Advanced Software Techniques

Debugging

Visual Studio

DataTips

Watch

Breakpoints

Debug Memory

Debug Disassembly

Conditional Breakpoints

We know about
breakpoints
from last semester

How about ...

a breakpoint

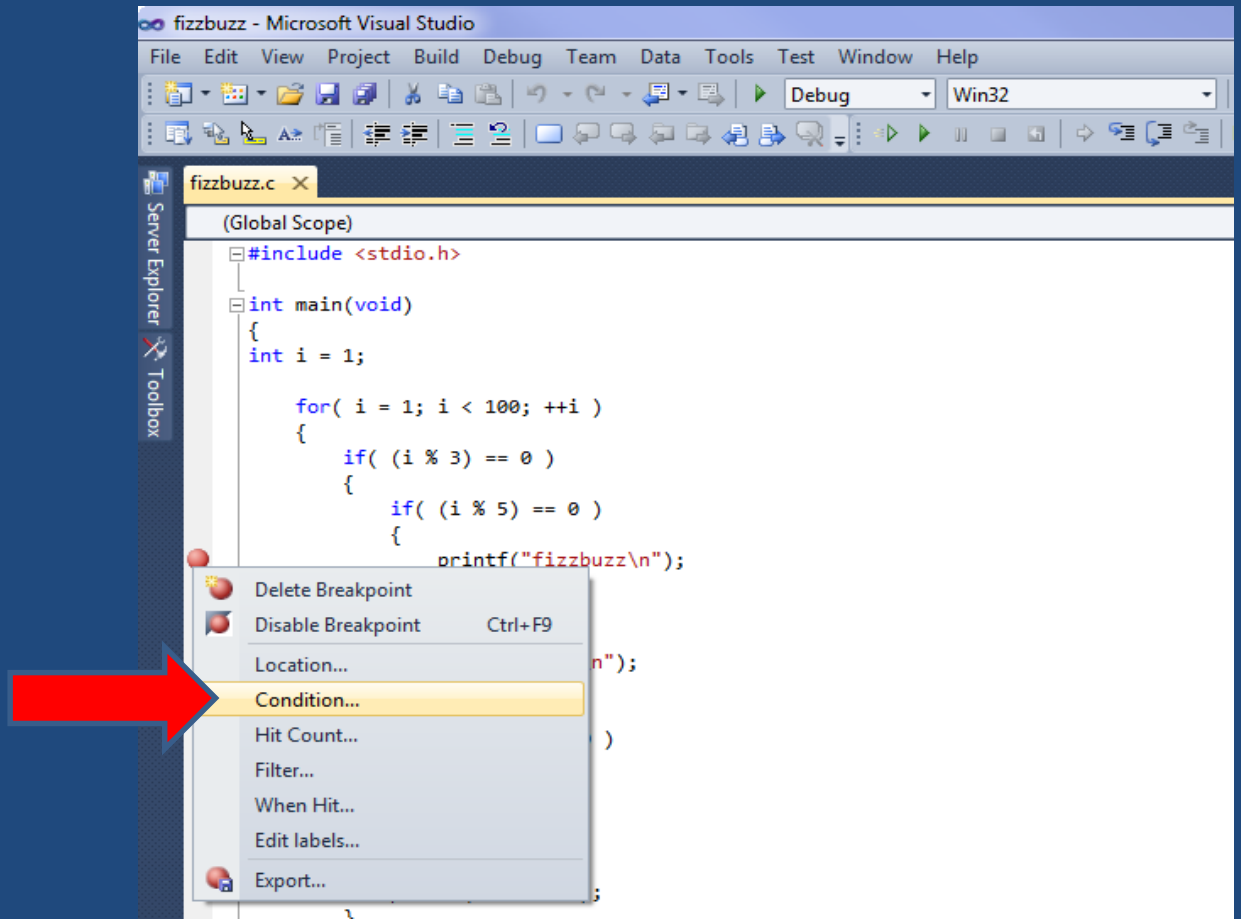
that breaks

when

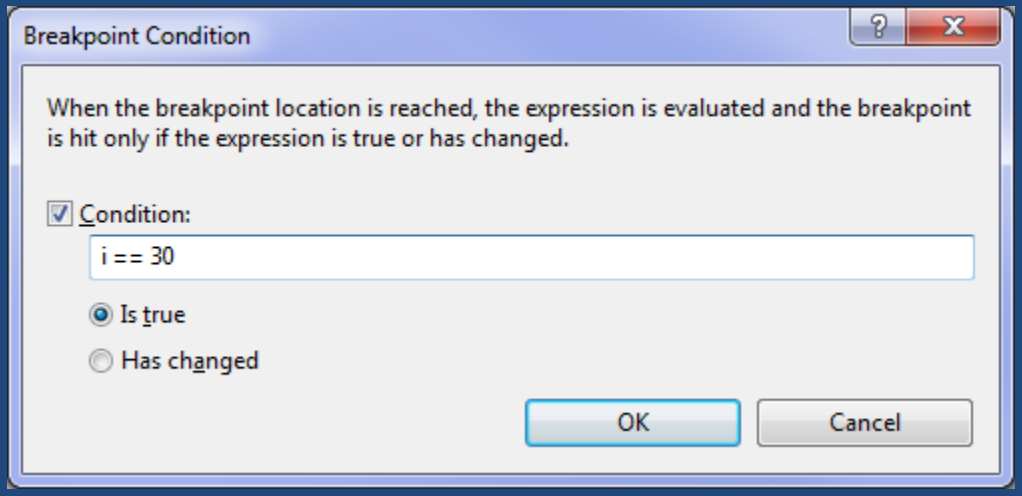
you

want it to?

