Computer Science 131 – Fall 2010

Instructor & Texts

Instructor: **Kim Bruce**
222 Edmunds, x7-1866
kim@cs.pomona.edu
Office Hours: MWF 11 – 11:50 a.m., MThF 2:00 - 3:00 p.m., & by appt.

Lectures: MWF 10:00 - 10:50 a.m., Edmunds 101
TAs: Charlotte Smail, Alejandro Lopez-Lago, & Elliot Godzich
TA hours
– *HMC Beckman B 105 (Hot Air Balloon lab)*: Alejandro, Elliot:
Wednesday and Thursday evenings, hours to be announced.
Text: *Concepts in Programming Languages*
by John C. Mitchell, Cambridge University Press, 2003
Course web page: http://www.cs.pomona.edu/classes/cs131/
Instructor's web page: http://www.cs.pomona.edu/~kim/
Prerequisites: (CSC062 or CSC070) and CSC081

Overview

A programming language is a programmer's principal interface with the computer. As such, the choice of an appropriate language can make a large difference in a programmer's productivity. A major goal of this course is to present a comprehensive introduction to the principal features and overall design of both traditional and modern programming languages. As such you will be examining language features both in isolation and in the context of a more complete language description.

Programming languages are fundamental to program design, analysis, and implementation. Every student passing this course should be able to:

1. Quickly learn programming languages and how to apply them to effectively solve programming problems.

2. Rigorously specify, analyze, and reason about the behavior of a software system using a formally defined model of the system's behavior.

3. Realize a precisely specified model by correctly implementing it as a program, set of program components, or a programming language.

This summer the Education Board of the ACM Special Interest Group in Programming Languages (SIGPLAN) released a first draft of "Why Undergraduates Should Learn the Principles of Programming Languages" that explains the importance of the principles of programming languages for undergraduate

CS education. That report is available at http://www.cs.pomona.edu/~kim/why.pdf. Please take a look. [Disclaimer, I was chair of the SIGPLAN Education Board when the report was written.]

We will examine features of a large variety of languages, though we will not study many of the languages themselves extensively. Like other CS courses, we will be discussing alternate ways of solving problems, looking at the pros and cons of different approaches. Because programming languages are so tied up with (and motivated by) programming problems, we will not only investigate language features, but also the software engineering problems that spawned them.

We will also investigate and practice writing interpreters and/or compilers for simple programming languages. This hands-on experience will provide a firm grounding in both the run-time characteristics of programming languages and the formal specification of programming language semantics. We will also see how the formal specification of different aspects of languages allows us to prove properties of the languages themselves, such as type safety, as well as reason properly about programs in the languages.

At the end of this course you should have a more thorough understanding of why certain programming language features provide more support for the production of reliable programs, while others are fraught with ambiguity or other problems. Since programming languages mediate between the programmer and the raw machine, we will also gain a deeper understanding of how programming languages are compiled, what actually happens when a program is executed on a computer, and how the programming language design affects these. As an example, by the end of the course, you should be able to understand why Java is generally considered a more reliable language than C or C++, while it will also generally execute slightly slower (at least on single processor machines). Similarly you will understand why Python must execute significantly slower than Java or C++.

In the last section of the course we will turn our atention to one of the biggest current problems in computer software: How to write concurrent and parallel programs. Our particular focus will be on languages that can support these kinds of programs. This is now a focus of considerable research, and we hope for a break-through in the near future, as we need to learn how to use the new multi-core (and soon many-core) computers effectively to create faster software.

This course will involve extensive reading on your part, both in the text and in outside sources. The segments of the course that introduce new programming language paradigms will also feature some relatively simple programming in newer languages supporting functional, object-oriented, and concurrent styles of programming, such as Haskell, Java, and Scala.

Lectures and Readings

The schedule on the following two pages shows the tentative schedule of topics to be covered at each class meeting during the semester. Consult the on-line version of the course syllabus (see URL above) regularly to see the most current version of the schedule of topics and readings. The on-line version of this schedule will be revised as the semester progresses.

I expect you to do the reading for a class before the lecture. I will not attempt to cover in lecture all the material in the readings. Instead my goal will be to cover the highlights or particularly difficult material. For this to work, you will need to already be familiar with the simpler aspects of the material. If you keep up your part of the bargain we should be able to have more interesting discussions in class, rather than just listening to me go over the text.

In the table below, M stands for the textbook by Mitchell.

Lecture	Date	Topic	Reading
1.	Jan. 19	Preview and History	M 1
2.	Jan. 21	Computability; LISP	M 2, 3
3.	Jan. 24	LISP; Compilers, Interpreters, & Virtual Machines	M 4.1
4.	Jan. 26	Lambda Calculus	M 4.2
5.	Jan. 28	More lambda calculus	
6.	Jan. 31	Typed lambda calculus & functional languages	M 4.4
7.	Feb. 2	Haskell	M 5
8.	Feb. 4	More Haskell	
9.	Feb. 7	Even More Haskell	Ch. 5 examples in Haskell
10.	Feb. 9	Monads & Functional Languages Evaluation	Tackling the Awkward Squad:, pp. 1–16
11.	Feb. 11	Data Types	M 6.1, 6.2, & 6.5
12.	Feb. 14	Type checking & inference	M 6.3
13.	Feb. 16	Type inference & Polymorphism	M 6.4
14.	Feb. 18	Lexing & Parsing	
15.	Feb. 21	More Parsing	
16.	Feb. 23	Semantics	M 4.3
17.	Feb. 25	More semantics	
18.	Feb. 28	More semantics	
19.	March 2	Type safety	Type Safety Handout
20.	March 4	Scala	Scala talk, Associated Scala slides
21.	March 7	Run-time storage management	M 7.1-7.2

