Debugging Techniques

HoangND1

Objectives

- ❑ Basic Debugging Technique
- ❑ Breakpoints
- ❑ Watches
- ❑ Stepping
- ❑ Stopping the Debugger
- ❑ Conditions and Hit Counts
- ❑ Break on Exception
- ❑ Step Into
- ❑ Trace and Assert

Basic Debugging Technique

- The debugger is a tool to help correct runtime and semantic errors
- note that no debugging tools are useful in solving *compiler errors.*
- Compiler errors are those that show at the bottom of the screen when compiling

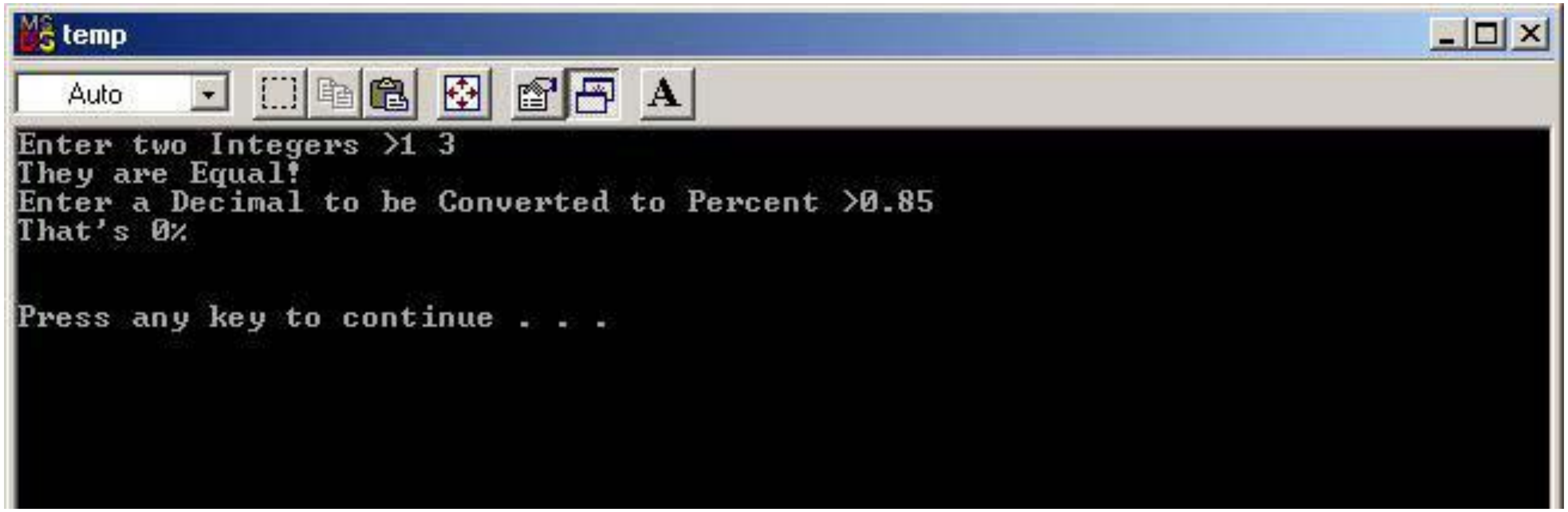
Basic Debugging Technique

- ❑ If the program isn't working correctly, one of two things could be going wrong:
  - ✓*Data is corrupt somewhere*
  - ✓The code isn’t correct
- ❑ Example

```
int a = 0;
int b = 1;
printf("%d", (b/a));
```

A Buggy Program

- Trying to debug a program that's working perfectly is rather pointless



```
Enter two Integers >1 3
They are Equal!
Enter a Decimal to be Converted to Percent >0.85
That's 0%


Press any key to continue . . .
```

The Buggy Code

```
#include <stdio.h>
int toPercent (float decimal);

int main()
{
        int a, b;
        float c;
        int cAsPercent;
        printf("Enter A >");
        scanf("%d", &a);
        printf("Enter B >");
        scanf("%d", &b);

        if (a = b) printf("They are Equal!\n");
        else if (a > b) printf("The first one is bigger!\n");
        else printf("The second one is bigger!\n");
        printf("Enter a Decimal to be Converted to Percent >");
        scanf("%f", &c);
        cAsPercent = toPercent(c);
        printf("That's %d %\n", cAsPercent);
        printf("\n\n");
        getchar();
        return 0;
}
```

The Buggy Code

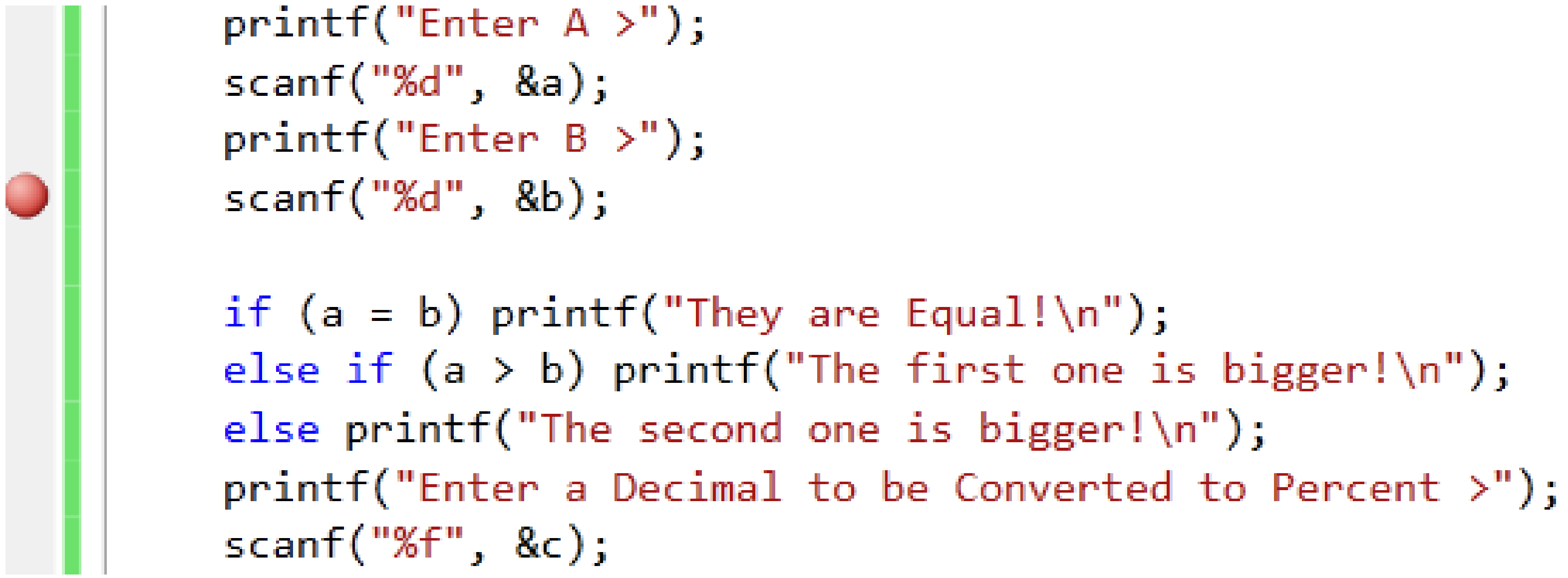
```
/* ToPercent():
Converts a given float (eg 0.9) to a percentage (90).
*/
int toPercent (float decimal) {
        int result;
        result = int(decimal) * 100;
        return result;
}
```

Debug Mode or Not?

