CSE 303:
Concepts and Tools for Software Development

Dan Grossman

Winter 2006

Lecture 15— Debuggers, e.g., gdb

Where are We

"Tools you may not know exist" – debuggers, profilers, library-makers, recompilation managers, version-control systems.

The concepts behind these tools are orthogonal to programming language and level of abstraction.

But tools may need to "understand" your PL of choice.

And we'll largely use C to give you more practice.

Today: debuggers (a terribly misnamed tool).

An execution monitor?

What would like to "see from" and "do to" a running program?

Why might all that be helpful?

What are reasonable ways to debug a program?

A "debugger" is a tool that lets you stop running programs, inspect (sometimes set) values, etc.

Issues

- Source information for compiled code. (Get compiler help.)
- Stopping your program too late to find the problem. (Art.)
- Trying to "debug" the wrong algorithm.
- Trying to "run the debugger" instead of understanding the program.

It's an important tool. I use it sometimes.

Debugging C vs. Java

- Eliminating crashes does not make your C program correct.
- Debugging Java is "easier" because crashes and memory errors do not exist.
- But programming Java is "easier" for the same reason!

gdb

`gdb` (Gnu debugger) is on `attu` and supports several languages, including C compiled by `gcc`.

Modern IDEs have fancy GUI interfaces, which help, but concepts are the same.

Compiling with debugging information: `gcc -g`

- Otherwise, gdb can tell you little more than the stack of function calls.

Running `gdb`: `gdb` *executable*

- Source files should be in same directory (or use the `-d` flag).

At prompt: `run` *args*

Note: You can also inspect core files, which is why they get saved. (I never do.)

Basic functionality

- backtrace
- frame, up, down
- print *expression*, info args, info locals

Often enough for "crash debugging"

Also often enough for learning how "the compiler does things" (e.g., stack direction, malloc policy, ...)

Breakpoints

- break *function* (or line-number or ...)
- conditional breakpoints
  1. to skip a bunch of iterations
  2. to do assertion checking
- going forward: `continue`, `next`, `step`, `finish`
  - Some debuggers let you "go backwards" (typically an illusion)

Often enough for "binary search debugging"

Also useful for learning program structure (e.g., when is some function called)

Advice

Understand what the tool provides you.

Use it to accomplish a task, for example "I want to know the call-stack when I get the NULL-pointer dereference"

Optimize your time developing software.

Use development environments that have debuggers?

See also: `jdb` for Java (on `attu`)