UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Prof. R. Fateman

Fall 2005

General Course Information: CS 164 Programming Languages and Compilers

Personnel

Instructor:	Richard J. Fateman, 789 Soda (fateman@CS, cs164@cs)
Teaching Assistant:	David Bindel, 515 Soda (dbindel@CS)
Class Home Page:	`http://www-inst.eecs.berkeley.edu/~cs164`.

Purpose of this Course

CS164 is an introduction to the design and implementation of programming languages.

In addition to study of the theory of programming language design and associated programming tools, you will implement a simple programming language through a set of programming exercises. These, taken together comprise most of the parts of a compiler as well as an interpreter.

We will also discuss a variety of historically influential programming language design ideas as well as current directions in languages.

Each instructor in CS164 provides his or her own stamp on the project sequence. This semester we will ultimately be implementing a language described in the main textbook for this semester. This is a very much cut down version of Java, referred to as MiniJava. Although the text uses Java as an implementation language, we will be using Common Lisp. Why?

1. Common Lisp will show you what an advanced language of the Lisp family can provide in terms of fast prototyping, on-line debugging, etc.

2. Common Lisp is already somewhat familiar to you from your acquaintance with Scheme.

3. Language implementation is a fundamental model of program development in Common Lisp: in CS61a you've seen how an interpreter can be used to define a language (Scheme); depending on the projects in your CS61a class, you may have also seen how the language Logo can be implemented. In Common Lisp we will find that a number of the necessary tools do not have to be invented in CS164: they already exist in the language. (creating and traversing trees, for example).

4. The model of language implementation by interpretation (the Scheme evaluator) is a powerful idea you've already seen. We will refresh your understanding of this — or perhaps you will appreciate interpretation for this first time in this course. In CS61a we do not talk about generating code (compiling), since you have not seen assembly language and machine structures, part of CS61C. Simple code generation, which is the role of a compiler, is surprisingly *not much harder than interpretation*, and fits a similar pattern. Since we will generate and execute "virtual machine code" some of the ideas behind Java's machine implementation will emerge as well.

5. Because the tasks you are required to perform can be done with so much less code (and we hope less time and effort), than if you were writing in Java or C++, we will have time to study programming language topics beyond those illustrated by the project. Also, if you allocate your time reasonably well, you will be able to finish your assignments without staying up all night.

Prerequisites

Please take this course *only if you have the prerequisites*: the CS 61ABC sequence. You are assumed to be familar with Scheme and *skilled* at reading and writing programs in at least one other higher-level language (most likely Java or C++). You are not expected to know Common Lisp yet, but you will pick up this language easily. It is like Scheme on steroids. We assume you are capable of using a network browser, email, newsgroups, emacs, etc. You will use only a small subset of tools in UNIX if you use the class accounts; since it is possible to run both emacs and Common Lisp on Windows 2000 or XP, or Apple Macintosh OS, you can do your assignments on a variety of platforms.

While I understand that people can become attached to other (non-emacs) editors, it seems to me the time needed to become familiar with emacs is repaid many times over by its greater level of efficiency in writing and debugging Lisp or other programming languages. (Many people unfamiliar with emacs and only slightly familiar with Lisp think that a major issue in Lisp programming is "balancing parentheses". This is not an issue since any decent editor shows you, by indentation, how expressions are grouped.) Of course there are many interesting additional features of emacs, not the least of which is the possibility of extending its features by writing emacs-lisp code!

Scheduling

There are two lecture meetings each week, meeting in Room 22, Warren Hall, Tu/Th 9:30-11:00. The course/lecture slides will be available on-line; sometimes copies may be handed out at lecture. The slides alone are not intended to be a total substitute for attendance. I understand that some students are not "morning people." Casual dress

Meeting Times

There were too many scheduled meeting times for discussion, based on the historical higher enrollment for CS164. Our plan is to consolidate the discussion sections as shown below, so you should choose section 101 or 104 if you are in 102, 103 or 105. If this presents problems for too many students, we will make other arrangements. We will schedule office hours after checking with students.

meeting	days	time	place	person
lecture 1	Tu/Th	9:30–11:00	22 Warren	Fateman
disc. 101	Tu	3:00–4:00PM	241 Cory	Bindel
disc. 102	Tu	4:00–5:00PM	433 Latimer	Bindel /cancelled?
disc. 103	W	9:00–10:00AM	71 Evans	Bindel /cancelled?
disc. 104	W	10:00–11:00AM	4 Evans	Bindel
disc. 105	Tu	11:00–12:00PM	5 Evans	Bindel /cancelled?

Grades

Important: Your grade will not depend upon the grades of others in the course. In particular, we will not limit the number of A grades to some percentage of the class. All students who have done excellent work on exams and assignments will get "excellent" grades.

