15213 Recitation 1
In case you didn’t get it (or you were too lazy to check)

• Class Web page: http://www.cs.cmu.edu/~213
• No Blackboard, no Piazza
• Questions? Email 15-213-staff@cs.cmu.edu
• Office hours: SMTWR, 6-8pm, WeH 5207
• Need help? 1:1 appointments available
• Sharks Machines: ssh shark.ics.cs.cmu.edu
Fun Stuff

• 7 labs, 1 midterm, 1 final
• All labs are individual
• All assignments due 11:59 pm on their respective due dates
• Conflicts – talk to us AHEAD of time
• Grade appeals only good for 7 days after grade release – formal procedures in syllabus
• 5 grace days, max of 2 per lab
• No grace days? 15% per day penalty afterwards
• No handin after 3 late days
Emailing the staff list

- Answers to these questions should be, “Yes”
  - Have I read the textbook?
  - Have I read the writeup?
  - Have I read the FAQs?
  - Have I read relevant man pages?
  - Have I spent a nontrivial amount of time pondering the problem myself?

- We want to help, but we also want you to learn how to problem solve on your own
Just in case

- Cheating is bad
- Don’t cheat
- Cheating is bad
Bit-Level Operators

- AND = &
- OR = |
- NOT = ~
- XOR = ^
- Applies to integer types: char, short, int, etc
- \( \sim 0x0 = 0xF \)
- \( 0x4 \mid 0x6 = 0x6 \)
- \( 0xA \& 0X6 = 0X2 \)
Logical Operators

- AND = &&
- OR = ||
- NOT = !
- 0 = “False”
- Nonzero = “True”
- (!0x0) = 0x1
- (!0x4) = 0x0
- 0xBEEF && OxDEAF = 0x1
- 0xFEED && 0xDEED = 0x1
- Short-circuit. If second expression does not need to be calculated, the machine does not. For example, if c = 0,
  - (++c || ++c)
  - Value of expression above is 1, but c is now 1, not 2
Fun Fact to Keep You From Insanity

• && and & are NOT the same thing. && applies to logical expression while & applies to bit (or bitwise vectors)
• Similarly for || & |
• If funny stuff is happening in controls, check your conditionals for these mistakes
Shifts

• Left shift: $x << y$.
  – Shift bit vector $x$ left by $y$ positions.
  – Discard the extra bits on the left.
  – Fill new bits on right with 0’s

• Right shift: $x >> y$.
  – Shift bit vector $x$ right by $y$ positions.
  – Discard extra bits on right.
  – Logical: Fill new bits on left with 0’s
  – Arithmetic: Fill new bits on left with most significant bit of $x$

• If $y < 0$ or $y >=$ word size, undefined behavior
Encoding Integers

- int /= integers
- Unsigned: $B2U(X) = \sum_{i=0}^{w-1} x_i 2^{i}$
- Signed: $B2T(X) = -2^{w-1} + \sum_{i=0}^{w-2} x_i 2^{i}$
- Sign bit: most significant bit indicates sign for two’s complement numbers
  - 0 for non-negative
  - 1 for negative
Two’s Complement

- $-x = \sim x + 1$
- $x = 00000110_2 = 6$
- $\sim x = 11111001_2 = -7$
- $-x = 11111010_2 = -6$
Range

• Unsigned
  – $U_{Min} = 0 = 000...000 \downarrow 2$
  – $U_{Max} = 2^{w} - 1 = 111...111 \downarrow 2$

• Signed
  – $T_{Min} = -2^{w} - 1 = 100...000 \downarrow 2$
  – $T_{Max} = 2^{w} - 1 - 1 = 011...111 \downarrow 2$`
Fun fact:

• In expressions mixing ints and unsigned ints, ints are casted to unsigned ints first!
  – (unsigned) 0 > (signed -1) returns 0 (false)
Lab 1: Data Lab

• Not a very hard lab
• Still, start early to prepare for any unforeseen issues with code/Autolab/Shark/Andrew/The World

**READ THE WHOLE HANDOUT FIRST**
  – Then ask questions :)

• More fun facts to save you from insanity:
  – Declare all your variables at the very beginning of each function, or the dlc will cause you much pain.
  – Work on the shark machines. Or else the dlc may not work and cause you much pain.
  – Don’t unzip files locally. Un-tar within shark.
How to access UNIX

• Know how to remotely access lab machines
  – SCP (WinSCP if you're on Windows, Filezilla on Mac)
  – SSH (PuTTY if you're on Windows)

• Familiarize yourself with a reasonable text editor
  – Using Sublime locally is fun, but you’ll have to transfer your files to shark all the time -> not fun, dangerous
  – Just edit code from terminal (Emacs, Vim)

• Figure out Unix and AFS file permissions
Style

• It’s important!
  – Your code should be easy for others to understand
• 80 characters per line
  – Over 80 causes line overflows when we read your code, doesn’t look good
  – Make sure to use tabs/spaces consistently
  – We read with tab width of 4. Use the same.
• Make your variables meaningful.
  – a, b, c, temp, tmp = bad
• Read the style guidelines on the course website for more
• TAs also willing to help with difficult style queries