Right-click on your breakpoint



Choose *Condition*



Enter a condition in the dialog box

The breakpoint
now only
activates when
the condition is true
and not otherwise

Watchpoints

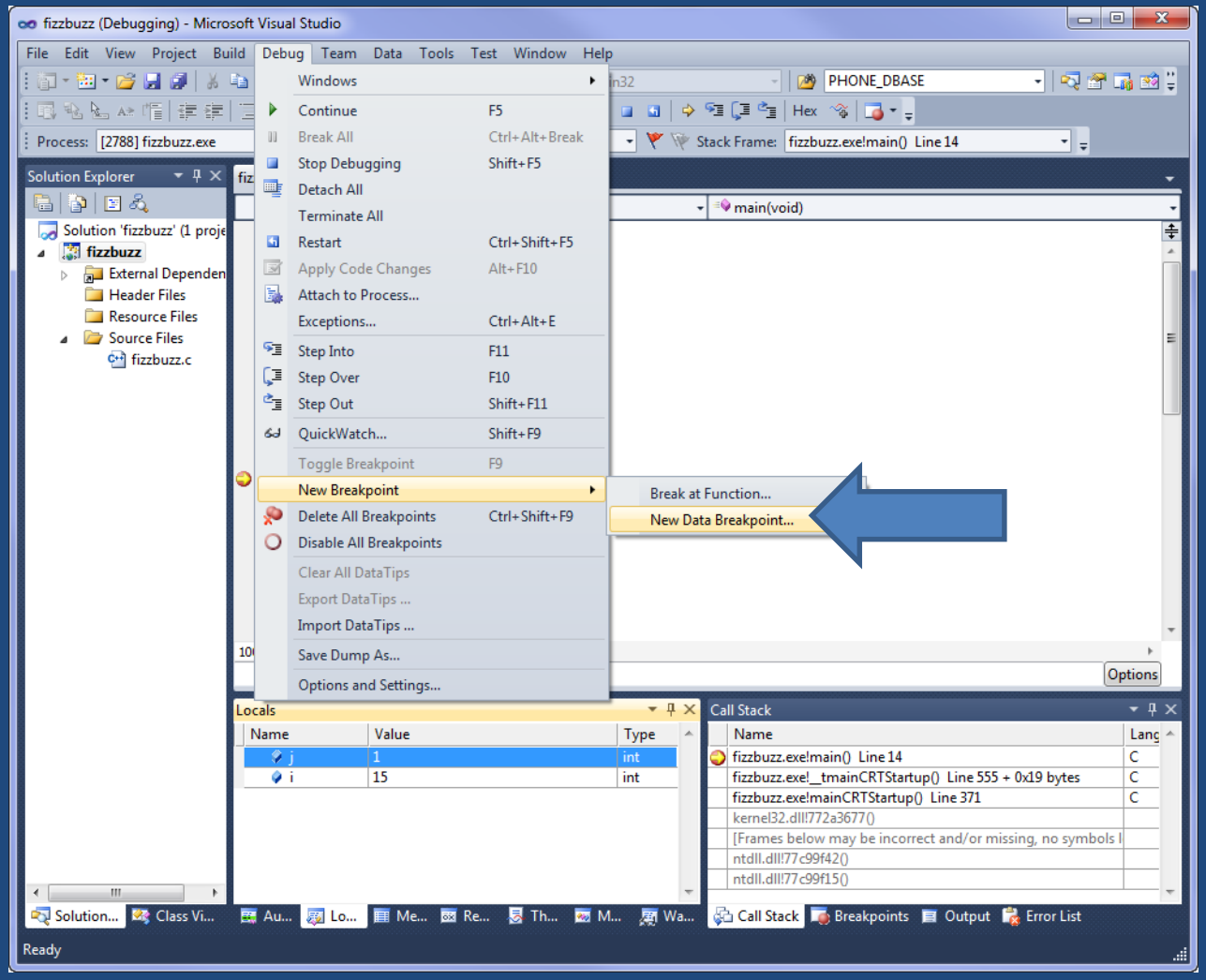
Ever wanted to break
when a variable changes ...

anywhere!?!?

Yes,

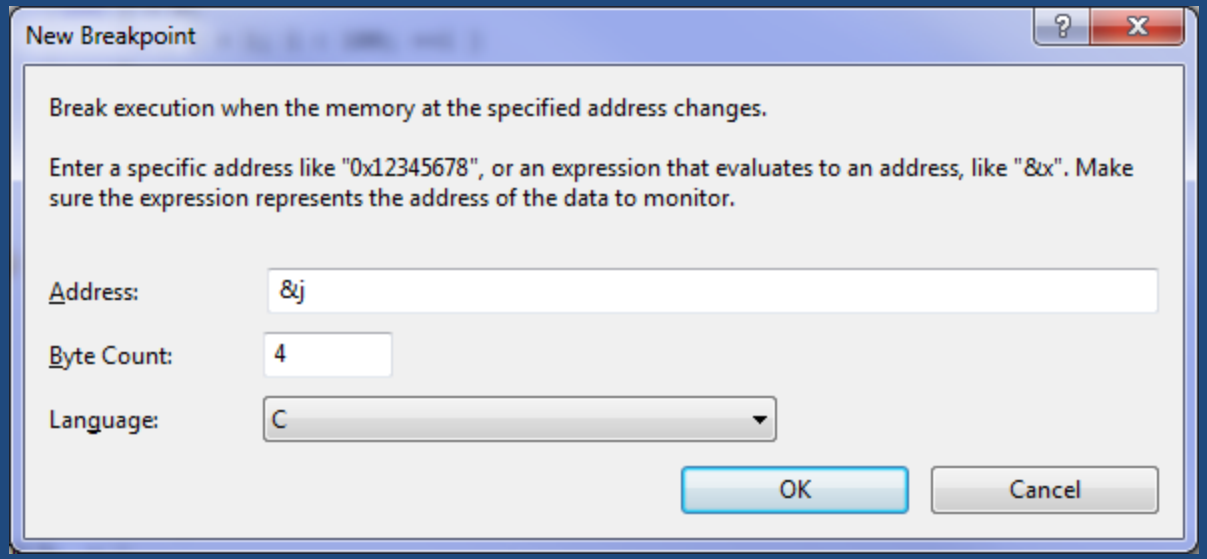
of course you have!

