CMPT 440/821
Advanced Topics in Programming Languages

Christopher Dutchyn

September 5, 2012

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.

Welcome to CMPT 440/812. We hope you learn a lot in return for the hard work you're going to invest.

From the Course Catalogue:

> CMPT 440.3—2(3L): Advanced Topics in Programming Languages Advanced topics in programming languages will be selected from: programming language design, programming language semantics, code optimization, memory management, garbage collection, closures, functional programming, logic programming, aspect-oriented programming, concurrent programming, history of programming languages, advanced programming language features and their implementation, polymorphic type systems, domain specific languages. *Prerequisites:* CMPT *340.*

> CMPT 821.3—1/2(3L): Advanced Topics in Programming Languages Advanced topics in programming languages will be selected from: programming language design, programming language semantics, code optimization, memory management, garbage collection, closures, functional programming, logic programming, aspect-oriented programming, concurrent programming, history of programming languages, advanced programming language features and their implementation, polymorphic type systems, domain specific languages. *Prerequisites: Open to graduate students in computer science who have at least one undergraduate course (3 credit units) in Programming Languages.*

1 Meetings

We will be meeting in room **Arts 108**, on Tuesdays and Thursdays from 11:30 am to 12:50 pm. First instructional day is September 6, 2012. Last instructional day is December 4, 2012. Holidays will not disrupt classes.

The midterm examination will be take-home, distributed in the October 18 class, and returned October 22. The final exam date will be scheduled by central timetabling, but will occur during the December 7 to December 21, 2012 window.

2 Staff

Instructor: Chris Dutchyn

email mailto:dutchyn@cs.usask.ca

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

office Thorvaldson 178.2

Instructor office hours are Wednesdays, from 10:00 am until noon. If this time does not fit, please contact the instructor to schedule a specific appointment.

3 Evaluation

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 Grading Schemes

The expectations and grading schemes differ between the undergraduate students and the graduate students. In particular, the graduate students have lower emphasis on exams, but will need to complete a coures project and a paper précis.

Item	*Weighting*	*Due Date*
Assignments	five yielding 45%	see topics below
Take-home midterm exam	20%	October 21
Final exam	35%	TBD
Total	100%	

Table 1: CMPT 440 (undergraduate) Marking

Item	*Weighting*	*Due Date*
Assignments	five yielding 45%	see topics below
Paper Précis	5%	November 21
Project	25%	December 20
Final exam	25%	TBD
Total	100%	

Table 2: CMPT 821 (graduate) Marking

3.2 Required Coursework

There are four assignments. In the main, they encompass extending the implementations of the various interpreters explored in class. The assignment topics and *approximate* due dates are:

1. **Scheme (5%)** – *September 19*: a number of exercises from chapters 1 – 2 of the textbook.

2. **Environment-Passing (10%)** – *October 9*: our higher-order, procedural programming language; roughly equivalent to LETREC in the textbook.

3. **Store-Passing (10%)** – *October 24*: adding mutable state, roughly equivalent to exercises 4.22–4.25 in the textbook.

4. **Continuation-Passing (10%)** – *November 7*: adding control effects, `letcc`.

5. **Types (10%)** – *November 28*: adding type-checking.

Paper *graduate only* Graduate students will need to read a recent research paper in programming languages, submit a five-page written précis of it. Topics for the paper will be chosen to complement the student's interest, in consultation with the instructor; usually in support of the project task.

Project *graduate only* Graduate students will also need to design and implement a substantive language extension. A variety of potential topics exist, including

- registerized and trampolined interpreter,
- garbage collection,
- type-checked objects (including generics),
- type-inference for higher-order procedural languages,
- 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 *October 31*. Please consult with the instructor in order to complete this description.

3.3 Attendance

All students are expected to attend all course meetings. Attendance may be taken.

Note: All students must be properly registered in order to attend lectures and receive credit for this course. Please ensure you have registered for both terms.

3.4 Final Exam Scheduling

The Registrar schedules all final examinations, including deferred and supplemental examinations. Students are advised not to make travel arrangements for the exam period until the official exam schedule has been posted.

