CSE P505: Programming Languages
Course Information and Syllabus
Winter 2009

Logistics and Contact Information: The instructor is Dan Grossman. See the course home page (`http://www.cs.washington.edu/education/courses/csep505/09wi/`) for all pertinent information. *You must join the class email list and check email at least once a day.*

Goals: Successful course participants will:

- Master universal programming-language concepts including control flow, datatypes, functions, continuations, threads, types, objects, and classes such that they can recognize them in strange guises.
- Learn to evaluate the power, elegance, and definition of programming languages and their constructs.
- Attain reasonable proficiency programming in a functional style.
- Find relevant literature somewhat more approachable.

Our primary intellectual tools will be *operational semantics* and *Caml programs*. Prior knowledge of neither is required nor expected.

Audience: This course is for software professionals and 5th-year students with substantial programming experience. No background in functional programming or programming-language theory is needed, though we will "move fast." Experience with imperative and object-oriented programming may be assumed occasionally.

Format: One weekly lecture will develop the course content. Homeworks will extend the material discussed in lecture; expect to learn as you do them. Programming exercises must be done in the Caml language, which will be discussed in class. The final exam will assess understanding of the lectures and homeworks.

There is no textbook, but substantial and free materials on Caml are linked from the course homepage. For those looking for a textbook, "Types and Programming Languages" by Pierce and, "Design Concepts in Programming Languages" by Turbak/Gifford have substantial but incomplete overlap. We will provide pointers to (optional) relevant research papers throughout the quarter.

Homeworks, Exams, Grading: We will have five homeworks (possibly +/- 1). Each homework will contribute equally toward the course grade and the final exam will contribute twice as much as a homework. The final exam is **Thursday, March 19th, 6:30-8:20PM**. Contact the instructor immediately if this proves impossible.

Academic Integrity: Any attempt to misrepresent the work *you* have done will result in immediate course failure. If there is any doubt, indicate on an assignment who assisted you and how. In general, you may discuss general approaches to solutions, but you should write your solutions on your own. You should not show your written solution to someone else or view someone else's solution. Particular assignments may include more specific instructions. If not, ask. *Violating the academic trust your instructor and classmates have placed in you will have a far worse effect on you than doing poorly on a homework assignment.*

Advice:

- In every course, there is a danger that you will not learn much and therefore lose the most important reason to take the course. In P505, avoid "thinking too generally" (not learning enough precise details to appreciate the rigor of programming-language definitions) or "thinking too specifically" (focusing so much on the details that the purpose of the definitions is lost). This balance takes getting used to, but in a ten-week course, time is of the essence.
- If you adopt the attitude, "I will have fun learning to think in new ways" then you will do well. If you instead say, "I will fit everything into the way I already view programming" then frustration is likely.
- While the homework mostly involves writing code, approach this code differently than industrial-strength programs. Your solutions should be short, simple, self-contained, and elegant. Be a minimalist. See the separate document, "Advice on approaching Caml programming in P505."
- Come to class. Ask relevant questions.

Approximate Topic Schedule (Very Much Subject to Change):

1. Purpose of studying programming languages
2. Functional programming (in Caml)
    - Lack of mutation
    - Higher-order functions
    - Datatypes
    - Tail recursion
3. Defining languages
    - Concrete vs. abstract syntax
    - Introduction/elimination forms
    - Semantics via interpretation
    - Semantics via translation
4. Formally defining languages via inference rules
5. Induction for Proving Program Properties
6. Lambda-Calculus
7. Abstract Machines
8. Exceptions, continuations, continuation-passing style
9. Types
    - Soundness vs. completeness
    - Type safety
    - Type inference
    - Static vs. dynamic typing
    - Curry-Howard isomorphism
10. Polymorphism
    - Subtyping
    - Generics
    - Parametricity
    - Polymorphic references
11. Laziness (thunks, streams, memoization)
12. Concurrency
    - Threads
    - Locks
    - Atomicity
    - Message passing (Concurrent ML)
13. Object-oriented programming
    - Dynamic dispatch
    - Classes vs. types
    - Multimethods
    - Classless OOP
    - Extensibility: contrast with functional programming
14. Memory management
    - Garbage collection
    - Regions (a.k.a. arenas)
    - Unique pointers