(or you just haven't known it!)


fizzbuzz (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Window Help
Windows
Continue F5
Break All Ctrl+Alt+Break
Stop Debugging Shift+F5
Detach All
Terminate All
Restart Ctrl+Shift+F5
Apply Code Changes Alt+F10
Attach to Process...
Exceptions... Ctrl+Alt+E
Step Into F11
Step Over F10
Step Out Shift+F11
QuickWatch... Shift+F9
Toggle Breakpoint F9
New Breakpoint
Delete All Breakpoints Ctrl+Shift+F9
Disable All Breakpoints
Clear All DataTips
Export DataTips ...
Import DataTips ...
Save Dump As...
Options and Settings...
Break at Function...
New Data Breakpoint...
Process: [2788] fizzbuzz.exe
Stack Frame: fizzbuzz.exe!main() Line 14
Solution Explorer
Solution 'fizzbuzz' (1 proje
fizzbuzz
External Dependen
Header Files
Resource Files
Source Files
fizzbuzz.c
main(void)
Options
Locals
Name Value Type
j 1 int
i 15 int
Call Stack
Name Lang
fizzbuzz.exe!main() Line 14 C
fizzbuzz.exe!__tmainCRTStartup() Line 555 + 0x19 bytes C
fizzbuzz.exe!mainCRTStartup() Line 371 C
kernel32.dll!772a3677()
[Frames below may be incorrect and/or missing, no symbols l
ntdll.dll!77c99f42()
ntdll.dll!77c99f15()
Call Stack Breakpoints Output Error List
Ready

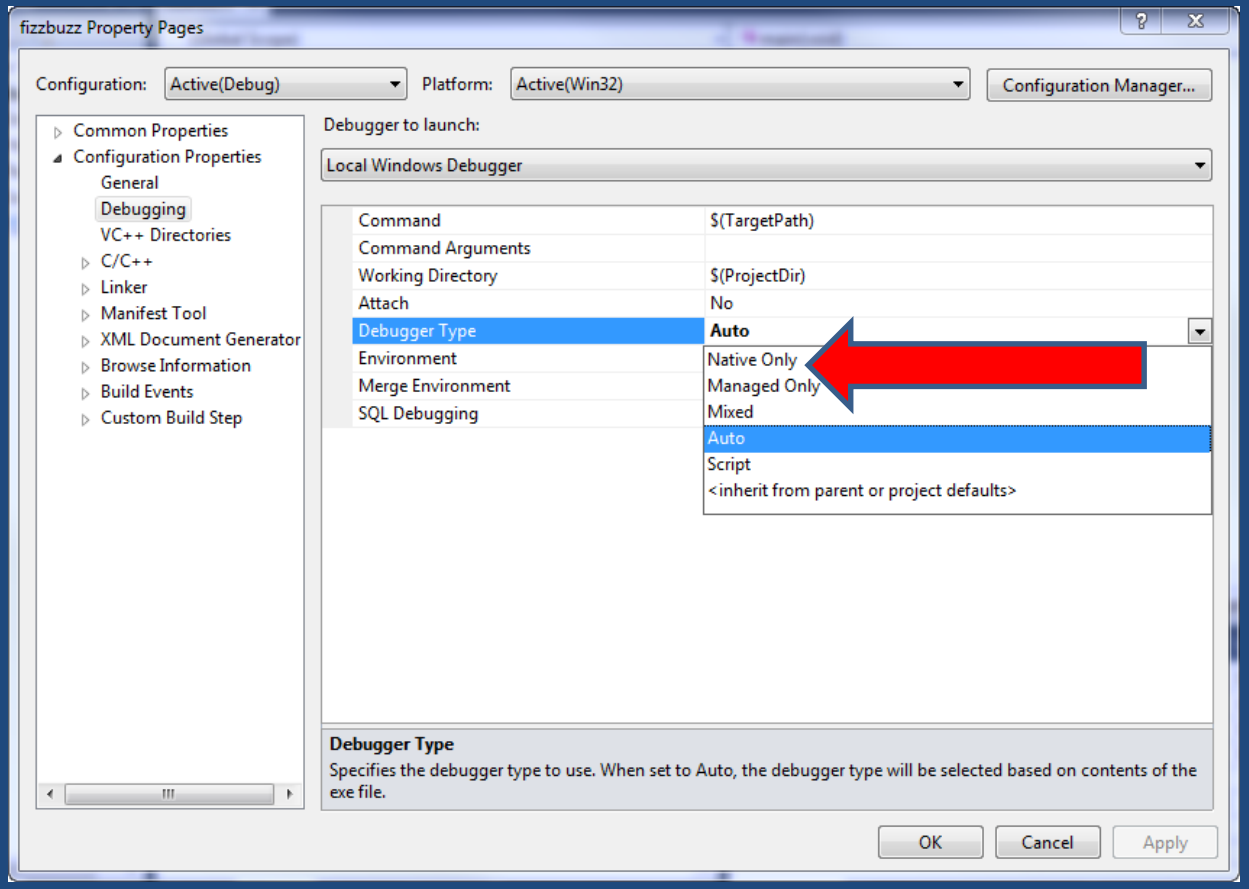
Specify:

1. an address of a variable
2. the size of the variable
3. Language: C


New Breakpoint
Break execution when the memory at the specified address changes.
Enter a specific address like "0x12345678", or an expression that evaluates to an address, like "&x". Make sure the expression represents the address of the data to monitor.
Address:
&j
Byte Count:
4
Language:
C
OK
Cancel

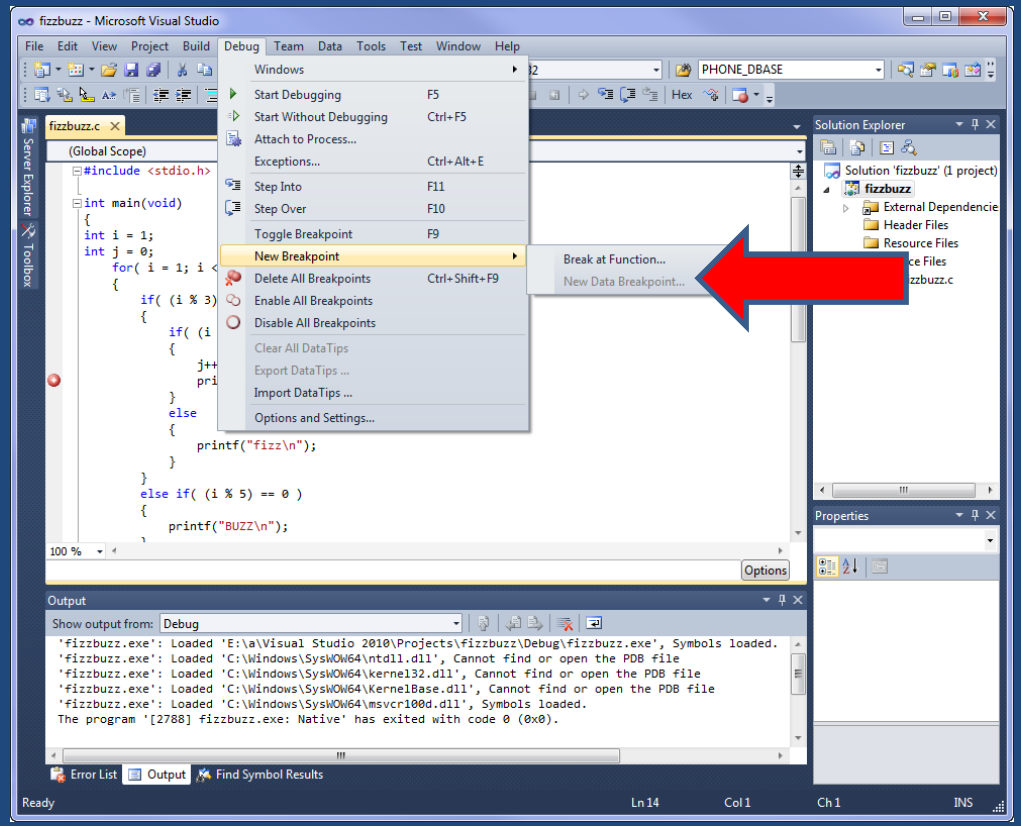
NOTE: Setup and use is **very** finicky!

