UNIT – V

Debugging Tools

REF : HTTP://WWW.CS.CORNELL.EDU/COURSES/CS501/2000FA/SLIDES/DEBUG.PPT

Prof. Reshma Pise

Comp Engg. Dept

Vishwakarma University

Bug Identification & Elimination

1. Bug reports should contain a test case, output, and the version number of the software.
2. Reproduce the bug using the same version the customer used.
3. Find the root cause of the bug.
4. Check if the bug still occurs with the latest version. If it does, fix it.
5. If it doesn't, make sure it is not just masked by other changes to the software.
6. Add test cases used to reproduce the bug to the regression test suite.
7. Keep Records!

Debuggers

Debuggers are tools that can examine the state of a running program.

Common debuggers: adb, dbx, gdb, kdb, wdb, xdb.

Microsoft Visual Studio has a built-in debugger.

This talk will focus on the Visual Studio debugger.

Visual Debugger

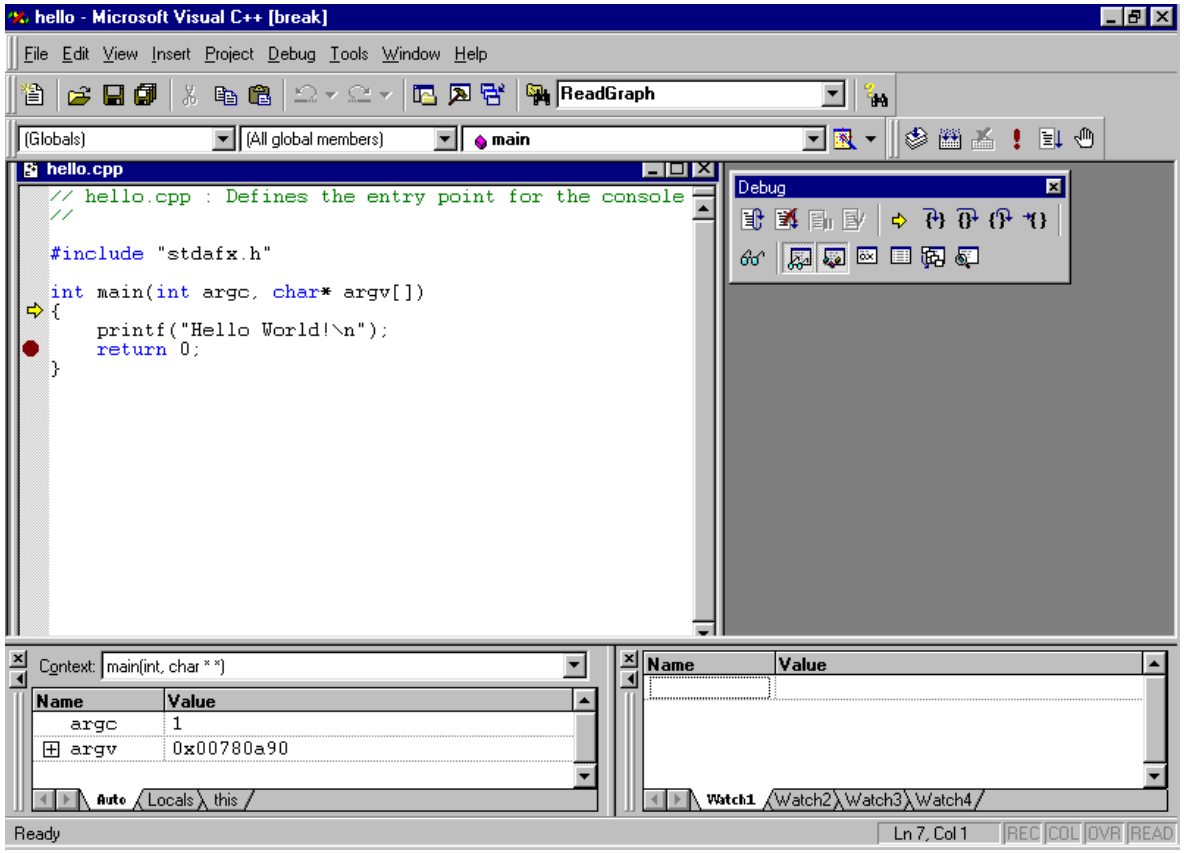
Graphically Oriented

Run from Visual Studio

Can debug a failed process by selecting the Yes button at “Debug Application” dialog after a memory or other failure occurs