- Ctrl+F5 to run your program
- The F5 key alone will also run in debug mode.
- Build for Debug
- Build for Release

Breakpoints

- Breakpoints are the lifeblood of debugging.
- Right-click and select "Insert/Remove Breakpoint" or press the F9 key

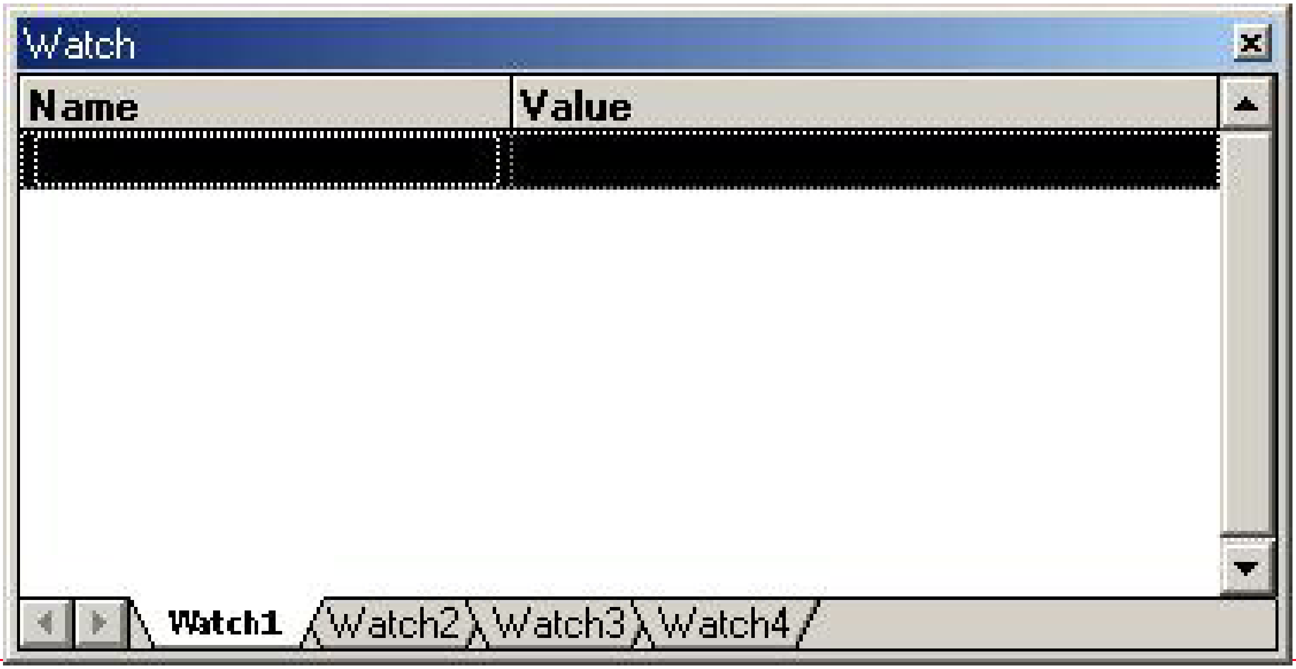


```
printf("Enter A >");
scanf("%d", &a);
printf("Enter B >");
scanf("%d", &b);

if (a = b) printf("They are Equal!\n");
else if (a > b) printf("The first one is bigger!\n");
else printf("The second one is bigger!\n");
printf("Enter a Decimal to be Converted to Percent >");
scanf("%f", &c);
```

Watches

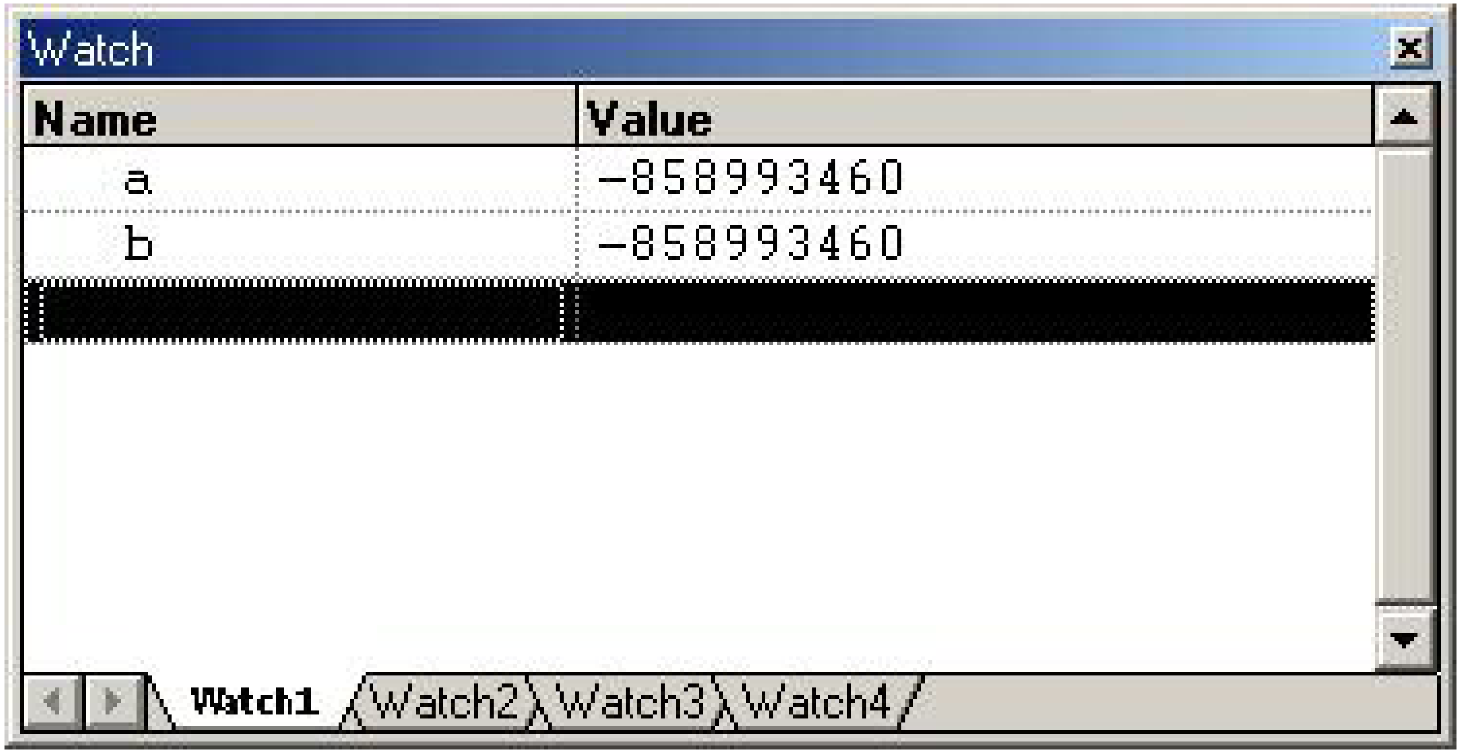
- The "Watch" window lets you *watch the contents of any variables you select as your program* executes.
- Open it from the View menu (Debug Windows >

