CS 59: Principles of Programming Languages

Sergey Bratus, Fall 2016

Objectives: To explore features and abstractions of programming languages that make them "modern" and motivate their ongoing development. To understand how these features are implemented in actual interpreters, compilers, and virtual machines. To get a taste for different styles of programming: functional, object-oriented, continuation-passing, "monadic",[1] and others. To look under the hood of language abstractions.

In practical terms, this course should give you some idea why languages like Scala and Clojure—and before them Perl, Python, and Ruby—were sought by the industry, and why leading companies continue developing new languages such as Go and Swift.

Required background: The course assumes that you wrote relatively complex programs in a language like C, C++, Java, Python, or Ruby, and have a good understanding of recursive data structures such as lists and trees. For this reason, CS 10 is a requirement and CS 30 is recommended. Having programmed in LISP, Scheme, ML, Haskell, or other functional languages will be helpful but not required.

You will be expected to either have or quickly develop an understanding of how a language like C is compiled down to machine code, and how that machine code executes on the actual CPU. For this reason, CS 51 is recommended, but if you are familiar with x86 or ARM machine code from other sources, or took a class in compilers, that will be sufficient.

Expect to do as much (or more) looking at how abstractions are implemented as coding.

The X-hour: As a rule, we will **not** be using the X-hour for lectures, but may use it for optional meetings (e.g., to catch up on a background subject). Each use of the X-hour will be announced on the class mailing list.

Office Hours: My office is 065 Sudikoff. Please email me for an appointment.

Grades: 20% homework, 40% midterm, 40% final project.

Homework will be given, but its primary purpose is for you to check your understanding of the material and fluency with new languages. The midterm will be take-home and hard; do not expect to develop the required fluency overnight!

Software and Hardware: Linux or MacOS preferred. We will make heavy use of the Unix shell and tools. Your mileage with Windows *will* vary.

Course Topics:

- How imperative languages such as C compile and run. How to look for language artifacts in compiled machine code.
- Understanding the runtime: the Application Binary Interface (ABI); compilation, linking, and loading of libraries; dynamic linking.
- Scripting languages. Internals of a Ruby interpreter.
- Functional programming: why aspire to program without "state" and "side effects" (or loops). Functions as "first-class objects".
- Recursion and induction.
- A taste of LISP(s); programs as their own fundamental data types.
- Lambda calculus, and what it has to do with programming. Other models of computation.
- Closures: the idea and the implementation.

[1]The mathematical foundations of monads, functors, and other concepts that Haskell and other cutting-edge programming languages explore are beyond the scope of this course; still, we'll touch on some ideas behind them.

- Continuations and the continuation-passing style.
- Types and type-checking.
- Polymorphism and generic programming. How they work in C++ and Java. The mechanisms behind C++'s overloading and inheritance.
- Strongly typed languages and type inference.
- A taste of ML and Haskell: functional, strongly typed languages with polymorphism.
- Algebraic data types and a new style of programming: pattern-matching, etc.
- Laziness and its puzzles.
- An idea of why "Propositions are Types" and "Types are Programs".

Textbooks:

– *Concepts in Programming Languages* by John C. Mitchel (available new or used from Amazon)

– *On Lisp* by Paul Graham, available as free download from `http://www.paulgraham.com/onlisp.html`

– *Ruby Under a Microscope: An Illustrated Guide to Ruby Internals* by Pat Shaughnessy

– *Real World Haskell* by Bryan O'Sullivan, Don Stewart, and John Goerzen, freely available online at `http://book.realworldhaskell.org/read/`

– We will also use chapters from the free *Structure and Interpretation of Computer Programs* (2nd ed.) by Harold Abelson and Gerald Jay Sussman with Julie Sussman `https://mitpress.mit.edu/sicp/`

Supplemental on Lisp:

– *Common Lisp the Language* by Guy Steele (2nd ed.), freely available at `https://www.cs.cmu.edu/Groups/AI/html/cltl/cltl2.html`

– *Practical Common Lisp* by Peter Seibel, freely available at `http://www.gigamonkeys.com/book/`

– The book I like best: *ANSI Common LISP* by Paul Graham. Unfortunately, not available freely, but worth owning if you are interested in LISP.

Supplemental on Haskell:

– *Learn You a Haskell for Great Good* by Miran Lipovača. Freely available online at `http://learnyouahaskell.com/`