4 Textbooks

4.1 Required

- Friedman and Wand: *Essentials of Programming Languages*, 3ed.; MIT Press, 2008)

4.2 Recommended

- Dybvig: *The Scheme Programming Language*, 4ed.; MIT Press, 2009. available online at `http://www.scheme.com/tspl4` and in the library.
- Krishnamurthi: *Programming Languages: Application and Interpretation*; 2007. available online at `http://www.cs.brown.edu/~sk/Publications/Books/ProgLangs`

4.3 Supplemental

- Abelson and Sussman; *Structure and Interpretation of Computer Programs*, 2ed.; MIT Press, 1996. available online at `http://mitpress.mit.edu/sicp`
- Flatt and Felleisen: *Programming Languages and Lambda Calculi*, 2006. available online at `http://www.cs.utah.edu/plt/publications/pllc.pdf`
- Friedman, Bird, and Kiselyov: *The Reasoned Schemer*; MIT Press, 2006
- Friedman and Felleisen: *The Little Schemer*, 4ed.; MIT Press, 1996
- Friedman and Felleisen: *The Seasoned Schemer*; MIT Press, 1996
- Harper: *Practical Foundations for Programming Languages*; MIT Press, 2012. available online at `http://www.cs.cmu.edu/ rwh/plbook/book.pdf`
- Kiczales: *The Art of the Metaobject Protocol*; MIT Press, 1991.
- Paulson: *ML for the Working Programmer*, 2ed.; Addison Wesley, 1996
- Pierce: *Types and Programming Languages*; MIT Press, 2004
- Quinnec: *Lisp in Small Pieces*; MIT Press, 1996

4.4 Software

- Racket v. 5.2.1, available online at `http://racket-lang.org`.

September 6	introduction and Scheme I
	A1 assigned
11	Scheme II
13	Interpreters, Parsers, Values
18	Expressions and Commands
19	**A1 due**
20	Primitives, Conditionals, Values
25	Environments, Variables, Scoping, Addressing
	A2 assigned
27	Procedures, Closures (x2), Meta-circularity
October 2	Meta-linguistics, partial evaluation
4	Recursion I: via arguments, combinators
9	Recursion II: direct implemented (x2)
10	**A2 due**
11	Mutation and Store-Passing
	A3 assigned
16	Macros and Monadic Construction
18	Other Monads, `amb`, and Logic Languages
	midterm exam distributed
21	**midterm exam due**
23	Continuations and Continuation-Passing
24	**A3 due**
25	Continuations in Monadic Form
	A4 assigned
30	Threads via Continuations
November 1	Logic Languages via Continuations
6	Types and Abstract Interpretation
7	**A4 due**
8	Type Annotations
	grads: paper selection
13	Type Checking as Theorem Proving
	A5 assigned
15	Type Inference
20	Objects and Classes
22	Object Constructions (x2)
27	Type-checking and Optimizing Objects
28	**A5 due**
29	Aspects
December 4	Summary
5	**grads: précis due**
20	**grads: project due**

Figure 1: Detailed Schedule

5 Policies

5.1 Late and Missed Assignments

Late and missed assignments will not be accepted, and a zero grade entered, absent a compelling reason. Extensions granted for compelling reasons will be individual if the reason is individual (e.g. illness), or apply to the whole class (e.g. university closure).

5.2 Incomplete Course Work and Final Grades

> When a student has not completed the required course work, which includes any assignment or examination including the final examination, by the time of submission of the final grades, they may be granted an extension to permit completion of an assignment, or granted a deferred examination in the case of absence from a final examination. Extensions for the completion of assignments must be approved by the Department Head, or Dean in non-departmentalized Colleges, and may exceed thirty days only in unusual circumstances. The student must apply to the instructor for such an extension and furnish satisfactory reasons for the deficiency. Deferred final examinations are granted as per College policy.
>
> In the interim, the instructor will submit a computed percentile grade for the course which factors in the incomplete course work as a zero, along with a grade comment of INF (Incomplete Failure) if a failing grade. In the case where the instructor has indicated in the course outline that failure to complete the required course work will result in failure in the course, and the student has a computed passing percentile grade, a final grade of 49% will be submitted along with a grade comment of INF (Incomplete Failure).
>
> If an extension is granted and the required assignment is submitted within the allotted time, or if a deferred examination is granted and written in the case of absence from the final examination, the instructor will submit a revised computed final percentage grade. The grade change will replace the previous grade and any grade comment of INF (Incomplete Failure) will be removed.

For provisions governing examinations and grading, students are referred to the University Council Regulations on Examinations section of the Calendar.

5.3 Academic Honesty

Students are expected to be academically honest in all of their scholarly work, including course assignments and examinations. Academic honesty is defined

and described in the Department of Computer Science Statement on Academic Honesty[1] and the University of Saskatchewan Academic Honesty Website[2].

The Student Academic Affairs Committee treats all cases according to the University Policy and has the right to apply strict academic penalties (see `http://www.usask.ca/university_secretary/honesty/academic_misconduct.php`).

6 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,

[1] `http://www.cs.usask.ca/undergrad/honesty.php`
[2] `http://www.usask.ca/honesty`

- closures (requested in almost every language) vs. inner classes in Java,
- type classes in Haskell and Scala,
- ...

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

- *Sept. 5, 2012: initial release*