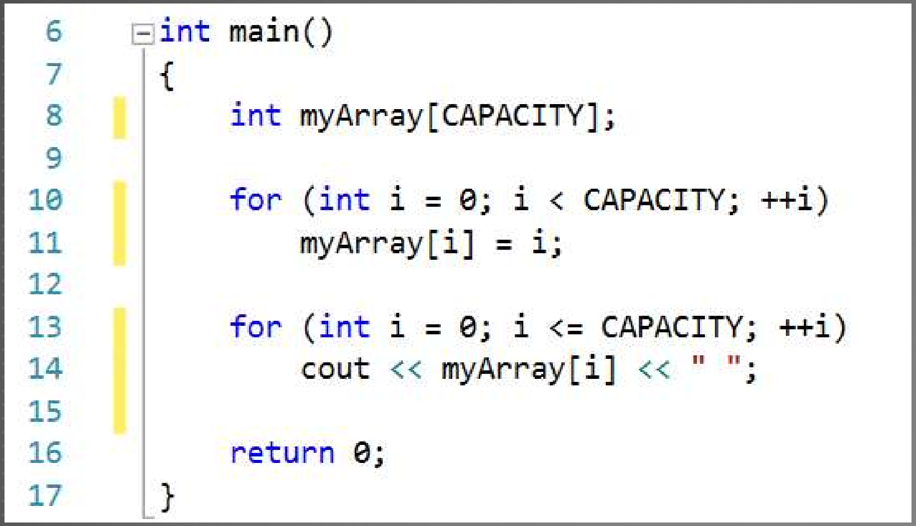
HOW TO USE THE DEBUGGER IN VISUAL STUDIO 2017

USING THE DEBUGGER

- The Debugger controls the execution of the application by
  - Stepping through the code one line at a time, OR
  - Running the program and stopping at a particular point in the program.
- At each point in your code where the debugger stops
  - You can inspect the code
  - You can view the value of each variable

EXAMINING AN OUT-OF-BOUND ERROR

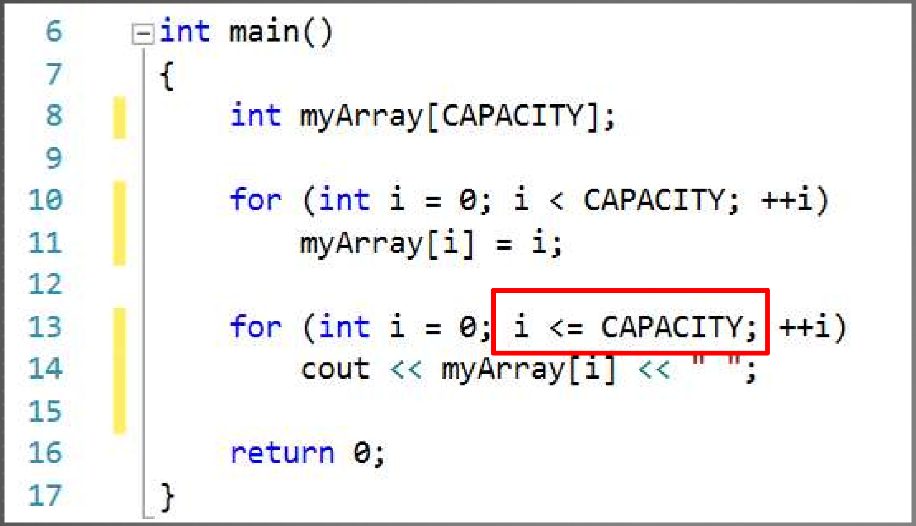
- Assume the code below is embedded in a program:



```
int main()
{
    int myArray[CAPACITY];

    for (int i = 0; i < CAPACITY; ++i)
        myArray[i] = i;

    for (int i = 0; i <= CAPACITY; ++i)
        cout << myArray[i] << " ";

    return 0;
}
```

EXAMINING AN OUT-OF-BOUND ERROR

- Assume the code below is embedded in a program:



```
int main()
{
    int myArray[CAPACITY];

    for (int i = 0; i < CAPACITY; ++i)
        myArray[i] = i;

    for (int i = 0; i <= CAPACITY; ++i)
        cout << myArray[i] << " ";

    return 0;
}
```

