# Welcome to CPSC 312

# Functional and Logic Programming

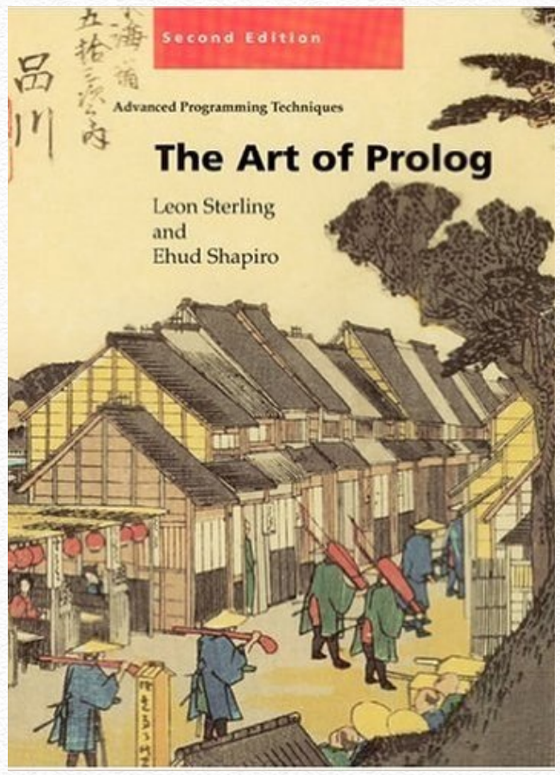
## Fall 2015

# Meet the Team

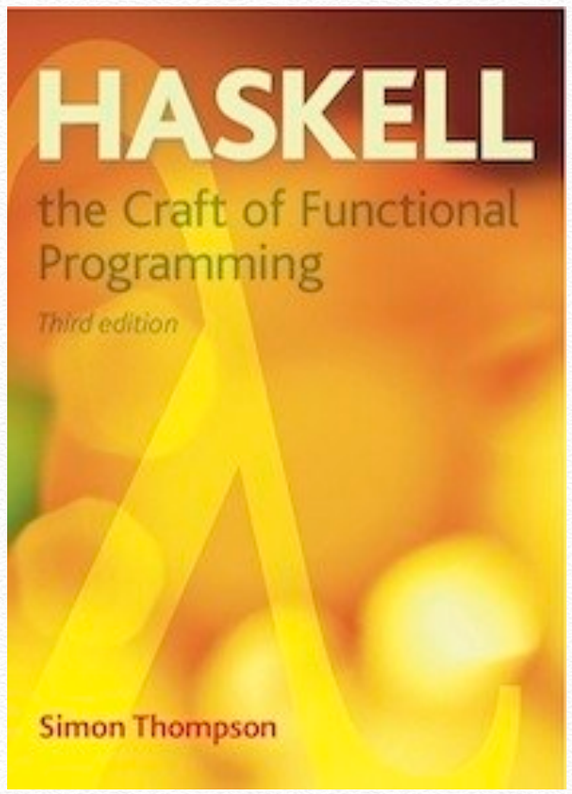
- Instructor
  - Sara Sagaii: sarams@cs.ubc.ca
  - Office Hours:
    - Tuesdays: 10-11am
    - Thursdays: 9:30-10:30am
    - Office: ICCS 187 (Sessional Instructors Office)
- Teaching Assistants:
  - Rui Ge <rge@cs.ubc.ca>
  - Susanne Bradley <smbrad@cs.ubc.ca>
  - Khurram Ali <khurram.ali@alumni.ubc.ca>

# Textbooks

- The Art of Prolog (second edition), by Leon Sterling and Ehud Shapiro, MIT Press.




- Haskell: The Craft of Functional Programming, by Simon Thompson, Addison-Wesley. (Third Edition)



# Course Material

- Piazza is where these lectures*, assignments, and everything else will be shared:

piazza.com/ubc.ca/winterterm12015/cpsc312

- No Course Website. Kurt Eiselt's Website:

http://www.techweenie.org/cpsc-312.html

- The Syllabus will also be on Piazza soon.