Can attach to a running process by choosing the Tools->Start Debug->Attach to Process menu option

The Visual Debugger



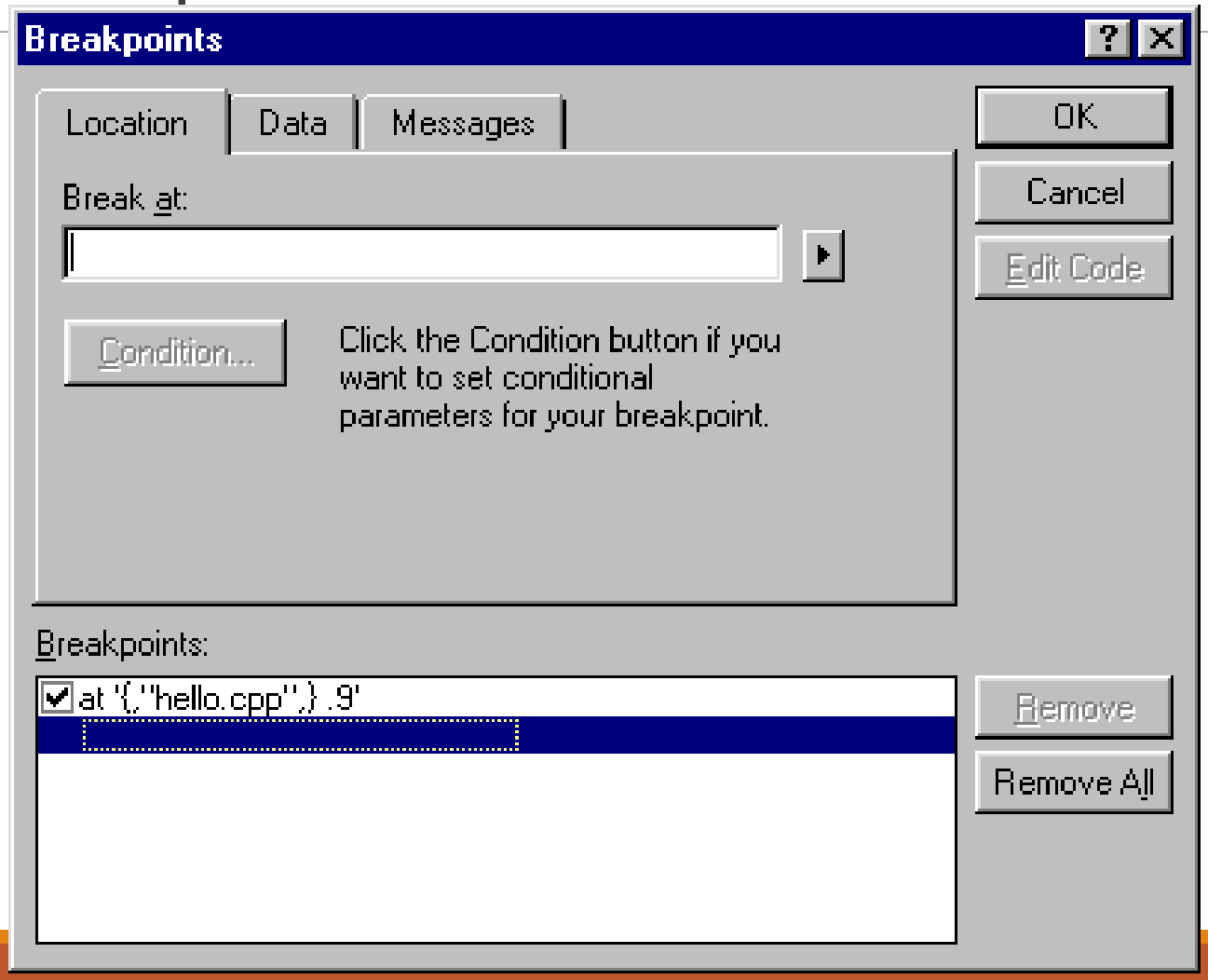
Breakpoints

Can stop execution at any line and in any function. (Location)

