UNIVERSITY OF NEWCASTLE

Writing C++ in Visual Studio

Proverb:
Work smarter not harder

Summary

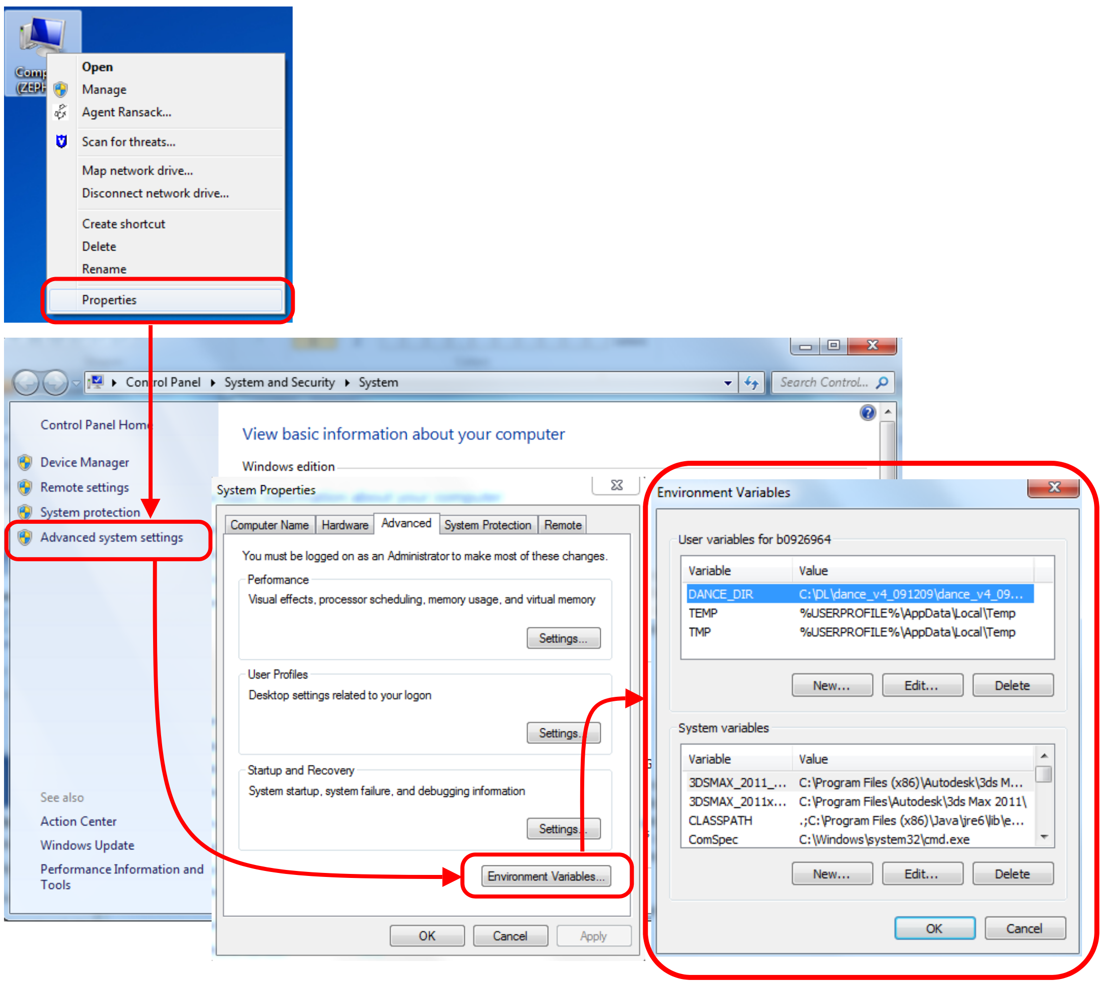
How to take advantage of Visual Studio's features.

Tutorial Overview

Visual Studio is more than a glorified syntax colour coder. It contains thousands of hidden tricks and abilities to make your life easy. Having a better understanding of your tools can help you produce more robust, cleaner code faster. Then when bugs and problems do arise for some reason, you should be able to go straight to the problem and rectify it without breaking a sweat.

Environment Variables

Setting environment variables for Visual Studio so paths and defines can be used in Visual Studio.



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

User variables for b0926964

Variable	Value
DANCE_DIR	C:\DL\dance_v4_091209\dance_v4_09...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
3DSMAX_2011_...	C:\Program Files (x86)\Autodesk\3ds M...
3DSMAX_2011x...	C:\Program Files\Autodesk\3ds Max 2011\
CLASSPATH	.;C:\Program Files (x86)\Java\jre6\lib\e...
ComSpec	C:\Windows\system32\cmd.exe

You can add a 'USER' define, so you can write user specific code for shared projects. In the environment variables add the variable "USER", and set it to something specific, e.g. "MYCODE".

In Visual Studio, add to the pre-processor for the project – "$(USER)".

