

Bug Class
or
Writing Production Quality Code

2009-2010 Supercomputing Challenge Kickoff,
Oct 25-26, 2009
Sacramento, NM
Bob Robey, Lori Liebrock

Production Quality vs Class Assignments

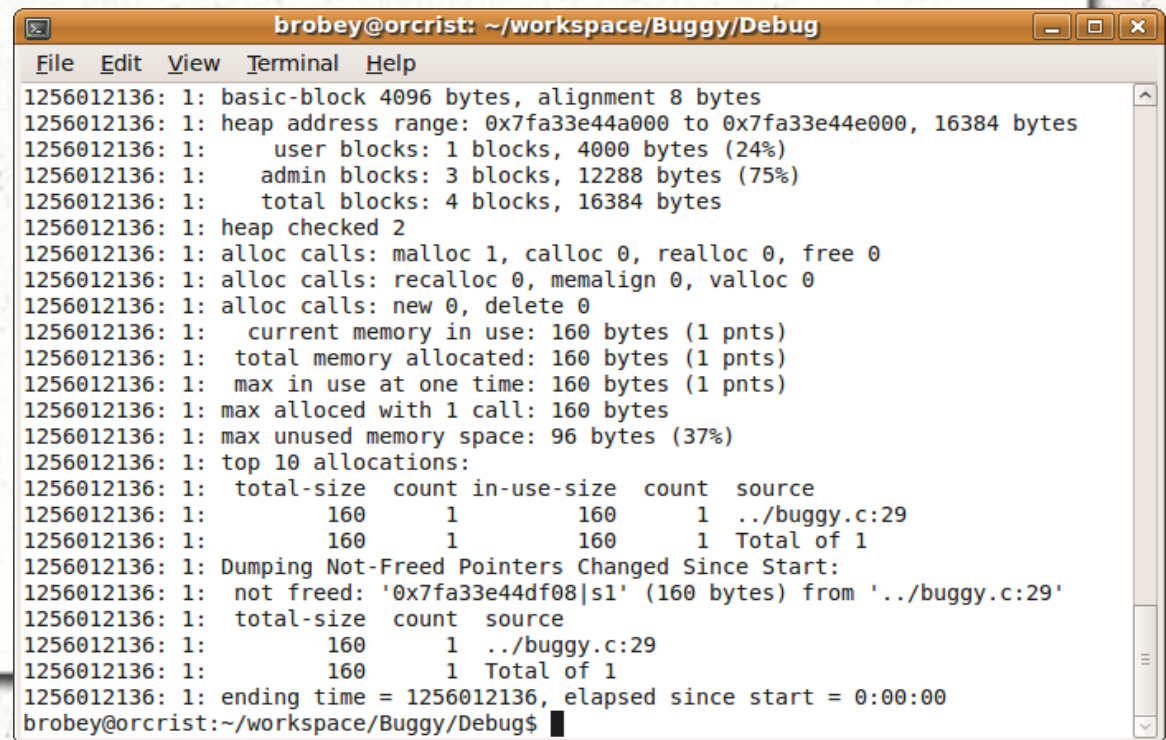
- Supercomputing projects need more production quality than class assignments
 - Memory leaks limit iterations and cause programs to “freeze”
 - Buggy routines crash randomly due to memory overwrites
 - Repeated runs give different results due to uninitialized memory
- Don't expect to learn all the techniques all at once – it takes many years.

Types of Bugs

- Memory Overwrites
 - Declare var x[10], write to x[1 to 20]
- Uninitialized Memory
 - Declare x[10], read before set
- Memory Leaks
 - Memory allocated, but not freed
- Invalid frees
 - Mixed types of malloc/free, new/delete, or 1d/2d

Correctness Tools (Linux Bias)

