

## How to Use the Microsoft Visual Studio Debugger

### Using the Debugger

The Debugger is a program that controls the execution of your application in such a way that you can step through the source code one line at a time, or run to a particular point in the program. At each point in your code where the debugger stops, you can inspect, or even change the values of variables before continuing. You can also change the source code, recompile, and then restart the program from the beginning.

---

### Example

We will use a simple application that contains an error to learn how to use the Debugger.

```
// Using the debugger
#include <iostream>
using namespace std;

const int CAPACITY = 5;

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] << " ";

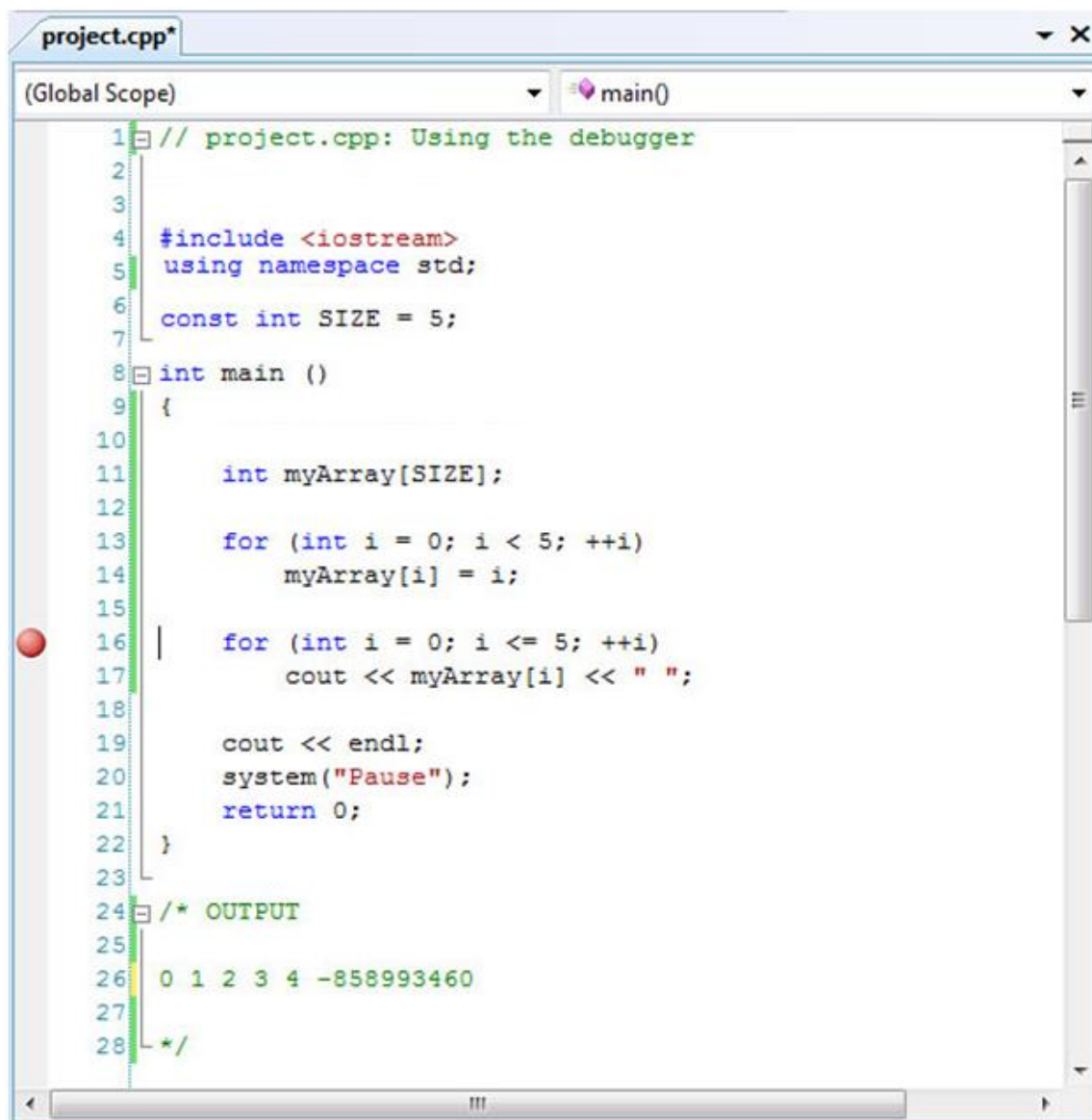
    cout << endl;
    system("Pause");
    return 0;
}

/* OUTPUT
0 1 2 3 4 -858993460
*/
```

In this example, you can see that the second **FOR** loop will output the value at index 5 of the array. Since the last index of the array is 4, the compiler will output some number (-858993460) or terminate the program.

The first thing you need to do is set a **BREAKPOINT**. A breakpoint is a point in your program where the debugger automatically suspends execution when in debugging mode. You can specify multiple breakpoints so that you can run your program, stopping at points of interest that were selected along the way.


To **set a breakpoint** at the beginning of a line of source code, you simply click in the grayed-out column to the left of the line number for the statement where you want execution to stop. A red circular symbol called a glyph appears showing the presence of the breakpoint at that line. To **remove the breakpoint**, you can double-click the glyph.



