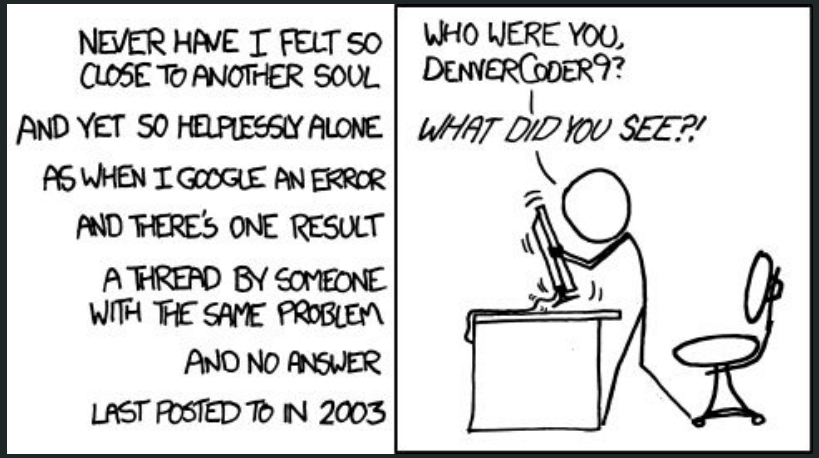
Debugging w/ GDB

feat. horrendous C code



Using GDB

- `-g` flag in Makefile for compiling

a) `gdb penn-shredder`
b) `gdb --args penn-shredder 3`
c) `gdb`
    - `(gdb) file penn-shredder`
    - `(gdb) set args 3`
    - `(gdb) run < in.txt`
    - `(gdb) help [command]`

Walking through code

Command	Shortcut	Description
`start`		• Start from beginning and stop there
`run`	`r`	• Start and run program from beginning
`continue`	`c`	• Run until next breakpoint / until program exits
`step`	`s`	• Run until next line* ○ Steps *into* a function
`next`	`n`	• Run until next line *in the current function* is reached / returns* ○ Steps *over* a function
`finish`	`fin`	• Run until the current function finishes*

*or until next breakpoint

Walking through code: example

```
int main(int argc, char* argv[]) {
  int n = 3;
  int fib = fibonacci(n);
  fprintf(stderr, "Fibonacci: %d\n", fib);

  return 0;
}
```

```
int fibonacci(int n) {
  int t1 = 0;
  int t2 = 1;
  int next = 0;

  if (n == 1) {
    return t1;
  } else if (n == 2) {
    return t2;
  }

  for (int i = 3; i <= n; i++) {
    next = t1 + t2;
    t1 = t2;
    t2 = next;
  }

  return next;
}
```

= continue

= step

= next

breakpoint

Where am I in the code?

Command	Description
`layout src`	● Changes the window layout to show source code
`refresh / ref`	● Refreshes the window incase it looks weird
`Ctrl-x + a`	● Close this window layout view
`list / l`	● Show some lines of source code before/around current
`backtrace / bt / where`	● Displays the call stack
`frame [number]`	● Selects and inspects a specific stack frame

Demo 0

Breakpoints (`b`/`break`)

Command	Description
`b [filename:]function` `b [filename:]linenum`	● Sets a breakpoint at the beginning of a function or at a specific line number
`info breakpoints` `info b`	● Lists all breakpoints w/ status and conditions
`disable [bnum]` `enable [bnum]`	● Disable or enable a specific breakpoint
`delete [bnum]` `d [bnum]`	● Deletes a specific breakpoint ● Deletes all if breakpoint num isn't specified
`clear [filename:]function` `clear [filename:]linenum`	● Removes breakpoints in a specific function or at a specific line number

Printing things (`p`/`print`)

Command	Description
`p var`	● Prints the value of a variable
`p/x var`	● Prints the value, in hex
`p var.field`	● Prints a field of a struct
`p var->field` `p (*var).field`	● Prints a field of a struct pointer
`p head->next->next->data`	● Example of printing data in a linked list
`p *arr[@len]`	● Prints the elements of an array, up to the specified length
`p var = value`	● Sets a different value to a variable

Inspection

Command	Description
`info args`	• Displays argos of the current function
`info locals`	• Displays local variables in the current function
`info variables [regex]`	• Lists all global and static variables + their data types • Can filter using regex
`info functions [regex]`	• Displays all functions in the program • Can filter using regex
`ptype [expression]`	• Shows the data type of the given expression • Can display the definition of a type (useful for structs)
`watch [expression]`	• Stops program whenever value of expression changes • Ex: `watch foobar if foobar > 3`

Demo 1 + 2

PennShell-specific debugging

Command	Description
`signal [signal]`	● Sends a signal (e.g. SIGINT) ● Useful to test Ctrl + C, Ctrl + Z, etc
`shell [cmd]`	● Executes a command as if you were in bash
`shell ps j`	● Lists process(es) info w/ job format output
`shell kill -9 <pid>`	● Sends a SIGKILL to a specific process (non-ignorable)
`kill`	● Kill the program being debugged
`set follow-fork-mode [parent|child]`	● After a fork, follow the child or parent process ● (parent by default)
`shell ls -l /proc/<pid>/fd`	● List a process' open fd's

Demo 3 + 4

Other Cool GDB Things

Command	Description
`disassemble / disas`	• View assembly instructions of the current function
`b [filename:]linenum condition`	• Make a breakpoint with an associated condition • e.g. `b 73 i > 4 && i % 2 == 0`
`call`	• Calls a function immediately • Helpful for on-the-fly behavior probing
`until, advance, jump, etc.`	• Even more ways to step through your code
`python`	• Yes you can do python scripting in GDB
`quit / q / Ctrl-D`	• Fixes your bugs instantly • Can touch grass

General Tips (for milestone and beyond)

- Use a debugger
  - gdb for C/C++ (and pdb for Python!)
  - VS Code extensions also available (with a GUI)
  - Take a break, go on a walk, etc.
- Use Valgrind to detect memory leaks / other memory issues
- Clean coding style
  - Use helper functions / helper files
  - Try to avoid nesting too many for/while/if/else's
  - Add comments for complicated bits
- Test incrementally
- Double check the man page / docs

Wrap Up

- We’ll post recording/slides on the website soon
- Quick reminder: Penn Shell due tomorrow (Wednesday)
- Open OH for the remaining time
- Any questions?