  Watch), or by clicking the "Watch" icon in the toolbar, or by pressing Alt+3



Watches

- Enter to add *variables to your Watch list:*

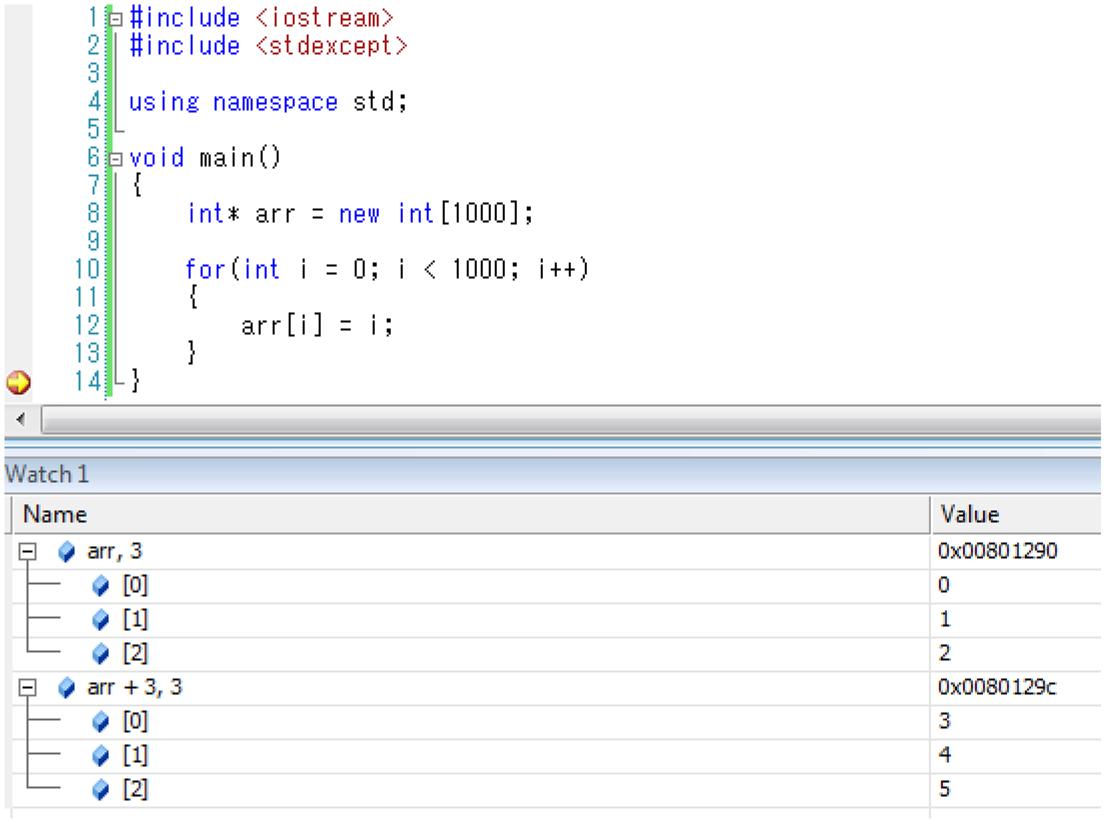


Name	Value
a	-858993460
b	-858993460

Watches

- Watch a range of values inside array:

  Syntax: array + <offset>, <range>



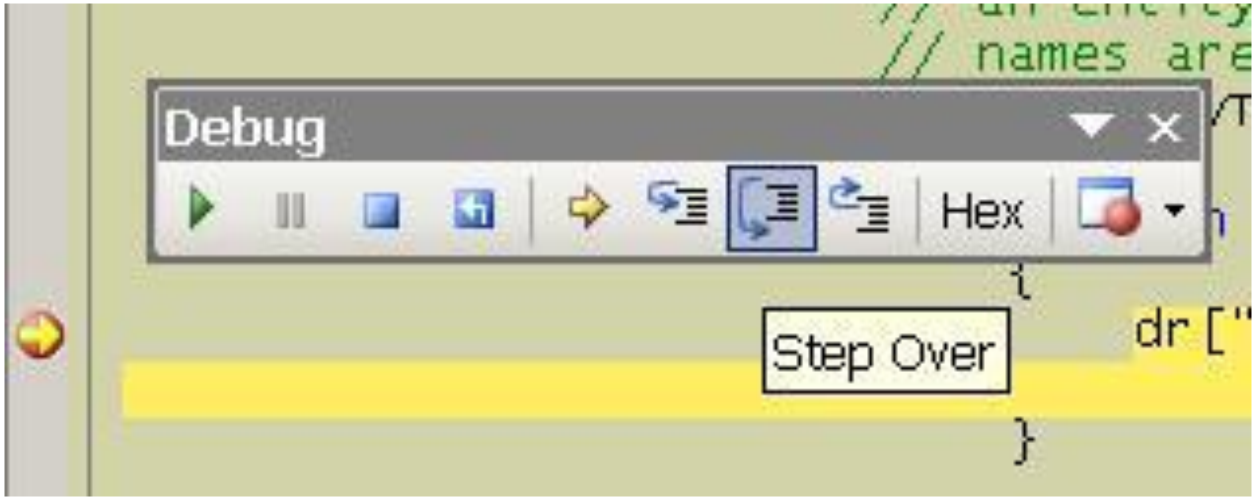
```cpp
#include <iostream>
#include <stdexcept>

using namespace std;

void main()
{
    int* arr = new int[1000];

    for(int i = 0; i < 1000; i++)
    {
        arr[i] = i;
    }
}
```

Watch 1

```
Name            Value
⊟ arr, 3        0x00801290
    [0]         0
    [1]         1
    [2]         2
⊟ arr + 3, 3    0x0080129c
    [0]         3
    [1]         4
    [2]         5
```

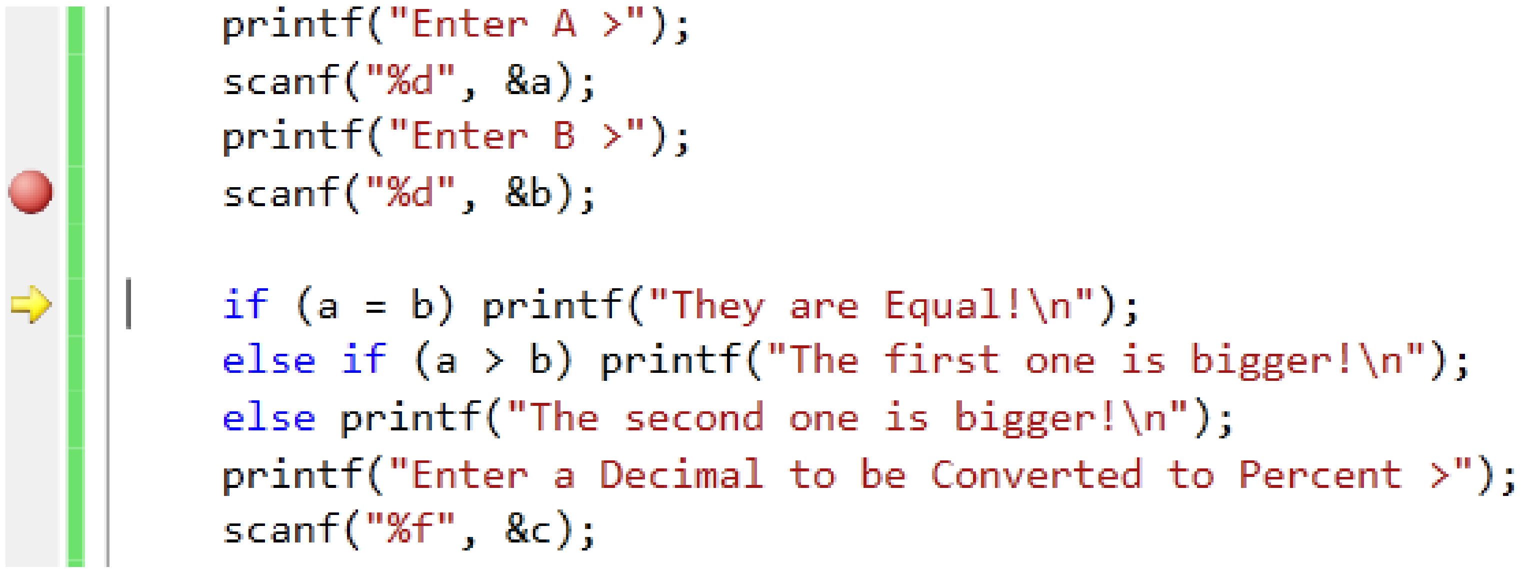
Stepping

- Step Over F10
- Step Into F11 (Some code inside a function may *or may not need to be examined*)
- Step Out Shift + F11