*I tend to only include the key points in my slides, so if you want to keep track of everything I say, you need to take your own notes

# (Tentative) Grading Scheme

| | | |
|---|---|---|
| 5 Assignments | 15% | |
| 2 Projects | 30% | |
| Midterm exam | 20% | -- early to mid-October |
| Final exam | 35% | |

Assignments are done individually.
Projects in pairs:

Both students must make a substantial programming contribution to the project and be able to discuss the contribution at length if asked.

Late submissions will not be marked.

# Questions

# What This Course is About

- We will explore the structure and logic of computation through two models of (declarative) programming:
  - *logic programming* (first half of the course)
  - *functional programming* (second half of the course)
- We will concentrate on *the process of problem-solving* and learn to think outside the imperative box.

# Why?

- Why learn a logic or functional programming language instead of yet another imperative (e.g., C) or object-oriented (e.g., Java, C++) language?

- Jobs. Yes, Jobs, but only for the few lucky ones who pass this course.
- Prolog -- popular in AI.
- You are your tools:

*"When All You Have Is a Hammer*

*Everything Looks Like a Nail"*

*(Abraham Maslow)*

# Why?

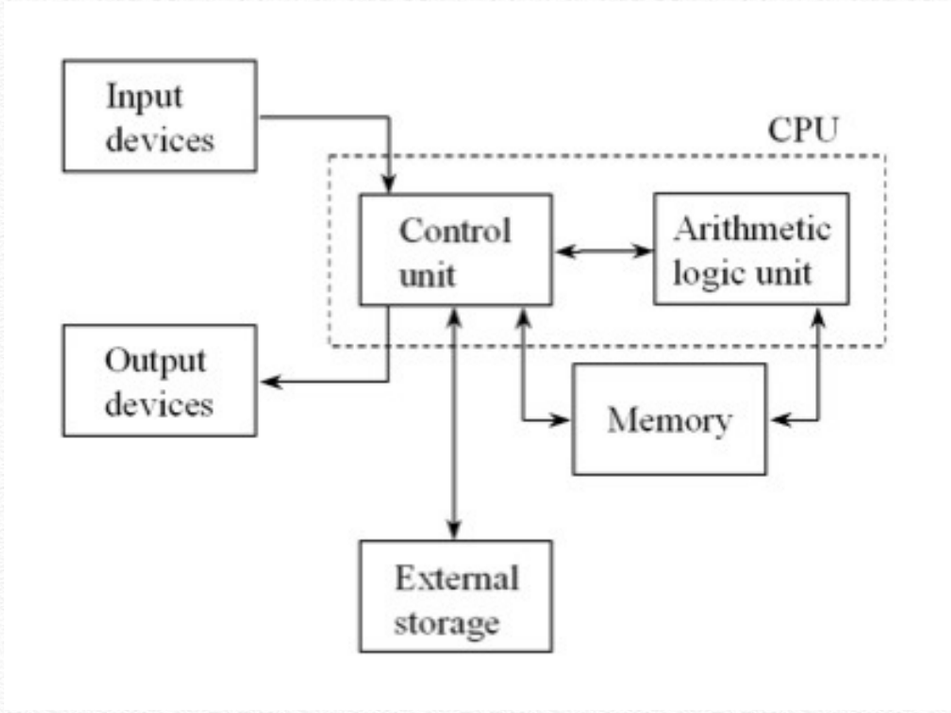
- When all you have is Java or C, everything looks like a problem to be solved by moving things in and out of memory locations.

- Imperative languages like C and object-oriented languages like Java and C++ encourage/require the programmer to map solutions to problems onto the traditional von Neumann architecture.

# Von Neumann Architecture

- The von Neumann architecture might be a great model for implementing a digital computer, but that doesn't mean it's the best perspective from which to perceive potential solutions to programming problems.

- The OO languages add bells and whistles like classes, objects, inheritance, and polymorphism, but ultimately you're still tied to the von Neumann framework.



# Why?

- Functional programming languages (such as Haskell, ML, Miranda, Lisp, Clojure, Erlang*) allow the programmer to map her solution onto a collection of math-like functions (which manipulate symbols as well as numbers).

- The gory details of the underlying von Neumann architecture are abstracted away, leading to greater clarity in the problem solving process and more elegant solutions.

*we will not deal with these languages, but it's important that you know many other ones languages exist and some are in somewhat popular use.