Finickiness #1:



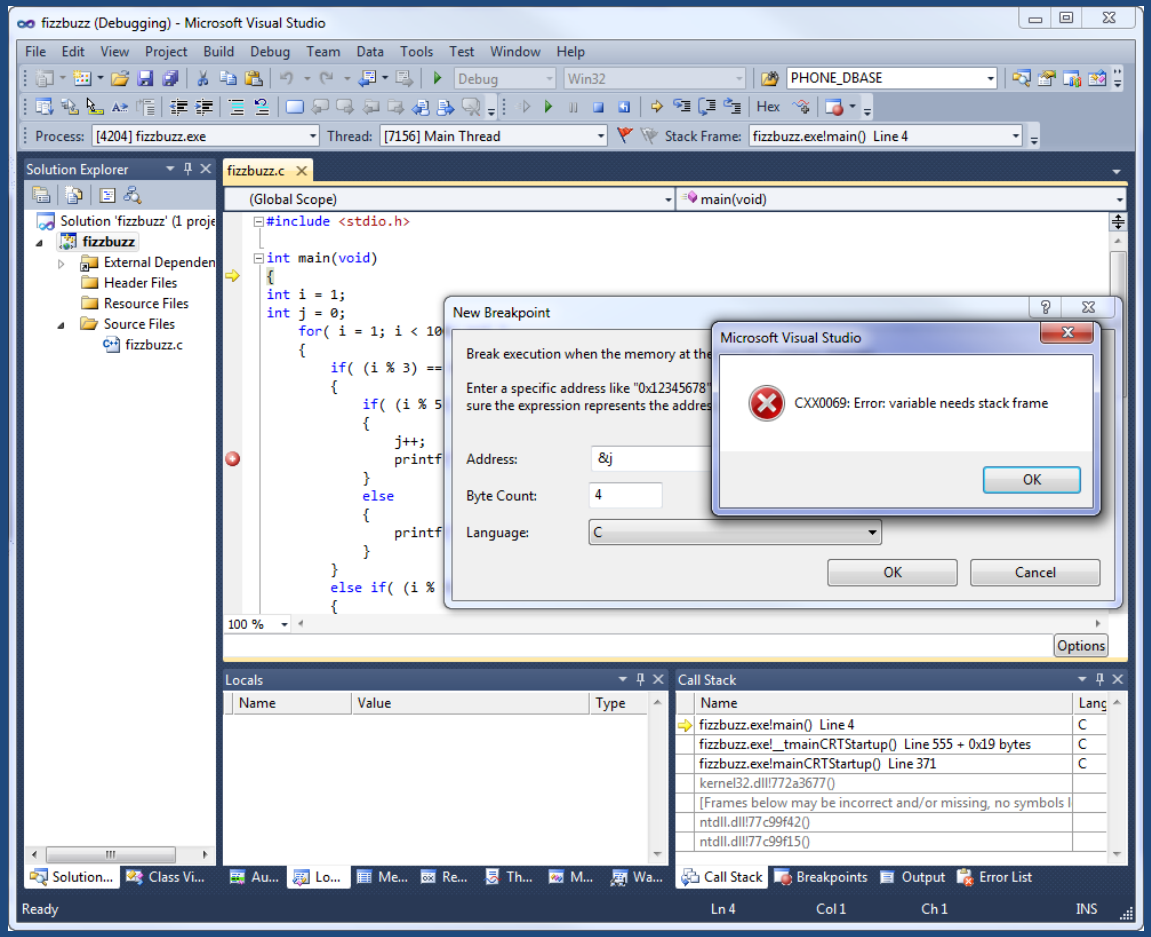
You might have to change the *Debugger Type* to *Native Only* (from *Auto*) in the *Project Properties / Configuration Properties / Debugging*

Finickiness #2:



If you're not currently debugging, *New Data Breakpoint* will be grayed out.

Finickiness #3:



If you're just starting a function, it won't know about the local variables yet.

Finickiness #4:

"Addresses of variables change from one debugging session to the next. For this reason, data breakpoints are **automatically disabled** at the end of each debugging session."

Quotes on next three slides from
http://msdn.microsoft.com/en-us/library/350dyxd0.aspx

Finickiness #5:

"If you set a data breakpoint on a local variable, the data breakpoint remains enabled when the function ends. However, the memory address it is set on no longer has the same meaning. Therefore, the results of such a breakpoint are unpredictable. If you set a data breakpoint on a local variable, the best practice is to remove or disable the breakpoint before the function ends."

Finickiness #6:

"Visual Studio supports a maximum of four data breakpoints per solution."

Not-a-finickiness-but-a-reality
#1:
Watchpoints / Data Breakpoints
often slow down
code execution

Portable Debugging Techniques

Conditional Compilation

Conditional Compilation

Code added to help
with debugging
can be
incredibly useful!

It's too easy

just to add

and add

and add

and add

That clutters the code

Then it
comes time
to hand it in


ARRGH!

