

# Computer Science 353—Architecture and Compilers

Fritz Ruehr • Willamette University • Fall 2010

## Introduction and goals

Learning to program in Java can be fun and empowering, but it may still leave you mystified about how computers really work: what are the components of a physical computer? How does the computer “know” what to do when it runs a Java program? What connects abstract ideas like objects and methods to the silicon hardware?

This hybrid course in computer architecture and high-level language processing (compilers) will provide you with the answers to these questions. The course is a “soup-to-nuts” overview of the various levels of structure embodied in a physical computer, as well as the various stages by which high-level languages are translated into a form which the computer can directly execute. In traditional curricula, these two topics are often covered in separate courses—we have combined them into a single comprehensive overview in order to expedite Willamette’s pedagogical goals.

We will start our journey with basic facts about binary encodings, how they are used to represent data and how logical operations can be used to transform these data. We will then see how basic electrical components called gates can be used to realize these operations on digital signals, and how they can be combined into circuits which implement more complex processes. Finally, we will see how these circuits are used to build higher-level components which constitute a typical computer, and how digital data stored inside the computer can be used to control its operation. In the second phase of our studies, we will start with the strings of characters which comprise a program file, and see how they can be analyzed in successive stages until the internal structure of the program is exposed. We will then see how this internal structure can be translated into the low-level codes which a computer can execute, in a way which respects the meanings of the original program. Finally, as time permits, we will survey a few important concepts which inform the design of high-level languages both like and unlike Java.

## Instructor

*Name:* Fritz Ruehr

*Office:* 208 Ford Hall

*Phone:* 370-6165

*E-mail:* fruehr@willamette.edu

*Course homepage:* <http://www.willamette.edu/~fruehr/353>

*Office hours:* Th 1-3; by appt.; or just drop by!

## Logistics and attendance

<i>Lectures</i>	MWF 12:40pm-1:40pm	Ford 204
<i>Lab</i>	MF 3pm-4pm	Ford 224 ( <i>Math Windows lab—East end</i> )
<i>Lab</i>	W 3pm-4pm	Ford 202 ( <i>CS Mac lab—West end</i> )
<i>Lab</i>	W 10:20am-11:20am	Ford 202 ( <i>ditto</i> )

Students are expected to attend all lectures, and lab sessions for help or to demo their labs. You are in any case responsible for all content, changes in assignments or policies, etc., which are made during the course of scheduled classes. I will try to make important announcements available on the course homepage or by e-mail.

***NB:*** *participation in class discussions may be a factor in determining course grades!*

**Textbook** (*or rather lack thereof...*)

Due to the non-traditional, hybrid nature of this course, it is difficult to find suitable textbooks: if we were to use traditional texts, we would likely need at least two expensive books, but would use only a

fraction of the material in each. In order to save you money, and because some excellent on-line resources are available on these topics, we will not be making use of a traditional text for study in this course. *This means that your regular attendance and participation in lectures and labs is crucial to your success in the class.* On-line resources will be linked from the course home page, and pointers to relevant materials, sites and tools will be provided regularly during lecture. I will also provide handouts and topical summaries to help you in your studies for the course.

## Grading policy

Grades will be based on programming projects, exams, in-class quizzes and written homework, and class participation. Individual grades will be given in numeric form and then combined to determine a composite at the end of the semester, according to weights given below. I grade on a modified curve basis—that is, I grade students relative to each other's performance, but not necessarily to fit a normal curve. I am happy to give out all high grades (and will give out all low grades if necessary), depending on the absolute performance of the class as a whole.

Grading of programming projects will be done in the lab, based on an interactive “demo” by the student for the instructor. This approach allows you to make some adjustments to your program in response to problems I find. We feel that this is a flexible and humane approach to the grading process which maximizes interaction between students and faculty.

I plan to give one mid-term examination and a longer, comprehensive final exam during the regular scheduled period. I may also give some in-class quizzes and written work to assess how the class is following along with lectures. Finally, I will assign 10 programming labs. The weights used for the final grade will be:

- 45% divided evenly among the programming projects;
- 10% split evenly between any quizzes or written homework;
- 20% for the mid-term exam;
- and 25% for the final exam;
- with class participation adjusting the final grade by as much as one “±” letter-grade modifier.

## Exams

The final exam is scheduled for **Monday, Dec. 13, 2010, from 2-5 pm**. The mid-term exam date will be announced in lecture.

*(Note: please make your holiday travel plans accordingly!)*

## Due dates

Generally speaking, programming projects will be due a week or two after they are assigned—on occasion, class-wide extensions may be announced for various reasons (difficulties with software tools, etc.). If you think you will be unlikely to be able to complete an assignment, contact the instructor as soon as possible regarding the problem; extensions may be granted for official business, emergencies, etc. It is always better to ask for an extension before the due date in question, although I recognize that this may not always be feasible.

## Collaboration and related issues

All exams and programming projects you hand in should be your own work. You are allowed (and encouraged) to seek help from other students for general study purposes, but you should never allow other people to do your work for you. You are also responsible for attributing any material you quote from outside sources (e.g., please leave all copyright and other attributions on Java library files, and clearly distinguish your own work from these resources).

Violations of these rules will result in penalties according to usual College policies, but would normally include **at least a zero for the assignment**, and likely **a failing grade for the course**, depending on the severity of the infraction.

## Tentative topic list

I hope to cover the following list of topics this semester at a rate of from 1 to 3 lectures per topic. Listen for classroom announcements or see the course homepage for any changes as the need develops.

- Introduction to the course
- Boolean logic
- Data representation
- Hardware implementation of logic (transistors and gates)
- Combinatorial circuits
- Sequential circuits (specifically, latched storage)
- The CPU and its components
- Instruction set architecture
- Machine code programming
- Assembly language programming
- The assembly process
- Programming in C, a low-level programming language *[depending on time available]*
- High-level languages and compilation (overview)
- Tokens and scanning
- Regular expressions and deterministic finite automata
- Syntax trees and parsing
- Context-free grammars
- Parsing techniques
- A simplified expression language
- Code generation
- Other high-level language features *[depending on time available]*
- The Java virtual machine *[depending on time available]*

## Accreditation information

In accordance with new college accreditation initiatives, I am listing here the **student learning objectives, outcome indicators** and **measuring devices** associated with this class (SLO numbers used here are relative to CS Department norms):

- **CS/SLO #2:** “Students will achieve proficiency in Computer Science skills (fundamentals of programming, computer organization, architecture, algorithms, theory, designing and implementing software).”
- **CS/SLO #5:** “Students will demonstrate the ability to work independently to analyze and solve problems.”

Outcome indicators and measuring devices will commence in the next teaching of the course and will follow the usual CS Department standards of pre-test and post-test using questions based on the topic list appearing above.