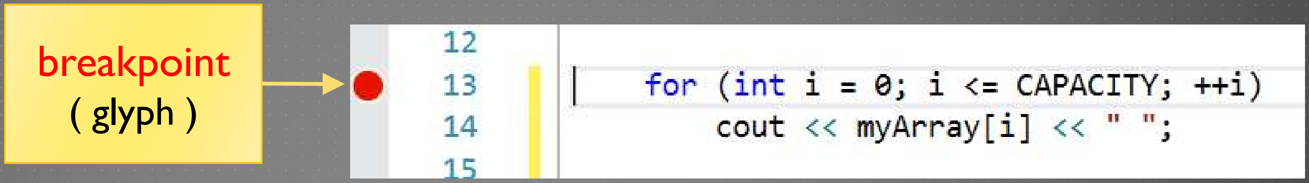
The second FOR loop will output a value at **index 5** of the array.

Since **the last index is 4**, the output will be incorrect:

```
0 1 2 3 4 -858993460
```

BREAKPOINTS

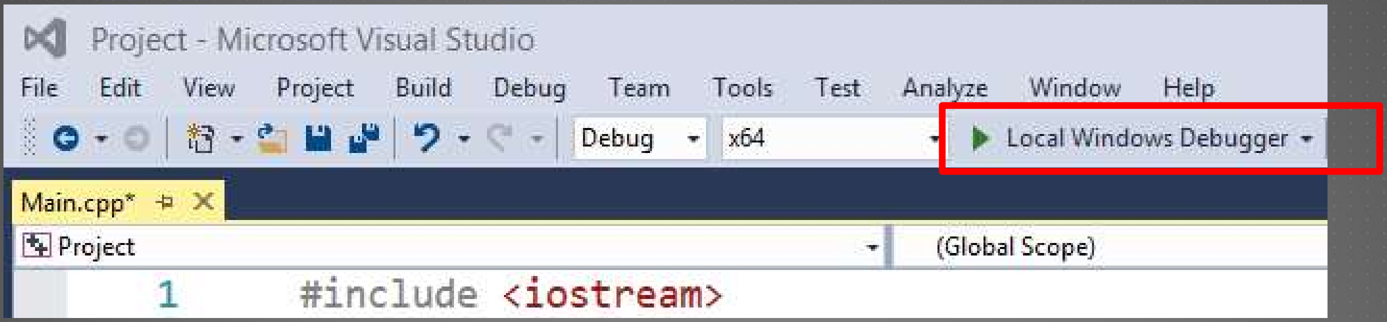
- When setting a **breakpoint** in debugging mode at a specific line of your code
  - The compiler will automatically suspend execution.
  - Note that you can set multiple breakpoints at the same time.
- To **set** and **remove** a **breakpoint**
  - Click in the grayed-out column to the left of the line number of the statement where you want execution to stop
  - A red circular symbol called a **glyph** appears showing the presence of a breakpoint at that line
  - To remove the breakpoint, you can double-click the glyph.



```
12
13 ●  for (int i = 0; i <= CAPACITY; ++i)
14         cout << myArray[i] << " ";
15
```

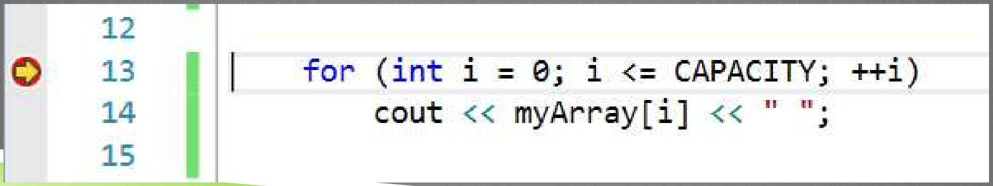
DEBUGGING

- To start debugging, simply press F5, or click the debugging icon.