Now you should be able to use your define in code.

```
#ifdef MYCODE
printf("Code only to compile/run for me \n");
#endif2
```

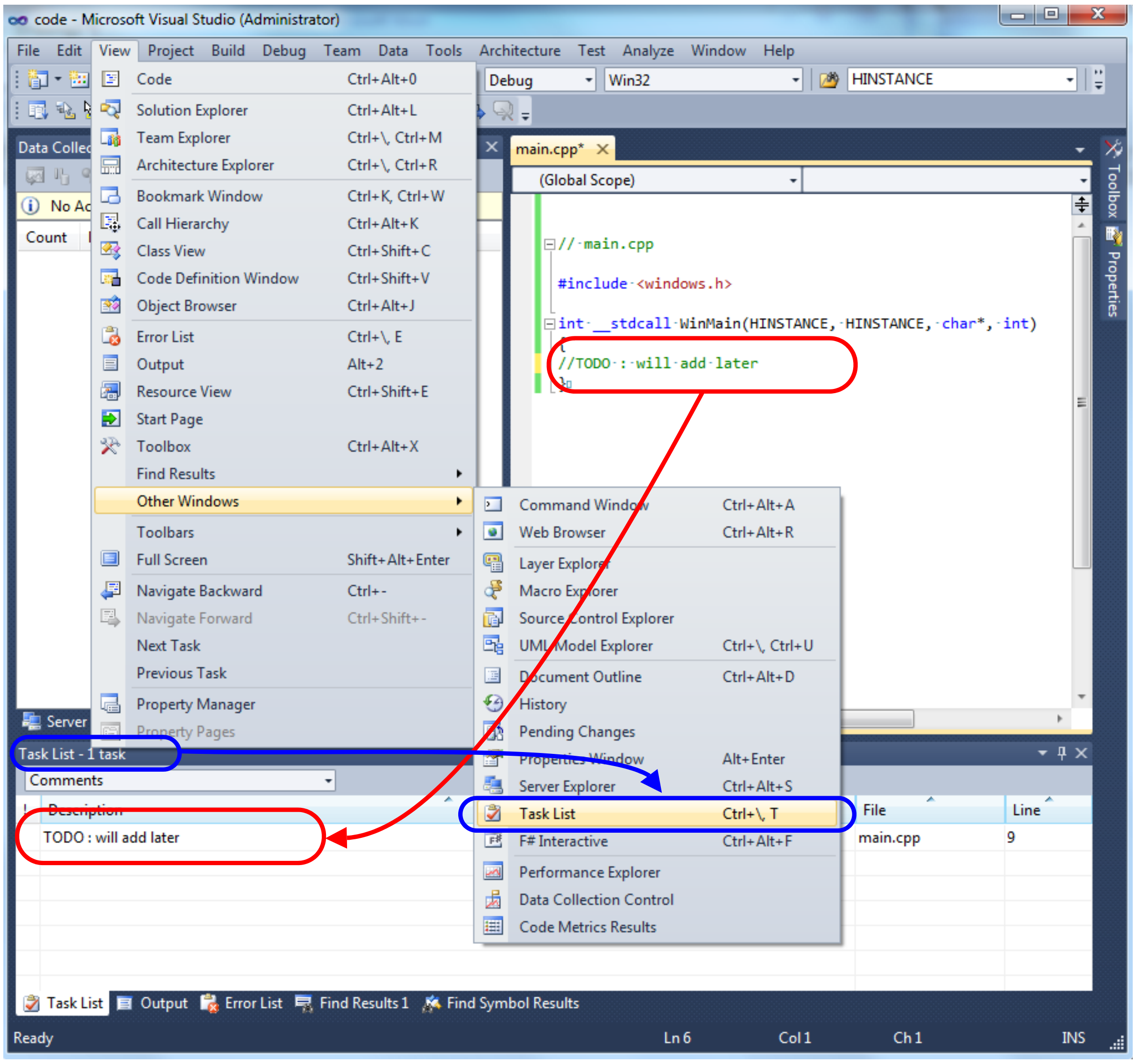
Code Messages and Reminders

At various points during code development you can write comments in your code to remind you to come back to them at a later date. Visual Studio can help you keep track of these by automatically adding them to a list. Visual Studio has a Task-List window, and will automatically add comments starting with "**//TODO**" to the list, so you know what still needs doing, or what others have left unfinished. Note that it's case sensitive, so "//todo" won't work, also no spaces between the double slashes and the word TODO.