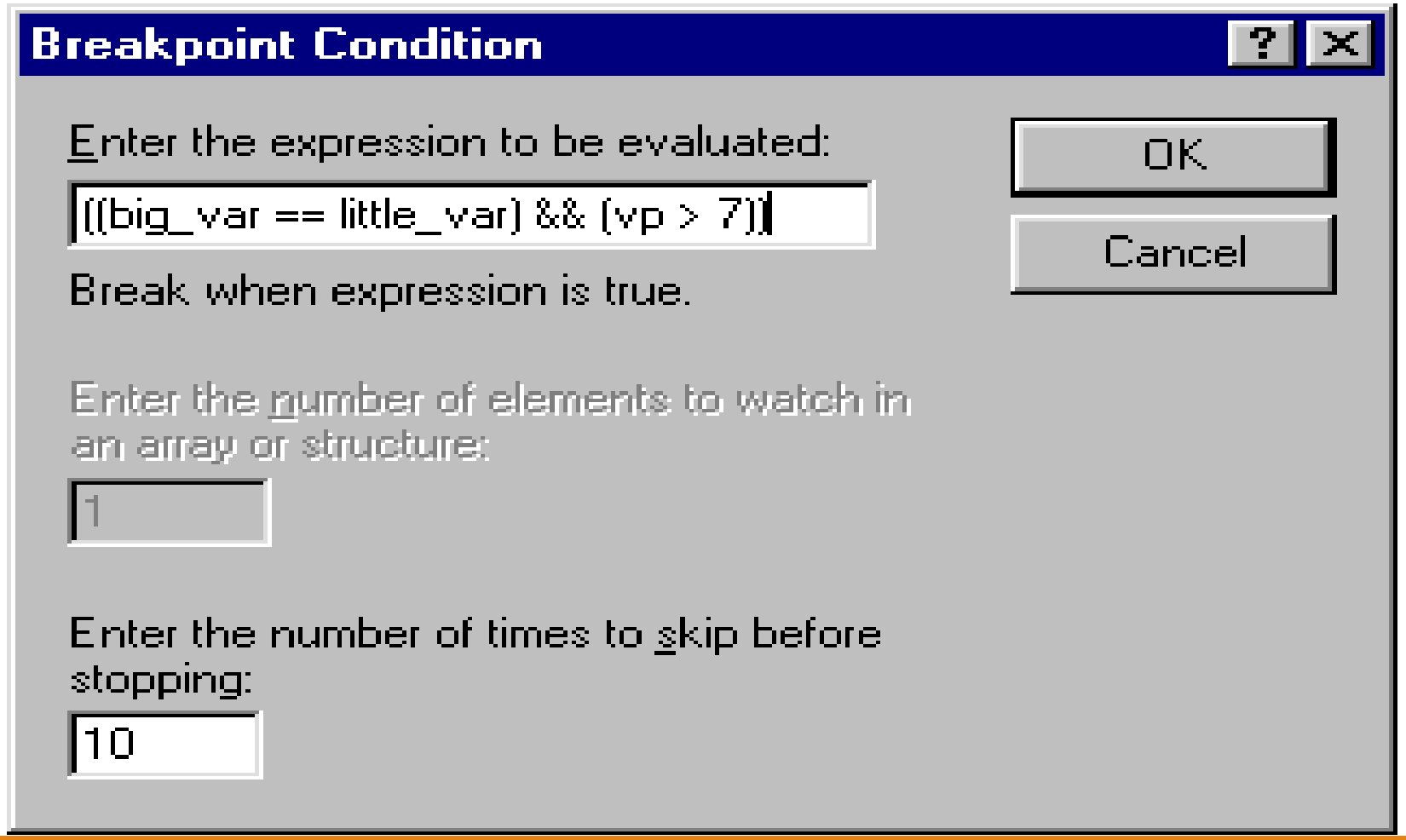
Can set conditions on breakpoints if you are only interested in specific passes through a piece of code (Location->Condition)

Conditional breakpoints detached from any one line in the program are also possible, but make program execution very slow (Data).