Project - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x64
Local Windows Debugger
Main.cpp*
Project
(Global Scope)

```
#include <iostream>
```

- When the debugger reaches the **breakpoint**, the glyph will show a highlighted arrow.



```
    for (int i = 0; i <= CAPACITY; ++i)
        cout << myArray[i] << " ";
```

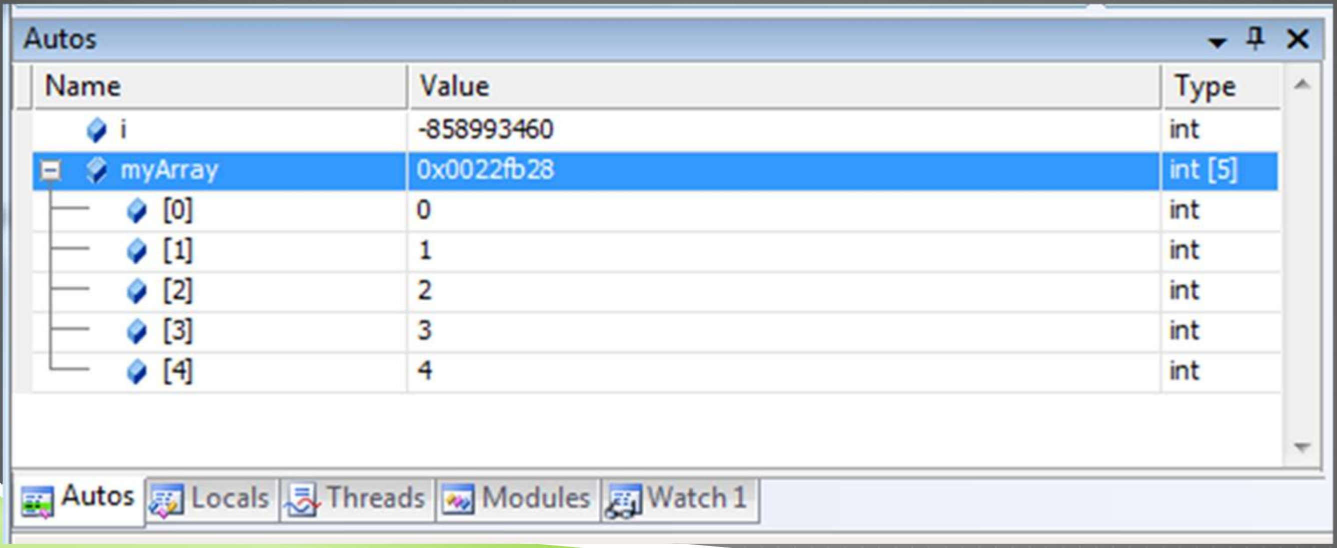
INSPECTING DATA

- When the Debugger starts, tabbed windows will appear
  - If they do not appear, select Debug | Windows and choose which window you want to display

- These are the most useful windows
  - **Autos**
    - Shows the content of variables in the context of the function.
  - **Locals**
    - Identifies the function calls currently in progress
  - **Watch**
    - Allows you to define any variable you wish to inspect

AUTOS WINDOW

- Shows current values for variables in the context of the function (the function in this example is **main()**) that is currently executing and its immediate predecessor (in other words, the statement pointed to by the **arrow** in the **Editor pane** and the one before it).



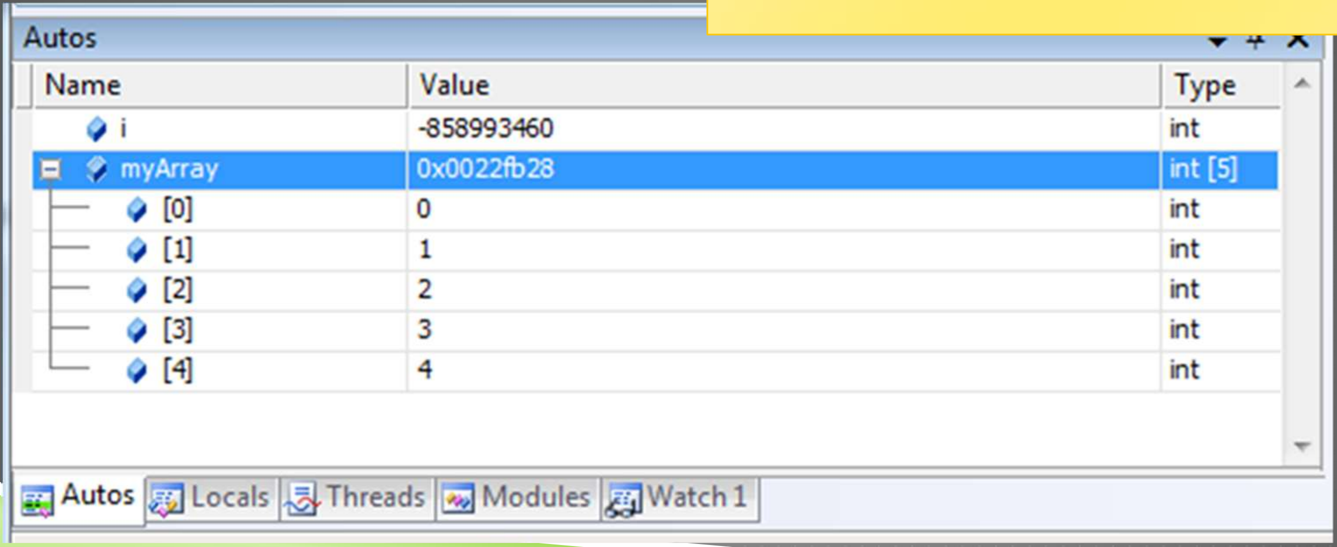
Name	Value	Type
i	-858993460	int
⊟ myArray	0x0022fb28	int [5]
[0]	0	int
[1]	1	int
[2]	2	int
[3]	3	int
[4]	4	int