In Visual Studio 2010 you have to enable this in:
Tools -> Options -> Formatting -> Enumerate comment tasks (Set to true to enable)

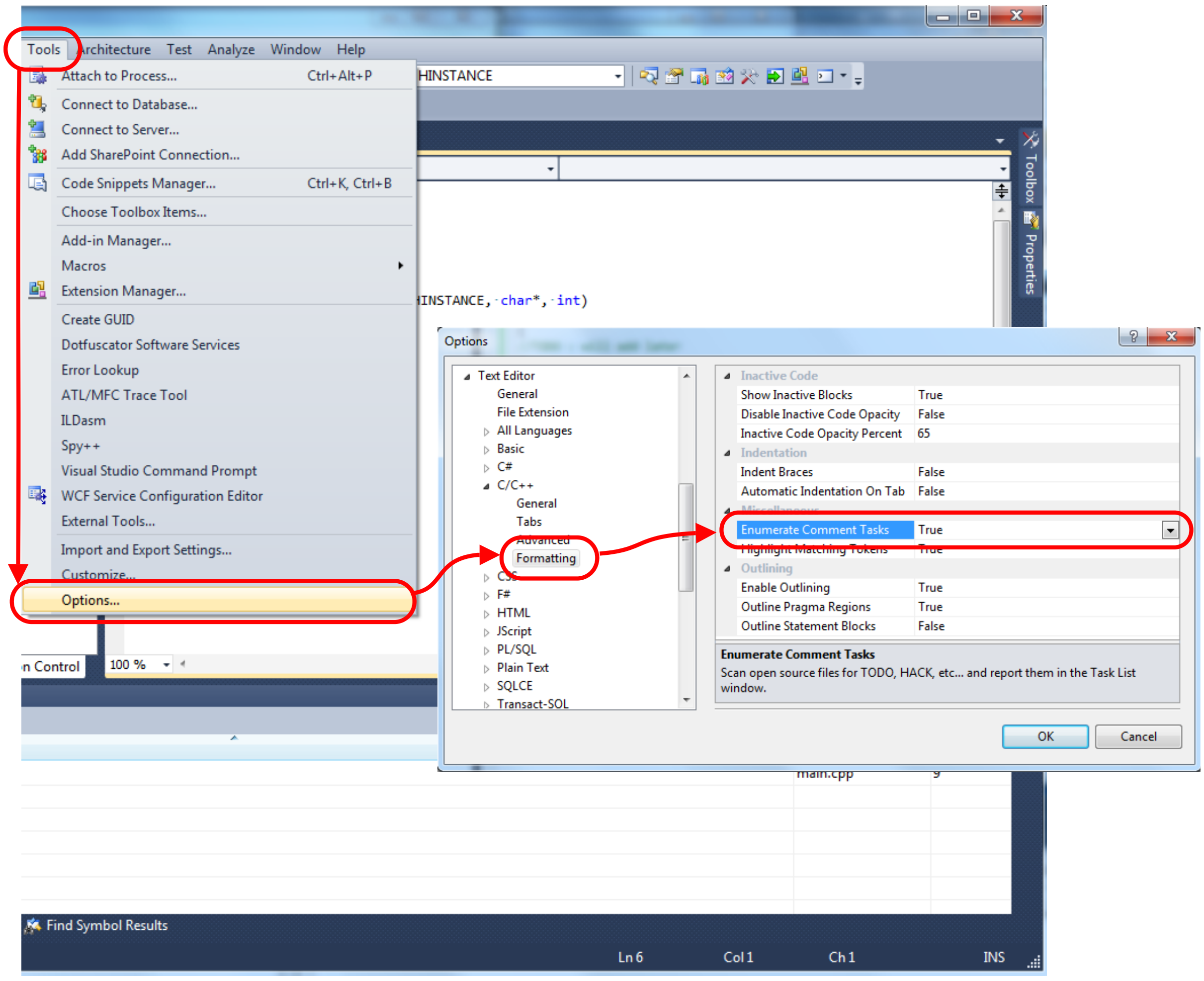
To view the task list window:
View -> Other Windows -> Task List

```
//TODO : Add extra check code later
```


code - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug Win32 HINSTANCE
Code Ctrl+Alt+0
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Architecture Explorer Ctrl+\, Ctrl+R
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+Shift+V
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Alt+2
Resource View Ctrl+Shift+E
Start Page
Toolbox Ctrl+Alt+X
Find Results
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Next Task
Previous Task
Property Manager
Property Pages
Command Window Ctrl+Alt+A
Web Browser Ctrl+Alt+R
Layer Explorer
Macro Explorer
Source Control Explorer
UML Model Explorer Ctrl+\, Ctrl+U
Document Outline Ctrl+Alt+D
History
Pending Changes
Properties Window Alt+Enter
Server Explorer Ctrl+Alt+S
Task List Ctrl+\, T
F# Interactive Ctrl+Alt+F
Performance Explorer
Data Collection Control
Code Metrics Results
main.cpp*
(Global Scope)
Toolbox
Properties
Data Collec
No Ac
Count
Server
Task List - 1 task
Comments
Description
File
Line
Task List Output Error List Find Results 1 Find Symbol Results
Ready Ln 6 Col 1 Ch 1 INS

```
// main.cpp

#include <windows.h>

int __stdcall WinMain(HINSTANCE, HINSTANCE, char*, int)
{
//TODO : will add later
}
```

Description	File	Line
TODO : will add later	main.cpp	9



Inactive Code	
Show Inactive Blocks	True
Disable Inactive Code Opacity	False
Inactive Code Opacity Percent	65
Indentation	
Indent Braces	False
Automatic Indentation On Tab	False
Miscellaneous	
Enumerate Comment Tasks	True
Highlight Matching Tokens	True
Outlining	
Enable Outlining	True
Outline Pragma Regions	True
Outline Statement Blocks	False

Enumerate Comment Tasks
Scan open source files for TODO, HACK, etc... and report them in the Task List window.

