CS251 Programming Languages **Handout # 1**
Prof. Lyn Turbak **January 28, 2004**
Wellesley College

CS251 Course Information

1 Contact Information

Professor:	Franklyn Turbak (please call me "Lyn")
Office:	SCI E126
Phone:	x3049
E-mail:	`fturbak@wellesley.edu` ("Franklyn Turbak" in FirstClass)
Lectures:	SCI E111, Mon./Wed./Thu. 9:50–11am
Office Hours:	Monday: 11am–12:30pm Tuesday: 7–9pm Wednesday: 4– 6pm Friday: 11am–1:30pm Appointments can be made for other times. This semester, I will be spending most of Thursdays after class doing research at Boston University, so I will almost never be available for appointments on Thursday. However, I am fairly flexible in terms of meeting at other times. I will sometimes need to cancel or shift office hours to attend a meeting or at talk. I will post a message to `CS251-S04 Announcements` to announce changes to office hours.
Web Site:	`http://cs.wellesley.edu/~CS251`
First Class:	`CS251-S04 Announcements`, `CS251-S04 Q&A`
Tutor:	Jue Wang (Wed. 7-9pm)

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: One of the best ways to understand the design space of programming languages is to study interpreters — programs that implement one programming language (the *object language*) on top of another (the *implementation language*). Reading, modifying, and building interpreters are key activities in this course. We will use Scheme, ML, and Haskell as the implementation languages for various "toy" object languages

3 Prerequisites

The official 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 CS11 and CS230 – e.g., abstraction, modularity, recursion, iteration, abstract data types – in the context of the programming languages we study.

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) 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.

4.2 Optional Textbooks

There are three textbooks that are recommended (but not required) for the course:

- *Structure and Interpretation of Computer Programs*, 2nd edition (*SICP*) by Abelson and Sussman with Sussman (MIT Press, 1996). This is the textbook 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 a purely-functional, lazy programming language (Haskell).

These 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.

Because these books are expensive and they are not the chief source of information in the course, you are not required to purchase them. You can instead rely on copies in the Computer Science resource room (SCI 173) or copies on reserve in the Science Center Library.

4.3 Reserved Materials

Several books relevant to this course are either available in the Computer Science resource room (SCI 173) and/or on reserve in the Science Center Library. (Note: books in SCI 173 should only be used in 173 and the microfocus area.) They are listed below. I encourage you to become familiar with this collection and to consult it often.

4.3.1 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 langauge 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).

4.3.2 Books on Particular Programming Languages or Paradigms

- *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.

4.3.3 Other Resources:

The Science Center Library houses many relevant books other than those on reserve. An easy way to find out what's available is to consult the on-line library catalog. To do this, execute `telnet library` from a Linux shell.

The MIT Laboratory for Computer Science has an excellent computer science library on the first floor of building NE43 (also known as "Tech Square") on the MIT campus. This is an especially good place to find journals and technical reports. For an on-line catalog, telnet to `reading-room.lcs.mit.edu`.

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. The on-line catalog is accessible by telneting to `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 the following URL:

`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. From Linux FTP, Fetch, or Winsock-FTP, the CS251 directory can be accessed by connecting to `cs.wellesley.edu` and navigating to `/home/cs251`.

Additionally, there is a `CS251-S04` conference in FirstClass with two subconferences:

- `CS251-S04 Announcements` will be used to make class announcements, such as corrections to assignments and clarifications of material discussed in class.
- `CS251-S04 Q&A` is 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.

You should plan on reading the CS251 conferences on a regular basis. It is strongly recommended that you add both subconferences to your FirstClass desktop

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 written in the OCAML, SCHEME, and HASKELL. Since the best way I know of understanding programming languages is by modifying interpreters, 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 Thursday at 6pm. 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.

Problem sets will be graded on a 100 point scale. 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:

- The work 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 problems of the 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 can only work with a given partner on a single problem set during the semester. So if you want to continue to collaborate, you must choose a different partner for every assignment. 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 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 previous semesters of CS251. It is my policy that consulting solutions from previous terms of CS251 constitutes a violation of the Honor Code.