AUTOS WINDOW

- Shows current values for variables in the contex **main()**) that is currently executing and its imm statement pointed to by the **arrow** in the **Edit**

Since the breakpoint is on the second loop, this window is showing only the current values (int **i** and array **myArray**).

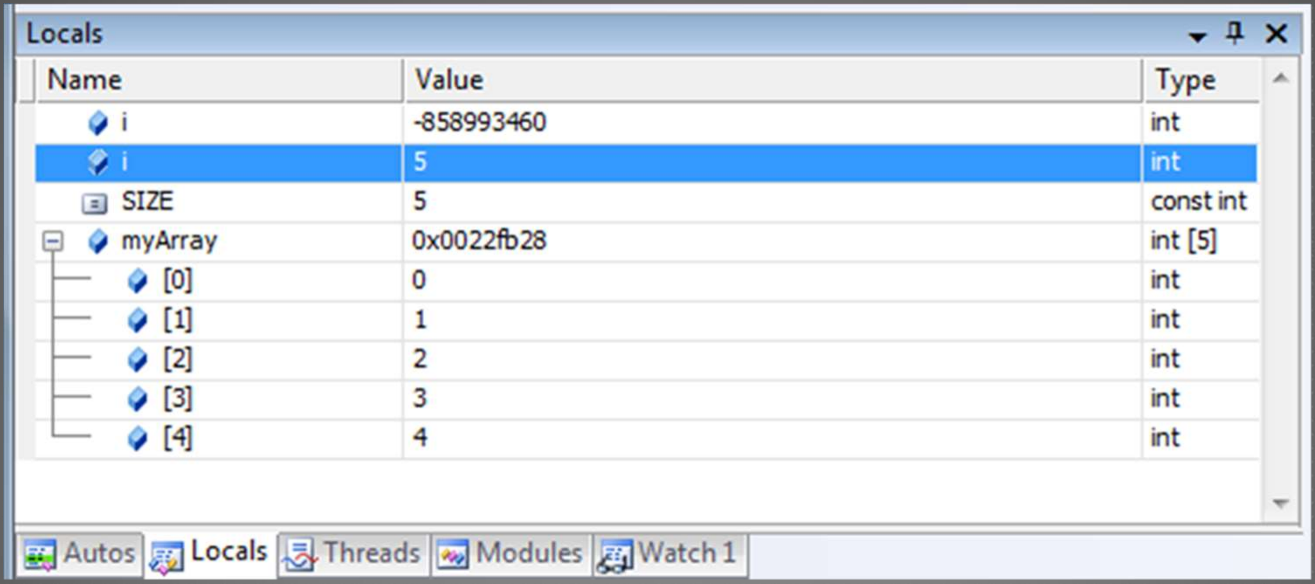
Note that int **i** has not been initialized yet and that **myArray** gives you only the address of the array.



Name	Value	Type
i	-858993460	int
myArray	0x0022fb28	int [5]
[0]	0	int
[1]	1	int
[2]	2	int
[3]	3	int
[4]	4	int

LOCALS WINDOW

- Identifies the function calls currently in progress (the function in this example is **main()**).