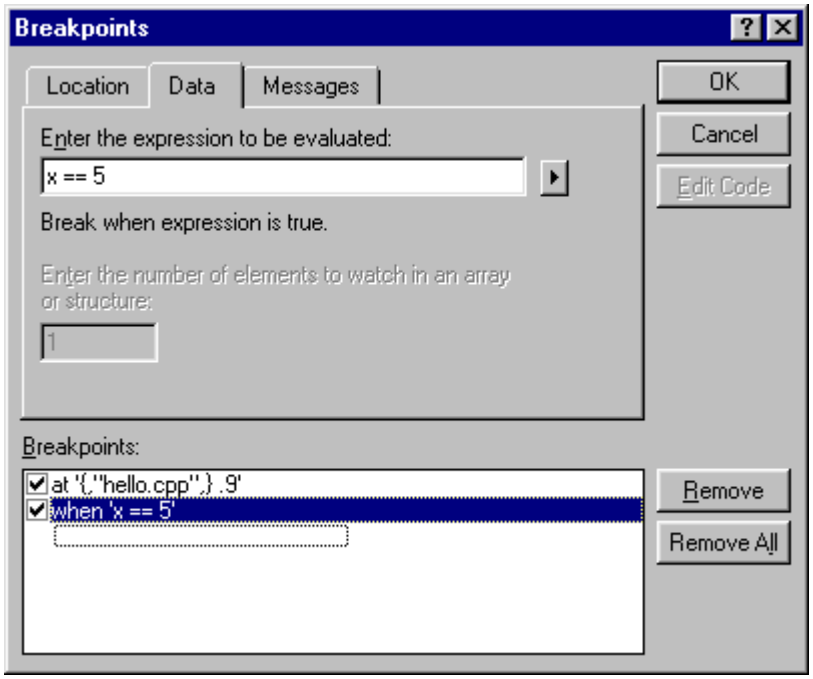
Breakpoint Window



Conditional Window



Conditional Data Breakpoint

