nano? Real programmers use Emacs.

Hey, real programmers use VIM.

Well, real programmers use ed.

No, real programmers use Cot.

Real programmers use a magnetized needle and a steady hand.

Excuse me, but real programmers use butterflies.

They open their hands and let the delicate wings flap once.

The disturbance ripples outward, changing the flow of the eddy currents in the upper atmosphere.

Which act as lenses that deflect incoming cosmic rays, focusing them to strike the drive platter and flip the desired bit.

Nice! Course, there’s an Emacs command to do that.

Oh yeah! Good ol’ C-x M-b "Butterfly..."

Damn, Emacs.
Michael Lee

- lee@iit.edu
- http://moss.cs.iit.edu
- Office: SB 226C
- Hours: Wed/Fri 11:30AM-1:30PM
  (make appointments on homepage)
TA: Xincheng Yang

- xyang76@hawk.iit.edu
- Hours: Tue/Thu (times TBA)
Agenda

- Course overview
- Administrivia
- Grading
- Assessments
- Resources
§ Programming Languages
Programming Languages …

- Theoretically all the same, yet practically very different!
- “Same” in a deep sense: *Turing completeness*
- Learning different languages and language features can vastly expand your repertoire of programming techniques
- PLs are among our most important and fundamental tools!
PL features

- Must learn to precisely dissect and discuss PLs

- Terminology: imperative, functional, compilers, interpreters, types, type checking, etc.

- Many terms are used imprecisely in conversation!
Reasoning about PLs

- What does a program (or PL construct) mean?
- Can we prove a program’s correctness?
- Many different ways of modeling and reasoning about program semantics
- Goal: inject mathematical rigor into programming
Not just a consumer!

- You will modify and create your own PLs
- Understand how PLs tick
  - Where is the overhead? Is it useful/necessary/worthwhile?
- Fun and useful skill!
We will …

1. Use a new language, Racket, to learn about different programming language constructs and ideas.
2. Learn different methods of language specification, focusing on semantics and verification.
3. Understand how programs are interpreted, compiled, represented, evaluated, and optimized.
4. Implement interpreters for our own programming language.
Topics

- Racket
- Syntax
- Higher order functions
- Recursion
- Closures
- Metaprogramming
- Continuations

- Grammars and Languages
- Semantics
- Operational / Axiomatic
- Evaluation strategies
- Interpreters and Compilers
- Type inference and Unification
- Memory management
§ Administrivia
Prior knowledge

- Programming experience (CS 115/116/201)
- First-order / Predicate logic (CS 330)
- Rules of inference and logical proofs (CS 330)
- Formal languages and Grammars (CS 330)
- Analysis of algorithms (CS 331 / 430)
Course website: http://moss.cs.iit.edu/cs440
Discord: TA class discussion and Q/A
(invite on course website)
References (in addition to notes)

- Programming Languages: Application and Interpretation, by Shriram Krishnamurthi
- Crafting Interpreters, by Robert Nystrom
Grading

- 60% Assignments
- 20% Midterm Exam
- 20% Final Exam (Cumulative)
Assignments

- 5-7 total
  - Some written, some machine problems (coding problems)
  - Written submitted via Blackboard, MPs via GitHub
Late Policy

- 7-day late pool, distributed however you like across labs (a day at a time)

- If you’re out of late days, late submissions will not be accepted for a grade!
Exams

- Scores may be linearly scaled so that median/mean (whichever lower) is 75%

- Midterm tentatively scheduled for October 14
A: $\geq 90\%$
B: 80-89\%
C: 70-79\%
D: 60-69\%
E: < 60\%
For Friday

- Read chapter 2 of *Crafting Interpreters*: “A Map of the Territory”

- Download and install DrRacket (https://racket-lang.org)

- Clone the class lecture repository from GitHub (https://github.com/cs440lang/lectures/)

- Bring your laptop to class!