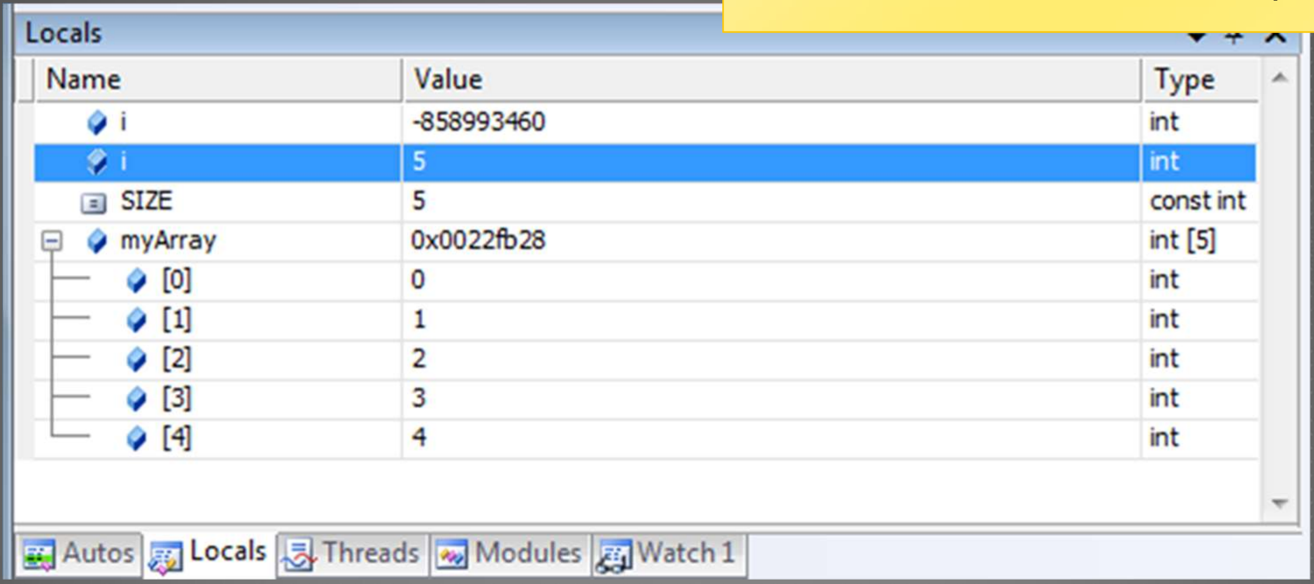
Locals

Name	Value	Type
i	-858993460	int
i	5	int
SIZE	5	const int
myArray	0x0022fb28	int [5]
[0]	0	int
[1]	1	int
[2]	2	int
[3]	3	int
[4]	4	int

LOCALS WINDOW

- Identifies the function calls currently in progress (t

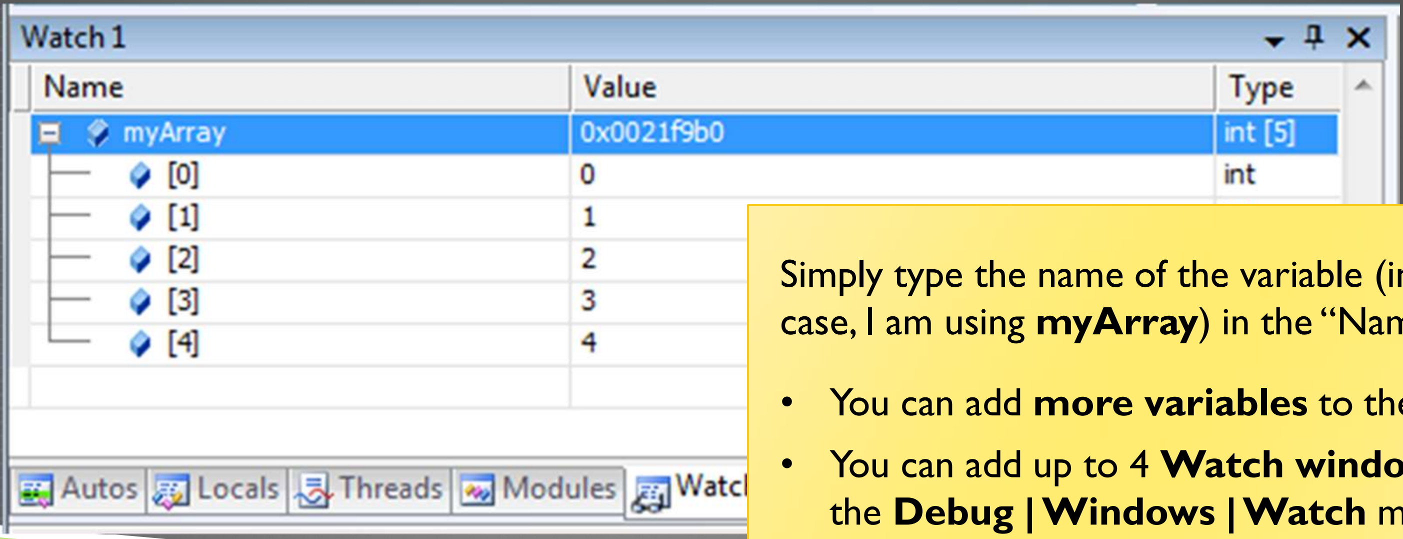
Note that there are **two** int **i**.
One corresponds to the first loop (highlighted), and the other to the second loop.



