Lecture 25: Victory Lap

Profs. Clarkson & George
Spring 2015

Today’s music: "We are the Champions" by Queen
Victory Lap

Extra trip around the track by the exhausted victors (us) 😊
Thank you!

Huge thank you to TAs and consultants!
Thank you!

Piazza heroes:

**Top Student Contributors**

- **Nikita Olegovich Torosyan**: 154 contributions; 86 days online
- **Genki Marshall**: 131 contributions; 97 days online
- **David Li**: 62 contributions; 93 days online
- **Jessica Lee**: 55 contributions; 41 days online
- **Eduardo Ferreira**: 53 contributions; 79 days online
Thank you!

And a huge thank you to all of you!

– You surmounted a daunting challenge
– You occasionally laughed at our dumb jokes 😊
What did we learn?

• You feel exhausted...
• You're tired of coding...

...step back and think about what happened along the way
What did we learn?

From the syllabus:
- Alternative programming paradigms, especially functional and concurrent programming
- Writing and using specifications
- Modular programming and data abstraction
- Reasoning about program correctness
- Reasoning about system performance
- Useful and efficient data structures

...and some cross-cutting, big ideas...
1. Syntax and Semantics

• Every language feature can be defined in isolation from other features, with rules for:
  – syntax
  – static semantics (typing rules)
  – dynamic semantics (evaluation rules)

• Divide-and-conquer!

• Entire language can be defined mathematically and precisely
  – SML is. Read *The Definition of Standard ML (Revised)* (Tofte, Harper, MacQueen, 1997).

• Learning to think about software in this “PL” way has made you a better programmer even when you go back to old ways
  – And given you the mental tools and experience you need for a lifetime of confidently picking up new languages and ideas
2. Benefits of immutability

• Programmer can alias or copy without worry

• Parallel programming easier with immutable data

• But mutability is appropriate when you need to model inherently state-based phenomena — or implement some efficient data structures
3. Programming languages aren’t magic

• Pattern matching, type inference, closures: all features you can implement yourself

• Interpretation of a (smallish) language is something you can implement yourself

• Compilation of a large language is also something you could implement!
4. Elegant abstractions are magic

From a small number of simple ideas...

...an explosion of code!

- map and fold
- numbers and list-like
- async
- monads
- Map-Reduce
5. Building software is more than hacking

- **Design**: think before you type
- **Empathy**: write code to communicate
- **Assurance**: testing and verification
- **Performance**: theory and experimentation
- **Group work**
What next?

• Follow-on courses:
  – CS 4110 Programming Languages and Logics (how to define and reason about programming languages)
  – CS 4120 Compilers (how to implement programming languages)

• Join the course staff?
  – CS department sent out email with URL for application site
  – Deadline is Friday, May 8, 4:30 pm for first round

• Stay in touch
  – Tell us when 3110 helps you out with future courses (or jobs!)
  – Ask us cool PL questions
  – Drop by to tell us about the rest of your time in CS (and beyond!)... we really do like to know

• GO DO AMAZING THINGS WITH YOUR LIFE
Q&A
App Demo Session

• Saturday, May 9, 12:30 pm, room TBA
• Free food!
  – Please RSVP in Piazza poll (even if not attending)
• You can demo whatever you like
• Awards and swag for student and staff favorites
Final Exam

• Sunday, May 17, 2:00-4:30 pm, Olin 155
• Covers everything in the course
• You may have three pages of notes
• (remaining details posted on Piazza as usual)
Final Grades

• Fill out the course eval to get your +1%
  – Take time on this, especially the free response questions

• Final grades:
  – uploaded to registrar approx. 24 hours after final exam scores are released on CMS
  – less than 24 hours after that, the registrar locks them, and we can't change them
Finally

• The most important idea of this course:
  – complicated artifacts can be broken down into small pieces
  – you can then study those small pieces and understand how they work in isolation
  – then you can understand why their aggregation achieves some goals

• Examples: the OCaml language, or a module you designed

• That kind of analysis is applicable anywhere, not just programming
THE END