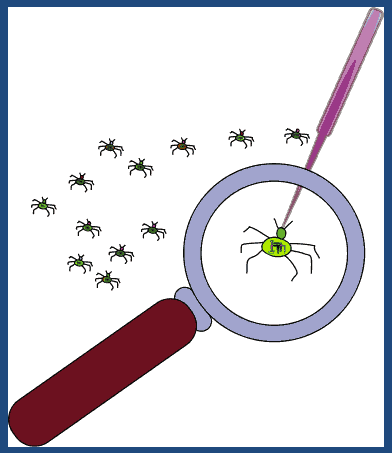
The solution?

Conditional compilation!

Surround the
debugging code
with
#ifdef
and
#endif

```
#ifdef DEBUGGING_CODE
   printf("Reached body of if\n");
#endif
```

You can do that around all of your debugging code



Then, when you need
to debug,
do

#define DEBUGGING_CODE

It doesn't have
to have a value

There's nothing special about DEBUGGING_CODE either

When you want to
not compile the code,
simply delete the
#define
(or comment it out)

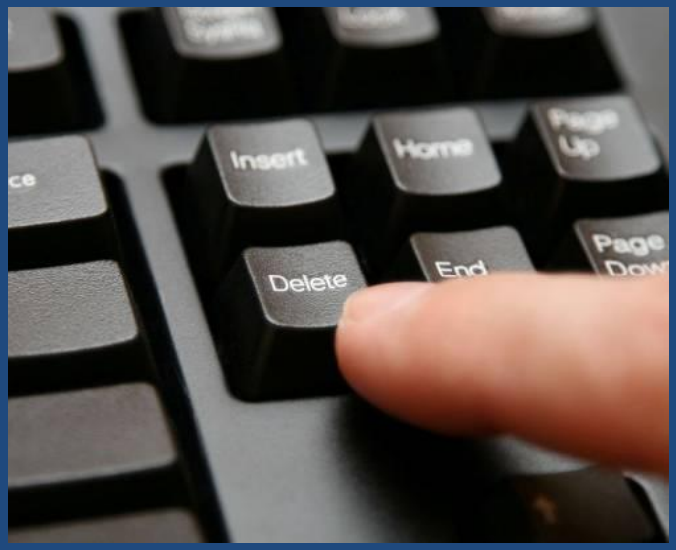
You can go further,
using
#if

```
#if (DEBUGLEVEL >= 2)
printf("Reached body of if\n");
#endif
```

#define DEBUGLEVEL 4

When you're sure
you won't need the
debugging statements
anymore
(like when you're handing in your
assignment)

just search for them and delete them



Library Specific Techniques

ASSERT

Why use assert?

There are
some things
that you just
don't want to happen

e.g.

- dereferencing a NULL pointer,
- having an out-of-range array index,
- having an invalid parameter for a function call

... or anything that
might produce a
bug that should
kill your program
immediately

assert can be used
to help during
development
to track down bugs

How to use assert

```
assert(ptr != NULL);
```

or

```
assert((index >= 0) && (index
<= kMaxIndex));
```

It looks like a function call but it's not

It's a macro (like #define with parameters)

Put the condition that you want to be true in the brackets

Example

assert(ptr != NULL);

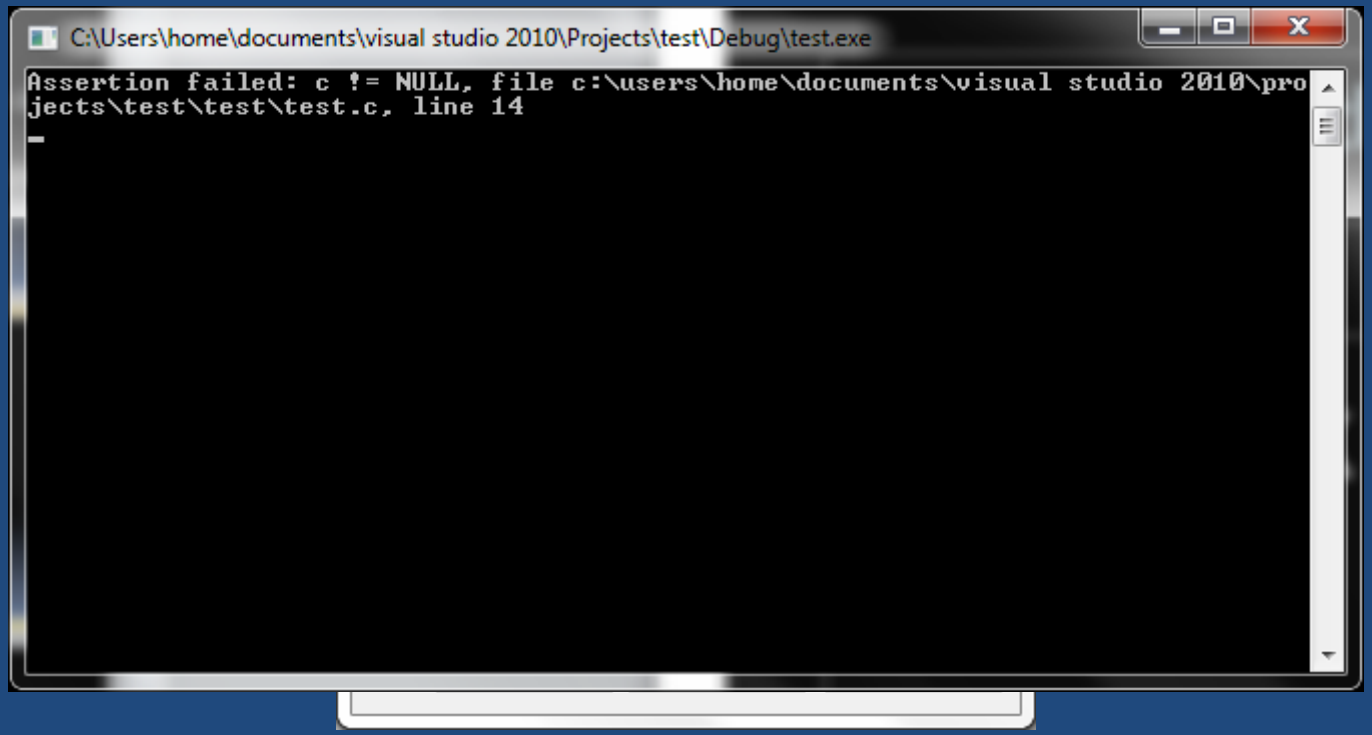
means

“Confirm that ptr is NOT NULL”

or

"If ptr is NULL, there's a bug in the program."