The screenshot shows a C++ IDE window titled 'project.cpp'. The code is as follows:

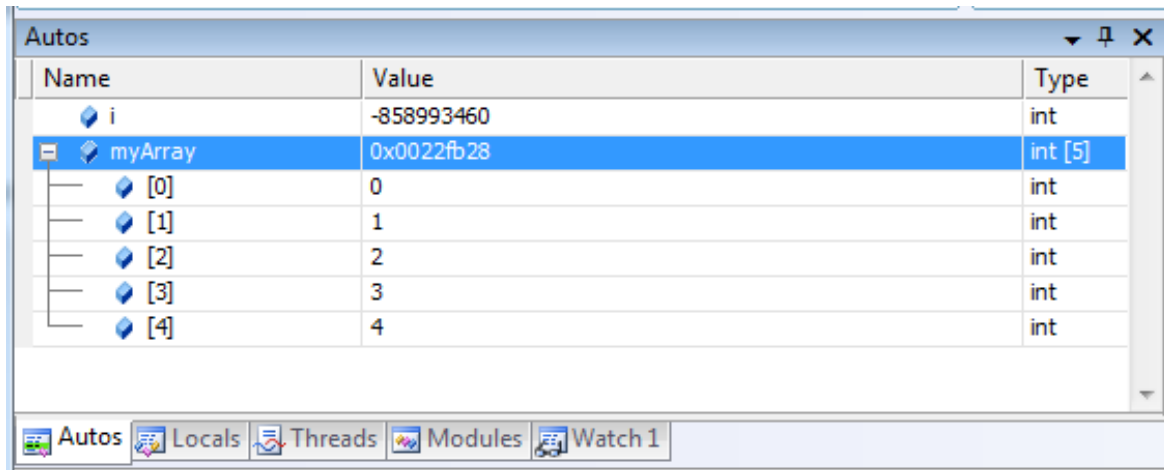
```
1 // project.cpp: Using the debugger
2
3
4 #include <iostream>
5 using namespace std;
6 const int SIZE = 5;
7
8 int main ()
9 {
10
11     int myArray[SIZE];
12
13     for (int i = 0; i < 5; ++i)
14         myArray[i] = i;
15
16     for (int i = 0; i <= 5; ++i)
17         cout << myArray[i] << " ";
18
19     cout << endl;
20     system("Pause");
21     return 0;
22 }
23
24 /* OUTPUT
25
26 0 1 2 3 4 -858993460
27
28 */
```

A red circular breakpoint glyph is visible in the left margin next to line 16. The IDE interface includes a toolbar at the top with a 'main()' button and a status bar at the bottom.

To **start debugging**, simply compile your program. The Debugger will halt execution when it reaches the breakpoint, and the breakpoint will change to display a highlighted arrow: 

When the Debugger starts, tabbed windows appear below the **Editor window**. If you select **Debug | Windows**, you can choose the windows you want to display. The following windows are the most useful at your level of programming:

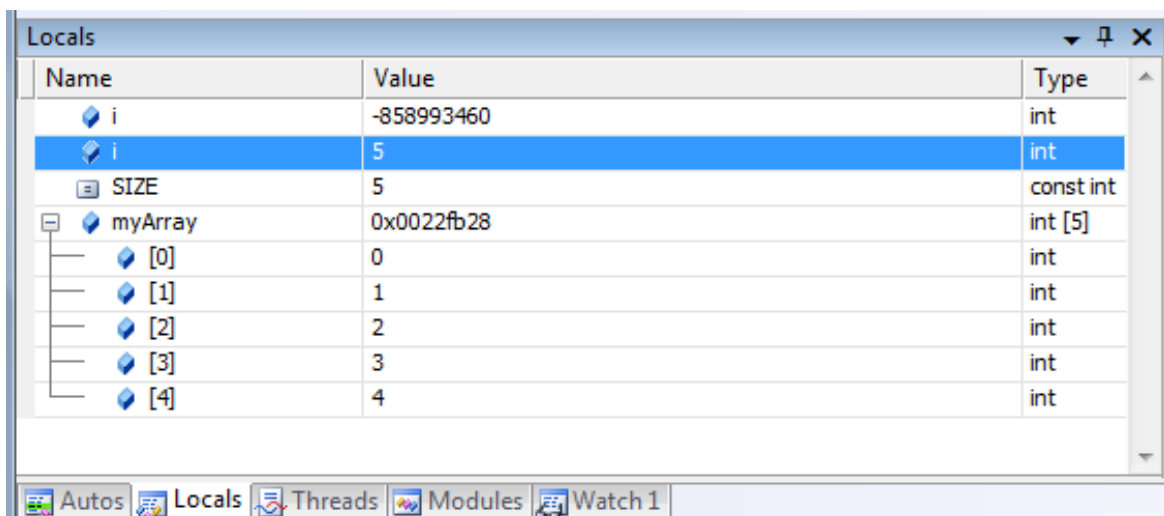
- **Autos window** – Shows current values for variables in the context of the function (the function in this example is **main()**) that is currently executing and its immediate predecessor (in other words, the statement pointed to by the **arrow** in the **Editor pane** and the one before it).