6.3 Late Homework Policy

I realize that it is not always possible to turn in problem sets on time. On the other hand, turning in one problem set late can make it more difficult to turn in the next problem set on time. I have decided on the following policy for this course this term:

> All problem sets will be due by 6pm on the due day (typically Thursday) A problem set can be turned in $24 \cdot n$ hours late if it is accompanied by n Lateness Coupons. If you work with a partner, each of you needs to attach one Lateness Coupon per person per day late.

At the end of this handout, you will find ten Lateness Coupons that you can use throughout the term. Use them wisely: you only get ten, and they are not copyable or transferable between students. (You also cannot use them on exams!)

You may turn in late problem sets by slipping them under my office door. Of course, if I post solutions before you turn in a late problem set, you are bound by the Honor Code not to examine these solutions.

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 Problem Set Header Sheets

I would like to get a sense for how much time it takes you to do your CS251 problem sets. I use this information to design problem sets later in the semester, as well as for future semesters.

Please keep track of the time you spend on each problem of your problem sets, and include this information on the problem set header sheets that I will provide at the end of each problem set. (Two time columns will be provided for the case of students working together on an assignment.) Turn in this header sheet as the first page of your hardcopy submission.

6.5 Extra Credit

This semester I do not plan to give any "official" extra credit problems. But if I am impressed by work that you have done on challenging problems related to the course, I may give extra credit points for such work.

6.6 Programming

We will be writing most programs in three programming languages, all of which 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 for various "toy 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.

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 OCAML, SCHEME, and HASKELL 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 Lyn to create one for you.

There are many free implementations of OCAML, SCHEME, and HASKELL 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 OCAML, SCHEME, and HASKELLenvironments 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.

6.7 Saving Work

Each CS251 student has a password-protected account on the CS Dept file server, `cs.wellesley.edu` (also known as `puma`). You will have a limited amount of space on this server to store your course-related files.

You are also expected to keep copies of all your course work on floppy disks or zip disks. Removable disks are a frail medium that you should handle carefully. Store and transport them in suitable protected containers. Do not subject them to temperature extremes, put them near magnetic fields, store them unprotected in your pockets, etc. Even if you handle floppy and zip disks carefully, they are still prone to failure. For this reason, you should regularly back up your floppy or zip disks!

Every time you insert a disk into a computer, you may be transmitting a computer virus! Viruses are nasty software fragments that can erase information on your computer or cause other malfunctioning. In order to reduce the spread of computer viruses, make sure that any personal computers you use have appropriate virus protection software installed.

7 Exams

There will be three exams, both open book and open notes:

1. Take-home Exam 1 will be handed out on Thu. Feb. 26 and will be due at 6pm on Thu. Mar. 4.

2. Take-home Exam 2 will be handed out on Tue. Apr. 20 and will be due at 6pm on Tue. Apr. 27.

3. There will be a Final Exam during the regular exam period.

Please mark these dates in your calendars. You are not allowed to collaborate with anyone else on any of the exams.

8 Grades

The course grade will be computed as shown below:

Problem sets (total)	50%
Take-home Exam 1	20%
Take-home Exam 2	20%
Final exam	10%

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.

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.

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 Jue Wang as a tutor for the course. Jue will be holding drop-in hours from 7-9pm on Wed. nights. If you are having trouble with the course, you can request a one-on-one tutor from the Learning and Teaching Center (LTC). This service is confidential and free of charge; please take advantage of it if you would like some extra help! Contact me or LTC 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 accomodations that may be helpful to you.

LATENESS COUPONS

Below are ten Lateness Coupons. A problem set that is $24 \cdot n$ hours late must be accompanied with n Lateness Coupons in order to be accepted. That is, each coupon gives you one extra day to turn in a problem set. You may use them in any manner in which you wish – e.g., turn in every problem set one day late, or turn in one problem set ten days late. Lateness coupons are not transferable between students, and may not be used on exams.

CS251 Lateness Coupon #1
CS251 Lateness Coupon #2
CS251 Lateness Coupon #3
CS251 Lateness Coupon #4
CS251 Lateness Coupon #5
CS251 Lateness Coupon #6
CS251 Lateness Coupon #7
CS251 Lateness Coupon #8
CS251 Lateness Coupon #9
CS251 Lateness Coupon #10