Stepping

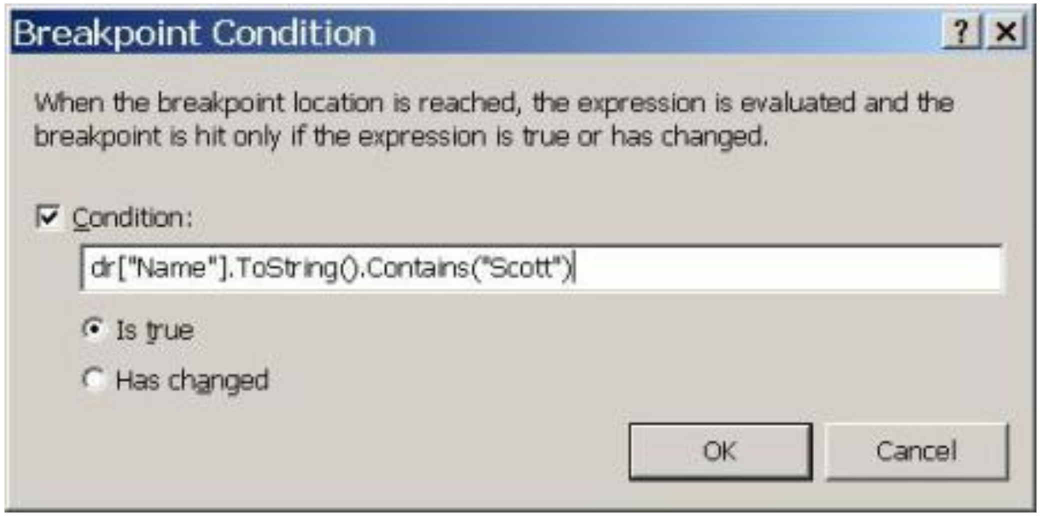
- When you are tired of stepping through the code, F5 resumes execution.



```
printf("Enter A >");
scanf("%d", &a);
printf("Enter B >");
scanf("%d", &b);

if (a = b) printf("They are Equal!\n");
else if (a > b) printf("The first one is bigger!\n");
else printf("The second one is bigger!\n");
printf("Enter a Decimal to be Converted to Percent >");
scanf("%f", &c);
```

Stopping the Debugger

When you've found a problem to correct, it may be tempting to press Ctrl+C in your program window to end the program


Breakpoint Condition

When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

☑ Condition:

```
dr["Name"].ToString().Contains("Scott")
```

● Is true

○ Has changed

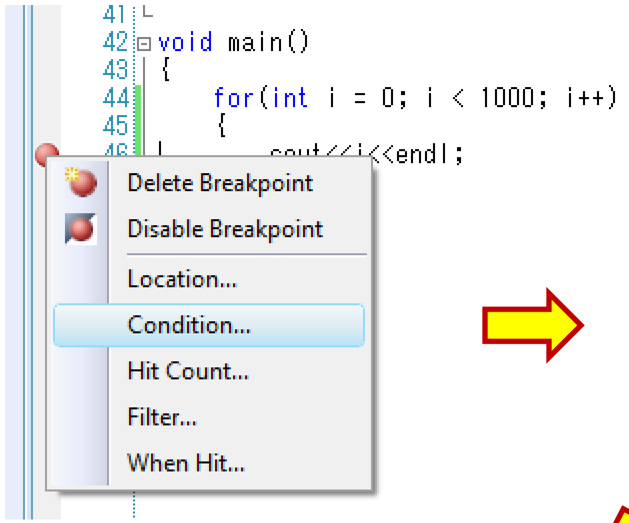
Select "Stop Debugging" from the Debug menu or on the toolbar or press Shift+F5.

Conditions and Hit Counts

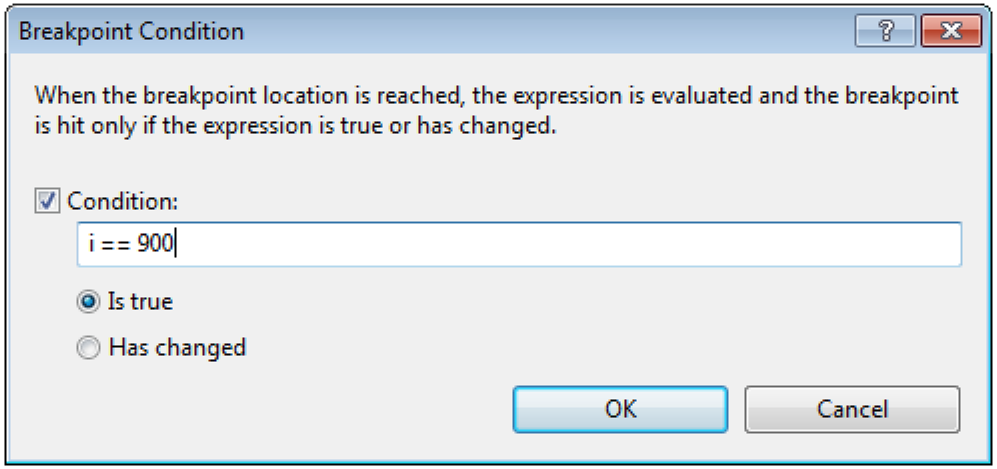
- Breakpoint can use conditions and hit counts
- Conditions and hit counts are useful if you don’t want the debugger to halt execution *every time the program reaches the breakpoint*
- Only when a condition is true, or a condition has changed, or execution has reached the breakpoint a specified number of times.

Conditions and Hit Counts

- Condition: Is true



```
void main()
{
    for(int i = 0; i < 1000; i++)
    {
        cout<<i<<endl;
```



Breakpoint Condition

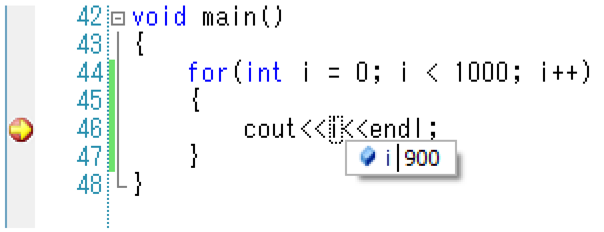
When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

[✓] Condition:

i == 900

[●] Is true

[ ] Has changed

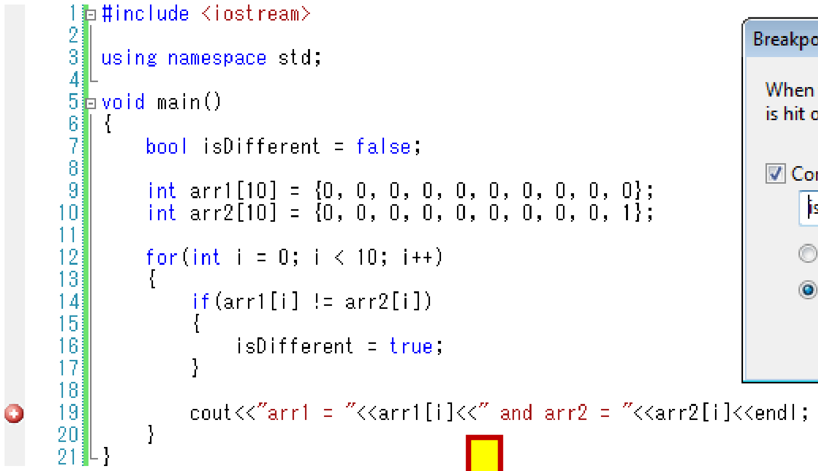


```
void main()
{
    for(int i = 0; i < 1000; i++)
    {
        cout<<i<<endl;
    }
}
```