If the condition is FALSE,
the program crashes with an
assertion message


C:\Users\home\documents\visual studio 2010\Projects\test\Debug\test.exe
Assertion failed: c != NULL, file c:\users\home\documents\visual studio 2010\projects\test\test\test.c, line 14

The console tells you:

- the assert failed
- what condition failed
- what file
- what line of code

When to assert

This is handy for watching dangerous code areas during development

assert()

should NOT be used

for things the program should be able to handle

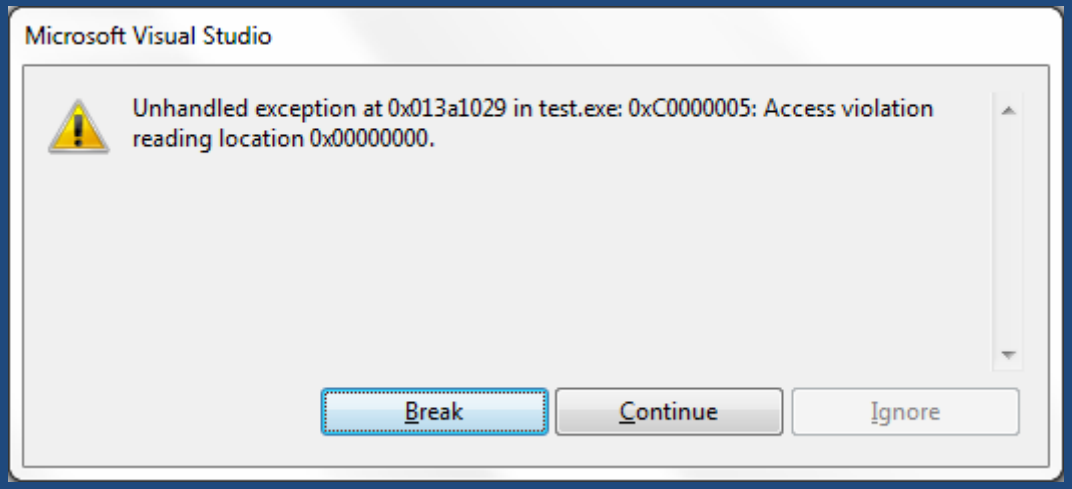
e.g. Out of Memory

e.g. variable out of range due to a measurement or user input

Why?

assert() is typically disabled in released code

So any protections
you add using assert()
won't be there
in your shipped code


Microsoft Visual Studio
Unhandled exception at 0x013a1029 in test.exe: 0xC0000005: Access violation reading location 0x00000000.
Break
Continue
Ignore


OOPS!!

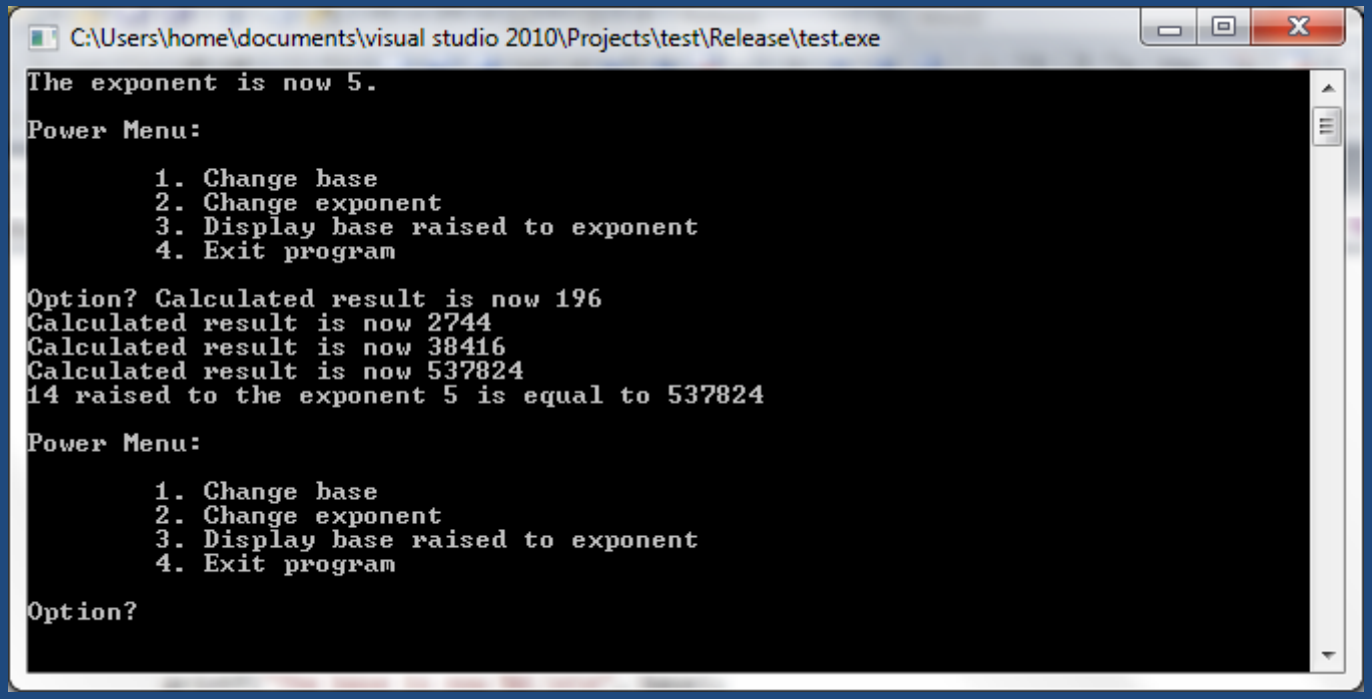
But it still is useful during development

And, you also need ...

