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

CS 164 **P. N. Hilfinger**
Spring 2005

CS 164: Programming Languages and Compilers
Handout 0: General Course Information

Personnel

Instructor:	Paul N. Hilfinger, 787 Soda (Hilfinger@CS, cs164@cs)
Teaching Assistants:	Johnathon Jamison and Paul Hale
Information:	See URL `http://www-inst.eecs.berkeley.edu/~cs164` (the class home page).

Purpose of this Course

CS164 is an introduction to the design of programming languages and the implementation of translators for them. In the process, we'll do some general exploration of programming language design and its implications for implementation, and look at a dialect of at least one particular language, which this semester is PYTHON.

One goal of this course is to explore the structure of programming languages and to consider alternatives to familiar programming language features. We'll also study the problem of translating programming languages into machine-executable forms, using PYTHON as a concrete example of a language to be translated, and the assembly language of the Intel ia32 family (used in PCs and some of our Solaris workstations downstairs) as a concrete example of a target machine. We study language translation first to learn some of techniques used that are useful for many programming problems outside of language translation, second to gain a better intuitive feel for the tools we use when programming and the costs of the programs we write, and third (possibly most important) to gain experience with the engineering problems associated with building and validating a substantial piece of software.

Prerequisites and Admission

Please do not take this course unless you have the prerequisites: CS61B and CS61C, or equivalent courses (at a junior college, for example). You should be familiar with Java or C++. You need not be familiar with Python.

Reference Material

There is no required textbook for this course, since none is altogether satisfactory. Officially, we'll rely on notes that I provide (on paper and on the web), but many people feel more comfortable if they can supplement this information with other treatments. I have put three "recommended" books on reserve at the Engineering Library for your perusal:

- *Compilers: Principles, Techniques, and Tools* by Aho, Sethi, and Ullman—also known as "The Red Dragon Book"
- *Modern Compiler Design* by Grune, Bal, Jacobs, and Lagendoen.
- *Engineering a Compiler* by Cooper and Torczon.

Take a look at them, and if you think one might be helpful, you can get it from the bookstore (either on campus or on-line).

For information on Python, there is the book *Python Essential Reference (2nd edition)*, by David M. Beazley. This covers Python version 2.1—not the most recent version, but we will restrict ourselves to some subset of these features. All of the information in this book is available in web pages on-line; we will have a link from the class home page.

You will be able to purchase any additional course documentation and notes at Copy-Central on Hearst as these documents become available (please don't go trying to buy them before I tell you they are ready).

Course Work

The major work in the course consists of building a system to translate programs written in PYTHON. The system will be divided into modules, with each module constituting an assignment. The implementation language is C++ or Java. In addition to the project, there will be a midterm, a final examination, and some ordinary written homework.

Collaboration

You may work on the various modules individually or in pairs. I encourage you to work in pairs. Collaborating with someone often encourages more attention to your programming technique and gives you practice in one of the "real-world" social aspects of computing that university courses must generally ignore. As an extra bonus, working in pairs substantially reduces the strain on our computing resources. Each pair will submit one module and each member of the pair will receive the same grade for that module. The exams may contain questions about the modules, so it is extremely unwise to allow your partner to do all the work. Although the Staff (elsewhere known as "we") will be happy to give advice, it is the responsibility of partnerships to work out their own disputes. Partnerships apply only to the project assignments, *not* to tests or other written homework. Students taking the course Pass/Fail may *not* take graded students as partners. Groups of more than two people will *not* be allowed except under certain unusual circumstances.

I encourage collaboration among groups, as long as it is restricted to questions of strategy as opposed to actual program text. Each partnership or individual is responsible for making it

clear that their work is really theirs. Except for general-purpose utilities (e.g., a doubly-linked list package) you should not generally borrow other people's code. For anything you do borrow (including ideas) you must give proper credit in comments. Naturally, over-dependence on others' ideas will adversely affect your grade on a module, but failure to give credit where credit is due falls under the heading of "cheating."

Cheating (also known as plagiarism) is the presentation of someone else's work as your own. It will not be tolerated. Violators will fail the course and will be reported to the appropriate disciplinary agency. It is not possible for only one member of a partnership to cheat on a project assignment; any transgressions on your part will reflect equally on your partner.

Grading

In an attempt to cut down on cutthroat competitiveness and to make your grade a bit more meaningful, I use an absolute grading scale. Your letter grade will be determined by the total number of points on all assignments: A for 170–200 points, B for 140–170, C for 110–140, and D for 80–110, with plusses and minuses awarded to the top and bottom third of each grade. These divisions are based on past classes' performances. Out of the total 200 points, the final counts for 65, the midterm for 35, and other written homework and project assignments total 100.

Ordinary homework doesn't get many points, because I see it mostly as a chance for exercise. You will be given full credit for handing in a solution in which you demonstrate that you have made a substantial effort on each problem. Since solutions will be available after the due date, late homework *won't* be accepted.

You should hand in programming assignments (as opposed to ordinary homework) electronically by the deadline given on the assignment. I'll penalize an assignment $5N/12$ percent for each N hours it is late, rounded off in some unspecified fashion. During the course of the semester, however, we'll give you a *total* of three late days (72 hours) for free. You may, for example, turn in all but one assignment on time and turn the other in three days late, or you may turn in each of three assignments one day late. Needless to say, you are not obliged to turn in *any* of them late, and it is advisable never to be more than one day late.

Labs, Software, and So Forth

If you don't already have an account, you should be able to get a personalized, named account by following the instructions posted on the green bulletin boards near the Cory and Soda Hall labs. We also have class accounts; you may want to take one just to keep your CS164 work separate from that of other courses.

Some of you probably have PC's at home and would like to use them directly (that is, without having to use a modem and remotely login to one of the Soda Hall machines). We will supply source versions of various tools we use. Other than that, you're on your own. Linux systems will give you a GNU C++ compiler and a recent Java implementation, and you will be able to compile a few other tools that we'll provide (a scanner generator and parser generator for Java). You can get much the same on MacOS X, although as of this

writing, you'll be restricted to Java 1.4 rather than the most recent version. However, you will not be able to run machine code (from later modules of the project) without some kind of ia32-based (Pentium, Celeron, Xeon, etc.) processor.

I will (daringly) assume your familiarity with UNIX and in particular with certain utilities provided with it. It is your responsibility to fill in any gaps in your knowledge. We have no specific requirements for which IDE (Integrated Development Environment) you use. Emacs will be available (with various additions to allow Java debugging and the like). There is growing interest in the Eclipse IDE; we will provide an installation on the instructional machines and pointers to where to go to install it yourself. Besides basic editing, you will need to learn your IDE's facilities for compilation control, documentation reading, and debugging.

If your IDE does not provide these things, it is not an IDE. Do *not* use `vi` to edit your programs. Watching you struggle through the edit/compile/debug cycle while switching amongst multiple instantiations of `vi` will make me despair of your sanity. `Jove` is slightly better, but is not as robust or flexible as `Emacs`.