Name	Value	Type
i	-858993460	int
i	5	int
SIZE	5	const int
myArray	0x0022fb28	int [5]
[0]	0	int
[1]	1	int
[2]	2	int
[3]	3	int
[4]	4	int

WATCH WINDOW

- Allows you to define any variable you want to inspect (note that the variable needs to be initialized first so that the debugger knows that the variable exists).



Watch 1

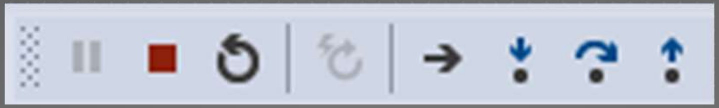
Name	Value	Type
myArray	0x0021f9b0	int [5]
[0]	0	int
[1]	1	
[2]	2	
[3]	3	
[4]	4	

Simply type the name of the variable (in this case, I am using **myArray**) in the "Name" line.

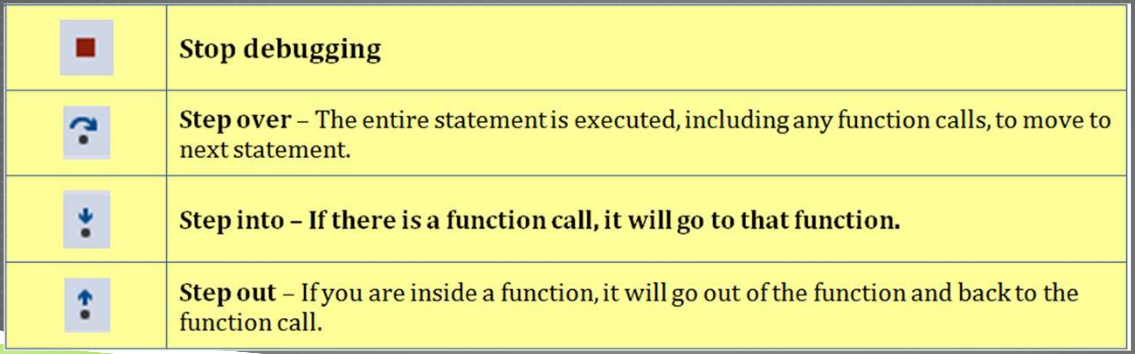
- You can add **more variables** to the list.
- You can add up to 4 **Watch windows** via the **Debug | Windows | Watch** menu.

THE DEBUG TOOLBAR

- The debugging toolbar is usually located on the top menu.

