CIS 624: Structure of Programming Languages  
Fall 2014

**Logistics and Contact Information:** The instructor is Boyana Norris. See the course web page (http://www.cs.uoregon.edu/Classes/14F/cis624/) for all pertinent information.

**Learning objectives:** We will investigate concepts essential to programming languages including control flow, scope, functions, types, objects, and threads. As time permits, we will explore other concepts such as continuations, exceptions, message passing, and connections to logic. Our primary intellectual tools will be operational semantics and OCaml programs. Prior knowledge of neither is required nor expected. We will build formal models and prove properties about them, which is a useful research skill throughout computer science (and other sciences).

Successful course participants will learn to:

- Give precise definitions to programming-language features.
- Prove properties of inductively defined sets (e.g., well-typed programs).
- Appreciate some programming-language theory jargon (e.g., inference rules) and find the research literature more approachable.
- Write better programs by exploiting modern language features such as higher-order functions and objects.

In short, our goal is to use theory to make us better programmers and better researchers. Always think, "how is this related to programs I have written?"  

**FORMAT**

Two weekly lectures will develop the course content. The recommended textbook (Types and Programming Languages by Pierce) covers some of the same material. References to other (free) materials will be added throughout the term. It can serve as an excellent second explanation, but we may not necessarily follow it closely, so you can consider it optional. Homeworks will extend the material discussed in lecture; expect to learn as you do them. Programming exercises must be done in the OCaml language, which will be discussed in class. Exams will assess understanding of the lectures and homework.

**Homeworks, Exams, and Grading:** We will have 4–5 homeworks, one three-part writing assignment, one midterm, and one final. The homeworks will be worth 30% of the course grade. The writing assignment will not be graded — you will receive full points worth 10% of your grade upon completion. The midterm will be worth 30% of the course grade. The final will be worth 30% of the course grade.

**Academic Integrity:** Any attempt to misrepresent the work that you have done will result in the maximum penalty the university allows. If there is any doubt, simply indicate on an assignment who assisted you and how. In general, you may discuss general approaches to solutions, but you must 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, do ask. Violating the academic trust your instructor and classmates have placed in you will have a far worse effect on your academic future than doing poorly on a homework
Course materials

Recommended:
• Programming Language Foundations, Aaron Stump, 2013, Wiley
• Types and Programming Languages, Benjamin C. Pierce, 2002, MIT Press. Errata

Suggested:
• Programming Languages : Theory and Practice, Robert Harper, draft
• Syntax and Semantics of Programming Languages, Kent Petersson
• Denotational Semantics: A Methodology for Language Development, David Schmidt
• Fundamental Concepts in Programming Languages, Christopher Strachey
• Theories of Programming Languages, John Reynolds, Cambridge University Press
• The Formal Semantics of Programming Languages , Glynn Winskel, 1993, MIT Press

See course web page for information on OCaml and additional references.

TOPICS
(Subject to change)

• Introduction to OCaml
• Inductive definitions
• Operational semantics
• "Pseudodenotational" semantics
• Assignment and basic control flow
• Scope and variable binding
• Lambda calculus
• First-class continuations
• Continuation passing style and CPS transform
• Simple types
• Type safety
• Universal and existential types
• Recursive types
• Shared–memory concurrency (threads, locks, transactions, memory–consistency models)
• Message–passing concurrency; Concurrent ML
• Parametric polymorphism