CS251 Programming Languages **Handout # 1**
Randy Shull
Wellesley College

CS251 Course Information

1 Contact Information

Professor:	Randy Shull
Office:	SCI E120
Phone:	x3102
E-mail:	`rshull@wellesley.edu`
Lectures:	Tues./Wed./Fri. 8:30am – 9:50am
Office Hours:	For the first week of classes my office hours are
.	Thursday: 8:00am – 9:30am
	Friday: 11:00am – 12:20pm
	Office hours for all other weeks will be determinged using student schedules and announced early the second week of classes. I will sometimes need to cancel or shift office hours to attend a meeting or at talk. I will post a message to the course conference to announce changes to office hours.
Web Site:	`http://cs.wellesley.edu/~CS251`
Tutor:	Ava Chang, Hanna Bond, and Shirley Lu drop-in hours TBA

2 Course Overview

There are thousands of programming languages, but only a small number of important programming language ideas. We will elucidate some of these ideas, and use them to understand, evaluate, and compare programming languages.

There are several themes that run through the course:

Dimensions: Programming languages can be analyzed along a number of dimensions. The dimensions we will use to study programming languages include first-class values, naming, state, data, control, types, safety, and memory management.

Programming Paradigms: Programming languages can embody many different models of computation. Java is representative of the object-oriented model, while C is a popular exemplar of the imperative model. In this course, we will study four programming paradigms: function-oriented programming, imperative programming, object-oriented programming, and logic-oriented programming. We will read and write programs in all of these paradigms, but most of our focus will be on the functional paradigm.

Interpreters and Translators: One of the best ways to understand the design space of programming languages is to study the implementation of simple languages. There are two fundamental approaches to programming language implementation: (1) interpreting one programming language (the *source language*) on top of another (the *implementation language*); and (2) translating a program in one programming language (the *source language*) into another programming language (the *target language*) via a translation program written in an *implementation language*. Reading, modifying, and building interpreters and translators are key activities in this course. We will use OCaml as our main implementation language for various "mini" source languages, but we will also dabble in a few other languages, such as Scheme, Haskell, and Java.

Interpreters and translators are excellent examples of *meta-programming*, in which programs manipulate other programs. The notion of meta-programming may seem confusing and somewhat incestuous at first glance, but many computer scientists consider meta-programming to be the most interesting kind of programming activity. We will get lots of hands-on experience with meta-programming this semester.

3 Prerequisites

The prerequisite for CS251 is CS230 *Data Structures*. We will often use standard data structures (e.g. lists, trees, stacks, queues, etc.) and algorithms (e.g., sorting, searching, traversing) both as examples for exploring the expressiveness of various programming languages and as tools for implementing interpreters. We will also revisit many of the "big ideas" of CS111 and CS230 – e.g., abstraction, modularity, recursion, iteration, abstract data types – in the context of the programming languages we study.

We will sometimes use the scanning and parsing tools presented in CS235. We will also make heavy use of the ML programming language introduced in CS235 (though in CS251 we will use the OCAML dialect of ML rather than the SML dialect from CS235).

Knowledge of some material from CS240, particularly assembly-level programming, will also be helpful for some parts of the course. However, neither CS235 nor CS240 are official prerequisites, and we will review materials from these courses that you need to know for CS251.

An unofficial prerequisite for CS251 is the willingness to work hard. The concepts covered in the course are very deep, and the programming (especially when it comes to writing interpreters and translators) is intrinsically more challenging than the programming you've done in CS111 and CS230. Moreover, you will be doing a *lot* of programming in this course. You should expect to work about ten hours every week on your assignments for this course.

4 Reading Materials

4.1 Lecture Notes and Papers

The material presented in CS251 is not neatly covered in any textbook or collection of textbooks. Most of the material will be presented in lecture (take detailed notes!) and in supplementary handouts and code that I provide. I will also hand out a few technical papers that cover some of the material.