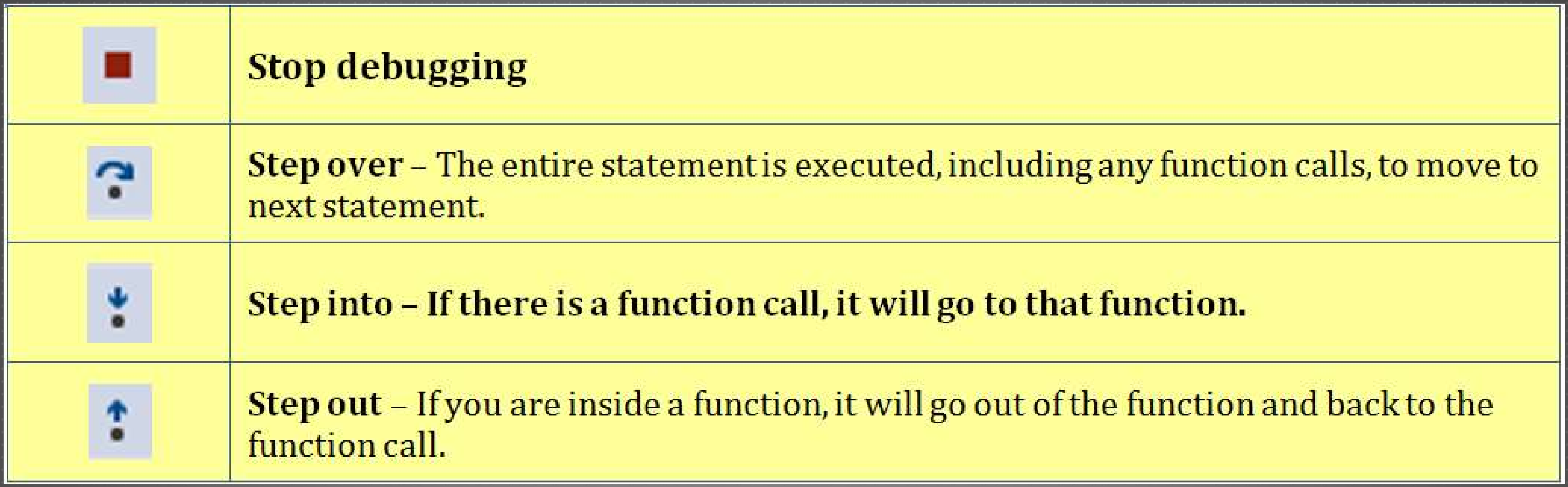


- To identify the function of each button, you can let the mouse cursor linger over the button.



■	**Stop debugging**
↷	**Step over** – The entire statement is executed, including any function calls, to move to next statement.
↓	**Step into – If there is a function call, it will go to that function.**
↑	**Step out** – If you are inside a function, it will go out of the function and back to the function call.

THE DEBUG TOOLBAR (CONT.)

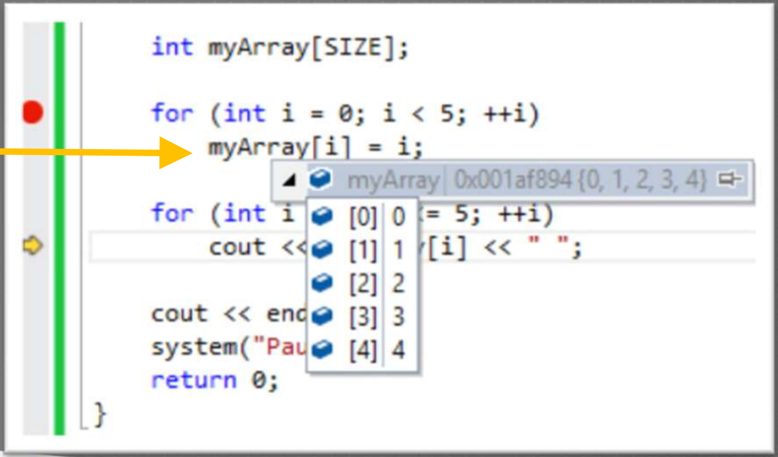


■	**Stop debugging**
↷	**Step over** – The entire statement is executed, including any function calls, to move to next statement.
↓	**Step into – If there is a function call, it will go to that function.**
↑	**Step out** – If you are inside a function, it will go out of the function and back to the function call.

VIEWING VARIABLES IN THE EDIT WINDOW

- You can also look at the value of a variable directly from the **Text Editor window**
  - Position the cursor over the variable for a second
  - A tool tip pops up showing the current value of the variable
  - To look at more complicated expressions, highlight the current value displayed and rest the cursor over the highlighted area.

cursor is positioned here



```
    int myArray[SIZE];

    for (int i = 0; i < 5; ++i)
        myArray[i] = i;
              ◢ myArray  0x001af894 {0, 1, 2, 3, 4}
    for (int i   [0] 0   = 5; ++i)
        cout <<  [1] 1  [i] << " ";
                 [2] 2
    cout << end  [3] 3
    system("Pau  [4] 4
    return 0;
}
```

ADDITIONAL HELP

- You can find additional resources at
  - https://msdn.microsoft.com/en-us/library/sc65sadd.aspx