You can also send messages to your output window during compile to give additional information by taking advantage of the #pragma define, e.g:

```
#pragma message( “This is a reminder...hello...” )
```

This can give additional information, e.g. the order files are compiled, or maybe just updates on which compile flags should be set etc, by placing the #pragma message inside different #defines you can give different messages for release and debug builds.

You can also insert custom compile errors, for example if you try and build the code for different platforms (PS3/PC/Win32) you can spit out compile error informing that this code isn’t supported, or needs certain fixes.

```
#ifdef MYFASTMODE
#error("Error : This code won't work in FastMode" )
#endif3
```

Temp Files Cleanup

Visual Studio generates numerous temporary files in various directories for debugging and during the build process. At various points you may want to a cleanup and remove all temp and compiled files before giving the code to someone else, or zipping them up.

A good method is to create a '.bat' script, which when run recursively deletes any temporary files in that directory. BE WARNED, bat files can be dangerous, deleted files cannot be recovered so be careful where you put the script and which files you delete.

```
REM Remove Temp Visual Studio Files
REM ******************************

del     /s      *.aps
del     /s      *.clw
del     /s      *.ncb
del     /s      *.opt
del     /s      *.plg
del     /s      *.pdb
del     /s      *.vsp
del     /s      *.sdf
del     /s      /A -H *.suo


del     /Q Debug\\*.*
rmdir   /Q Debug
del     /Q Release\\*.*
rmdir   /Q Release

FOR /F "tokens=*" %%G IN ('DIR /B /AD /S *ipch') DO RMDIR /S /Q "%%G"

FOR /F "tokens=*" %%G IN ('DIR /B /AD /S *obj') DO RMDIR /S /Q "%%G"

FOR /F "tokens=*" %%G IN ('DIR /B /AD /S *bin') DO RMDIR /S /Q "%%G"

```

Code 1. Cleanup.bat to remove temporary Visual Studio files.

Libs and Includes

If you've got multiple projects or your including libs from external sources (e.g. DirectX/OpenGL/Bullet), then you'll need to tell your Visual Studio project which libs you want to include and where their located.

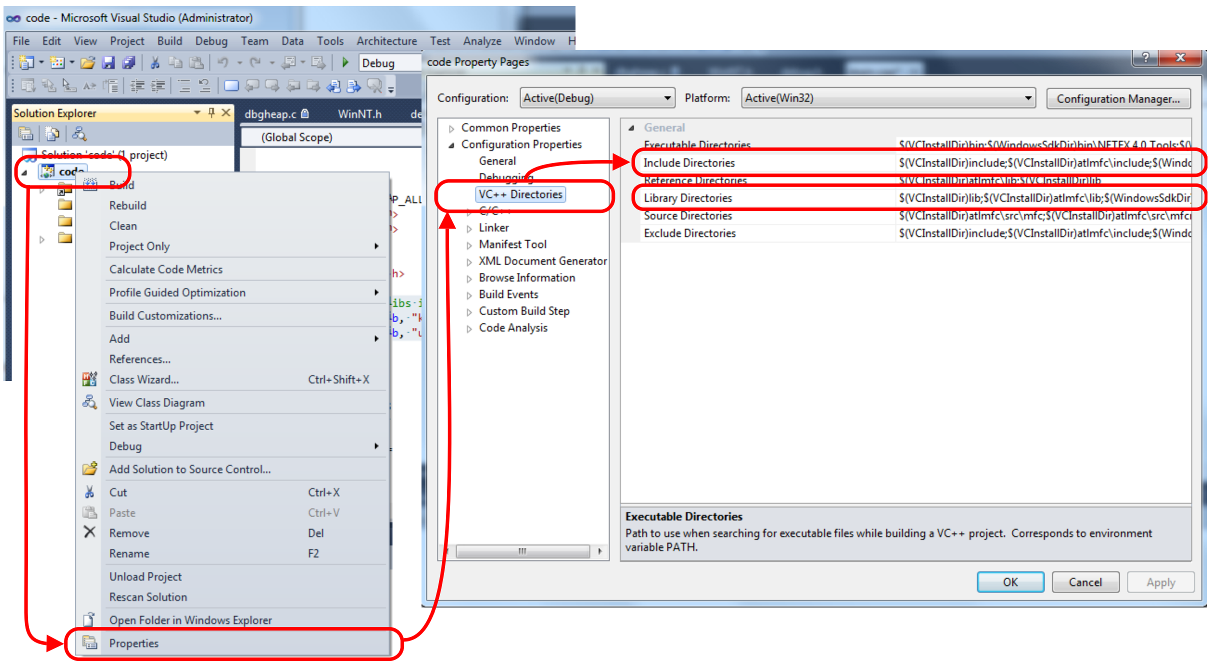
You can reference your libs in code by adding #pragma defines which will inform Visual Studio to include those libs.

```
// Include these libs in our build
#pragma comment(lib, "kernel32.lib")
#pragma comment(lib, "user32.lib")
```

Alternatively you can add them directly to the project through the IDE:
Right click the project in solution explorer -> Properties ->Linker -> Input -> Additional Dependencies, and then add your libs directly.