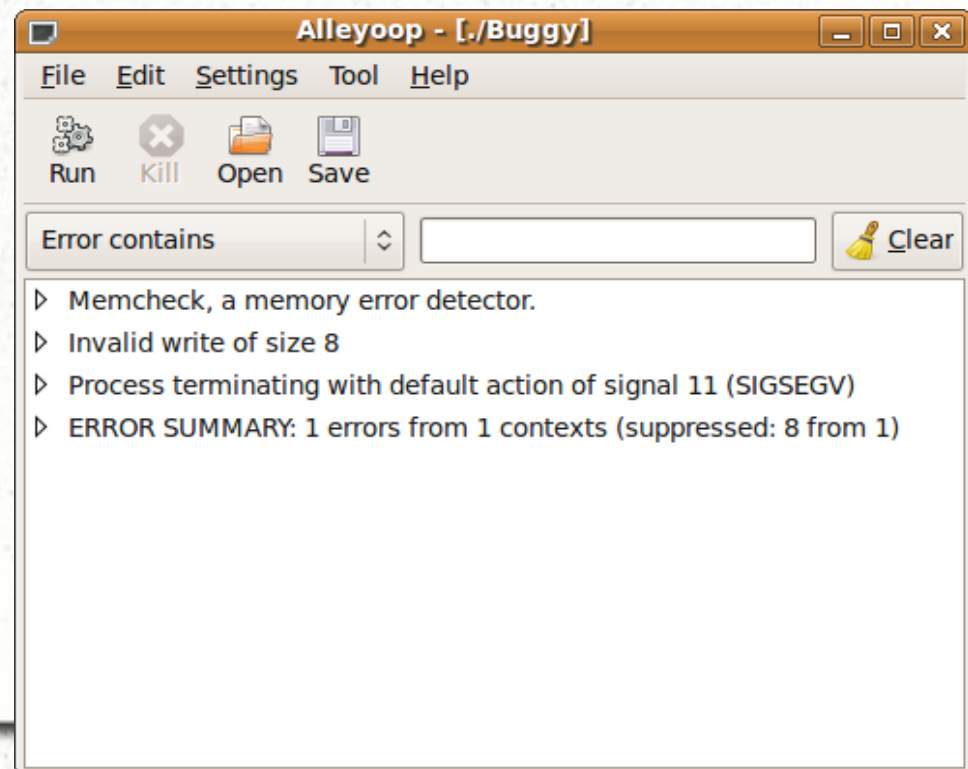
- Lint tools or static code analyzers
- Bounds Checkers -- Fortran and Java built-in
- Memory leak tools
 - Dmalloc, <http://dmalloc.com> – malloc lib replacement

A terminal window titled "brobey@orcrist: ~/workspace/Buggy/Debug" showing the output of the dmalloc tool. The output provides a detailed memory usage report, including heap address ranges, block counts, allocation statistics, and a list of top 10 allocations. It also shows a dump of not-freed pointers and the ending time of the process.

```
brobey@orcrist: ~/workspace/Buggy/Debug
File Edit View Terminal Help
1256012136: 1: basic-block 4096 bytes, alignment 8 bytes
1256012136: 1: heap address range: 0x7fa33e44a000 to 0x7fa33e44e000, 16384 bytes
1256012136: 1:   user blocks: 1 blocks, 4000 bytes (24%)
1256012136: 1:   admin blocks: 3 blocks, 12288 bytes (75%)
1256012136: 1:   total blocks: 4 blocks, 16384 bytes
1256012136: 1: heap checked 2
1256012136: 1: alloc calls: malloc 1, calloc 0, realloc 0, free 0
1256012136: 1: alloc calls: realloc 0, memalign 0, valloc 0
1256012136: 1: alloc calls: new 0, delete 0
1256012136: 1:   current memory in use: 160 bytes (1 pnts)
1256012136: 1:   total memory allocated: 160 bytes (1 pnts)
1256012136: 1:   max in use at one time: 160 bytes (1 pnts)
1256012136: 1:   max allocated with 1 call: 160 bytes
1256012136: 1:   max unused memory space: 96 bytes (37%)
1256012136: 1: top 10 allocations:
1256012136: 1:   total-size count in-use-size count source
1256012136: 1:     160      1      160      1  ../buggy.c:29
1256012136: 1:     160      1      160      1 Total of 1
1256012136: 1: Dumping Not-Freed Pointers Changed Since Start:
1256012136: 1: not freed: '0x7fa33e44df08|s1' (160 bytes) from ' ../buggy.c:29'
1256012136: 1:   total-size count source
1256012136: 1:     160      1  ../buggy.c:29
1256012136: 1:     160      1 Total of 1
1256012136: 1: ending time = 1256012136, elapsed since start = 0:00:00
brobey@orcrist:~/workspace/Buggy/Debug$
```

