CSE P505: Programming Languages
Course Information and Syllabus
Spring 2006

Logistics and Contact Information: The instructor is Dan Grossman. See the course home page (`http://www.cs.washington.edu/education/courses/csep505/06sp/`) for all pertinent information. *You must join the class email list and check email at least once every 24 hours.*

Goals: Successful course participants will:

- Master universal programming-language concepts (including datatypes, functions, continuations, threads, macros, 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.

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

Format: One weekly lecture will develop the course content. Code examples accompanying the slides may prove useful so students can "keep the big picture" of an extended example in front of them. 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. We will provide pointers to relevant research papers throughout the quarter.

Homeworks, Exams, and Grading: We will have four, five, or six homeworks (probably five), and one final exam. 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, June 8th, 6:30-8:20PM**. Please be there and contact the instructor immediately if this proves impossible.

Academic Integrity: Any attempt to misrepresent the work *you* have done will result in the maximum penalty the university allows. 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, this danger can result from "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 approach the course by saying, "I will have fun learning to think in new ways" then you will do well. If you instead say, "I will try to fit everything I see into the way I already look at programming" then you may get frustrated.

Approximate Topic Schedule (Very Much Subject to Change):

1. Purpose of studying programming languages
2. Functional programming (in Caml)
    - Datatypes
    - Higher-order functions
    - Tail recursion
    - Lack of mutation
3. Defining languages
    - Concrete vs. abstract syntax
    - Introduction/elimination forms
    - Semantics via interpretation
    - Semantics via translation
4. Induction for Proving Program Properties
5. Program and Programming-Language Equivalence
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
10. Polymorphism
    - Generics
    - Parametricity
    - Subtyping
    - Polymorphic references
11. Laziness (thunks, streams, memoization)
12. Macros
13. Concurrency
    - Threads
    - Locks
    - Message passing (Concurrent ML)
    - Atomicity
14. Object-oriented programming
    - Dynamic dispatch
    - Classes vs. types
    - Multimethods
    - Classless OOP
    - Extensibility: contrast with functional programming
15. Memory management
    - Garbage collection
    - Regions (a.k.a. arenas)
    - Unique pointers