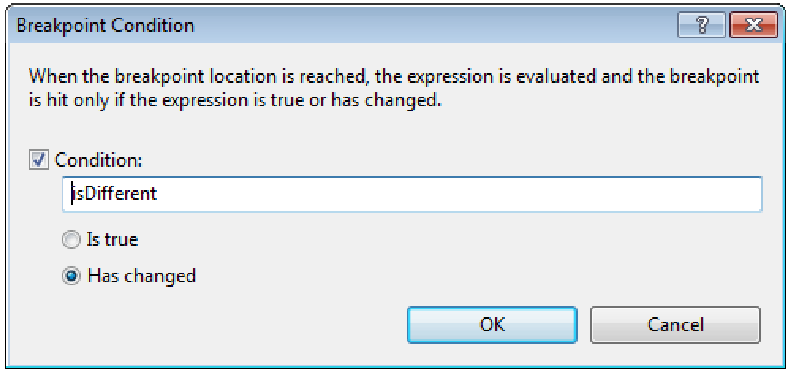
i | 900

Conditions and Hit Counts

❑ Condition: Has changed



```
#include <iostream>

using namespace std;

void main()
{
    bool isDifferent = false;

    int arr1[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    int arr2[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 1};

    for(int i = 0; i < 10; i++)
    {
        if(arr1[i] != arr2[i])
        {
            isDifferent = true;
        }

        cout<<"arr1 = "<<arr1[i]<<" and arr2 = "<<arr2[i]<<endl;
    }
}
```



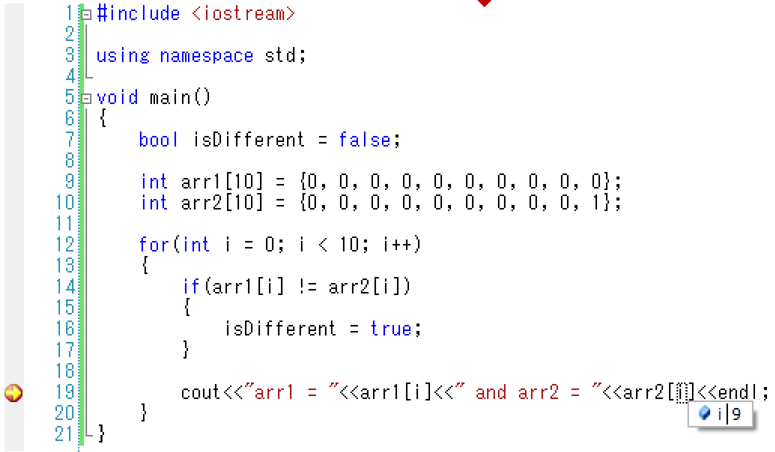
Breakpoint Condition

When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

[✓] Condition: isDifferent

( ) Is true

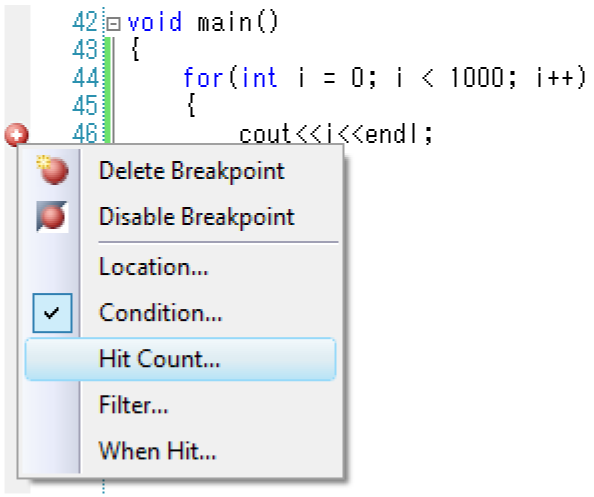
(●) Has changed



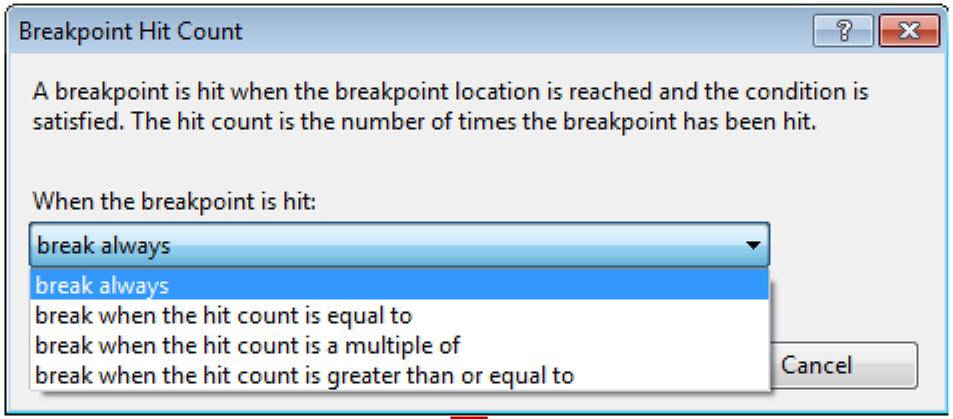
```
#include <iostream>

using namespace std;

void main()
{
    bool isDifferent = false;

    int arr1[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    int arr2[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 1};

    for(int i = 0; i < 10; i++)
    {
        if(arr1[i] != arr2[i])
        {
            isDifferent = true;
        }

        cout<<"arr1 = "<<arr1[i]<<" and arr2 = "<<arr2[i]<<endl;
    }
}
```

Conditions and Hit Counts

- Hit Count: is a multiple of



```
void main()
{
    for(int i = 0; i < 1000; i++)
    {
        cout<<i<<endl;
```



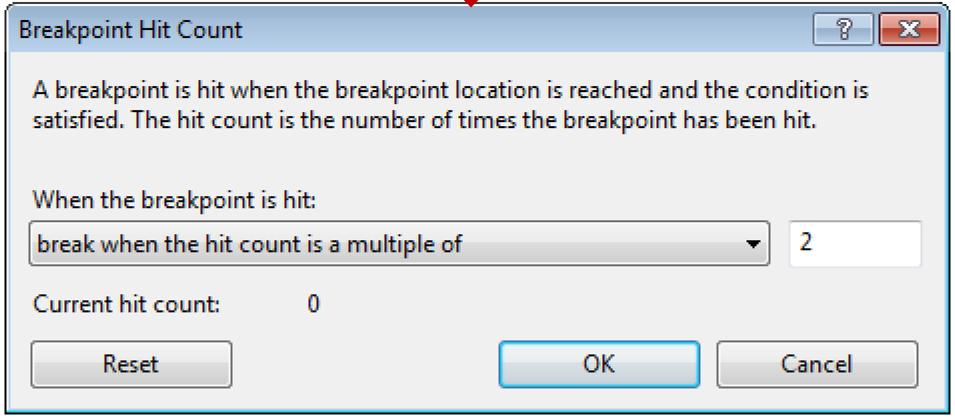
Breakpoint Hit Count

A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.