Correctness Tools (Linux Bias)

- Uninitialized memory
 - Some compilers (rare)
 - Valgrind, <http://valgrind.org> (with alleyoop gui frontend)



Productivity Tools

- Make
 - Dependency driven language, only compiles files with a newer date.
- Revision Control – CVS or Subversion
 - Important for team code development
- Ctags – variable cross-reference

Tools – Free Sources

- Comprehensive list at <http://www.thefreecountry.com>
 - Includes compilers, libraries, utilities and tools
 - Many old commercial tools are now available free and are listed here
 - Use sparingly, you could spend a lifetime trying all the listings

Debuggers

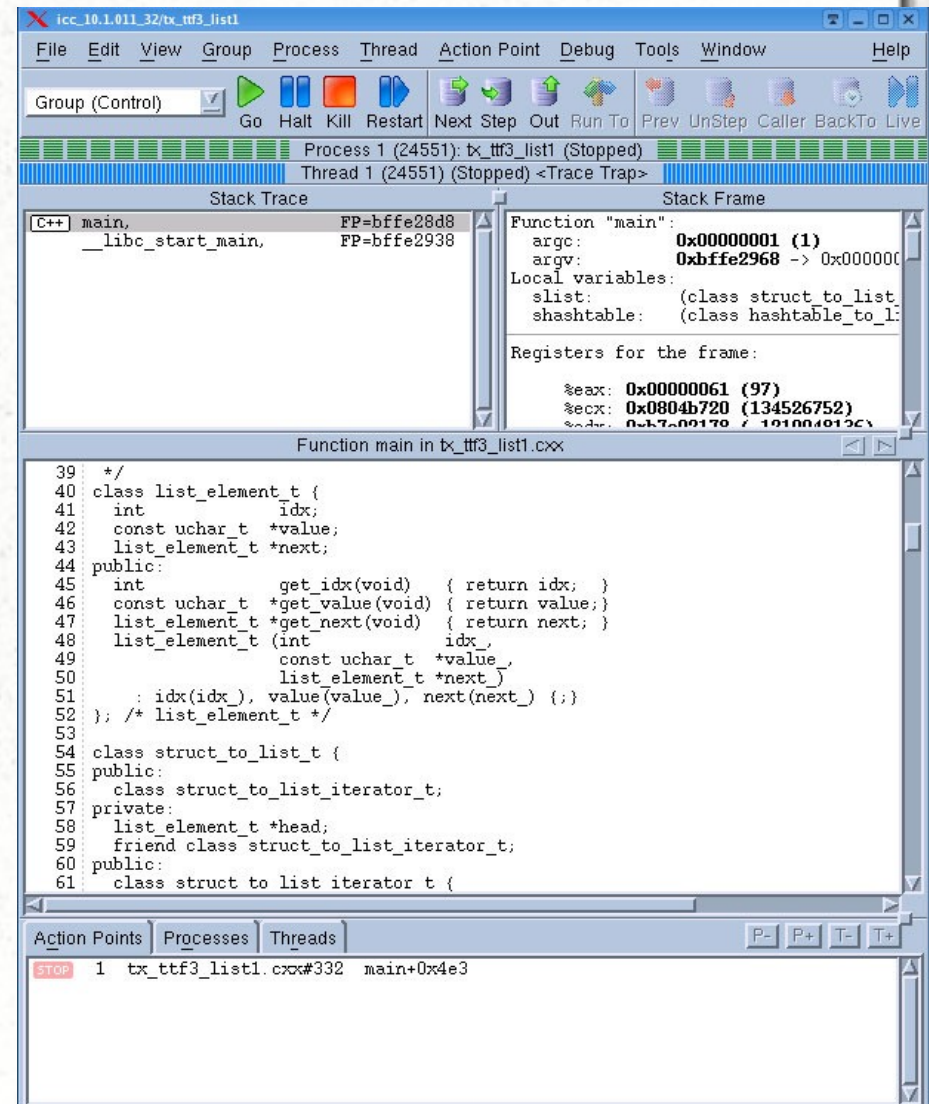
- Not just for “debugging”
- Good practice is to step through newly written code to see if it goes through the expected paths
- Critical for object-oriented programs because of jumping from routine to routine
- Can check that allocate/deallocates work correctly
- Stop at end of routine and look for any still allocated arrays