As well as specifying which libs you want to include, you also have to tell the Visual Studio project where their located. To add include directories and lib paths to your project:

Right click project in the Solution Explorer -> Properties -> VC++ Directories -> Include Directories / Library Directories.



Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

General

Executable Directories	$(VCInstallDir)bin;$(WindowsSdkDir)bin\NETFX 4.0 Tools;$(
Include Directories	$(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(Windo
Reference Directories	$(VCInstallDir)atlmfc\lib;$(VCInstallDir)lib
Library Directories	$(VCInstallDir)lib;$(VCInstallDir)atlmfc\lib;$(WindowsSdkDir)
Source Directories	$(VCInstallDir)atlmfc\src\mfc;$(VCInstallDir)atlmfc\src\mfc
Exclude Directories	$(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(Windo

Executable Directories
Path to use when searching for executable files while building a VC++ project. Corresponds to environment variable PATH.

You can set all the paths and directories in code, by placing the relative and absolute paths at the start, or set them inside Visual Studio's settings:

```
#define DX_LIB_PATH "C:\\Program Files (x86)\\Microsoft DirectX SDK (June 2010)\\Lib\\x86\\"
#pragma comment(lib,DX_LIB_PATH "d3d10.lib")
#pragma comment(lib,DX_LIB_PATH "d3dx10.lib")
#include <C:\\Program Files (x86)\\Microsoft DirectX SDK (June 2010)\\Include\\d3dx10.h>

// OR - If you've added the paths to visual studio

#pragma comment(lib,"d3d10.lib")
#pragma comment(lib,"d3dx10.lib")
#include <d3dx10.h>
```

Breakpoints in code

Having your code halt at lines when it has performed an invalid action is very useful. When I say halt, I mean stop in Visual Studio as if you had set a breakpoint on that line. You can trigger Visual Studio to halt on any line as if you'd set a breakpoint on that line by calling an interrupt 3 call. You can examine the code in Visual Studio to determine what caused the problem, and optionally press f5 and have the code continue.

```
#define HALT __asm {  int 3  }
```

You can combine the breakpoint with checks to make your code more robust, e.g:

```
  #define DBG_ASSERT(exp) {if ( !(exp) ) {DBG_HALT;}}

  #define INVALID_FLOAT 2139095040
  #define DBG_CHECKFLOAT(f) \
      {if (((*(unsigned int *)((void *)&f)) & INVALID_FLOAT) == INVALID_FLOAT) {DBG_HALT;}}
```

For cross platform breakpoints (e.g. PS3/360/Win32)

```
#define __asm volatile( "trap" ) ; // Halts a program running on PPC32 or PPC64 (e.g. PS3).
```

```

__asm {int 3}          ; // Halts a program running on IA-32.
__debugbreak()         ; // Halts Win32 (basically a wrapper for int 3)
```

Output Window (or DOS Prompt)

You should direct all your text debug information through a macro or single function, this enables you to turn it off for speed. You can also redirect it to Visual Studio's output window (the preferred way), a txt log file, or even a DOS output window from a single location.

The function 'OutputDebugString(..)' let's you send strings to the output window while your program is running.

```
OutputDebugString( char* );
```

Taking advantage of vsprintf(..) you can create a debug print function that takes variable arguments, which you pass along to your output/log file.

```
//Saving debug information to a log file
__forceinline void dprintf(const char *fmt, ...)
{
    va_list parms;
    static char buf[2048]; // Dangerous - buffer overrun issues?

    // Try to print in the allocated space.
    va_start(parms, fmt);
    vsprintf (buf, fmt, parms);
    va_end(parms);

    // Dirty quick write the information out to a txt file
    #if 0
    FILE *fp = fopen("output.txt", "a+");
    fprintf(fp, "%s",  buf);
    fclose(fp);
    #endif

    // Output to the visual studio window
    OutputDebugString( buf );

}// End dprintf(..)
```

Memory Data

When using Visual Studio it will use the debug heap, which will initialize memory and variables to default magic values.

Value	**Usage**
0xCDCDCDCD	Allocated in heap, but not initialised.
0xCCCCCCCC	Allocated on the stack, but not initialized.
0xDDDDDDDD	Released heap memory.
0xFDFDFDFD	"NoMansLand" fences are placed around heap allocations to help you detect if you're walking off the edge of an array.

Watch Window

The watch window lets you see the value of any variable, but it also has a lot of hidden tricks to let you see additional information.