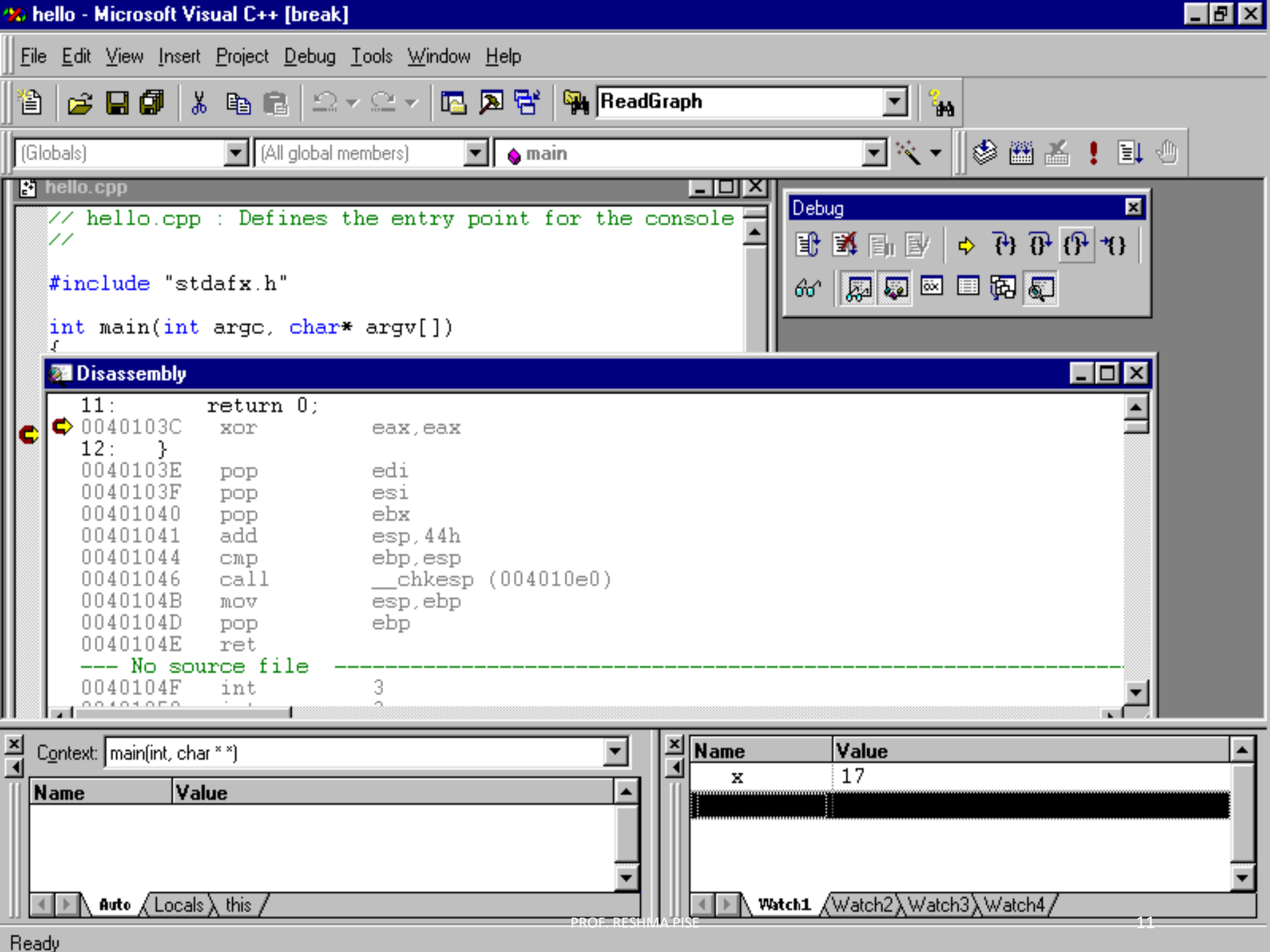


Examining Program State

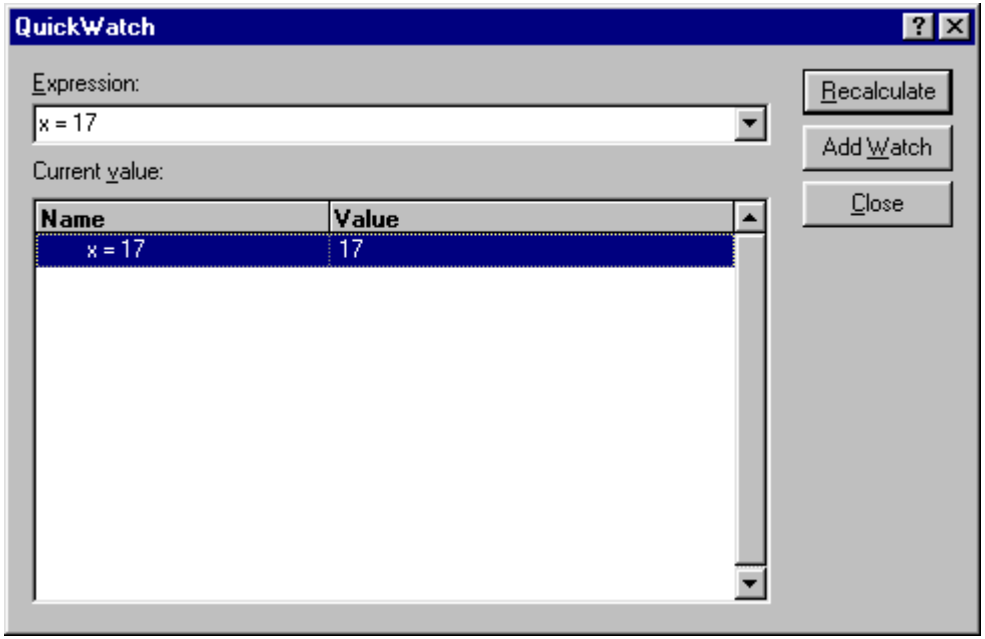
Print and/or Change variable state.