When the breakpoint is hit:

break always

- break always
- break when the hit count is equal to
- break when the hit count is a multiple of
- break when the hit count is greater than or equal to



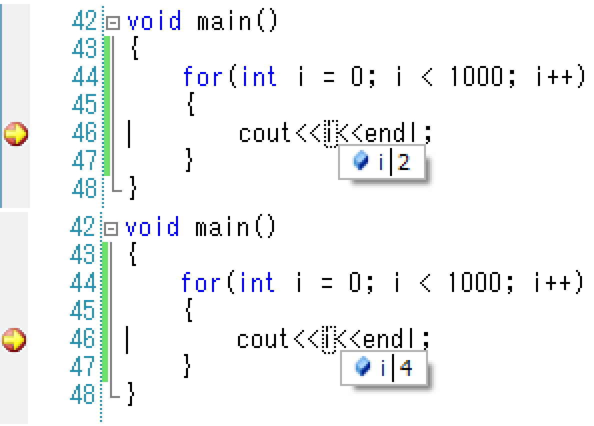
Breakpoint Hit Count

A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.

When the breakpoint is hit:

break when the hit count is a multiple of — 2

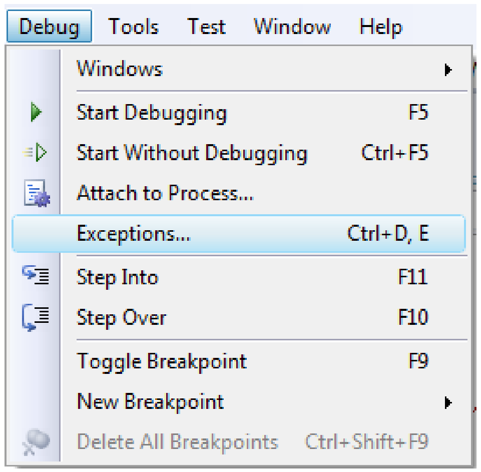
Current hit count: 0

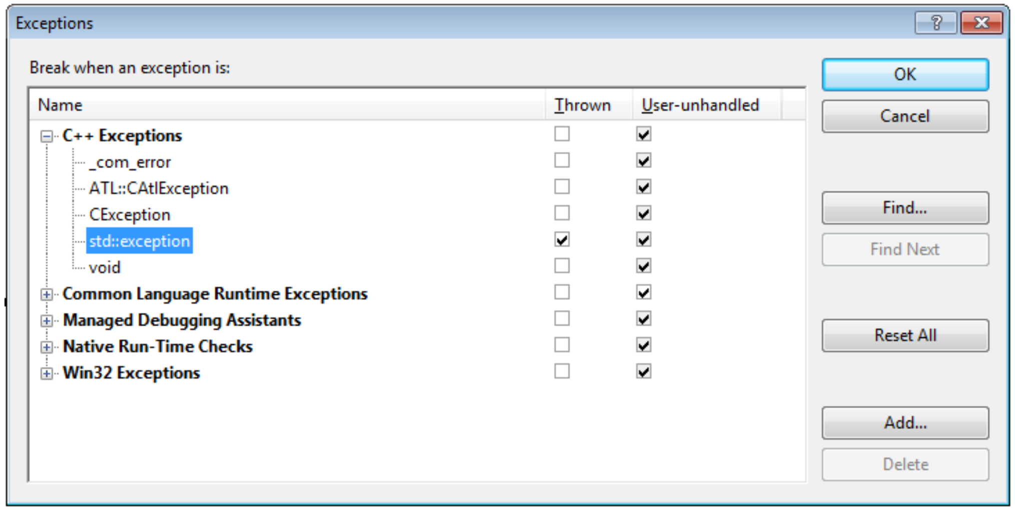


```
void main()
{
    for(int i = 0; i < 1000; i++)
    {
        cout<<i<<endl;     i | 2
    }
}
```

```
void main()
{
    for(int i = 0; i < 1000; i++)
    {
        cout<<i<<endl;     i | 4
    }
}
```

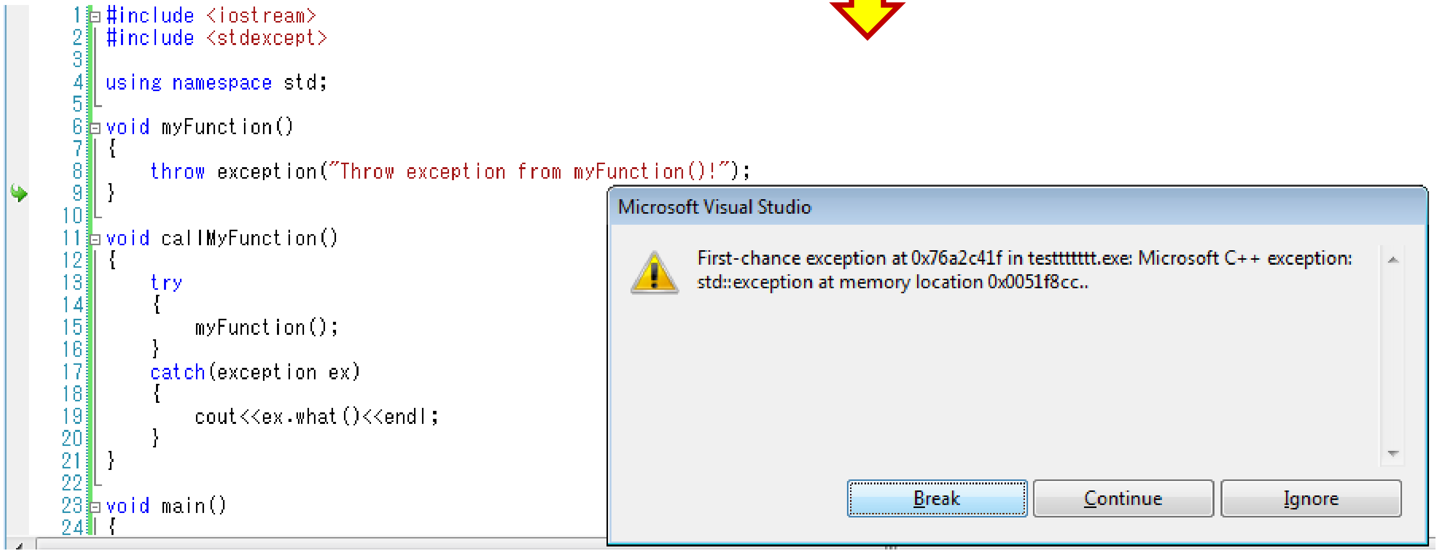
Break on Exception





Break when an exception is:

Name	Thrown	User-unhandled
C++ Exceptions	☐	✓
_com_error	☐	✓
ATL::CAtlException	☐	✓
CException	☐	✓
std::exception	✓	✓
void	☐	✓
Common Language Runtime Exceptions	☐	✓
Managed Debugging Assistants	☐	✓
Native Run-Time Checks	☐	✓
Win32 Exceptions	☐	✓



