CS301 Compiler Design and Implementation **Handout # 1**
Prof. Lyn Turbak **September 2, 2003**
Wellesley College

CS301 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 E211, Tue./Wed./Fri. 8:30–9:40am
Web Site:	`http://cs.wellesley.edu/~CS301`
First Class:	`CS301-F03 Announcements`, `CS301-F03 Q&A`
Office Hours:	Tue. 9:50-noon Wed. 4-6pm Fri. 9:50-noon Appointments can be made for other times. This semester, I am unavailble on Monday and Tuesday afternoons and all of Thursdays. However, I am fairly flexible in terms of meeting at other times. If you have questions when I am not around, please email me or post to `CS301-F03 Q&A`.

2 Course Overview

An **interpreter** is a program written in one language that executes programs in another (possibly the same) language. A **compiler** is a program that translates programs from one language to another, typically from a high-level language (such as JAVA or SCHEME) to a low-level language (such as Pentium machine code). These are two ends of a spectrum: programming language implementations often consist of several stages that involve aspects of both. For instance, JAVA is compiled to so-called bytecodes that can be executed on any processor running a Java Virtual Machine (JVM) interpreter. Microsoft's .NET architecture is another example of this "compile once/run everywhere" architecture. Many compilers are actually a composition of translators. For example, you can translate Scheme to C, and then compile this with a C-to-Pentium compiler.

In this course, you will learn about the many stages of interpreters and compilers and the techniques and tools for implementing these stages. This semester we will emphasize programming language implementations for robot controllers like the HandyBoards and LogoChips used in *CS115: Robotic Design Studio* and *PHYS 219: The Art of Electronics*. We will also consider other target architectures.

There are many different dimensions of the course. Among the most important are:

Theory: A significant corpus of theory has been developed to describe each of the many stages of compilation. Examples include automata theory for lexical analysis, context free grammars for parsing, type theory for type checking, and fixed point theory for data flow analysis. We will study the theory associated with the stages and learn about many standard algorithms that have been developed for them (e.g., scanning, parsing, type checking, instruction selection, register allocation).

Practice: You will be getting significant hands-on experience with simple interpreters and compilers by modifying existing ones and building others from scratch. We will follow an

onion skin approach in which we begin with a complete implementation of a very simple source language, and then use more sophisticated techniques as we extend the source language with more complex features.

Tools: A number of tools have been developed to simplify the process of constructing and testing compilers. We will be using two standard tools: `ocamllex`, a lexical analyzer generator that simplifies the construction of a lexical scanner; and `ocamlyacc`, a parser generator that facilitates parser construction. We will also use the SPIM simulator for the MIPS computer architecture, and several tools developed by Jue Wang '04 for programming the HandyBoard.

Programming Language Design: As an implementor of programming languages, you will gain new appreciation for various aspects of programming language design.

Software Engineering: Compilers are infamous for their complexity. They tend to be very big programs constructed out of many parts. There are usually many ways to implement each part, with lots of tradeoffs involved (e.g. time vs. space, simple vs. complex). Furthermore, the compiler must handle a plethora of tiny details in order to correctly implement the source language.

All of this means that constructing a compiler is a significant software engineering project. In order to manage the complexity of the compiler, we will use a combination of standard tools and algorithms and good programming methodology based on modularity and abstraction. Our implementation language (OCaml) is particularly well-suited for supporting good programming methodology.

3 Prerequisites

The official prerequisites for *CS301* are *CS240, Machine Organization*, and *CS251, Programming Languages*:

- From *CS240*, you are assumed to have familiarity with assembly language and low-level computer organization (binary representations of data and instructions, registers, stacks, memory, etc.). This knowledge will be helpful for understand code generation and other issues in the back end of the compiler. You are not assumed to be familiar with the 68HC11 (HandyBoard) or MIPS assembly languages we will be using this semester; that will be taught in class.

- From *CS251* and/or previous programming experience, you are assumed to have some familiarity with functional programming (say in Scheme). This will be helpful for programming in OCaml, which will be our implementation language this semester. Understanding various features of the programming languages we will be compiling (e.g., namespaces, scope of names, types, data structures, etc.) will also be important. These topics will be reviewed/taught as necessary.

Although *CS231 (Algorithms)* and *CS235 (Languages and Automata)* are not official prerequisites, we will be making use of some of the ideas from these courses (e.g., automata, grammars, graph algorithms), which will be introduced on an as needed basis.

Also helpful are (1) significant programming experience and (2) experience in working with the Linux operating system. Don't worry: if you don't have these now, you will by the end of the course!

It's pretty clear from the above discussion that the compiler course depends on a lot of other material. For this reason, it is often referred to as a **capstone course** - one that nicely ties together the material from the rest of the computer science curriculum.

4 Ten Reasons You Should Take CS301 This Fall

1. The coolness factor of this course is off the charts! Really – how many times do you get a chance to learn how to be a programming language designer and implementer and to control devices like the HandyBoard?