4.2 Textbook

There is one textbook in this course from which I will assign readings: *Introduction to the Objective Caml Programming Language*, by Jason Hickey. We will use parts of this book to learn the OCAML programming language, which will be our main implementation language in the course. This book will soon be published by Cambridge University Press; we will be using an on-line draft copy.

4.3 Reserved Materials

Several books relevant to this course are either available in the Computer Science resource room (SCI E125) (Note: books in SCI E125 should only be used in 125 and the microfocus area so that they do not stray too far from home!) They are listed below. I encourage you to become familiar with this collection and to consult it often.

4.3.1 Recommended Scheme, OCaml, and Haskell Books

The following three books contain important examples of programming language dimensions and interpretation. Moreover, they cover programming in OCaml, Scheme, and Haskell, which will be the three main programming languages we use for coding this semester. (We'll use a lot of OCaml and only a little of Scheme and Haskell.)

- *Structure and Interpretation of Computer Programs*, 2nd edition (*SICP*) by Abelson and Sussman with Sussman (MIT Press, 1996). This is the textbook that was used in 6.001 at MIT. Many people rate it as the best computer science text ever written (I am among them). It is a must read for any serious computer science student, but it is not a quick read. Plan to read it carefully many times through; you will learn something new on each pass. Although it is not about programming languages per se, it has more insight into the essence of programming languages than most self-proclaimed programming language texts. Chapter 4 on interpreters is particularly relevant to CS251. This textbook is also a good way to learn how to program Scheme, although that is not its primary purpose.

- *The Functional Approach to Programming*, (*FAP*) by Guy Cousineau and Michel Mauny (Cambridge University Press, 1998). This is a nice resource for learning functional programming in the context of OCaml.

- *Haskell: The Craft of Functional Programming*, 2nd edition (*HCFP*) by Simon Thompson (Addison-Wesley, 1999). This book gives a very good introduction to programming in the functional paradigm in the context of Haskell, a purely-functional, lazy programming language.

4.3.2 Books on Programming Languages in General

- *Programming Languages: A Grand Tour*, edited by Ellis Horowitz. An excellent collection of classic programming languages papers, some of which you will be required to read this term. There are two copies on reserve, neither of which may leave the library: a first edition (1983) and a third edition (1987). They are basically the same, although the later edition has some newer articles.

- *Principles of Programming Languages: Design, Evaluation, and Implementation*, by Bruce MacLennan (1987). Discusses principles of programming language design in the context of actual programming languages. Used as a textbook in previous terms of CS251.

- *Programming Languages: An Interpreter-Based Approach, by Samuel Kamin* (1990). Uses Pascal-based interpreters to explore toy versions of the following modern programming languages: Lisp, APL, Scheme, SASL, CLU, Smalltalk, and Prolog. The strength of this book is that each chapter contains a discussion of the real language on which the corresponding toy language is based; I heartily recommend that you read these discussions. Used as a textbook in previous terms of CS251.

- *Essentials of Programming Languages (EOPL)* by Friedman, Wand, and Haynes (MIT Press, 1992). This book uses interpreters written in Scheme to explore programming language features and paradigms. This is the strategy that we will follow throughout much of the course, although the particular interpreters we use are different than those in the book. The book is well worth reading; the initial chapters are especially helpful for learning Scheme. Used as a textbook in previous terms of CS251.

- *Programming Languages: Concepts and Constructs*, by Ravi Sethi (1989).

- *Programming Languages: History and Fundamentals*, by Jean Sammet, (1969).
- *Introduction to the Theory of Programming Languages*, by Bertrand Meyer (1991).
- *Principles of Programming Languages*, by R. D. Tennent (1981)
- *Programming Language Concepts*, by Carlo Ghezzi and Mehdi Jazayeri (1987).