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

Since the breakpoint is on the second loop, this window is showing only the current values (the int **i** and the array **myArray**). Note that int **i** has not been initialized yet and that **myArray** gives you the address of the array.

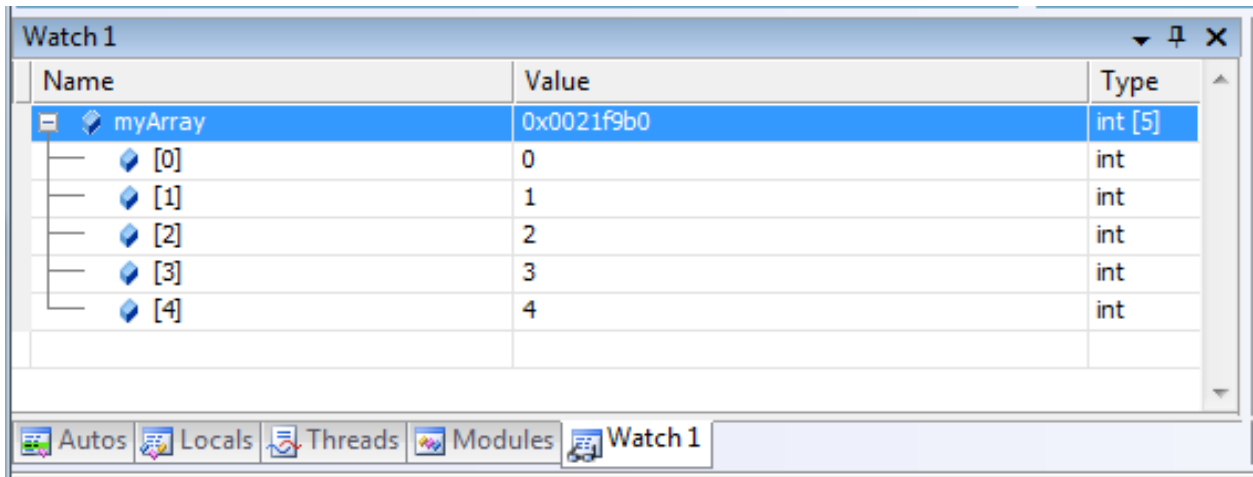
- **Locals window** – Identifies the function calls currently in progress (the function in this example is **main()**).



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

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

- **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).







To accomplish this, you simply type the name of the variable (in this case, I am using **myArray**) in the “Name” line. Of course, you can add more variables to the list.

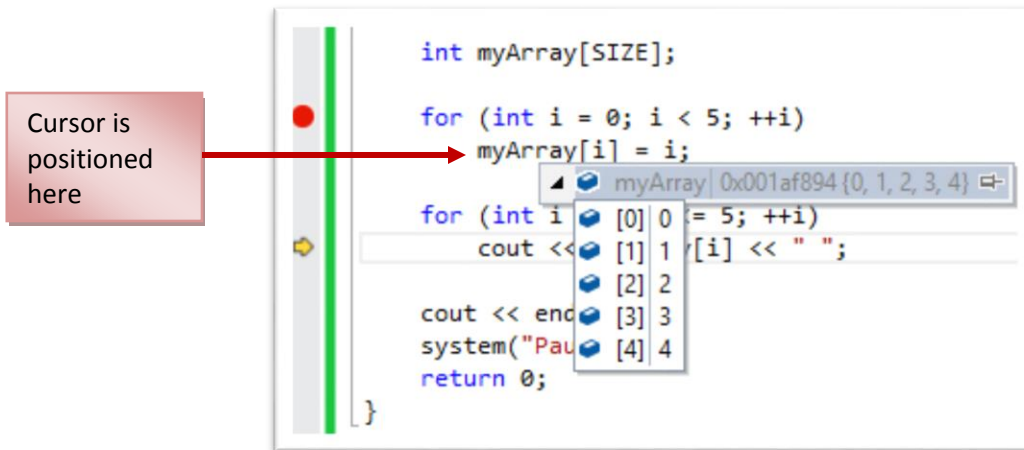
You can add up to four **Watch windows** via the **Debug | Windows | Watch** menu.

**The Debug Toolbar** – The debugging toolbar is usually located on the top menu. You can find out what the toolbar buttons are for by letting the mouse cursor linger over a toolbar button. A tool tip for that button appears that identifies its function.



	<b>Stop debugging</b>
	<b>Step over</b> – The entire statement is executed, including any function calls, to move to next statement.
	<b>Step into</b> – If there is a function call, it will go to that function.
	<b>Step out</b> – 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** – If you need to look at the value of a single variable, and that variable is visible in the **Text Editor window**, the easiest way to look at its value is to position the cursor over the variable for a second. A tool tip pops up showing the current value of the variable. You can also look at more complicated expression by highlighting them and resting the cursor over the highlighted area.



These are only a few techniques. To learn more about how to debug in Visual Studio go to <http://msdn.microsoft.com/en-us/library/sc65sadd.aspx>.