CS 536

Introduction to Programming Languages and Compilers

Charles N. Fischer

Spring 2008

http://www.cs.wisc.edu/~fischer/cs536.html

Class Meets

Mondays, Wednesdays & Fridays,
1:20 — 2:10

1325 Computer Sciences

Instructor

Charles N. Fischer

6367 Computer Sciences

Telephone: 262-6635

E-mail: fischer@cs.wisc.edu

Office Hours:

10:30 - Noon, Tuesdays & Thursdays,
or by appointment

Teaching Assistant

Min Qiu
5393 Computer Sciences
E-mail: qium@cs.wisc.edu
Telephone: 608.262.5340

Office Hours:

Monday: 9:15 - 10:45,

Wednesday: 4:00 - 5:30
or by appointment

Key Dates

- February 4: Assignment #1 (Symbol Table Routines)
- February 27: Assignment #2 (CSX Scanner)
- March 28: Assignment #3 (CSX Parser)
- April 2: Midterm Exam (tentative)
- April 21: Assignment #4 (CSX Type Checker)
- May 9: Assignment #5 (CSX Code Generator)
- May 17: Final Exam 7:45 am-9:45 am

Class Text

- Draft Chapters from:

  Crafting a Compiler, Second Edition.

  (Distributed directly to class members)
- Handouts and Web-based reading will also be used.

Reading Assignment

- Chapters 1-2 of **CaC** (as background)

Class Notes

- The transparencies used for each lecture will be made available prior to, and after, that lecture on the class Web page (under the "Lecture Nodes" link).

Instructional Computers

Departmental Unix (Linux) Machines (king01- king12, emperor01-emperor40) have been assigned to CS 536. All necessary compilers and tools will be available on these machines.

You may also use your own PC or laptop. It will be *your* responsibility to load needed software (instructions on where to find needed software are included on the class web pages).

The Systems Lab teaches brief tutorials on Unix if you are unfamiliar with that OS.

Academic Misconduct Policy

- You must do your assignments—no copying or sharing of solutions.
- You may discuss general concepts and Ideas.
- All cases of Misconduct *must* be reported to the Dean's office.
- Penalties may be severe.

Program & Homework Late Policy

- An assignment may be handed in up to one week late.
- Each late day will be debited 3%, up to a maximum of 21%.

Approximate Grade Weights

Program 1 - Symbol Tables 5%
Program 2 - Scanner 12%
Program 3 - Parser 12%
Program 4 - Type Checker 12%
Program 5 - Code Generator 12%
Homework #1 9%
Midterm Exam 19%
Final Exam (non-cumulative) 19%

Partnership Policy

- Program #1 and the written homework must be done individually.
- For undergraduates, programs 2 to 5 may be done individually *or* by two person teams (your choice). Graduate students must do all assignments individually.

Compilers

Compilers are fundamental to modern computing.

They act as *translators*, transforming human-oriented *programming languages* into computer-oriented *machine languages.*

To most users, a compiler can be viewed as a "black box" that performs the transformation shown below.

A compiler allows programmers to ignore the machine-dependent details of programming.

Compilers allow programs and programming skills to be *machine-independent*.

Compilers also aid in detecting programming errors (which are all too common).

Compiler techniques also help to improve computer security. For example, the Java Bytecode Verifier helps to guarantee that Java security rules are satisfied.

Compilers currently help in protection of intellectual property (using *obfuscation*) and provenance (through *watermarking*).

History of Compilers

The term *compiler* was coined in the early 1950s by Grace Murray Hopper. Translation was viewed as the "compilation" of a sequence of machine-language subprograms selected from a library.

One of the first real compilers was the FORTRAN compiler of the late 1950s. It allowed a programmer to use a problem-oriented source language.

Ambitious “optimizations” were used to produce efficient machine code, which was vital for early computers with quite limited capabilities.

Efficient use of machine resources is still an essential requirement for modern compilers.

Compilers Enable Programming Languages

Programming languages are used for much more than "ordinary" computation.

- TeX and LaTeX use compilers to translate text and formatting commands into intricate typesetting commands.
- Postscript, generated by text-formatters like LaTeX, Word, and FrameMaker, is really a programming language. It is translated and executed by laser printers and document previewers to produce a readable form of a document. A standardized document representation language allows documents to be freely interchanged, independent of how

they were created and how they will be viewed.

- Mathmatica is an interactive system that intermixes programming with mathematics; it is possible to solve intricate problems in both symbolic and numeric form. This system relies heavily on compiler techniques to handle the specification, internal representation, and solution of problems.
- Verilog and VHDL support the creation of VLSI circuits. A *silicon compiler* specifies the layout and composition of a VLSI circuit mask, using standard cell designs. Just as an ordinary compiler understands and enforces programming language rules, a silicon compiler understands and enforces the design rules that dictate the feasibility of a given circuit.

- Interactive tools often need a programming language to support automatic analysis and modification of an artifact.
  How do you *automatically* filter or change a MS Word document? You need a text-based specification that can be processed, like a program, to check validity or produce an updated version.