UCLA Computer Science 131: Programming Languages

Overview

This course teaches students the principles underlying programming languages and their design. These principles are conveyed via the discussion and usage of several programming languages. The languages also serve as representatives of various programming paradigms, including functional, object-oriented, and logic programming. Students will learn each paradigm's novel language features, its characteristic programming idioms, and its relationship to the other paradigms. We will focus on the languages ML, Java, and Prolog.

PTEs

The advising office handles all undergraduate enrollment issues (including PTEs). Graduate students have the option to audit the class and take the exams in order to pass the Breadth Requirement. CS131 will be offered again in the Winter Quarter.

Logistics

All of the course information and materials are available on the CS131 homepage. You must login using your BOL account in order to gain full access to the materials.

The instructor is Todd Millstein. The TAs are Saswat Padhi and Tomer Weiss.

Getting Help

We will use the Discussion Forum that is part of the course website. If you have a general question that other students may be able to answer and/or could benefit from knowing the answer to, you should ask it there. Of course, **you should never post anything that gives away part of your answers to an assignment or violates our academic integrity policy (see below) in any way. For example, posting your code is obviously out of bounds. If in doubt, don't post.**

If you have a specific question about your work you should attend either my or one of the TA's office hours.

Prerequisites

This course assumes that you have proficiency in at least one programming language. You must have already taken CS31 and CS32, or an equivalent sequence. Experience with trees and related data structures, propositional logic, and formal language theory (e.g., finite-state machines and context-free grammars) will also be useful.

Text

This book does a good job of describing many of the key concepts underlying programming languages and their design. It also specifically describes three languages -- Standard ML, Java, and Prolog. We are studying the same languages in this class, except we will use the OCaml dialect of ML rather than Standard ML.

Adam Brooks Webber, *Modern Programming Languages: A Practical Introduction*, Franklin, Beedle & Associates, ISBN 1-887902-76-7 (2002-10-01).

Other resources for the languages we will study are available here.

Grading

- homeworks: 35%
- midterm exam: 30%
- final exam: 35%

The midterm exam will be in class on Thursday, October 30. The final exam will be on Monday, December 15 from 11:30am-2:30pm.

Homework

Homework assignments constitute the majority of the course workload, and they are the primary means by which you actually learn the concepts taught in lecture. Homeworks have you actively explore different programming paradigms by writing code in various languages.

Late Policy

You can turn in each homework up to 3 days late for partial credit. If your score would normally be S, then being 1 day late will make your score S * 0.9, being 2 days late will make your score S * 0.7, and being 3 days late will make your score S * 0.5. **One second late is equivalent to 23 hours, 59 minutes, and 59 seconds late -- both cost one late day.**

Academic Integrity

I trust you, and I take violations of this trust quite seriously. Both SEAS and the university as a whole have strict policies on academic integrity. Our course additionally has its own policy on academic honesty, which can be found on the course web page. **We will adhere to these policies strictly.**

Major Course Topics

functions

- motivation
- recursion and tail recursion

- activation records
- anonymous functions
- first-class functions
- parameter passing

names

- names, binding, visibility, scope, lifetime
- static vs. dynamic scope

types

- primitive and compound types
- types, values, operations
- dynamic vs. static typing
- strong vs. weak typing
- typechecking vs. type inference

polymorphism

- overloading
- parametric polymorphism
- subtype polymorphism

objects

- object-oriented design
- encapsulation and data abstraction
- classes and class hierarchies
- inheritance

parallelism and distribution

- fork-join parallelism
- MapReduce

control

- pattern matching
- dynamic dispatch
- unification
- backtracking
- exceptions