4.3.3 Books on Particular Programming Languages or Paradigms

- *ML for the Working Programmer*, 2nd edition (*MLWP*) by Lawrence Paulson (Cambridge University Press, 1996). This is an excellent resource for learning how to do higher-order typed programming in Standard ML, a dialect of ML that is different from OCaml. It contains many nice examples of typeful programming, higher-order functions, immutable data structures, and interpreters.
- *The Haskell School of Expression: Learning Functional Programming Through Multimedia*, Paul Hudak, (2000). This is a nice introduction to HASKELL in the context of programming with graphics, animations, and sound.
- *LISP*, Patrick Henry Winston and Berthold K. P. Horn (1984)
- *Common Lisp: The Language*, Guy L. Steele, Jr. (1990).
- *APL: An Interactive Approach*, by Leonard Goodman and Allen J. Rose. (1984).
- *Scheme and the Art of Programming*, by George Springer and Dan Friedman (1989).
- *Functional Programming: Application and Implementation*, by Peter Henderson (1980).
- *Abstraction and Specification in Program Development*, by Barbara Liskov and John Guttag (1986). The textbook for 6170, MIT's course in software engineering. Includes an overview of CLU and a CLU language manual.
- *Smalltalk-80, the Language*, by Adele Goldberg and Dave Robson (1985).
- *The Art of Prolog*, by Leon Sterling and Ehud Shapiro (1986).
- *Programming in Prolog*, by W.G. Clocksin, C.S. Mellish (1987).
- *On to Java*, by Patrick Henry Winston and Sundar Narasimhan (Addison-Wesley, 1996). This is a good and relatively short and inexpensive introduction to programming in Java. The *On to C* and *On to C++* books by these authors are also recommended for an introduction to these other languages.

4.3.4 Other Resources:

The Science Center Library houses many relevant books other than those on reserve. Use `http://luna.wellesley.edu/search` to search the collection.

The Barker Engineering Library at MIT (on the 5th floor of building 10, under the big dome) houses an extensive collection of computer science books. You can visit their on-line catalog at `http://library.mit.edu`.

5 Course Web Pages, Directories, and Conferences

All handouts and various course-related links can be found on the CS251 home page at:

`http://cs.wellesley.edu/~cs251`

The CS251 course directory is located on `cs.wellesley.edu` in the directory `/home/cs251`. This directory contains material relevant to the class, and is where the problem set drop folders are located. This semester, we will be using CVS to download problem set materials from `cs.wellesley.edu` (see Handout #4), so there is no reason to directly connect to `cs.wellesley.edu` for this purpose. However, in order to submit softcopies of your problem sets, you will need to use the Linux `scp` program to copy directories into the drop folders in `/home/cs251/drop`. Note that you can also copy files to/from `cs.wellesley.edu` using WinSCP on a PC or Fetch on a Mac.

Additionally, there is a course conference which will be used to make class announcements such as corrections to assignments and clarifications of material discussed in class. It is also a forum for you to post questions or comments. They will be answered by me or a classmate. This is also a good place to find people to form a study group.

6 Homework

6.1 Problem Sets

There will be weekly problem sets during the semester. These will include pencil and paper problems and programming problems. Programs will range from simple programs to substantial interpreters and translators. Since the best way I know of understanding programming languages is by modifying interpreters and translators, many problems will be along these lines.

Many of the assignments will be challenging. Keep in mind that programming often consumes more time than you think it will. **Start your assignments early!** This will give you time to think about the problems and ask questions if you hit an impasse. Waiting until the last minute to begin an assignment is a recipe for disaster.

Problem sets will typically be due on either Mondays or Thursdays at 5pm. You need to submit both a "hard" (paper) copy of your assignment as well as a "soft" (electronic) copy of any programs (so that I may test them if necessary). Hardcopies should be slipped under my office door.

I will strive to have problem sets graded as soon as possible. At this time, solutions will be distributed with the graded homework.

6.2 Collaboration Policy

I believe that collaboration fosters a healthy and enjoyable educational environment. For this reason, I encourage you to talk with other students about the course and to form study groups.

Because the programming assignments in this course are particularly challenging, you will be allowed on any assignment to form a two-person "team" with a partner. The two team members can (in fact, must; see below) work closely together on the assignment and turn in a single hard- and soft-copy of the assignment for the team. The grade received on such a submission will be given to both team members.