2. The course removes the shroud of mystery surrounding how high-level software can be implemented on low-level hardware.

3. Writing programs that manipulate programs is the most interesting kind of programming!

4. For the first time in Fall '03, CS301 will focus on robot control languages.

5. As noted above, it's a great capstone experience to tie together much of your undergraduate CS experience. CS301 has connections to programming languages, computer architecture, algorithms, automata, software engineering, operating systems, etc.

6. There are practical benefits to taking the course:

   - If you plan to get any job that involves programming, or if you plan to go to grad school, you will be expected to know this material.
   - Even if you're not planning on this, the material is so central to CS that you should know it.
   - It will open up research opportunities to you. Handy Logo/Cricket/ LogoChip development is done here at Wellesley, at the MIT Media Lab, and at UMass Lowell. If you learn the details of these systems, you can do summer internships, UROPS, thesis projects, and independent study projects involving them. BTW, this will make you more marketable for industry and grad school.

7. Fall'03 your only chance to take this course at Wellesley. It's only offered every 2-3 years. *CS249 Networks* is also in this category, but *CS235 Languages and Automata* and *CS307 Graphics* are offered every fall.

8. Programming language design and implementation is my main research area. It's what I love! I want to share my knowledge, and more importantly the sense of excitement and fun that I have for this area, with you.

9. I've never had many of you as students and would like to get to know you better. Besides, you haven't had the complete Wellesley CS experience unless you've taken a course with me!

10. Did I mention that I bake cookies and brownies for my students?

5 Reading Materials

5.1 Notes

The material presented in *CS301* 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. We will also depend heavily on a number of resources available on the Internet; these are listed in the **Resources** section of the *CS301* web page.

5.2 Books

In the CS Lounge (room SCI 173), I will maintain a "Compiler Corner" – a set of shelves with books related to compilation and programming languages. You are encouraged to browse through these books frequently. Here are some books that are particularly worthwhile:

- *Modern Compiler Implementation in ML* by Andrew Appel (Cambridge University Press, 1998) was used as a textbook in the Fall 2000 incarnation of CS301. While this book embodies a much more modern approach to compilation than many other texts, it is rather uneven. Some sections are very good while others are terse to the point of being incomprehensible.

- *Compilers: Principles, Techniques, and Tools*, by Alfred Aho, Ravi Sethi, and Jeffrey Ullman (Addison-Wesley, 1986) is called the "dragon book" because of its cover. Although somewhat dated, this book is considered by many to be canonical compiler book.

- *Advanced Compiler Design and Implementation*, by Steven S. Muchnick (Morgan Kaufman, 1997) is a relatively new book with encyclopedic coverage of many compiler topics.

- *The Functional Approach to Programming* by Guy Cousineau and Michel Mauny (Cambridge University Press, 1998) is a nice introduction to functional programming in CAML.

- *ML for the Working Programmer*, 2nd edition by Lawrence Paulson (Cambridge University Press, 1996). This is a good resource for learning how to do higher-order typed programming in STANDARD ML(SML), a different dialect of ML than OCAML. It contains many nice examples of typeful programming, higher-order functions, immutable data structures, and interpreters. Although there are a few syntactic differences between SML and OCAML, they are close enough for you to get a lot out of this book.

5.3 Other Resources

There are a number of other resources that you may find handy during the semester:

- The *CS301* home page (`http://cs.wellesley.edu/~cs301`) contains a **Resource Links** section with links to on-line materials relevant to many parts of the course.

- The ACM digital library (`http://portal.acm.org/dl.cfm`) is a great place to search for technical papers on compilation and programming languages. You can do simple searches for free; contact me if you want to do advanced searches or download articles.

- The Science Center Library houses many relevant books on compilers and programming languages. 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 machine.

- 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`.

6 Course Web Pages, Directories, and Conferences

All handouts and various course-related links can be found on the **CS301 home page** at:

`http://cs.wellesley.edu/~cs301`

The **CS301 course directory** is located on `cs.wellesley.edu` in the directory `/home/cs301`. 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 CS301 directory can be accessed by connecting to `cs.wellesley.edu` and navigating to `/home/cs301`.

Additionally, there is a `CS301-F03` conference in FirstClass with two subconferences:

- `CS301-F03 Announcements` will be used to make class announcements, such as corrections to assignments and clarifications of material discussed in class.
- `CS301-F03 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 CS301 conferences on a regular basis. It is strongly recommended that you add both subconferences to your FirstClass desktop

7 Class Work

7.1 Problem Sets

There will be weekly problem sets during the first two thirds of the semester. The main focus of most problem sets will be part of a programming language implementation project, but there will sometimes be pencil and paper problems as well.

Most of the assignments will be rather 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 a Friday. 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).

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.