Debuggers (cont)

- Open-source are limited
 - gdb is text based, difficult to use
 - Gui frontends to gdb like ddd, xxdbg, or insight are weak
 - sdm in eclipse workbench adequate, but buggy
- Integrated Development Environments (IDE), Workbenches, Windows, MacOS
 - May have adequate debuggers depending on language, platform, etc

Totalview Student Program

- commercial debugger
- Much better quality and more powerful
- Has a student program for free
- Has powerful enhancements like parallel debugging, gpu debugging
- Replay engine (recorded debugging) great for instruction



Thoughts on Debugging

- The primary difference between a programmer right out of college and one with five years' experience is the ability to debug programs. [robelle.com]
- Start with small problems that can be easily checked by hand. [drpaulcarter.com]
- As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs. [Wilkes, Maurice]



Thoughts on Debugging (cont.)

- Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. [Brian W. Kernighan]
- Bloody instructions, which, being taught, return
To plague the inventor ...
[William Shakespeare, Macbeth, act 1, scene 7]
- If debugging is the process of removing bugs, then programming must be the process of putting them in. [Edsger W. Dijkstra]



Basic Debugging

- Breakpoints – set a breakpoint by clicking in the left margin of a line. Hit the go button and the program will run to that point
- Basic Motion
 - Next, or Step Over – progress one line statement forward in current routine
 - Step or Step Into – move one line forward but drop into subroutine if that is the next line
 - Step Out – move forward until the line just after the exit of the current subroutine
 - Run to – highlight a line and run to will move the program forward to that point


Exercise

- Start up Eclipse. If you have a different C workbench, you can use that instead.
- Create a new project called Buggy. Import the file `buggy.c`
- Compile. Switch to Debug perspective. Hit debug icon. Wait until job launches. Be patient, it is slow.
- Use next button  to run to the next line. Observe the variables in the upper right window. Open up the x array by clicking on the . Notice that x is not initialized. Try initializing it by adding `= {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}` to the declaration of the x array. Compile and run again. Is it initialized now?

Exercise (cont.)

- Line 26, `a = x[10];` is a read out-of-bounds. `x` is an array of 10, but in C that is from 0 to 9. What does the debugger show that `a` is assigned? Do you think this is reliable and repeatable?
- Line 27 is a write out-of-bounds. What does the debugger show?
- At line 31, there is a subroutine call. When the cursor is at that line, click on step into . Use next to go a few times through the loop. It will take a long time to go 100 iterations, so use step return  to get out of the routine.

Exercise (cont.)

- Set a breakpoint by right clicking in the left margin of the source at the line where you want to set the breakpoint. Use the resume  to run to the breakpoint.
- Try adding `=NULL` to declaration of pointer `b` (`*b=NULL`). Move `b=NULL;` after the `free`. Try stepping through with the debugger now.
- Line 37 writes to an array that is already freed. Note in the upper window after stepping through it that it shows a segmentation fault.

Exercise (cont)

- Fix the problem on line 37 (easiest to comment out with two backslashes – `//b[3] = 1.0;`);
- Right click on the main job in the upper window and select terminate and relaunch. Now step through and see what happens with the invalid free statements.
- How many errors crashed the program? How many were detected by the compiler? The debugger?

Debugger – Documentation

- GDB – <http://dirac.org/linux/gdb>
- Totalview –
<http://www.totalviewtech.com/support/documentation.html>
-

Appendices

Linux setup (64 bit Ubuntu)

- Eclipse for C programs
 - Install Eclipse by running the eclipse-cpp-galileo-SR1-linux-gtk-x86_64 installer
 - Install Java by running the jdk-6u16-nb-6_7_1-linux-ml.sh installer
 - Install compilers and supporting tools with the package manager
 - build-essential

Windows Setup

Eclipse for C programs

Install Wascana by running the Wascana installer

Add to system path by opening up control panel, search for system environment and adding to the end of the path

;C:\Program Files (x86)\GnuWin32\bin;C:\Program Files (x86)\Wascana\mingw\bin

When setting up the compiling in Eclipse

Add to GCC C Compiler:Directories

"C:\Program Files (x86)\GnuWin32\include"

Add to MinGW C Linker:Libraries

Library search path

"C:\Program Files (x86)\GnuWin32\lib"