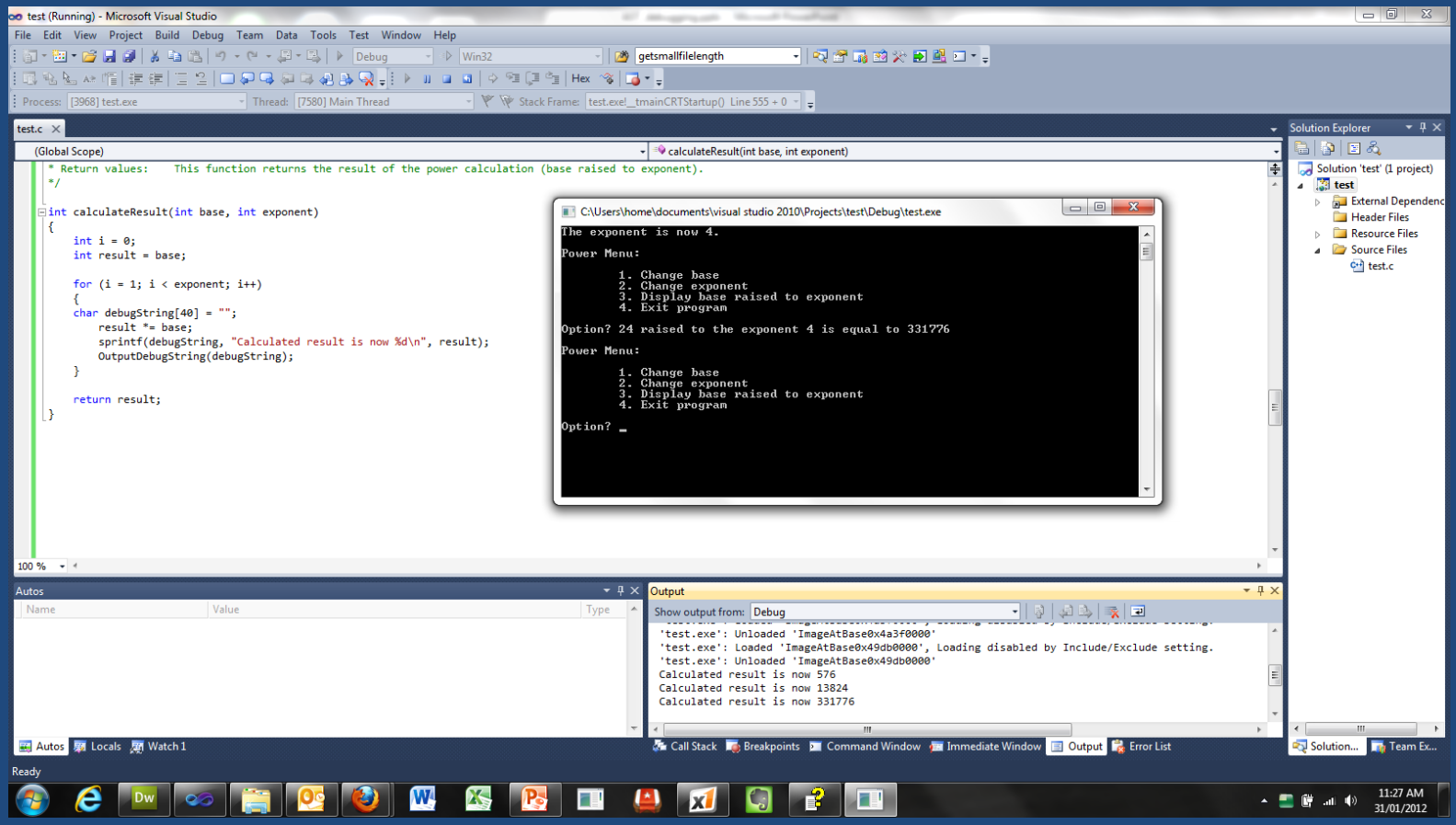
```
#include <assert.h>
```

Outputdebugstring()

It's a pain
to have debugging output
get in the way of your
normal output

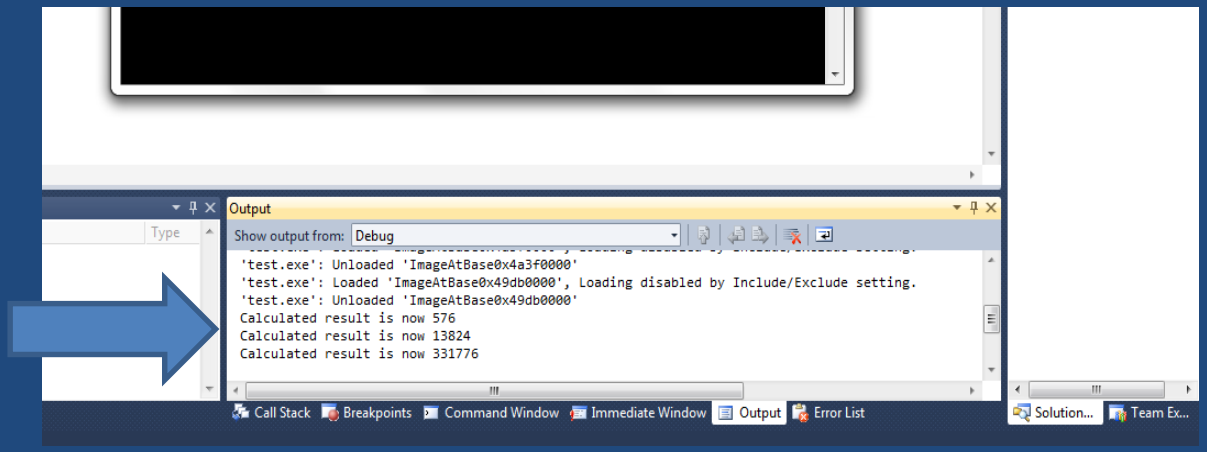

C:\Users\home\documents\visual studio 2010\Projects\test\Release\test.exe
The exponent is now 5.
Power Menu:
1. Change base
2. Change exponent
3. Display base raised to exponent
4. Exit program
Option? Calculated result is now 196
Calculated result is now 2744
Calculated result is now 38416
Calculated result is now 537824
14 raised to the exponent 5 is equal to 537824
Power Menu:
1. Change base
2. Change exponent
3. Display base raised to exponent
4. Exit program
Option?