7.2 Final Project

During the last month of the course, you will focus on a programming language implementation project of your own choosing. There are many kinds of projects you can undertake. For example:

- Experiment with program optimization techniques.
- Extend one of the source languages we study in class with a new feature and implement it. E.g. add ML-style dataypes and pattern-matching to a language.
- Explore the techniques and tradeoffs in some portion of the programming language design space – e.g., the pros and cons of lazy data structures, different approaches to concurrency, various garbage collection techniques, etc.
- Implement a tool that will be useful to future CS301 students – e.g., a 68HC11 simulator or a convenient way to read and display the 68HC11 memory map.

We will discuss possible projects in a brainstorming session near the end of October, and you will have time to discuss possible projects before you choose one.

As part of the final project, you will have to:

- Complete whatever implementation work is required for your project.
- Submit a write-up describing your implementation project.
- Give a brief talk to the class about your project.

7.3 Exams (or Lack Thereof)

There are no exams in this course. WooHoo!

7.4 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!

- In order to encourage people to swap partners, you are not allowed to work with the same partner on two straight 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.

- I have not decided what to do about partners in the context of the final project. If you have a compelling reason to work with a partner for the final project, I will consider this as an option.

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 materials handed out in class. It is also reasonable to consult public literature (books, articles, web pages 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 CS301 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 CS301 It is my policy that consulting solutions from previous terms of CS301 constitutes a violation of the Honor Code.

7.5 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 at the advertised time (typically 5pm on a Friday). A problem set can be turned in n days 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 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.

7.6 Problem Set Header Sheets

I would like to get a sense for how much time it takes you to do your CS301 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.

7.7 Extra Credit

To make up for points lost on problem sets and exams, students often request extra credit problems. In order to give everyone the same opportunity, I will sometimes include extra credit problems on the problem sets. The extra credit problems will often be more difficult than the other problems, but they provide the opportunity to earn extra points toward your course grade. You should only attempt extra credit problems after completing the assigned problems.

Extra credit problems are entirely optional. Extra credit points will only be factored into course grades after I have partitioned the grade scale into letter grades. Thus, doing he extra credit problems may raise your course grade, but not doing extra credit problems will not lower your course grade.

For maximum flexibility, you may turn in extra credit problems at any time during the term (through the end of finals week). However, experience has shown that students who leave extra credit problems until the end of the term rarely turn them in. It is in your best interest to complete extra credit problems in a timely fashion. I will not hand out solutions to extra credit problems, but you are encouraged to discuss them with me in person.

7.8 Grades

The course grade will be computed as shown below:

Problem sets (total)	65%
Final Project	35%

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. If you are dissatisfied with the grade you will receive based on the above scale, I encourage you to turn in extra credit problems to raise your grade.

8 Programming/Computers

We will be using the OCaml programming language to implement interpreters and compilers this semester. This is a language that will be new to almost all of you. The first few lectures and first problem set will be devoted to bringing you up to speed in OCaml.

You will also be learning to program in several assembly languages this semester:

- 68HC11 assembly language is used to program the HandyBoards.
- We will use MIPS assembly language as an example of a more full-featured assembly language than that for the 68HC11. We will use the SPIM simulator for simulating MIPS on our Pentium-based machines.
- If time permits, we will also consider the LogoChip and Pentium as compiler targets. If not, investigating one of these would be a good final project!

All programming will be done on Linux workstations, which are located in the micro-focus, the nano-focus (the Lyn-ex lab), and the hardware lab (E125). These machines are configured so that (1) you can log into them with your CS username and password (2) you can access your CS home directory in `/students` "transparently" on any Linux machine. If you are not familiar with the Linux environment, browse the on-line Linux documenation, or ask me or a classmate for a tutorial. Information about how to use OCaml on the Linux machines is provided in a separate handout.

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.

The "default" place for you to work on your assignments will be at the CS Department's Linux workstations. However, you need not be sitting in front of a Linux machine console in order to work on your assignments. You can connect remotely to any of the department's Linux machines via a client like `telnet` or `ssh` and do your work remotely. Alternatively, if you have a personal computer, you can install Linux and the course software on it.

Even though you can do much of your work in this class remotely, there are still a compelling reason to work in the Science Center: there are likely to be other students working on their CS301 there, increasing the probability of collaboration.

8.1 Saving Work

Each CS301 student will save her work in her password-protected account on the CS Dept file server, `cs.wellesley.edu`. 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.

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. 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.

Finally, when looking for help, don't overlook your fellow students. 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 n days 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. If you are a member of a two-person team and are turning in a late assignment, *each* member must turn in one coupon for each day late.

CS301 Lateness Coupon #1
CS301 Lateness Coupon #2
CS301 Lateness Coupon #3
CS301 Lateness Coupon #4
CS301 Lateness Coupon #5
CS301 Lateness Coupon #6
CS301 Lateness Coupon #7
CS301 Lateness Coupon #8
CS301 Lateness Coupon #9
CS301 Lateness Coupon #10