Type "@**err**" in the watch window to display the value from GetLastError().

For array pointers, you can expand the list by adding ",n" to the variable in the watch window. Where n is how many entry's you want to see (e.g. "ptr, 20").

Other watch flags:
"**@esp**" in the watch window, it display the stack pointer. You can even add to it, e.g. (esp + 10) to look at values.
"**@clk**" displays the clock timer value.
"**@eip**" show the instruction pointer.
"**@eax**", shows eax register value.
"**@tib**", thread information block.

Data Breakpoints

Data breakpoints are a very powerful tool which is underused because of the difficult interface.

Typical use would be to break when address "*(long*)0xCD23AABD, length == 5".

Visual Studio only supports four hardware breakpoints, if you add more than four, Visual Studio will just disable older ones. The data breakpoints can be very useful if you suspect corruption, and you know which part of memory is being damaged, you can set a data breakpoint to trigger if anything changes that memory address, letting you track down the culprit.

DLL Breakpoints

Using Visual Studio you can set breakpoints on dll functions in the watch window. So when that dll function is called anywhere it will trigger a breakpoint and let you examine who's calling it.
When your program starts up, the output window should show you which dlls it has loaded. There are dll explorer programs so you can open up any dll and it will list all the function names. Using the known dll name and function call, you can use the following syntax to call it from the watch window in Visual Studio:

{ , , dllname }FunctionName

Visual Studio will trigger a breakpoint when the dll function is called. (You can also use this method to change variables in the loaded dll if you know the variable name).

Memory Leaks

The debug libs let you add extra header files to catch memory leaks. These debug libs will determine how much memory is being used at some point and then when you trigger a stop point, it will let you know if any memory is still being allocated and what their addresses are.

First, include the header at the start before anything.

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
```

Second, at your program entry point (e.g. main() or WinMain()), enable memory tracking.

```
// Enable memory leak detection
_CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
_CrtSetReportMode( _CRT_ERROR, _CRTDBG_MODE_DEBUG );

// Any point in your code you can dump all the memory allocations
_CrtDumpMemoryLeaks();  // dump memory leaks to the output window
```

If you use the 'CrtDumpMemoryLeaks()' you get a list of memory that is being allocated. Each allocation has a corresponding number id so you can set a breakpoint when its allocated.

For example, if you call CrtDumpMemoryLeaks(), you would get something like this:

```
Detected memory leaks!
Dumping objects ->
{76} normal block at 0x001F1290, 400 bytes long.
 Data: <                > CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD
```

In your code, you would set a breakpoint to trigger when the number {76} is hit by adding an extra line of code:

```
   _crtBreakAlloc = 76; // Break at the 76th memory allocation.
     // Or
   _CrtSetBreakAlloc(76);  // Break at the 76th memory allocation.
```

Or alternatively you can link to the debug dll while it's running in the watch window, and set the debugger to break when the allocation number is hit:

