CS 3110

Lecture 1: Course Overview

Profs. Clarkson & George
Spring 2015

Today’s music: Prelude from Final Fantasy VII by Nobuo Uematsu (remastered by Sean Schafianski)
Welcome!

We have 15 weeks to finish your university education as a programmer

- Programming isn’t hard
- Programming well is very hard
  - High variance among professionals’ productivity: 10x or more
  - With hard work, patience, and an open mind, this course makes you a better programmer
Evolution

- **CS 1110**: Write code for your professor

- **CS 2110**: Write code for yourself

- **CS 3110**: Write code for others
  - Emphasis on design, performance, correctness
  - Also, with others: techniques and tools for collaboration
  - **MAIN GOALS for this course:** write code for and with others
How we’ll approach goals

1. Functional programming (OCaml)
   – Challenge you to think outside the Python/Java imperative family of languages
   – Realize that programming transcends programming in a language
     • Language features: syntax, semantics, idioms, tools
How we’ll approach goals

2. Data structures and modern programming paradigms
   – Challenge you to think about abstraction
   – Rigorously analyze performance and correctness
   – Learn to write concurrent programs
   – Learn to write scalable programs
How we’ll approach goals

3. Software engineering
   – Experience with modular design, specification, integrated testing, source control, code reviews
   – Expose you to tools used in the real world (git, Linux)
Challenges in our way

You might think *programming = Java*

- For the next five weeks, please *let go of Java*
- Learn OCaml as a totally new way of programming
- Thinking “Oh, that’s like this thing in Java” will confuse you, slow you down, make you learn less
Challenges in our way

You might think programming = hack until it works

– As you begin this semester, please develop the mindset of a professional: disciplined work habits

– Common challenge: type first, think later
  • “A year in the lab saves an hour at the library”
  • **Fact:** there is an infinite number of incorrect programs
  • **Corollary:** tweaking your code is unlikely to help
  • ...we hope you’ll think first and type less
Keep the end in sight

We want to help you learn to write code that is

– Reliable
– Efficient
– Readable
– Testable
– Provable
– Maintainable
– BEAUTIFUL
OCaml

A pretty good language for writing beautiful programs

O = Objective, Caml=not important
ML is a family of languages; originally the “meta-language” for a tool
OCaml is awesome because of...

• **Immutable programming**
  – Variable’s values cannot destructively be changed; makes reasoning about program easier!

• **Algebraic datatypes and pattern matching**
  – Makes definition and manipulation of complex data structures easy to express

• **First-class functions**
  – Functions can be passed around like ordinary values

• **Static type-checking**
  – Reduce number of run-time errors

• **Automatic type inference**
  – No burden to write down types of every single variable

• **Parametric polymorphism**
  – Enables construction of abstractions that work across many data types

• **Garbage collection**
  – Automated memory management eliminates many run-time errors
Why immutability?

Imperative (mutable) programming:

• *commands* specify **how to compute** by destructively changing *state*
  - \( x = x + 1; \)
  - \( a[i] = 42; \)
  - \( p.\text{next} = p.\text{next}.\text{next}; \)

• and functions/methods have **side effects**
  - `int wheels(Vehicle v) {
      v.size++; return v.numWheels;
    }`
Why immutability?

The fantasy of mutability:

– There is a single state
– The computer does one thing at a time

The reality of mutability:

– There is no single state
  • Programs have many threads, spread across many cores, spread across many processors, spread across many computers... each with its own view of memory
– There is no single program
  • Most applications do many things at one time

...mutable programming is not well-suited to modern computing!
Why immutability?

Functional (immutable) programming: expressions specify what to compute
- Variables never change value
- Functions never have side effects

The reality of immutability:
- No need to think about state
- Powerful ways to build concurrent programs
Functional vs. imperative

**Functional languages:**
- Higher level of abstraction
- Easier to develop robust software

**Imperative languages:**
- Lower level of abstraction
- Harder to develop robust software

You don’t have to believe me now. You will by the end of the course. 😊
Functional languages predict the future

Dismissed as “beautiful, worthless, slow things professors make you learn in school”:

- **Garbage collection**
  Java [1995], LISP [1958]

- **Generics**
  Java 5 [2004], ML [1990]

- **Higher-order functions**
  C#3.0 [2007], Java 8 [2014], LISP [1958]

- **Type inference**
  C++11 [2011], Java 7 [2011] and 8, ML
Functional languages matter in the real world

- F#, C# 3.0, LINQ (Microsoft)
- Scala (Twitter, LinkedIn, FourSquare)
- Java 8
- Haskell (dozens of small companies/teams)
- Erlang (distributed systems, Facebook chat)
- OCaml (Jane Street)
A GLIMPSE OF OCAML...
Example 1: Sum Squares

// yields \( \sum_{1 \leq i \leq n} i^2 \)
int sum_squares(int n) {
    sum = 0;
    for (int x = 1; x <= n; x++) {
        sum = sum + x*x
    }
    return sum;
}

How can you do that without mutability?
Example 1: Sum Squares

// yields $\sum_{1 \leq i \leq n} i^2$

```c
int sum_squares(int n) {
    if (n == 0) {
        return 0;
    } else {
        return n*n + sum_squares(n-1);
    }
}
```


Example 1: Sum Squares

(* yields $\sum_{1\leq i\leq n} i^2$*)

```ocaml
let rec sum_squares (n:int) : int =
  if n=0 then 0
  else n*n + sum_squares (n-1)
```

Better yet...