Windows provides a better option


test (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Window Help
test.c
(Global Scope)
calculateResult(int base, int exponent)
* Return values: This function returns the result of the power calculation (base raised to exponent).
*/
int calculateResult(int base, int exponent)
{
int i = 0;
int result = base;
for (i = 1; i < exponent; i++)
{
char debugString[40] = "";
result *= base;
sprintf(debugString, "Calculated result is now %d\n", result);
OutputDebugString(debugString);
}
return result;
}
C:\Users\home\documents\visual studio 2010\Projects\test\Debug\test.exe
The exponent is now 4.
Power Menu:
1. Change base
2. Change exponent
3. Display base raised to exponent
4. Exit program
Option? 24 raised to the exponent 4 is equal to 331776
Power Menu:
1. Change base
2. Change exponent
3. Display base raised to exponent
4. Exit program
Option?
Solution Explorer
Solution 'test' (1 project)
test
External Dependenc
Header Files
Resource Files
Source Files
test.c
Autos
Name
Value
Type
Output
Show output from: Debug
'test.exe': Unloaded 'ImageAtBase0x4a3f0000'
'test.exe': Loaded 'ImageAtBase0x49db0000', Loading disabled by Include/Exclude setting.
'test.exe': Unloaded 'ImageAtBase0x49db0000'
Calculated result is now 576
Calculated result is now 13824
Calculated result is now 331776
Autos
Locals
Watch 1
Call Stack
Breakpoints
Command Window
Immediate Window
Output
Error List
Ready
11:27 AM
31/01/2012

OutputDebugString()
is a Windows function.

It puts strings
to the Output window
in Visual Studio.


Output
Show output from: Debug
'test.exe': Unloaded 'ImageAtBase0x4a3f0000'
'test.exe': Loaded 'ImageAtBase0x49db0000', Loading disabled by Include/Exclude setting.
'test.exe': Unloaded 'ImageAtBase0x49db0000'
Calculated result is now 576
Calculated result is now 13824
Calculated result is now 331776
Call Stack
Breakpoints
Command Window
Immediate Window
Output
Error List
Solution...
Team Ex...

You can do your own pre-formatting using sprintf()

```
char buffer[100]= "";
sprintf(buffer,
            "Calculated result is now %d\n",
            result);
OutputDebugString(buffer);
```

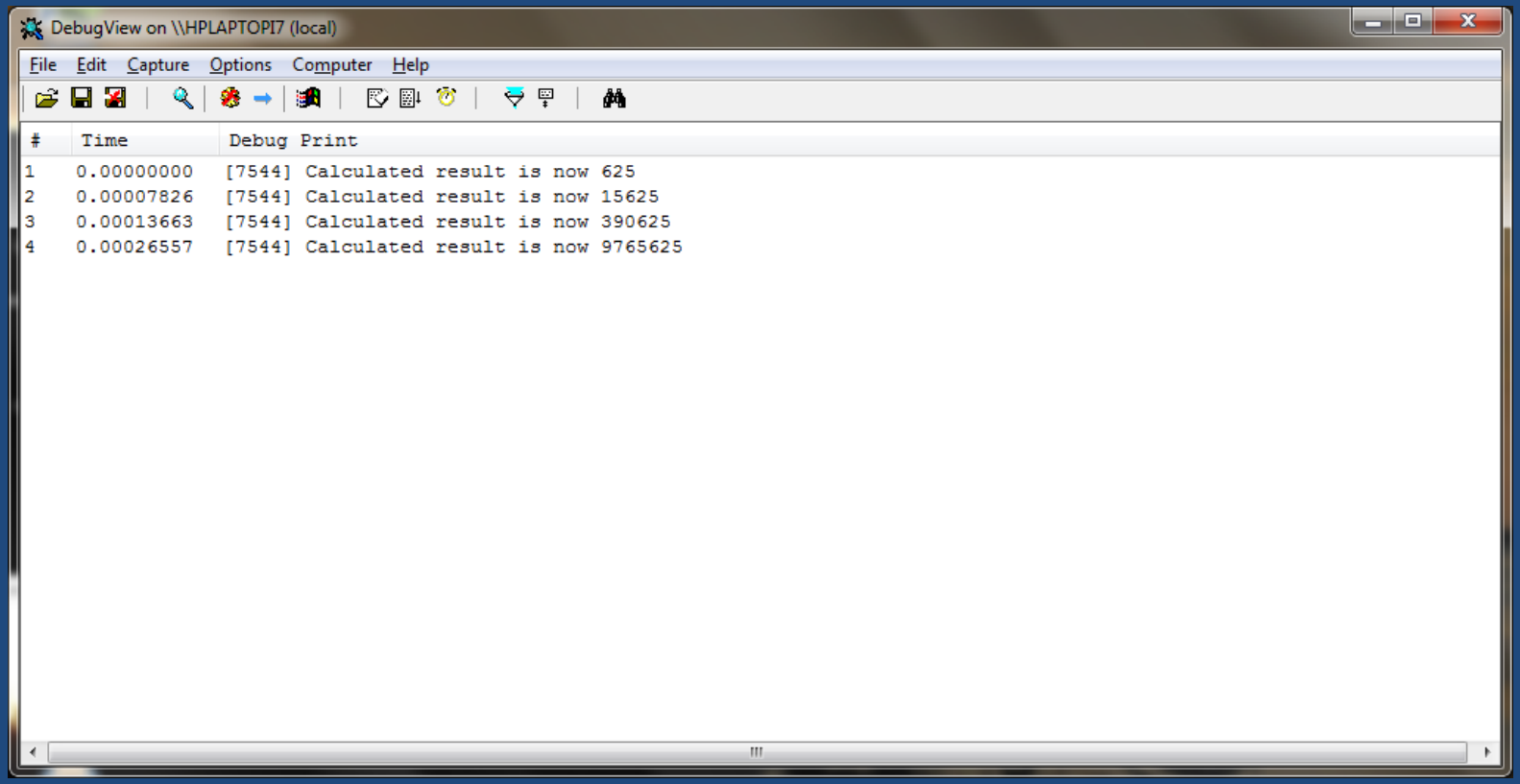
Don't forget the prototype
(found in:

#include <windows.h>

Even better,

if you're not running in the debugger,

you can still see the debugging information


DebugView on \\HPLAPTOPI7 (local)
File Edit Capture Options Computer Help
Time Debug Print
1 0.00000000 [7544] Calculated result is now 625
2 0.00007826 [7544] Calculated result is now 15625
3 0.00013663 [7544] Calculated result is now 390625
4 0.00026557 [7544] Calculated result is now 9765625

DebugView is a downloadable utility that will show you the messages

if you're not in

Visual Studio

http://technet.microsoft.com/en-us/sysinternals/bb896647

On Linux,
there's a similar functionality
built in, called syslog

There's more info at:

http://linux.die.net/man/3/syslog