```
{,,msvcr100d.dll}_crtBreakAlloc = 76
```

Optimized Code

Once you run optimized code, Visual Studio will attempt to match the actual code with the corresponding source code point. But due to optimisation methods this is not always the case and you may need to look at the disassembly window for the truth.

Release vs Debug

When your program works in 'Debug' but not in 'Release', some common problems are:

- Memory corruption (Over/ Under-runs of memory).
    - Enable the debug heap to help try and track it down.
- Uninitialized variables
    - Often set to some magic value in debug (e.g. 0).
    - Unless /GZ switch is enabled.
- Calling convention (e.g. GetProcAddress)
    - Use /GZ in compiler
- Memory alignment
    - Alignment in memory might need to be 16/32/128 bytes aligned (and just luckily is working in debug).
- Local variables being overwritten.
    - Variable with the same name being used multiple times inside a function.
    - Local variables being passed on the stack.
- Optimiser
    - Modifying the code causing it to break, thinks functions might not be getting used, or variables order being changed for speed. (Very rare but happens).

Outside Debugger

If you find that your application doesn't run when you run it standalone, this is can be because of numerous problems.

- When you run your program under the debugger it uses the debug heap. To avoid this, you can run the program standalone and then inside Visual Studio use 'attach to process' to debug your program.
- PATH may be different. Call GetEnfiromentString(..) in your application to get the working paths and check.
- Thread timing might be different.
- Using different dlls.

Hacking Code
On occasion while your code has stopped you might want to modify your assembly code while running. Such as blank out a specific line of code (e.g. function call), or have the assembly jump somewhere else, such commands would be:

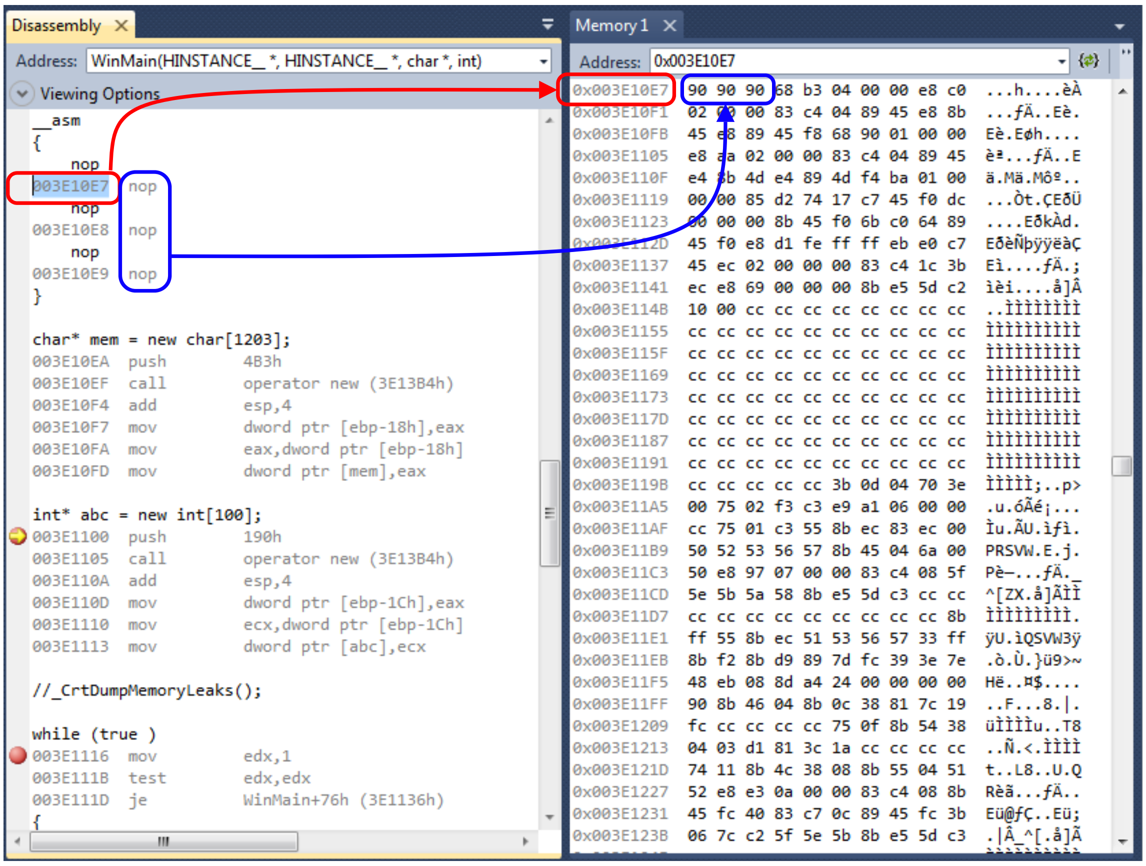
0x90 NOP
"Comment-out" chunks of code by using the NOP instruction in the assembly window.

0x74, 0x75 JE, JNE
Modify if statement to redirect your code in another direction.

To view the assembly, stop your program at a breakpoint while running then goto:
Debug->Windows->Disassembly

To view any piece of memory, view the memory viewer:
Debug-> Windows-> Memory

Note, the program must be running and be stopped before you can view the disassembly and memory windows.



Address: WinMain(HINSTANCE__ *, HINSTANCE__ *, char *, int)

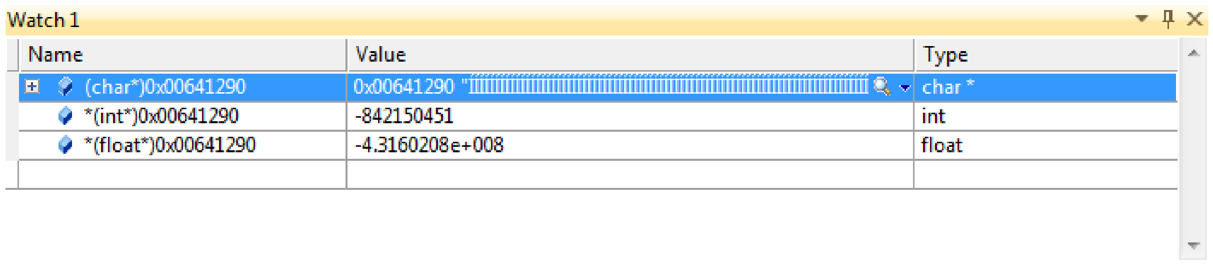
```
__asm
{
    nop
003E10E7  nop
    nop
003E10E8  nop
    nop
003E10E9  nop
}

char* mem = new char[1203];
003E10EA  push        4B3h
003E10EF  call        operator new (3E13B4h)
003E10F4  add         esp,4
003E10F7  mov         dword ptr [ebp-18h],eax
003E10FA  mov         eax,dword ptr [ebp-18h]
003E10FD  mov         dword ptr [mem],eax

int* abc = new int[100];
003E1100  push        190h
003E1105  call        operator new (3E13B4h)
003E110A  add         esp,4
003E110D  mov         dword ptr [ebp-1Ch],eax
003E1110  mov         ecx,dword ptr [ebp-1Ch]
003E1113  mov         dword ptr [abc],ecx