Walk up/down the stack trace.

View disassembled code.


hello - Microsoft Visual C++ [break]
File Edit View Insert Project Debug Tools Window Help
ReadGraph
(Globals)
(All global members)
main
hello.cpp
// hello.cpp : Defines the entry point for the console
//
#include "stdafx.h"
int main(int argc, char* argv[])
Debug
Disassembly
11: return 0;
0040103C xor eax,eax
12: }
0040103E pop edi
0040103F pop esi
00401040 pop ebx
00401041 add esp,44h
00401044 cmp ebp,esp
00401046 call __chkesp (004010e0)
0040104B mov esp,ebp
0040104D pop ebp
0040104E ret
--- No source file ---
0040104F int 3
Context: main(int, char * *)
Name
Value
Auto
Locals
this
Name
Value
x
17
Watch1
Watch2
Watch3
Watch4
Ready

Quick Print/Change Variables



Execution Flow

Step Into - Execute code, step into a function if one is called

Step Out - Continue execution until N-1’st region of stack frame reached

Step Over - Execute code, execute any functions that are called without stopping.

Debugging Pointer and Dynamic Data Structure Problems

Pointers and explicitly allocated dynamic data structures are a central feature in many popular procedural and object-oriented languages

- Great power - especially in extreme cases (eg C/C++)
- Can be very painful to debug

Common Pointer Problems

Pointer to bogus memory

Corrupt data structure segments

Data sharing errors

Accessing data elements of the wrong type

Attempting to use memory areas after freeing them

Debugging Multitasking Programs

Multiple process/multi-threaded code ubiquitous in modern programs

Many debuggers will work with these programs, but it is not always elegant or easy.

Fallback method: Put new processes to sleep and then attach a debugger to them before they awake.

Better solution: Read debugger documentation, find better one if it is weak in this area.

A Few Tips

Pointers and multithreading together can be extremely difficult to debug

Try to debug parts by themselves before tackling combined system

Analogous strategies to those used in pointer debugging can be a big help

Thread/process timing an important concern in the debugging process

Core Dumps

(Unix) If you run your code outside of the debugger and there is a fault a core file may be generated (depending on your system settings) where the current program state is stored.

Can debug your code post-mortem via: gdb executable-file core-file

Debug Prompts

Windows does not use core files.

If you run your code outside of a debugger and a problem occurs you will be given the option of either debugging the code or killing the executing process.

Blame the Compiler

Sometimes software crashes in debugged code but not in optimized code

The tendency is to blame the compiler and de-optimize the file or function where the bug occurred

Most often the problem is in the code and is just exposed by the optimizer, typically an uninitialized global variable

Of course, sometimes it really is an optimizer bug. In that case, please submit a bug report to the compiler vendor with a nice short test program

Debugging Techniques

Use assertions liberally

Add conditionally compilable debugging code

Multiple platform execution has a way of bringing bugs to the surface

Assertions

Can be used to enforce function pre and post conditions

Make your implicit assumptions explicit

Can be turned off in final release for a performance boost or left in with messages to help in bug report creation

Ex : In C / C++

Assertions are statements used to test assumptions made by programmer. For example, we may use assertion to check if pointer returned by malloc() is NULL or not.

Syntax for assertion.

void assert(expression);

If expression evaluates to 0 (false), then the expression, source code filename, and line number are sent to the standard error, and then abort() function is called.