(* yields $\sum_{1\leq i\leq n} i^2$*)

```ocaml
let rec sum_squares n =
  if n=0 then 0
  else n*n + sum_squares (n-1)
```
Example 2: Reverse List

// return a copy of x, 
// with the order of its elements reversed
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 List

(* return the reverse of list *)

let rec reverse lst =
  match lst with
  | [] -> []
  | h::t -> (reverse t) @ [h]
Example 3: Quicksort

• Describe quicksort in English.
• Describe quicksort in Java. (No.)
• Describe quicksort in OCaml:

(* returns lst sorted according to < *)
def qsort(lst):
    match lst with
    | [] -> []
    | pivot::rest -> (* poor choice of pivot *)
        let (left,right) = partition ((<) pivot) rest
        in (qsort left) @ [pivot] @ (qsort right)
THE SYLLABUS
Course staff

Instructors:
• Michael Clarkson
  – PhD 2010 Cornell University
• Mike George
  – ABD Cornell University
• Research areas: security and programming languages
• We go by “Prof. Clarkson” and "Prof. George" in this course

TAs and consultants: 28 at last count
  – Course administrator (“head TA”): Remy Jette (rcj57)
Course meetings

• **Lectures:** TR 10:10-11:00 am
  – Attendance is expected and will be checked
  – If you miss, get notes from another student

• **Recitations:** mostly MW
  – Attendance is expected
  – TR sections are effectively MW delayed one day
  – You may attend any, regardless of registration, subject to room capacity; best to stick to one

• **Consulting:** coverage every day except Monday
Course website

http://www.cs.cornell.edu/Courses/cs3110/2015sp/

• Full syllabus *(required reading)*
• Lecture and recitation notes
  – Typically go live the night after the lecture
  – Supplement, do not replace, attendance
Piazza

- Online discussion forum
- Primary vehicle for announcements
  - Set up email notifications now
- Monitored almost continuously by staff
- Ask for help, don’t post solutions
- Post **anonymously** (to classmates)
- Post **privately** (only seen by staff)
Email

• If it's about content (e.g., you have a question about a homework problem), post a message on Piazza instead
  – Messages restricted only to instructors probably will get lost

• If it's about your own personal logistics (e.g., you need to leave town for a funeral), send email to cs3110-instructors-L@cornell.edu

• But better than that...come talk to us in person!
CMS

• Course Management System
• Grades, regrades, materials we don’t want to post publicly
• **Make sure you have access to CS 3110 now**
  – If not, notify Course Administrator and provide your full name and NetID
• Gets overloaded at due time; **submit early**
Course materials

• No textbook
  – Online course notes
  – If you want a book, *Real World OCaml* is good and written 2/3 by Cornellians
  – Other free resources linked on website

• i>clickers
  – Required; will be used to take attendance
  – Buy one at Cornell Store
  – Will not use i>clicker GO app in this course
  – We start using them on Tuesday in lecture
Problem Sets

Five problem sets (PS’s)

– Plus an ungraded, uncollected PS0
– Usually soft deadline on Thursdays at 11:59 pm, followed by hard deadline 48 hours later
  • 25% late penalty after soft deadline
– Electronic submission by CMS, never by email
– Length of time usually about 1.5 weeks
– First individual, then pairs, then small teams
Exams

• Two prelims
  – Prelim 1: 03/10/15
  – Prelim 2: 04/21/15
  – Put them on your calendar now
  – Two offerings each night: 5:15-7:15 and 7:30-9:30
    • No other makeups will be offered
    • If you miss without advance permission, you get a zero

• Final
  – University will announce date and time later
Grading

• Problem sets: 40%
• Prelims: 15% each
• Final: 25%
• Other factors: 5%

Historical median grade: B/B+ range
**Academic integrity**

- You are bound by the Cornell Code of Academic Integrity and the CS Department Code of Academic Integrity
  - Both linked from course website
  - You are responsible for understanding them
- In 3110, you may not share code with anyone
  - (except partner or teammates)
- In 3110, you may not copy code from online sources
- If you have a question about what is or is not allowed, please ask
- The course staff uses automated software to detect cheating. **It works.**
Upcoming events

• [today] PS 0 is out now
  – Start by getting OCaml installed and working
  – We provide a virtual machine (VM) that makes this relatively easy
• [Saturday] OCaml Install Session
  – Upson 315, noon to 8 pm
  – Drop in at any time if you need help with VM or OCaml
• [Monday] Recitations begin (none today)
• [Tuesday am] i>clickers start in lecture
• [Tuesday pm] Consulting hours and office hours start
  – Drop by Profs offices this afternoon if you need something immediately
  – Regular hours will be posted on Piazza next Tuesday

...why are you still here? Get to work! 😊

THIS IS 3110