CMPT 440/821
Advanced Topics in Programming Languages

Christopher Dutchyn

January 3, 2017

Programmers spend a lot of time understanding and improving their tools: editors, profilers, debuggers, and so forth. Technical magazines sometimes call to mind stores that sell outdoor gear: It's a rough world out there, you need all the equipment and gadgetry you can get. You, too, may have stared in admiration and longing at a particularly powerful syntax highlighter at some point in time.

Often lost in this analysis is a proper understanding of what tools and technologies can have the greatest impact. Irrespective of how you choose to write code and where you might run it, perhaps the single most important technology is the programming language itself. Languages both enable solutions and inhibit them; they save time and waste it; and most importantly, they either expand or contract our imagination. Yet how much have you thought about this, and how well do you understand the issues?

Whereas prior courses may have taught you how to program, this course teaches you how to analyze programming languages. What are the questions one asks when confronting a new language? What intellectual tools do we have for studying languages? What does a language designer need to know? How can we implement new languages? You should have much better answers to these questions when we're done than I expect you have now.

A major difference between this course and ones with a similar title at other universities is that we will use a much better way of classifying languages. In particular, we will move past clichéd and relatively useless divisions such as *functional*, *object-oriented* and *imperative*. We will instead decompose languages into building blocks, and understand these building blocks in depth. The goal is to give you a richer verbal and intellectual vocabulary so that, when you are confronted with a new language, you have a broad set of concepts, each of which you understand well, to use to dissect the language.

As a model, we will attempt to reconstruct the Go language, including exceptions, nominal and structural typing, goroutines, modules, and other features. But, the emphasis will be on the building blocks of languages in general.

1 Staff

Instructor: Chris Dutchyn

email mailto:dutchyn@cs.usask.ca

web http://www.cs.usask.ca/faculty/cjd032

office Thorvaldson 178.2

Marker: Taher Ghaleb

email mailto:tag263@mail.usask.ca

2 Meetings

We will be meeting in room **Arts 105**, on Mondays, Wednesdays, and Fridays, from 10:30 am to 11:20 am.

First instructional day is January 4, 2017. Last instructional day is April 5, 2017. Spring Break will disrupt classes: no lectures will be held on February 20–24, 2017.

The midterm for undergraduate students, will occur on March 1, during class time.

The final exam date will be scheduled by central timetabling, but will occur during the April 7–29, 2017 window.

Instructor office hours are Wednesdays, from 14:00–16:00. If these times do not suit, please contact the instructor to schedule a specific appointment. The TA will offer tutorial assistance in the Spinks help centre, according to their schedule.

3 Grading

Intangibles may count in the determination of your grade. Regular attendance and productive classroom participation may slightly ameliorate some weaknesses elsewhere; the converse is also true.

3.1 Undergraduate

Item	Description	Weighting
Midterm Examination	(in-class, 80 minutes)	15%
Assignments	(six)	60%
Final Examination	(180 minutes)	25%
Total		100%

Table 1: CMPT 440 (undergraduate) Marking

3.1.1 Assignments

There is one warm-up assignment (5%) and four substantive assignments (10% each), and a final completed interpreter (15%), due approximately every two weeks. In the main, they encompass completing the implementations of the various interpreter features explored in class. The assignment topics and *approximate* due dates are:

1. **Scheme** – *Jan. 16*: a number of exercises on designing data and writing recursive functional programs.
2. **Environment-Passing** – *Jan. 30*: our higher-order, procedural programming language; roughly equivalent to LETREC in Scheme.
3. **Store-Passing** – *Feb. 16*: adding mutable state.
4. **Continuation-Passing** – *Mar. 20*: adding control effects.
5. **Types** – *Apr. 6*: adding type-checking.
6. **Modules** – *Apr. 20*: completed full interpreter.

3.2 Graduate

Graduate students will not have examinations, but will need to read and present a research paper, complete a larger-scale project, and will have larger-scale assignments.

3.2.1 Paper

Graduate students will need to read a recent research paper in programming languages, submit a five-page written précis of it, and make a short (five–ten minute) presentation. Depending on how many graduate students attend, presentations may be scheduled outside of lecture hours. Topics for the paper will be chosen to complement the student's interest, in consultation with the instructor.

Item	Description	Weighting
Assignments	(six)	65 %
Research Paper	précis	10%
	presentation	5%
Final Examination		20%
Total		100%

Table 2: CMPT 821 (graduate) Marking

3.2.2 Assignments

There are six substantive assignments (5%, 10%, 10%, 10%, 10%, 20%), due on the same schedule for the undergraduate students. They correspond to the undergraduate assignments, except at a larger scale. Graduate students will also need to implement a language extension as part of the last assignment beyond the undergraduate level. A variety of potential topics exist, including

- imperative or trampolined interpreter for a more mainstream language,
- guarded algebraic datatypes with inference
- type-checked objects (including generics)
- modules and functors
- partial evaluation and just-in-time compilation

Graduate students are recomended to begin thinking about their project early in the term; because, a written description must be submitted for approval by the instructor, no later than *March 1*. Please consult with the instructor in order to complete this description.