This is a rather unusual collaboration policy, and it is only allowed subject to the following ground rules:

- There are two kinds of problems on problem sets this semester: *group problems* and *individual problems*. Team members may collaborate only on group problems. Individual problems are effectively "take-home quizzes" that **must** be completed by each individual student without collaborating with anyone else.

- The work on group problems must be a true collaboration in which each member of the team will carry her own weight. It is *not* acceptable for two team members to split the group problems of an assignment between them and work on them independently. Instead, the two team members must actively work together on all parts of the assignment. In particular, almost all programming on the assignment should be done with the two team members working at the same computer. It is strongly recommended that both team members share the responsibility of "driving" (i.e., typing at the keyboard), swapping every so often.

  The fact that team members have to program together means that you need to carefully consider a potential partner's schedule before forming a team. You cannot be a team if you cannot find large chunks of time to spend at a computer together!

- You are encouraged to work with different partners on different assignments. Rotating through partners is a good way to build community in the class and is helpful for avoiding situations where one individual feels pressured to continue working with another.

- You are not *required* to have a partner on any assignment, but you are *encouraged* to do so. Based on past experience, working with a partner can significantly decrease the amount of time you spend on an assignment, because you are more likely to avoid silly errors and blind alleys. On the other hand, certain individual may take more time on an assignment than they would alone. In this case there are still benefits to working with a partner. but they may be outweighed by the time cost.

Unless otherwise instructed, teams are allowed to discuss the group problems (but **never** the individual problems) on problem sets with other teams and exchange ideas about how to solve them. However, there is a thin line between collaboration and plagiarizing the work of others. Therefore, I require that each (one-person or two-person) team must compose its own solution to each assignment. In particular, while you may discuss strategies for approaching the programming assignments with other teams and may receive debugging help from them, **each team is required to write all of its own code**. **It is unacceptable (1) to write a program with another team and turn in two copies of the same program or (2) to copy code written by other teams. Such incidents will be interpreted as violations of the Honor Code.**

In keeping with the standards of the scientific community, you must give credit where credit is due. If you make use of an idea that was developed by (or jointly with) others, please reference them appropriately in your work. E.g., if person X gets a key idea for solving a problem from person Y, person X's solution should begin with a note that says "I worked with Y on this problem" and should say "The main idea (due to Y) is ..." in the appropriate places. It is unacceptable for students to work together but not to acknowledge each other in their write-ups.

When working on homework problems, it is perfectly reasonable to use code from the textbooks and other materials handed out in class. It is also reasonable to consult public literature (books, articles, etc.) for hints, techniques, and even solutions. However, you must cite any sources that contribute to your solution. There is one extremely important exception to this policy: **assignments and solutions from previous terms of CS251 are *not* considered to be part of the "public" literature. You must refrain from looking at any solutions to problem sets or exams from previous semesters of CS251. It is my policy that consulting solutions from previous semesters of CS251 constitutes a violation of the Honor Code.**

6.3 Late Homework Policy

All problems set will be due on the advertised date and time Since the material in this course depends so strongly on what has come before, turning in one problem set late will make it difficult to turn in the next problem set on time. In addition, solutions to problems sets will be discussed in

lecture in order to build upon earlier work. For these reasons, late homework will not be accepted. If you have not completed the homework set, please turn in what you have finished by the due date.

In extenuating circumstances (e.g., sickness, personal crisis, family problems), you may request an extension without penalty. Such extensions are more likely to be granted if they are made before the due date.

6.4 Programming

We will be writing most programs in the following five programming languages. All except Java (which you already know) will be taught during the semester:

- OCAML will be used to (1) explore the function-oriented programming paradigm in a statically typed context and (2) implement interpreters and translators for various mini-languages that illustrate important programming language features or dimensions.
- SCHEME, a dialect of Lisp, will be used to function-oriented programming paradigm in a dynamically typed context.
- HASKELL will be used to explore programming in a purely functional language (no side effects at all) with "lazy" evaluation.
- JAVA will be used to explore object-oriented programming.
- C will be used to explore imperative programming, especially in the context of manual storage management.

