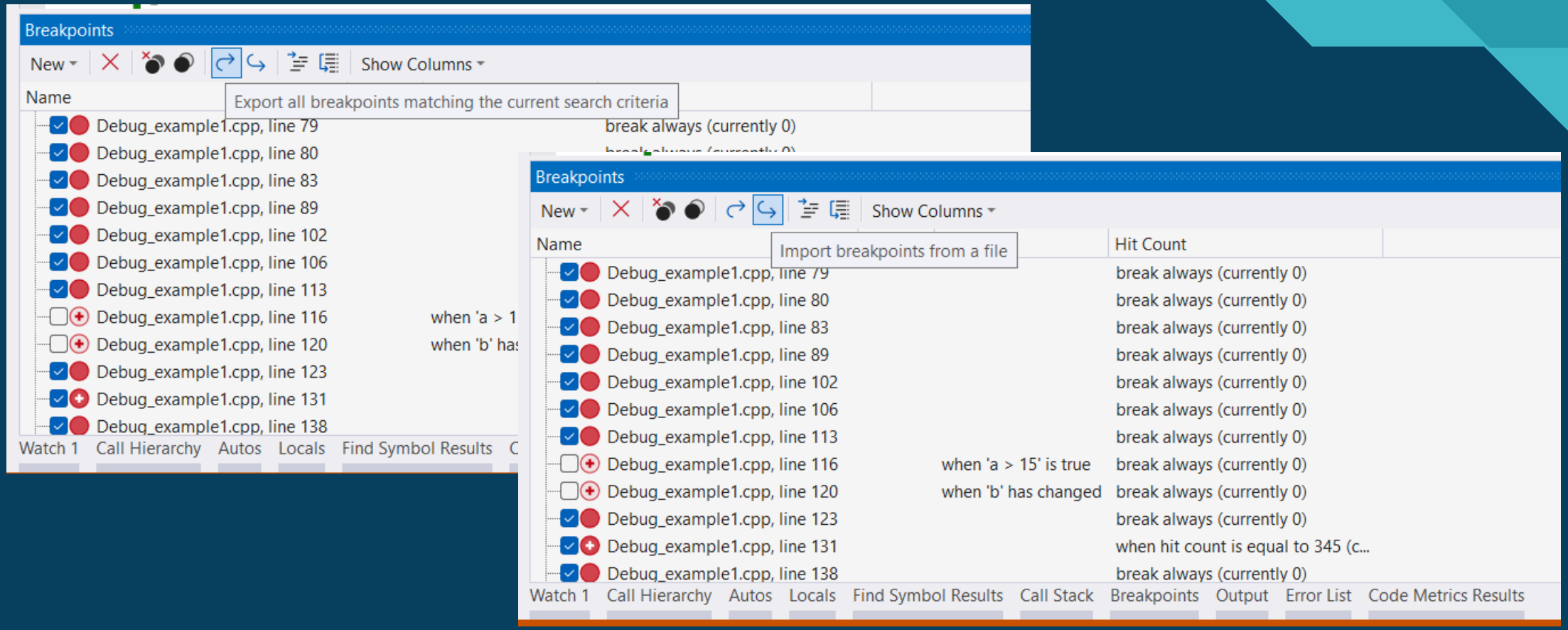
# Temporary Breakpoint

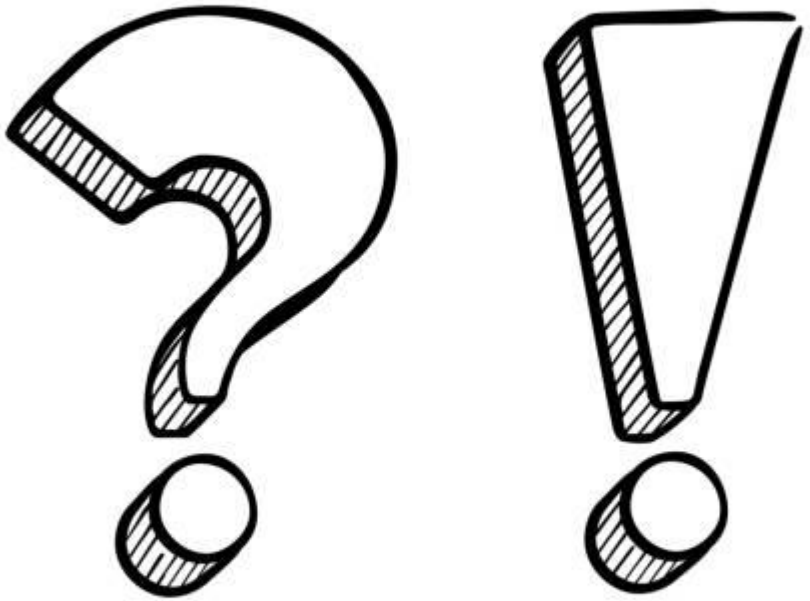
- This will break the code only once.
- Only pause the running application once for this breakpoint and then removes it immediately after it has been hit



# Import and export breakpoint

- You can import and export breakpoints using Breakpoint window





# Thank You