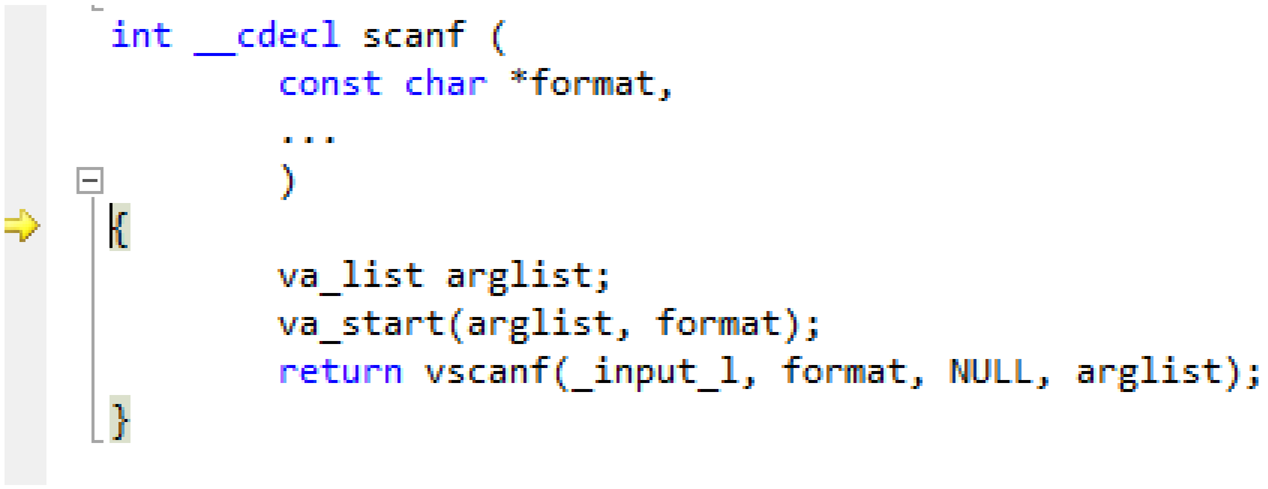
```
#include <iostream>
#include <stdexcept>

using namespace std;

void myFunction()
{
    throw exception("Throw exception from myFunction()!");
}

void callMyFunction()
{
    try
    {
        myFunction();
    }
    catch(exception ex)
    {
        cout<<ex.what()<<endl;
    }
}

void main()
{
```

Microsoft Visual Studio

First-chance exception at 0x76a2c41f in testttttt.exe: Microsoft C++ exception: std::exception at memory location 0x0051f8cc..

Stepping Into Assembly

- Be careful when you "Step Into" lines involving printf, scanf, or other system functions!



```
int __cdecl scanf (
        const char *format,
        ...
        )
{
        va_list arglist;
        va_start(arglist, format);
        return vscanf(_input_l, format, NULL, arglist);
}
```

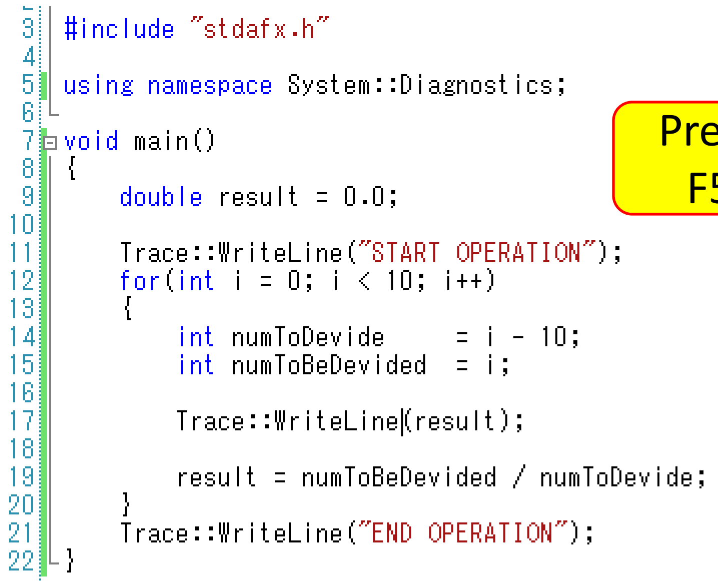
Debug commands

Command	Meaning
Ctrl+F5	Run program
F5	Run in debug mode
F9	Create breakpoint
F10	Step over
F11	Step into
Shift + F11	Step out
Shift + F5	Stop debugging
Ctrl + Tab	Change window

Trace and Assert

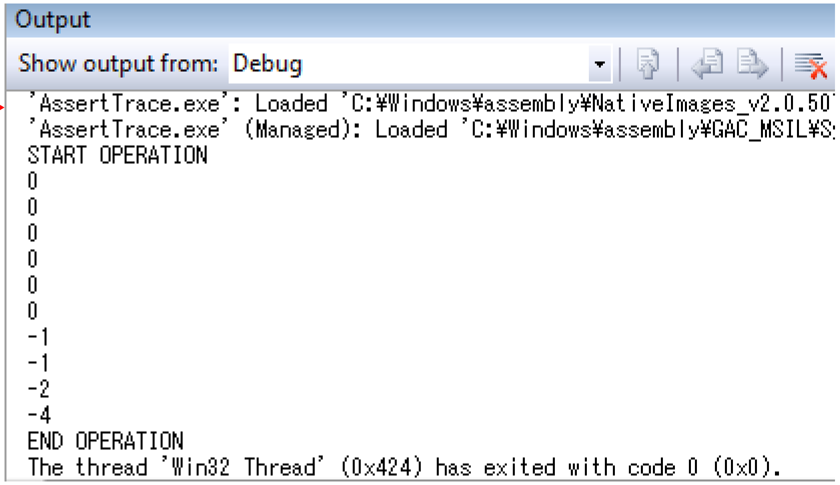
- Trace: Allows the programmer to put a log message onto the main output window
- Assert: To check program assumptions

Trace and Assert



```
#include "stdafx.h"

using namespace System::Diagnostics;

void main()
{
    double result = 0.0;

    Trace::WriteLine("START OPERATION");
    for(int i = 0; i < 10; i++)
    {
        int numToDevide     = i - 10;
        int numToBeDevided  = i;

        Trace::WriteLine(result);

        result = numToBeDevided / numToDevide;
    }
    Trace::WriteLine("END OPERATION");
}
```

Press F5



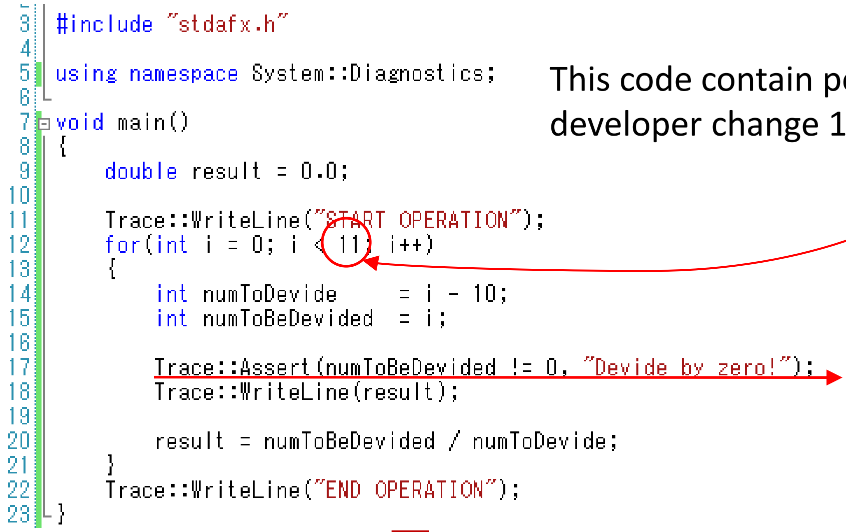
Output

Show output from: Debug

```
'AssertTrace.exe': Loaded 'C:¥Windows¥assembly¥NativeImages_v2.0.50
'AssertTrace.exe' (Managed): Loaded 'C:¥Windows¥assembly¥GAC_MSIL¥S
START OPERATION
0
0
0
0
0
0
-1
-1
-2
-4
END OPERATION
The thread 'Win32 Thread' (0x424) has exited with code 0 (0x0).
```