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

int main()
{
        int x = 10;

        /* Some big code in between and let's say x
        is accidentally changed to 9 */
        x = 20;

        // Programmer assumes x to be 7 in rest of the code
        assert(x==10);

        /* Rest of the code */
        return 0;
}
```

Output :

```
Assertion failed: x==10, file test.cpp, line 13 This
application has requested the Runtime to terminate it
in an unusual way. Please contact the application's
support team for more information.
```

Assertion Vs Normal Error Handling

Assertions are mainly used to check logically impossible situations. For example, they can be used to check the state a code expects before it starts running or state after it finishes running. Unlike normal error handling, assertions are generally disabled at run-time. Therefore, it is not a good idea to write statements in assert() that can cause side effects.

For example writing something like assert(x = 5) is not a good ideas as x is changed and this change won't happen when assertions are disabled.

Ignoring Assertions

In C/C++, we can completely remove assertions at compile time using the preprocessor NODEBUG.

```
// The below program runs fine because NDEBUG is defined
# define NDEBUG
# include <assert.h>


int main()
{
        int x = 10;
        assert (x==20);
        return 0;
}
```

Conditional Compilation

Maintain multiple customized versions in one code base.

Typically have one debug version of your code for bug killing and a release version (sans debug code) for high performance.

Caveat 1: You do need to test the release version before shipping.

Caveat 2: Conditional Compilation not available in all languages.

Multiple Platform Execution

Additional initial design effort

Great debugging aid

Can be a commercial selling point

Debugging Aids

Lint for stricter code checks

Garbage Collectors for C/C++

Lint

- Lint is a semantic checker that identifies potential bugs in C programs
- **Lint is a mistake!**
- In the early days of C on UNIX complete semantic checking was removed from the C compiler as a design decision. This allowed for smaller, simpler, and faster compilers at the expense of potentially buggy code.
- Lint exists on UNIX systems (but not LINUX)
- Most modern ANSI C compilers include Lint semantic checking functionality but only some of Lint's other features
- **Use Lint Early and Often!**

What does Lint Do?

- Checks for consistency in function use across multiple files
- Finds
  - bugs
  - non-portable code
  - wasteful code
- Typical Bugs Detected include
  - Argument types transposed between function and call
  - Function with wrong number of arguments takes junk from stack
  - Variables being used before set or never used

More about Lint

- See Unix man page
- OR “Checking C Programs with lint” By Ian F. Darwin

Purify

- Purify is a tool for locating runtime errors in a C/C++ program
- Purify can find
  - Array bounds errors
  - Accesses through dangling pointers
  - Uninitialized memory reads
  - Memory allocation errors
  - Memory leaks
- Purify is available on Windows and UNIX systems and is a product of Rational Software www.rational.com

How Purify Works

- Purify instruments a program by adding protection instructions around every load and store operation
- When program is executed a viewer will be created to display errors as they happen
- Purify is flexible and can be run standalone with any executable (written in C) or within a debugging environment like Visual Studio
- Purify is customizable and can be set to ignore certain types of errors

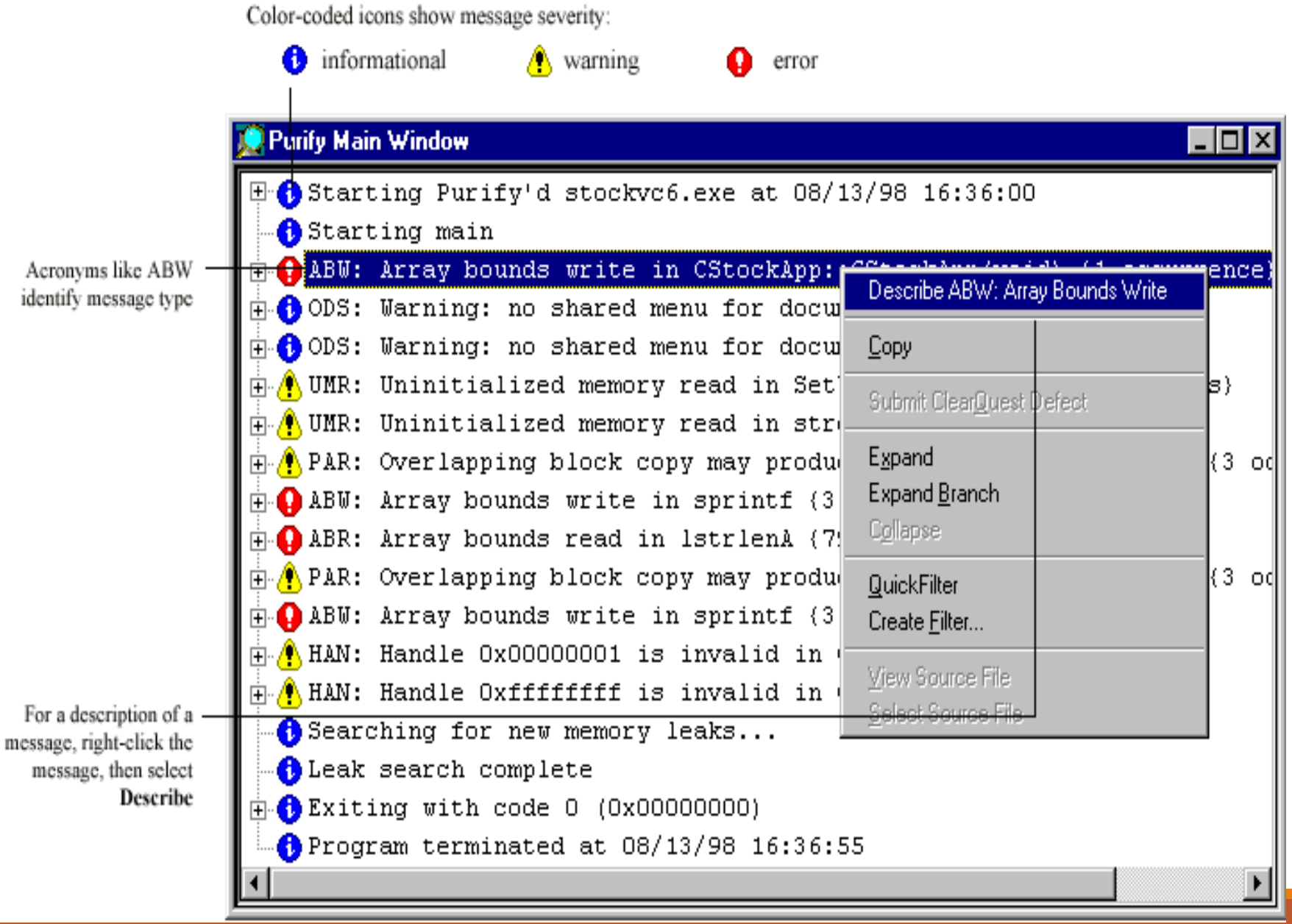
How to Use Purify

- add purify command to link command
- program: $(OBJS)