3.3 Late Homework

We will not accept late assignments. Assignments are often timed to be due for a classroom discussion on the assigned material, because you can better follow a difficult topic if your struggles are fresh in your head. The class will sometimes even discuss solutions to the homework problems. Once we do this, we can no longer accept your solution.

4 Textbooks

4.1 Recommended

- *The Scheme Programming Language*, 4ed (Dybvig – MIT Press, 2009) available online at `http://www.scheme.com/tspl4`
- *The Go Programming Language*, (Donavan and Kernighan – Addison Wesley, 2015).

4.2 Supplemental

- *Programming Languages: Application and Interpretation* (Krishnamurthi – 2012) available online at `http://www.cs.brown.edu/∼sk/Publications/Books/ProgLangs`
- *Essentials of Programming Languages*, 3ed (Friedman and Wand – MIT Press, 2008)
- *Lisp in Small Pieces* (Quiennec – MIT Press, 1996)
- *Programming Languages and Lambda Calculi* (Flatt and Felleisen – 2006) available online at `http://www.cs.utah.edu/plt/publications/pllc.pdf`
- *Structure and Interpretation of Computer Programs*, 2ed (Abelson and Sussman – MIT Press, 1996) available online at `http://mitpress.mit.edu/sicp`
- *The Little Schemer*, 4ed (Friedman and Felleisen – MIT Press, 1996)
- *The Seasoned Schemer* (Friedman and Felleisen – MIT Press, 1996)
- *The Reasoned Schemer* (Friedman, Bird, Kiselyov – MIT Press, 2006)
- *The Art of the Metaobject Protocol* (Kiczales – MIT Press, 1991)
- *ML for the Working Programmer*, 2ed (Paulson – Addison Wesley, 1996)
- *Types and Programming Languages* (Pierce – MIT Press, 2004)
- *Advanced Types and Programming Languages* (Pierce et al – MIT Press, 2004)
- *Practical Foundations of Programming Languages* (Harper – MIT Press, 2015)

5 Software

- Racket v. 6.7, available online at `http://racket-lang.org`.

6 Topics

The order and depth with which the following topics are covered may still be altered.

1. Programming language anatomy
   - Scheme
2. Definitional interpreters
   (a) abstract syntax
      - expressions
      - statements

   (b) parsing

   (c) procedures

   (d) scoping
      - dynamic
      - lexical

   (e) parameter passing variations
      - call-by-value
      - call-by-reference
      - call-by-name
      - laziness

   (f) compilation
      - de Bruijn indexing / lexical addressing
      - partial evaluation

   (g) meta-circularity
3. Continuations
   (a) threads and co-routines

   (b) *web programming*

   (c) exceptions
4. *Garbage collection*
   (a) reference counting

   (b) mark-sweep

   (c) mark-copy

   (d) generational
5. Typing

   (a) varieties
      - nominal / structural
      - dependent
      - pure / monadic
   (b) checking
   (c) inference
   (d) polymorphism
      - ad-hoc
      - parametric
         - predicative / impredicative / polymorphic recursion
      - subtypes

6. Program Structure
   (a) modules
   (b) functors
   (c) objects and classes
   (d) aspects

7. Syntactic extension and macros
   (a) Object-oriented style
   (b) Aspect-oriented style
   (c) Logic style

8. Computational reflection (*if time permits*)
   (a) meta-object protocols
   (b) aspect-oriented programming

7 Summary

The purpose of this course is to give you a deep and practical understanding of programming languages. Deep in the sense that we will study the core semantic elements of modern programming languages. You will be able to understand various programming languages in terms of their constituent features and understand, in a deep way, how those fit together. Practical in the sense that this knowledge will help you, in the future, to make decisions about when and how to use programming-language techniques in systems you build. You will have a basis for making decisions such as when should I use a mini-language? What scripting language should I use here? Why does this language contain one feature and not another? How can I emulate a feature missing from this language?

The course is *implementation oriented*. We will build interpreters for each of the languages we study. The interpreters will be written in Scheme, using a programming style that lets us easily focus on the core semantics of the languages, without having to worry about details of syntax, parsing, and grammars (a focus of CMPT 442). Working with Scheme allows us to rapidly prototype language features and quickly experiment with a number of different variations.

You may be concerned that this course is not going to provide you with skills in any particular commercially-important language, i.e. Java or C or C++ or C#. That is true; this course is going to teach you much more: a practical understanding of the foundations of modern programming languages. With that knowledge you will be able to understand a wide range of languages and their unique features easily:

- constructor classes and monads in Haskell,
- covariance and contravariance in Java generics,
- CLR JIT compilation for C#,
- multiple inheritance and `virtual` in C++, C#, and others,
- resumable exceptions in Smalltalk and Lisp,
- co-routines and threads in Python and many others,
- generators and `yield` in Ruby,
- closures (requested in almost every language) vs. inner classes in Java,
- duck typing
- ...

Then, when the successor to whatever comes out you will be able to quickly understand it, and do the same every five years when new 'hot languages' come out.

- *Jan. 3, 2017: initial release*