# Why?

- The functional programmer creates programs consisting of functions that compute with values passed through parameters and return the newly- computed values. Like math functions, Haskell functions have no mutable state...there are no side effects, no modifiable variables, no assignment statements.

- That's a long way from imperative or object-oriented programming.

# Program State

"the state of a program is the binding of all active objects to their current values. The state has two maps: 1) the pairing of active objects with specific memory locations, 2) and the pairing of active memory locations with their current values.

The current statement to be executed in a program is interpreted relative to the current state. The individual steps that occur during a program run can be viewed as a series of transformations. The final state of the program happens when the ideal value or values are written to the right memory cells, which are then bound to the right program objects."

*Programming Language: Principles and Paradigms  by Allen Tucker, Robert Noonan*

# State is BAD

- "State is any data the program stores that can possibly be changed by more than one piece of code. It is dangerous because if the code's behavior is dependent on a piece of state, it is impossible to analyze what it might do without taking into account all the possible values of that state, as well as every other part of the program that might modify that state. This problem is exponentially magnified in parallel programs, where it is not always easy to tell even what order code will execute in. It becomes nearly impossible to predict what a given state might be."

Practical Clojure, VanderHart and Sierra, p. 5. Springer-Verlag (Apress), 2010.

# Side Effects

- "Side effects are anything a function does when it is executed, besides just returning a value. If it changes program state, writes to a hard disk, or performs any kind of IO, it has executed a side effect. Of course, side effects are necessary. But they also make a function much more difficult to understand and to reuse in different contexts."

Practical Clojure, VanderHart and Sierra, p. 5. Springer-Verlag (Apress), 2010.

# Why?

- Logic programming languages (Prolog and its dialects being the dominant example, but see also Mercury and Oz) allow the programmer to map her solution onto a collection of logical declarations or assertions.

- The gory details of the underlying von Neumann architecture are abstracted away, leading to greater clarity in the problem solving process and more elegant solutions.

# Why?

- The Prolog programmer creates programs consisting of only logical facts and rules that allow new facts to be generated from existing facts. The Prolog programmer tells Prolog what to do, but not how to do it...Prolog already knows how.
- That's even further from imperative or object-oriented programming...much further.
- In short, one good reason for learning these languages is to break away from traditional problem solving approaches.

# Why?

- Moore's law won't apply for much longer. The doubling of transistor density on integrated circuits every 24 (or 18) months is limited by physics.
- More transistors require more power and generate more heat. Shrinking the circuits leads to electron leakage from one circuit to another. Circuits may shrink, but electrons can't.
- But a technology-based economy is dependent upon continued increases in processing power..
- Functional and logic languages may be better suited to programming in a multiple processor environment than imperative languages (more on this later).
- They are also closer to the way human brain operates.

# Why?

Lots of little reasons: You might...

- fall in love with one of these languages.
- see some of their ideas in mainstream languages.
- force some ideas into mainstream languages.
- implement some techniques within mainstream languages.
- become a programming languages researcher or developer (functional languages are now sexy).
- take CPSC 311.
- take CPSC 422.
- work for an employer who has seen the light.
- find your own reasons.

# WHY?

- On a small piece of paper, write:
  - Your name
  - Why you are taking this course.
  - Have you taken any courses similar or related to this course?
  - Are you currently registered in CPSC322?
  - Anything else you would like to tell me about yourself.

# A word of caution?

- Please note that the course title is "Functional and Logic **Programming**"
- This course will challenge your mental models of programming. Wrestling with current mental models while building new ones is never easy. Doing lots of programming will help...more programming than you'll be assigned as homework. If you don't like programming, you won't do the extra work.
- This is a fun and not too difficult course if you like to solve problems and write programs. We try to go slow enough so that a third-year CS major can keep up. Yet every year, some students fail the course and others just barely pass.
- Keep up, do the work, do more if you need to.

# A word of wisdom

- "A language that doesn't affect the way you think about programming is not worth knowing."

Alan Perlis

# Next Class

- Download SWI-Prolog from

http://www.swi-prolog.org/download/stable

- For next class: Read Chapter 1 from the Art of Prolog
- Stay tuned for Piazza page and course syllabus