//_CrtDumpMemoryLeaks();

while (true )
003E1116  mov         edx,1
003E111B  test        edx,edx
003E111D  je          WinMain+76h (3E1136h)
{
```

Address: 0x003E10E7

```
0x003E10E7  90 90 90 68 b3 04 00 00 e8 c0  ...h....èÀ
0x003E10F1  02 00 00 83 c4 04 89 45 e8 8b  ...ƒÄ..Eè.
0x003E10FB  45 e8 89 45 f8 68 90 01 00 00  Eè.Eøh....
0x003E1105  e8 aa 02 00 00 83 c4 04 89 45  èª...ƒÄ..E
0x003E110F  e4 8b 4d e4 89 4d f4 ba 01 00  ä.Mä.Môº..
0x003E1119  00 00 85 d2 74 17 c7 45 f0 dc  ...Òt.ÇEðÜ
0x003E1123  00 00 00 8b 45 f0 6b c0 64 89  ....EðkÀd.
0x003E112D  45 f0 e8 d1 fe ff ff eb e0 c7  EðèÑþÿÿëàÇ
0x003E1137  45 ec 02 00 00 00 83 c4 1c 3b  Eì....ƒÄ.;
0x003E1141  ec e8 69 00 00 00 8b e5 5d c2  ìèi....å]Â
0x003E114B  10 00 cc cc cc cc cc cc cc cc  ..ÌÌÌÌÌÌÌÌ
0x003E1155  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E115F  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E1169  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E1173  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E117D  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E1187  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E1191  cc cc cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌÌÌÌÌ
0x003E119B  cc cc cc cc cc 3b 0d 04 70 3e  ÌÌÌÌÌ;..p>
0x003E11A5  00 75 02 f3 c3 e9 a1 06 00 00  .u.óÃé¡...
0x003E11AF  cc 75 01 c3 55 8b ec 83 ec 00  Ìu.ÃU.ìƒì.
0x003E11B9  50 52 53 56 57 8b 45 04 6a 00  PRSVW.E.j.
0x003E11C3  50 e8 97 07 00 00 83 c4 08 5f  Pè—...ƒÄ._
0x003E11CD  5e 5b 5a 58 8b e5 5d c3 cc cc  ^[ZX.å]ÃÌÌ
0x003E11D7  cc cc cc cc cc cc cc cc cc 8b  ÌÌÌÌÌÌÌÌÌ.
0x003E11E1  ff 55 8b ec 51 53 56 57 33 ff  ÿU.ìQSVW3ÿ
0x003E11EB  8b f2 8b d9 89 7d fc 39 3e 7e  .ò.Ù.}ü9>~
0x003E11F5  48 eb 08 8d a4 24 00 00 00 00  Hë..¤$....
0x003E11FF  90 8b 46 04 8b 0c 38 81 7c 19  ..F...8.|.
0x003E1209  fc cc cc cc cc 75 0f 8b 54 38  üÌÌÌÌu..T8
0x003E1213  04 03 d1 81 3c 1a cc cc cc cc  ..Ñ.<.ÌÌÌÌ
0x003E121D  74 11 8b 4c 38 08 8b 55 04 51  t..L8..U.Q
0x003E1227  52 e8 e3 0a 00 00 83 c4 08 8b  Rèã...ƒÄ..
0x003E1231  45 fc 40 83 c7 0c 89 45 fc 3b  Eü@ƒÇ..Eü;
0x003E123B  06 7c c2 5f 5e 5b 8b e5 5d c3  .|Â_^[.å]Ã
```

Figure 1. View disassembly and memory in real-time to modify/alter code.

Casting in Watch Window
The watch window lets you cast variables to anything, e.g. (char*)0x12345678, let's you look at memory address 0x12345678, because we cast it to char*, it will show the data as ascii.


Watch 1

Name	Value	Type
(char*)0x00641290	0x00641290 "ÍÍ	char *
(int)0x00641290	-842150451	int
(float)0x00641290	-4.3160208e+008	float

Append to PATH in Visual Studio

When running programs through Visual Studio for debugging, you may need to set environment variables temporarily. One such example would be to modify the PATH environment variable so that it includes an additional directory where dll's are loaded (OpenGL, Glut..).

Goto:
Project>Properties>Configuration Properties>Debugging "Environment" and "Merge Environment".

The syntax is NAME=VALUE and macros can be used (for example, $(OutDir)).

For example, to prepend "C:\Windows\Temp" to the PATH:

```
PATH=C:\WINDOWS\Temp;%PATH%
```

Similarly, to append $(TargetDir)\DLLS to the PATH:

```
PATH=%PATH%;$(TargetDir)
```

Recommended Reading

The tutorials only give you a starting point from which you can get a foot hold, so that you can get started and begin to explore and understand the power...

Work in progress...