CS 312
Data Structures and Functional Programming

Radu Rugina
Fall 2006

Course staff

- Prof. Radu Rugina
- Two TAs:
  - Bruno Abrahao
  - Olga Belomestnykh
- Consultants:
  - Ben Weber
  - Edward McTighe
  - Kareem Amin
  - Paul Lewellen
  - Tyler Steele
- Office, consulting hours posted on web
- One hour of consulting Mon, Tue, Thu evening
- TAs, instructor have office hours : use them!

Course meetings

- Lectures Tuesday, Thursday : Kimball B11
- Recitations Monday, Wednesday
  - Snee Hall 1120, at 2:30pm
  - Hollister Hall 314, at 3:35pm
  - TBA
- New material is presented in lecture *and* recitation
- Attendance is expected at recitation and lecture
- Participation counts

Course web site

http://www.cs.cornell.edu/courses/cs312

- Announcements
- Lecture notes
- Assignments
- Course software
- ML documentation
- Other resources

Course newsgroup

cornell.class.cs312

- A great place to ask questions!
- A great place to see if your question has already been asked
- (But don't give information about your solutions)

Readings

- Course material in lecture notes on website
  - But also responsible for in-class material…
- Some other useful texts:
  - *Elements of ML Programming*, Ullman
  - *ML for the working programmer*, Paulson
  - *Programming in Standard ML*, Harper (on-line)
  - *Notes on Programming in SML*, Pucella (on-line)
  - *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*. Liskov, Guttag.
    - Material on abstraction and specification, but in Java

Assignments

- 6 problem sets
  - PS1 assigned today!
- Programming and written problems
- Submitted electronically via CMS
- Three single-person assignments
- Three two-person assignments
- Final Project

Exams

- Prelim 1: October 12
- Prelim 2: November 14
- Makeup exams must be scheduled within the first two weeks of class
  - Check your schedule and let the instructor know
- No final exam
- Final project due during final exam period

Academic integrity

- Strictly enforced
- Please don't make us spend time on this
- Start assignments early and get help from course staff!

Course Contents

What this course is about

Three aspects of **Computation**

1) Paradigms	2) Techniques	3) Structures
Learn about new programming language concepts, and constructs	Learn about how to better design, reason about, and analyze programs	Learn about new data structures and algorithms

Why do you need to know this?

- Science and craft of programming
  …what you didn't learn in 211
- *You'll acquire invaluable skills that will help you become better programmers*
- Needed in many upper level courses
- Needed for any serious programming task
- Needed for managing programming projects

1) Programming Paradigms

- Functional programming
- Polymorphism
- Pattern matching
- Module systems
- Concurrent programming
- Type inference
- Garbage collection
- We'll use ML to convey these concepts
  - You'll need to learn ML
  - The important part are the concepts, not the ML syntax!

2) Programming Techniques

- *We will stress the importance of design and reasoning upon the development of robust, trustworthy software systems.*
- Design and planning:
  - Modular programming
  - Data abstraction
  - Specifications, interfaces
- Reasoning about programs
  - Program execution models
  - Reason about program correctness
  - Asymptotic complexity
  - Using induction to reason about program behavior
- Testing

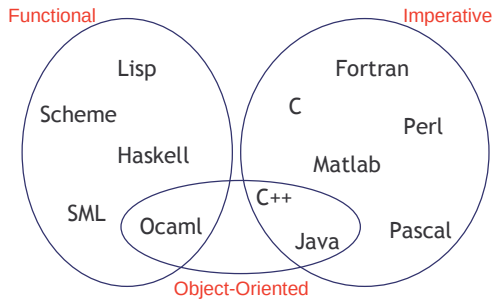
3) Data Structures & Algorithms

- Standard structures: lists, trees, stacks, graphs, etc.
  - Functional versions of these structures
- Advanced structures:
  - Balanced trees: AVL, Red-Black, B-trees
  - Hash tables
  - Binary heaps
- Algorithms on these data structures

Common Misconception

- "This course is useless; ML is not as widely used as C/C++/Java!"
- Answer:
  - The course is not about ML. It is about *paradigms* present in ML, plus many other things. *You're here to learn general principles and concepts.*
  - Many of concepts are applicable to C/C++/Java.
  - Learning new concepts makes your mind more flexible. You'll be able to learn new languages quicker.
  - Languages are constantly evolving. Who knows what tomorrow's languages will be? You'll be more prepared to those changes.

Programming Languages Map



Imperative vs. functional

- ML: a *functional* programming language
  - Encourages building code out of functions
  - Like mathematical functions; f(x) always gives the same result
- Opposite: *imperative* programming language
  - E.g., FORTRAN, Java
  - Imperative = execution is a sequence of commands that change the program's state

    x = x + 1; y = y + x; …
- Functional style usable in ML, Java, C, …
  - No side effects: easier to reason about what happens
  - Equational reasoning

Imperative vs. functional

- Functional languages:
  - Higher level of abstraction
  - Closer to specification
  - Easier to develop robust software
- Imperative languages:
  - Lower level of abstraction
  - Sometimes more efficient
  - More difficult to maintain, debug
  - More error-prone

Example 1: Sum

```
y = 0;
for (x = 1; x < n; x++) {
  y = y + x*x;
}
```


Example 1: Sum

```
y = 0;
for (x = 1; x < n; x++) {
  y = y + x*x;
}


fun sum(n: int): int =
  if n=0 then 0 else n*n + sum(n-1)
```


Example 2: Reverse

```
List reverse(List x) {
  List y = null;
  while (x != null) {
     List t = x.next;
     x.next = y;
     y = x;
     x = t;
  }
  return y;
}
```


Example 2: Reverse

```
fun reverse(l : int list) : int list =
  case l of
     [] => []
   | h :: t => reverse(t) @ [h]
```


Announcements

- Problem set 1 is released
  - Due September 6, at 11:00pm
  - Posted on the course web site.
- Consulting Monday and Tuesday next week:
  - *Help session*: on installing and using SML + Emacs
  - A brief introduction to the ML Basis Library
  - Both sessions in Upson B17