CSci 2041: Advanced Programming Principles
Administrative Issues and Course Introduction

Gopalan Nadathur

Department of Computer Science and Engineering
University of Minnesota

Lectures in Spring 2015

People Involved

- *Instructor*

  Name: Gopalan Nadathur Office: Keller Hall 6-215
  Email: `gopalan@cs.umn.edu`
  Hours: Mon, Fri 14:30 - 15:30

- *Graduate Teaching Assistants*

  Name: Dan DaCosta and Mary Southern
  Email: `{dacosta,marys}@cs.umn.edu`
  Office: Keller Hall 2-209
  Hours: TBD

- *Undergraduate Teaching Assistants*

  Kesha Hietala, Lucas Meyers, William Haltom
  Daniel Pederson, Connor McMahon, Nipun Parasrampuria

  8-10 office hours, time to be decided

Do not use email except for personal matters

Electronic Resources

- Course Web Page

  `http://www-users.cselabs.umn.edu/classes/Spring-2015/csci2041/`

- Resources available from the course web page:
  - Forum bulletin boards
  - Assignments, Manuals, Handouts, Papers, Lecture slides
  - Information between lectures

  Accessing protected area:
  Id: app2014, Password: subl!ime
- Github Resources
  - Lab writeups, assignments (maybe)
  - submission repositories for labs and assignments

Other Resources

- Course Text?

  No official textbook (none exists, really)

  However, we will periodically put up readings, links to online material, notes specific to this course, etc

- Lab Component on Tuesdays
  - Intended to build familiarity with concepts and programming environment in preparation for independent work
  - May include work to turn in but primarily to assess participation
  - Your input on what would be useful to do in labs?

Course Prerequisites

- *Formal Prerequisite*: CSci 1913 or 1933 and CSci 2011
- *Conceptual Meaning of the Prerequisites*:
  - Good grasp of programming fundamentals
  - Exposure to basic data structures like lists and trees
  - Understanding of “computer science” math
    - Knowledge of functions, relations, etc
    - Exposure to recursion and induction
    - Understanding of logical reasoning and proofs
- *A bonus for enjoyment:*

  Enthusiasm for programming and a desire to think about it in new and unexpected ways

Required Work

- Attending lectures and completing readings

  No direct contribution to grade by still extremely important to doing well in the parts that count
- Attending lab sessions (7.5% of the grade)
- Completing **all** homeworks

  Must show sufficient effort in every homework to pass

  Accounts for 45% of the grade overall
- Participating in forum and class discussions (2.5%)
- Taking two midterm and one final closed book exams
  - Midterms on Feb 23 and April 6 (in class)
  - Final on May 15, 08:00–10:00

  Exams count for 10%, 10% and 25%, respectively

Grades will be based on a curve, 55% or more needed to pass

Lateness and Grading Policy

- There will be *no* written makeup exam
  Oral exam for an exam missed for an excusable reason
- No credit for late homeworks
  You must still submit them to pass the course
  Excusable lateness will be made up in some other way
- Homeworks must adhere to submission protocols
  No credit for programming problems if our tools fail on them
- Form and content in homework and exams matter!
  - Program structure is more important than input-output correctness
  - How you express yourself in written answers counts
  - Legibility is *essential* for credit
- Grade issues must be resolved in a two-week window

Academic Honesty

Discussions about course material in class or outside class are *strongly* encouraged

However, any work submitted for a grade *must be independently done*

In particular

- consultations on solutions prior to submission is illegal
- material obtained from an external source must be explicitly acknowledged with associated loss in credit

A breach of these requirements constitutes *academic dishonesty* that will be reported and will be severely penalized

Resist the temptation to cheat: *you will be caught and it also does not help you*

You are culpable even if you only enable cheating and you too will draw punishment

Decorum in the Classroom

The guiding principle:

> *Avoid doing anything that unnecessarily distracts from the central purpose of learning*

Some initial rules in keeping with this idea:

- No laptop or cell phone use in class
- If you come in late, settle in as unobtrusively as possible
- If you have to turn in something, do it *before* the lecture starts or *after* the lecture
- Avoid private conversations except as required

These rules may be refined over time

One distraction that is *not* in this category: asking questions!

Reading Assignments

- Make sure to familiarize yourself with github (first lab, assignment 0)
- Read and work through the notes I have created for getting started with OCaml (first lab)
- Read the “popular computer science” paper with title *OCaml for the Masses*
- Start reading the first four chapters of *Introduction to Objective Caml*—skip Section 3.3 though
- Read also the section entitled *The Core Language* in the OCaml online manual

Objectives for this Course

We will be covering three separate but interrelated themes:

- advanced principles that should guide us in programming

  In particular, we will discuss
  - ways in which to structure the process of programming and the programs we write
  - ways in which to use such structure in reasoning about our programs
  - advanced language level tools that help in both writing and reasoning about programs

- functional programming as a means for putting such principles into practice

- the OCaml language as a particular brand of functional programming

What Kinds of Advanced Principles Anyway?

Some of the themes we will develop during the term

- organizing programs around *data values* to be manipulated rather than their *representation*
- using rich types to structure such programs

  types as a means for visualizing data, constructing programs and reasoning about them
- treating programs in almost the same way as we treat data
- new programming techniques based on evaluation strategies
- logical methods for building programs from pieces
- different computing approaches: search-based, parallel, etc

This is just a flavour and the handout has more

Why Functional Programming?

This is an approach to programming that allows us to discuss the advanced principles easily

For example

- it treats data values not data representations
- it is based on expression evaluation and hence allows experimentation with evaluation strategies
- it treats functions in almost the same way as data
- expression evaluation and higher-order features can be exploited to support many powerful programming styles
- many functional languages treat cutting-edge mechanisms like rich types, modularity, etc

Such languages are *real*: e.g., see the Yaron Minsky article

Moreover, what we learn in this setting can be transferred later to other languages

Why OCaml?

A starting point: OCaml is a functional language

Beyond this, it is a natural choice for the following reasons:

- it is a language with a rich and useful type system
- it is a well-engineered system

  its implementations compete for efficiency with ones for lower-level languages, support interoperability, etc
- it is a well-supported system

  INRIA in France, Jane Street, a large community of academic and industrial users
- it mixes useful mainstream language ideas with advanced functional ones
- it includes modularity features

My hope: OCaml will become a language that *you* really use in applications in the future