The "default" place for you to work on your assignments will be at the CS Department's Linux workstations in the mini-Focus. Implementations of all five of the above languages are installed on them. Information about how to use these programming environments on these machines will be provided at relevant points during the semester. Documentation on these systems will be accessible from the CS251 web page.

In order to use the Linux workstations, you will need a Linux account. If you do not already have one, please ask me to create one for you.

There are many free implementations of the above languages that you can install on your personal computer, if you own one; see the documentation accessible from the CS251 home page for details. However, there are several reasons to prefer working on the Linux workstations:

- There are likely to be other students working on their CS251 there, increasing the probability of collaboration.
- Most alums say that getting Unix (of which Linux is an instance) experience is one of the most important skills you can get acquire while at Wellesley. You will lose your chance to get this experience if you stick with your Mac or PC (unless, of course, you install Linux on them).
- The programming language environments on the Linux workstations are the only ones I will officially support. There are often small differences between implementations that can cause many frustrating headaches. Although I can try to help with installing/using other systems, I will have very little time for such activities.

6.5 Saving Work

You have a limited amount of space on the CS department fileserver (`cs.wellesley.edu`, a.k.a. `puma`) to store your course-related files. Although regular backups are made on `puma`, sometimes files are irretrievably lost.[1] For this reason, you should **not** depend on `puma` as the only repository for your course files. You are also expected to keep backup copies of all your course work a flash drive and/or the hard disk of your own computer.

7 Exams

There is only one exam this semester: a Final Exam during the regular final examination period.

Although there will no dedicated take-home exams, the individual problems (as opposed to group problems) on problem sets effectively count as take-home quizzes. Most problem sets will have an individual problem. Keep in mind that you may not collaborate with anyone on an individual problem.

8 Grades

The course grade will be computed as shown below:

Group problems on problem sets (total)	40%
Individual problems on problem sets (total)	40%
Final exam	15%
Class participation	5%

The default ranges for grades are expressed as a percentage of total points (excluding extra credit points):

A	93.33 – 100
A-	90 – 93.32
B+	86.66 – 89.99
B	83.33 – 86.65
B-	80 – 83.32
C+	76.66-79.99
C	73.33-76.65
C-	70 – 73.32
D	60 – 69.99
F	below 60

I reserve the right to lower boundaries between grades, but I will not raise them. This means that I can grade on a curve, but only in your favor.

Keep in mind that this is a very challenging course in which a B is considered a *good* grade (not a slap in the face) and an A is an *outstanding* grade. In the past, the average grade in this course has tended to be in the B range.

The above information is intended to tell you how I grade. It is not intended to encourage a preoccupation with point accumulation. You should focus on learning the material; the grade will take care of itself.

[1] For example, several years ago a significant number of `puma` files were deleted by a hacker before they had been backed up.

9 Finding Help

If you have any questions at all about the class (whether big or small, whether on problem sets lectures, reading, or whatever) please contact me. **That's what I'm here for!**

The best time to see me is during my scheduled office hours (which are listed at the top of this handout). If these times are not convenient, we can set up an appointment at some other time. You can set up an appointment by talking with me in person, calling me on the phone, or sending me email. You can also ask questions by sending me email. I read my email on a regular basis, and will check it even more frequently in the few days before an assignment is due.

This year we are fortunate to have two excellent tutors, Karishma Chadha and Smaranda Sandu as tutors for the course. Their drop-in tutor schedule will be posted early in the semester.

If you are having trouble with the course, you can request a one-on-one tutor from the Pforzheimer Learning and Teaching Center (PLTC). This service is confidential and free of charge; please take advantage of it if you would like some extra help! Contact me or PLTC for more information about this service.

Finally, when looking for help, don't overlook your fellow students — not only those who have taken the course in the past, but your classmates as well. Your classmates are a valuable resource; make good use of them!

10 Students With Special Needs

If you have any disabillities (including "hidden" ones, like learning disabilities), I encourage you to meet with me so that we can discuss accommodations that may be helpful to you.