- Keep tracing code processing by output value during debugging

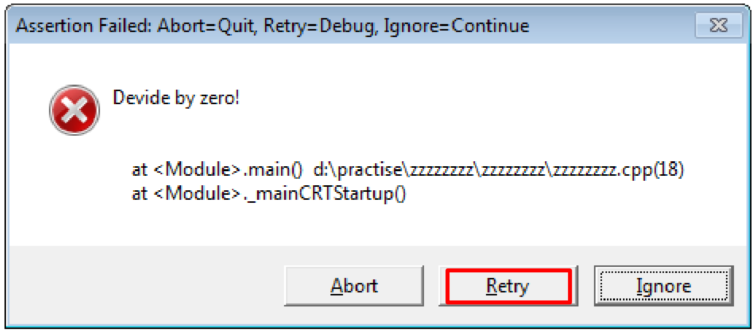
Trace and Assert



```
#include "stdafx.h"

using namespace System::Diagnostics;

void main()
{
    double result = 0.0;

    Trace::WriteLine("START OPERATION");
    for(int i = 0; i < 11; i++)
    {
        int numToDevide     = i - 10;
        int numToBeDevided  = i;

        Trace::Assert(numToBeDevided != 0, "Devide by zero!");
        Trace::WriteLine(result);

        result = numToBeDevided / numToDevide;
    }
    Trace::WriteLine("END OPERATION");
}
```

This code contain potential bug, if another developer change 10 to other values (such as 11)

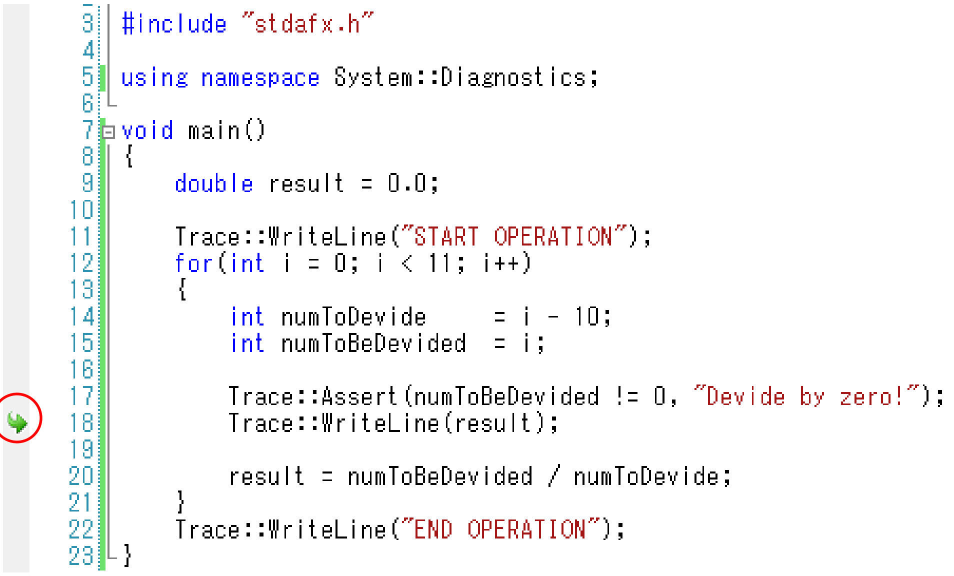
We use Assert to validate that the value is valid or not



Assertion Failed: Abort=Quit, Retry=Debug, Ignore=Continue

Devide by zero!

at <Module>.main() d:\practise\zzzzzzzz\zzzzzzzz\zzzzzzzz.cpp(18)
at <Module>._mainCRTStartup()



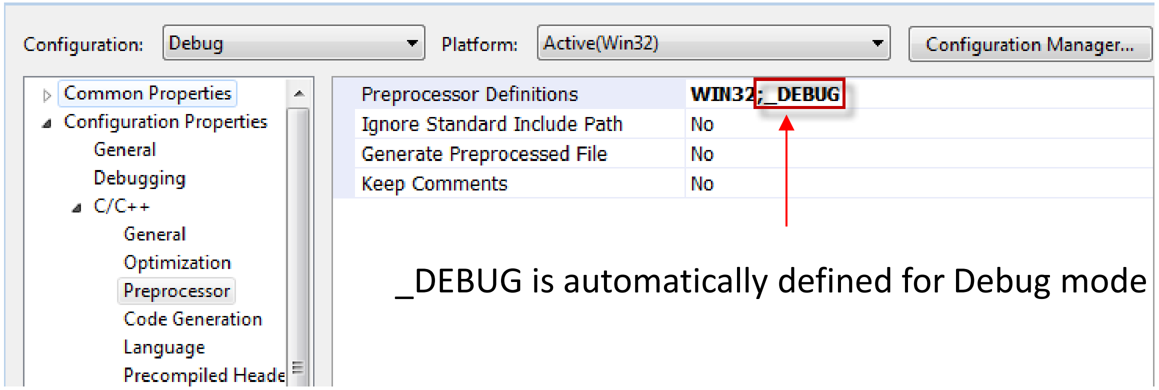
```
#include "stdafx.h"

using namespace System::Diagnostics;

void main()
{
    double result = 0.0;

    Trace::WriteLine("START OPERATION");
    for(int i = 0; i < 11; i++)
    {
        int numToDevide     = i - 10;
        int numToBeDevided  = i;

        Trace::Assert(numToBeDevided != 0, "Devide by zero!");
        Trace::WriteLine(result);

        result = numToBeDevided / numToDevide;
    }
    Trace::WriteLine("END OPERATION");
}
```

Press Retry allow us to debug after Assert

Trace and Assert

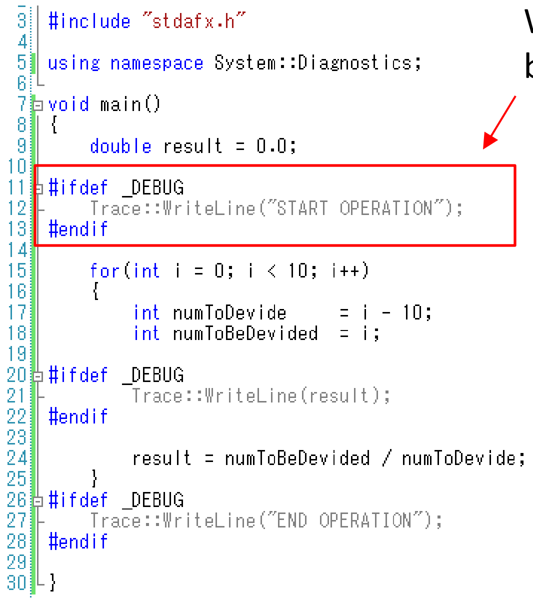
- The behavior for Trace will not change between a debug and a release build
- This mean that we must #ifdef any Trace-related code to prevent debug behavior in a release build

Trace and Assert



Preprocessor Definitions	WIN32;_DEBUG
Ignore Standard Include Path	No
Generate Preprocessed File	No
Keep Comments	No

_DEBUG is automatically defined for Debug mode



```
#include "stdafx.h"

using namespace System::Diagnostics;

void main()
{
    double result = 0.0;

#ifdef _DEBUG
    Trace::WriteLine("START OPERATION");
#endif

    for(int i = 0; i < 10; i++)
    {
        int numToDevide     = i - 10;
        int numToBeDevided  = i;

#ifdef _DEBUG
        Trace::WriteLine(result);
#endif

        result = numToBeDevided / numToDevide;
    }
#ifdef _DEBUG
    Trace::WriteLine("END OPERATION");
#endif

}
```

We can use #ifdef _DEBUG to prevent debug behavior in release mode

Questions and Answers