```
        purify [-option ...] $(CC) $(CFLAGS) -o\
    program $(OBJS) $(LIBS)
```

- OR run purify in Visual Studio
- OR load file in purify executable

Click **Run** to begin


Color-coded icons show message severity:
informational
warning
error
Purify Main Window
Starting Purify'd stockvc6.exe at 08/13/98 16:36:00
Starting main
ABW: Array bounds write in CStockApp:
ODS: Warning: no shared menu for docu
ODS: Warning: no shared menu for docu
UMR: Uninitialized memory read in Set
UMR: Uninitialized memory read in str
PAR: Overlapping block copy may produ
ABW: Array bounds write in sprintf (3
ABR: Array bounds read in lstrlenA (7
PAR: Overlapping block copy may produ
ABW: Array bounds write in sprintf (3
HAN: Handle 0x00000001 is invalid in
HAN: Handle 0xffffffff is invalid in
Searching for new memory leaks...
Leak search complete
Exiting with code 0 (0x00000000)
Program terminated at 08/13/98 16:36:55
Describe ABW: Array Bounds Write
Copy
Submit ClearQuest Defect
Expand
Expand Branch
Collapse
QuickFilter
Create Filter...
View Source File
Select Source File
Acronyms like ABW identify message type
For a description of a message, right-click the message, then select Describe

Linux Garbage Collection Aids

If you are using C then checker-gcc is an excellent tool - compile your code using modified gcc compiler and memory errors flagged

Options exist in C++ (checker-g++, ccmalloc, dmalloc), but they tend to be fragile and/or very slow.

Thank You