Lecture	Date	Topic	Reading
22.	March 9	Functions, Procedures, & Tail Recursion	M 7.3
23.	March 11	More run-time storage and access	M 7.4–7.5
	March 14–18	**Spring Break**	
24.	March 21	Higher-Order Functions & Managing the Heap	
25.	March 23	Heap & Control	M 8.1-8.2
	March 25	**Chavez Day – no classes**	
26.	March 28	Commands, Exceptions, & Continuations	M 8.1-8.2
27.	March 30	Data Abstraction	M 9.1-9.2
28.	April 1	Modules	M 9.3-9.5
29.	April 4	ML Modules & Subtyping	Subtypes chapter
30.	April 6	OOLs	M 10
31.	April 8	OOLs: Simula & Smalltalk	M 11
32.	April 11	C++ & Implementation	M 12
33.	April 13	Java & Typing Issues	M 13.1-13.4
34.	April 15	Java 5 generics & wild cards	M 13.5
35.	April 18	More Scala	
36.	April 20	Concurrency	M 14.1
37.	April 25	Actors in Scala	M 14.2, Scala Actors
38.	April 27	Shared memory Concurrency: Semaphores & Monitors	M 14.1, Concurrent Programming with Java2SE 5.
39.	April 29	More Concurrency:Monitors & Ada Tasks	M 14.4
40.	May 2	Concurrent ML	M 14.3
41.	May 4	Concurrency & Summary	

Programs from Lecture

As the course progresses links will be provided to take you to sample programs from classes

Homework and Programming Assignments

Programs Programs for this course will be run on the Pomona College Computer Science department's lab facilities, based in Edmunds 227. You are welcome to use other computers to write and test your programs, but they must run on our facilities. You may log in remotely to any of the lab machines using ssh. Please do not log into any of our servers (e.g., vpn or xserv) to do homework.

If you do not have an account on the Pomona College Computer Science network, please go to https://www.dci.pomona.edu/ immediately to request an account.

Turning in Homework Runnable copies of programs for homework should be turned in via the web page at http://www.dci.pomona.edu/tools-bin/cs131upload.php. If you have more than one file for an assignment, please put all associated files in a directory, zip up the directory, and then turn in the zipped directory.

If you are running your programs on other computers, please make sure that you are using compatible versions of software. I will specify in class the versions that we have loaded on the Pomona College computers. Different versions of compilers or interpreters often use slightly different libraries that are incompatible with other versions. You are responsible for making sure that your program, as turned in, will run successfully without any extra work on my part. Include instructions on how to run your programs either at the top of the program file or in a separate README.

I prefer to receive copies of all problems, whether program or not, in the web-based dropbox. Non-program solutions can be turned in either as plain text or pdf (preferably generated by LaTeX). If you would prefer to write your solutions by hand, please turn in printouts of all problems (including the programs), but also turn in programs in the dropbox. All items turned in must have your name on them (e.g., as comments for programs), as I will normally be looking at printouts, and will not know who to give credit to if there are no names printed.

An important criterion in grading homework will be clarity of solution. Thus you should attempt to explain your solutions, program or not, as clearly as possible. This also means that programs should be carefully documented so that I can understand them. At a minimum, each function defined should include a comment on what it does. The comment should explain what each input parameter stands for and how the output depends on the input.

Late policy Problems involving analysis of programming language features will be assigned and due most weeks during the term. Homework will generally be due at the beginning of class on Fridays. Each student may use a maximum of three late days during the course of the semester (note that weekend days count). Once those late days are used up, late homework will not be accepted.

Exams and Grading

There will be open-book take-home midterm and final exams covering both lectures and readings. The midterm will be handed out after spring break. The final will be a 24 hour take-home that can be picked up from the CS secretary during office hours during the final exam period.

Student grades will be determined as follows: Midterm: 25%, Final Exam: 35%, Homework and programs: 40%.

Collaboration & Academic Honesty Policy

I highly encourage students to get together in small groups to go over material from the lectures and text, work problems from the text, study for exams, and to discuss the general ideas and approaches to material in the course.

However, work to be turned in, including programming assignments, must be done independently, unless I explicitly designate an assignment as one in which collaboration is allowed. As explained in the student handbook, this means that the work you turn in must represent only your own work. It must not be based on help from others or information obtained from sources other than those approved by the instructor (e.g., the text, web pages linked from the course web page, and materials provided in lecture). Effective learning is compromised when this is not the case.

Accordingly, you should never read or copy another student's code or solutions, exchange computer files, or share your code or solutions with anyone else in the class until after the assignment is due. However, students may collaborate or receive help from each other on an occasional basis as long as all parties contributing are given explicit credit for their contributions to the homework. I will inform students if I believe they are collaborating too much. Uncredited collaborations will be considered a violation of college policies and will be handled appropriately. If there is any doubt about whether a collaboration is legal, you should cite it or ask me. I will let you know if it is within the rules.

An important exception to the above rules has to do with learning to use new computer languages. Students are explicitly allowed to help each other with difficulties in setting up or running interpreters or compilers to run programs, and to help explain error messages. However, any collaboration beyond that point should be cited as explained in the prior paragraph.

Failure to abide by these rules is considered plagiarism, and will result in severe penalties. The first offense typically results in failure in the course and referral to the appropriate college office or committee. See the Academic Honesty Policy in the Student Handbook for further information. Please do not put me, yourself, or anyone else in this unpleasant situation.