Past experience suggests that students want to keep track of how they are doing relative to others, and how they should allocate their time among their courses. Most of the activities within CS164 are aimed at helping you learn the material and demonstrate your knowledge on exams. But we will also count other aspects of your performance toward your grade.

Your ranking in this course will depend on four components:

1. Programming Assignments: We are planning on 7 programming assignments. You will be expected to turn in your answers in electronic form by the time given in the assignment. For late assignments we will generally assign a grade of 0 for all questions whose answers have been discussed in class or section (whether or not you attended). To be safe, please turn in your results by 11:59PM on the day assigned (unless otherwise indicated in the assignment). After that, your grade will be decreased by 50% per 24-hour period, or fraction. Not all the programming tasks are expected to be equally challenging. Assignments 1 and 4 should be easier; the hardest is probably assignment 6.

   Programming Assignments in total will account for 30% of your course grade.

2. Exams: There will be 2 examinations in class, and a final. These will represent 10, 10, and 30% of your course grade for a total of 50%. We may experiment with in-class brief quizzes factored in to this portion of your grade. Since the class is small enough, we propose to have oral examinations on your projects. E.g. "Explain what this piece of code does." Naturally answering such questions are made easier if your code has extensive comments. If students raise strong objections we will reconsider (Like if you get nauseous from nervousness just thinking about talking to a professor.)

   Exams will probably be open-book (closed computer/PDA/Cellphone).

   This course is scheduled for final exam group 7, which is Thursday, December 15, 2005 from 12:30-3:30PM. It should be impossible for you to have scheduled another class in the same final exam group.

3. Problem Sets: We may experiment with a few of these. These may be more likely near the beginning of the semester when we are doing more "theory." The intent is to help you anticipate questions on the exams and final. It will ordinarily be simpler for you to write these out on paper rather than type them in to a computer system, if we use them, they will be turned in at class meetings or at alternate place to be determined. The grade here will represent *no more than* 15% of your course grade.

4. Participation in discussion: Remaining points: the teaching assistant will judge your performance on the basis of in-class participation, any scores on discussion-section quizzes, partnership cooperation.

Our intention is to provide an absolute grading scale to emphasize that students are *not* competing against each other. The number of A grades is not limited. In the past we have been successful in starting out with 200 possible points, 185–200 is an A+, 175–184 is an A, 165-174 is an A-, with 10 points each for B+, B, B-, C+, C, C-, D+, D, (and below is F). If exams turn out to be harder than expected, we may modify the scale.

Partnerships and Original Work

There is a tradition in this course providing students an opportunity for programming in teams. These partnerships (maximum 2 persons, unless you get prior special permission) will receive the same grade on programming assignments. (Problem sets and exams must be done individually.) Since the exams will contain questions about the program modules, it is extremely unwise to divide up your projects so that each partner does all the work on "half" the modules. In fact, a few questions on the exams (written or oral) will be quite unanswerable if you have not done the programming (or studied your partner's pieces *verrrry* carefully).

The advantage of partnerships is largely in the ability to practice the social aspects of computing, including having someone else comment on your programs. Partnership is especially helping to find elusive bugs. Partnerships are completely optional, however. You may work alone for one or all of your assignments. Your TA will be monitoring the health of your project partnership.

While you may discuss questions of strategy in your projects with staff, or another student who is not your partner, you *must not take someone else's code and make it seem as if it is your own.* If you have discussed ideas with others, including the TA, the professor, and other students, you should give proper credit in comments. For example, "Thanks to xyz@eecs who helped me find the bug in this" or "the idea for storing symbols this way was suggested by abc@eecs". Clever computer programs have been devised to detect unusual similarities between submitted programs, and it is embarrassing to everyone if we discover "coincidences".

In summary: If you visit us during office hours and we tell you how to solve a problem, please give us credit. If you use ideas from other students, e.g. from the newsgroup, or friends, you should also say so. If you find ideas or pieces of code on the internet, say so. Wholesale appropriation of program code without attribution is plagiarism, and is unacceptable.

