Overview, C, and Datalab

15-213/18-243

Eric Faust, Summer 2011

Agenda

- Class overview
  - Syllabus
  - Expectations
- Remember your C
  - GCC, GDB, and Make
  - C preprocessor
  - Pointers
- Datalab (Due Thursday)
  - Bitwise operations: Logical Shift, Arithmetic Shift, And, Or, Xor, Complement
  - Big vs. Little endian
  - Useful approaches for datalab
  - Old examples

Overview and Expectations

- Syllabus questions?
- Autolab or Shark machine troubles?
  - PLEASE see me after
- Our expectations
  - Understanding of the C language
  - More debugging maturity than 122 or 123
  - Put in the effort
    - We are here to help, but if you lean on us too hard, you only cheat yourself.
- Lab grading
  - Style

Debugging Quotes

- “As soon as we started programming, we found out 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.” ~ Maurice Wilkes
- “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 Kernighan

Debugging (Lovingly stolen from 15-410)

- Debugging is hard! Dive in!
- A tale of two stories
  - Looking for divergence
  - Fractal (you probably won't need much zoom)
- At every level:
  - What happened?
    - How can I tell what happened?
  - What **should** have happened?
    - If you struggle here, come talk to us
- “It crashes” is only the first level
- As always, we are here to help you if you get stuck

Your C Toolchain

- Source code → Compiler → Binary
  - Run binary
  - Binary probably crashes
    - Don't worry, that's what they do.
- Debugger
  - Step by step
  - Examine world state
    - Symbolic debugging
- Build systems
  - Make

GCC

- Compiles C source code to x86 assembly, and then assembles into machine code.
  - Compilers are a huge convenience.
- Interesting flags:
  - ansi
  - Pedantic
  - Wall
  - Werror
  - O2
    - !Debugging with optimizations!
  - g
    - This one will be important to you

GDB

- Single step symbolic debugger
  - step, finish
  - continue
  - print my_list->head
- Invoked via ./gdb <binary>
  - At the prompt, give the “run” command to start execution
  - If your binary takes command line arguments, wait to give those when you give gdb the “run” command
- Breakpoints
  - “Stop when you get here”
  - break <file>:<line number>

Make

- Scriptable build system
- Compile and link large programs together, while allowing multiple files.
  - Rebuild things based on “dependencies”
- Change in one file might mean recompiling the whole project
  - Might mean recompiling only that file
- We provide Makefiles for you for all coding labs.

The C Design Template

- Modularity
- Think about interfaces
- Headers
  - .h files
  - Define interfaces, contain struct definitions
- Code files
  - .c files
  - Contain implementations and internal helpers
- Link it all together (more on this later)
- static keyword
  - “Please don't show this to any other files but me”

C pre-processor

- Sometimes abbreviated cpp
- Runs over the code before the compiler has a chance to get to it.
- **Does not attempt to “understand code”**
  - Only does textual substitution
- Pre-processor has global state
- Macros
- Magic Numbers
- Headers

#include

- Does a literal line by line inclusion of the text of the other file.
- Useful for modularity
- Can #include .c or .h files
- <> for “system headers”
  - Things in your include path
  - Think <stdio.h>
- “” for “local headers”
  - Things not in your include path
  - Probably in local directory
  - Think “btest.h”

#define

- Macros
  - “I have a bunch of code that needs to vary at the variable name level instead of the variable value level”
  - Takes arguments, does code insertion with those arguments replaced in the appropriate place
    - REMEMBER: The pre-processor does not “understand” your code, it only replaces
- Macros can also have no arguments
  - Convenient to hide away those constants that could change
  - #define INITIAL_LENGTH

#ifdef/#ifndef/#undef/#endif

- We can know at pre-process time whether or not something has already been defined
  - Remember, global state
- Useful for the sake of doing “header guards”
  - Error: this_will_not_work already defined
  - Might anger linker
  - What we do is:
    - #ifndef __MAGIC_NAME__
    - #define __MAGIC_NAME__
    - …
    - #endif
- #undef un-defines something

Pointers Review

- Memory is a byte-adressable array
  - A pointer is just an address to read from or write to
  - Big row of PO boxes
  - Think Excel spreadsheet with a single row
    - Can “merge” cells with multi-byte type pointers
    - Take on lowest addess as “pointer number”
- Using pointers
  - & takes the address of something
  - * dereferences something
  - If you need to, “count stars”, and make sure you have the right type
  - Need a pointer to do an “external change”

Pointer Arithmetic

- When we increment a pointer, we step by sizeof(type) bytes.
- If we have int * foo, foo + 3 is actually 12 bytes forward!
- Cleverly, this means that foo[3] = *(foo + 3)

Hexadecimal

- Use digits for 0-15 to represent 4 binary bits
  - We use 0-F
- Numbers written 0x<hex number>
- Allows fast binary translation, which drastically shrinking the number of needed digits

Bitshifts

- Logical
  - Pad with zeroes
  - Left shift always logical
  - Right shift logical (for our purposes) on unsigned types
  - 0x0F >> 4 = 0x0F
- Arithmetic
  - Sign extend
    - Highest order bit the “sign bit” (more tomorrow)
    - Repeat highest bit to maintain signedness
  - Signed datatypes arithmetic shift by default
  - 0x0F >> 4 = 0xFF

Big vs. Little Endian

- We can encode bytes in one of two orders in memory
- Big Endian
  - Encode the bytes just like humans would read them
  - Most significant bit of number is at lowest address
  - 0xdeadbeef → de ad be ef
- Little Endian (what we use)
  - Encode the bytes with the least significant bit at the lowest address
    - Don't change order of bits per byte, just reorder
  - Makes typecasts down in size cheap
    - We just read fewer bytes
  - 0xdeadbeef → ef be ad de

Other Bitwise Operations

- And
  - Final bit is on if both input bits are on
- Or
  - Final bit is on if either of input bits is on
- Xor
  - Final bit is on if exactly one of input bits is on
- Complement
  - Final bit is on if input bit was off, off if input bit was on
  - “Flip every bit”
- Bang (!)
  - Maps 0 to 1 and everything else to 0
  - Logical negation

Helpful thoughts for Datalab

- “What operations are useful to me here?”
  - “What is the problem?”
  - “What is this operation good for?”
  - We often give you access to more ops than you would need to solve the problem
- “What about this input makes it special that I can exploit?”
  - Datalab solutions are founded around manipulating what you get, no matter what it is, into a much smaller set of things that you can count on and use.

Solution removed by F11 course staff