General hints: start your assignments early and ask for help from the staff if you need it. Leave copies of early and intermediate development files on the CLASS's file systems so you will not lose everything if you home computer fails. Evidence of partial efforts on our file system is also prudent to show that your work is your own. We will not require use of a version control system (e.g. CVS), and do not expect that use of such a system would be of much benefit, but you can use one. If you work at home, you should nevertheless read and use the newsgroup `ucb.class.cs164`, where many questions and answers will appear. Usually answers to well-posed questions appear very rapidly, at least during the day. Frequently the answers are correct. *Most people benefit greatly from the newsgroup. Provide credit in your submissions! Students already know not to post big sections of code or solutions to homework: let other students figure out the project!*

Required Texts

There is still no ideal text for CS164 as taught at UCB, so instructors tend to use course notes. In addition to the course notes and slides, we have decided to require the Appel text (see below). It is more up-to-date than the Aho text (optional), but is somewhat brisk in places; it advocates writing a language implementation in Java. Design of the Java code given in the text may be helpful, but detailed examination of the jumble of stuff needed for Java to run might possibly lead you astray. Programming in Lisp should be simpler.
Andrew Appel, Jens Palsberg, *Modern Compiler Implementation in Java*, (SECOND EDITION, 2002.) Cambridge Univ. Press.

Note: don't get the first edition (1998 or 1999), or a version of this book based on ML or C++.

The next (also required) book explains Common Lisp in a very readable and literate manner. It has especially fine sections on extended examples, and we will use this as a source for programming language features issues.
Paul Graham, *ANSI Common Lisp* Prentice Hall, 1996.

Optional/Recommended

Among the optional books, we include: G. L. Steele, Jr, *Common Lisp: the Language* 2nd edition, Digital Press, 1990.

This is the standard reference; massive when printed out, but entirely available on-line. Some time after working on this, Guy Steele joined Sun to work on the design of Java.

You may find it useful to have the old CS61A text, as well as your favorite book on Java. If you have never seen Java (but know C++, for example), Flanagan's *Java in a nutshell* may be helpful.

The classic compiler book, used in past CS164 classes, is difficult to read and somewhat out of date. Coverage of early topics in the course is an alternative to Appel. You may hear it referred to as the "Red Dragon" book because of its cover illustration. It is
A. V. Aho, R. Sethi, J. D. Ullman. *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 1986.

As the course proceeds, we will provide additional notes, programs, etc. These will be provided on-line in various forms, typically as powerpoint, pdf, or text files.

Programming Languages

All your programming assignments should be written in COMMON LISP. We hope that by the end of the semester you will agree that —even given the time to learn COMMON LISP, you saved time over writing in C++ or Java.

Computer Accounts

If you are a CS or an EECS major, you are expected to have your own "named" computer account. Nevertheless, you should pick up a class account at the first lecture. While it is possible for you to use your named account for much of the course, there appear to be advantages to class accounts, sending email, setting up shared files with a partner, submitting projects, etc.

If you insist on using a named account, be sure to fix up your `.emacs` file so that it includes the class-emacs initialization from the class home page.

If you are using your own home computer, it is especially important to get a copy of the same software we have on our computers. You can download a (free) "Lite" version of Allegro Common Lisp that should provide you with all the facilities you need for this course, as well as documentation, on Windows, Mac, and Linux. You must also make sure to use emacs. The enormously complex Allegro CL "IDE" based on the Microsoft Windows IDE is quite unnecesary for writing non-graphical programs such as we will be doing. (On the other hand, if you were writing a menu-based graphical user interface, the IDE would probably help.) You can download from `www.franz.com.`

We wish to be clear: if you depend on your home/dorm computer, and something goes wrong, we may not be able to help much. You take full responsibility for getting material to and from the computer systems on campus, making periodic backups of your project-in-progress to the class machine's file system, etc. (And once again, make sure you have emacs set up right at home).

Web access

You should read the class newsgroup often; you should bookmark the home page for CS164 where you can find lecture notes, links to software, hints, etc.
`http://www-inst.eecs.berkeley.edu/~cs164`

If you need physical access to the workstations in Soda Hall after 6:30PM, you should get a card-key allowing you in to the labs. *These cards will be available to students whose names are on the official enrollment list for CS164.* We expect these will be available in 395 Cory after the first week or so of classes; they will require a deposit.

Office Hours

person	days	time	place	e-mail	telephone
R. Fateman	T/Th	1:00–2:30PM, OBA	789 Soda	fateman@cs	642-1879
D. Bindel	TBA	TBA	TBA Soda	dbindel@cs	
readers				???	

(OBA = Or By Appointment, TBA = To Be Announced) Other times may be posted from time to time. OBA means meetings can be arranged if you cannot make any of these times. It is a good idea to first contact us by e-mail to ask questions or set up appointments. The class newsgroup can be invaluable and very fast response for some kinds of questions.

This is a course where 30 seconds of educated help from the staff or other students can save you 3 hours of pointless, frustrating thrashing.

While you will be doing a substantial amount of programming, we hope it is MUCH LESS than if you were writing in other languages.

Course assignments, homeworks, exam schedule

This is a *tentative* course outline including lecture topics, readings, topics for discussion section, and homeworks. Reading marked AWA are in the text by Andrew W. Appel, *Modern Compiler Implementation.* Reading marked PG are in the text by Paul Graham, *ANSI Common Lisp.*

Note that the lectures are intended to provide perspective on the readings and homework. We hope you are mature enough to read the assigned material in the week given, to help you get full value out of the lectures. **The material in the reading assignments anticipate the lectures by one week.**

In addition to helping you understand the lectures and answer questions on exams, another benefit of carefully reading the material is that you will be able to impress your fellow students and the professor with your ability to ask (and answer!) pertinent questions in discussion and lecture.

Assignment 0

Turn in, in class on Thursday, a small photograph or sketch (!) of yourself with your full name, login and preferred form of reference (Ms. Jones, Sneezy, etc.) on the back. This could be a photocopy of your photo on your registration card.

Course Outline TENTATIVE, Fall 2005 ***

Week 1:	(Aug 30, Sept 1)	
	Lect:	Course overview; Languages, translation, Common Lisp.
	Read:	AWA: 1.1–1.3(lightly on Java specifics)
	Read:	PG: 1,2,3,4 (you know most of this)
	Read:	PG: 7.1-7.3, maybe 7.4
		get comfy with Common Lisp
	Sec:	set up computer accounts, Review Lisp vs. Scheme vs. Java
	Due:	Programming Assignment 0 due 9/1
Week 2:	(Sep 6, 8)	
	Lect:	MiniJava, Intro to Lexical Analysis
	Read:	AWA: 2, (light on p 31-33)
	Read:	PG: Appendix A (p 287)
	Sec:	More LISP. turning in programs; Q & A
	Due:	Programming Assignment: 1 due 9/8
Week 3:	(Sep 13, 15)	
	Lect:	Lexical analysis, Regular expressions, finite automata
		Highlights of **MiniJava**.
	Read:	AWA: 3.1-3.2
	Sec:	tools for automated LEX
Week 4:	(Sep 20, 22)	
	Lect:	CF Languages, Parsing, Syntactic analysis.
	Read:	AWA: 3.3-3.5
	Sec:	Review for exam, recognizers, NDA to DA conversion
	Due:	Programming Assignment 2 (lexical analysis) due 9/22
Week 5:	(Sep 27, 29)	
	Lect:	Review for test; Recursive Descent, Predictive Parsing
	Read:	AWA: 4.1-4.2
	Sec:	Q & A, LL parser Lisp tool;
	Exam 1	this week. Sept 29 in class
Week 6:	(Oct 4, 7)	
	Lect:	Abstract Syntax, LR parsing
	Read:	AWA: 5.1-5.2
	Read:	PG: 13
	Sec:	examples for LALR parser generator, post-exam discussion
Week 7:	(Oct 11, 13)	
	Lect:	Static analysis
	Read:	AWA: 6.1-6.2, 7.1-7.3
	Sec:	discuss parsing problems
	Due:	Programming Assignment 3 (parser) due 10/13

Week 8:	(Oct 18, 20)	
	Lect:	Semantic Analysis, Intro to run-time
	Read	AWA: 13.1-13.7
	Sec:	Discuss type-checking, polymorphism, etc.
	Due:	Programming Assignment 4 (parser, AST)
Week 9:	(Oct 25, 27)	
	Lect:	Language Run-time (continued)
	Read:	notes on interpreters, AWA 15.1-15.7 ***
	Sec:	Review successful parser/checker techniques
Week 10:	(Nov 1, 2)	
	Lect:	Review for Test; Interpretation
	Read:	AWA: 8, 10
	Sec:	properly tail recursive interpreter for **MiniJava**
	Exam 2	Nov 2
	Due:	Programming Assignment 5 (interpreter for AST)
Week 11:	(Nov 8, 10)	
	Lect:	Virtual Machine, Intermediate Code generation
	Read:	AWA: 14, notes
	Sec:	Review
		discussion of stack machine architecture, assembler
Week 12:	(Nov 15, 17)	
	Lect:	Code generation and optimization
	Read:	notes
	Sec:	More architecture review
	Due:	Programming Assignment 6 (type checker)
Week 13:	(Nov 22, –)	
	Lect:	Prog. language features: macro/string processors, other Examples/Critiques
	Read:	PG: 11, 17, Allegro CL on-line notes
	Sec:	Code generation
Week 14:	(Nov 29, Dec 1)	
	Lect:	Storage (GC), OOP
	Read:	notes
	Sec:	Linking, loading, testing.
	Due:	Programming Assignment 7 (code generator) due 12/1
Week 15:	(Dec 6, 9)	
	Lect:	Other languages and their implementation; review
	Sec:	Review for final, (Dec 15)