PROGRAMMING LANGUAGES & TRANSLATORS

Baishakhi Ray

Fall 2018

Instructor

Prof. Baishakhi Ray

rayb@cs.columbia.edu

http://rayb.info

CEPSR 604

Office Hour: Monday 3pm-4pm/by Appointment

Prof. Stephen A. Edwards and Prof. Ronghui Gu also teach 4115

* Borrowing a lot of materials from Prof. Edward

Goals

- Theory
  - Principle of modern programming languages
  - Fundamentals of compilers: parsing, type checking, code generation
  - Traditional Usage: register allocation, optimization, etc.
  - Modern Usage: static analysis is bug detection, OO compilation, etc.

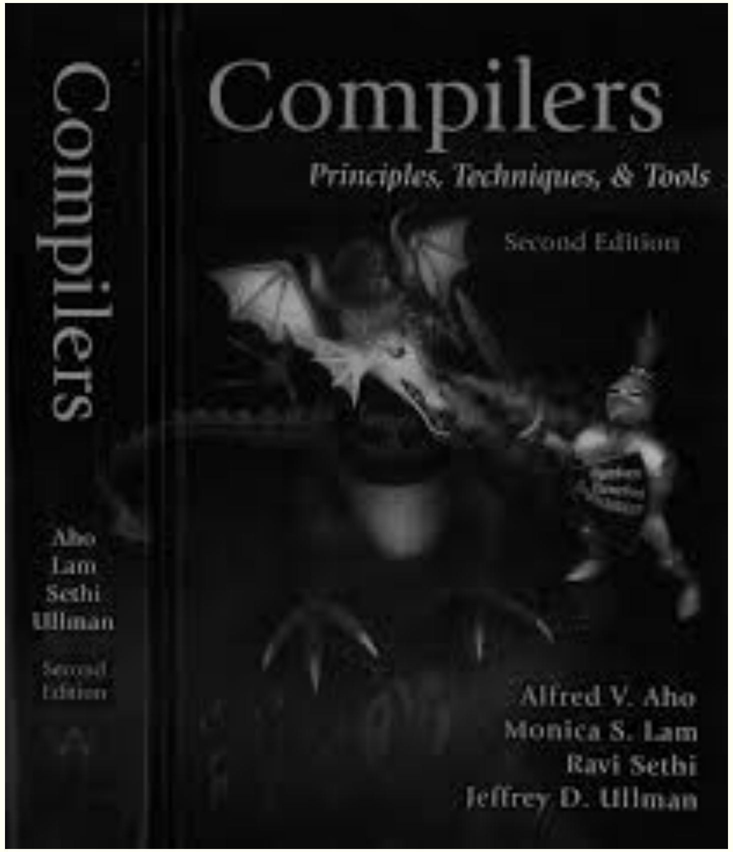
- Practice: Semester-long team project
  - Design and implement your own language and compiler
  - Code it in the OCaml functional language
  - Manage the project and your teammates; communicate

Recommended Text

- Compilers: Principles, Techniques, and Tools
  - By Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.
  - 2nd Edition
  - Addison-Wesley, 2006

- **Programming Languages: Application and Interpretation**
  - By Shriram Krishnamurthi
  - PDF for reading online
    PDF for printing (2 pages per sheet)

- Research Papers
  - Distributed by the instructor



Assignments and Grading

Team Programming Project	**40%**
Midterm Exam	20%
Final Exam (cumulative)	30%
Three individual homework assignments	10%
Effort*	0%

∗Do or do not; there is no try —Yoda

Team project is most important, but most students do well on it.
Grades for tests often vary more.

Schedule

- Lectures:
  - Mondays and Wednesdays, 1:10 PM-2:25 PM @ 703 Hamilton Hall
  - September 5 – December 10

- Exams:
  - Midterm : October 17
  - Final: December 10

- Team Project:
  - Presentations: TBD (All team members must present)
  - Report due date: December 19

Prerequisites

- COMS W3157 Advanced Programming
  - How to work on a large software system in a team
  - Makefiles, version control, test suites
  - Testing will be as important as coding

- COMS W3261 Computer Science Theory
  - Regular languages and expressions
  - Context-free grammars
  - Finite automata (NFAs and DFAs)

Collaboration

- Collaborate with your team on the project.
- Do your homework by yourself.
- Tests: Will be closed book with a one-page "cheat sheet" of your own devising.

Don't be a cheater (e.g., copy from each other).
If I catch you cheating I will send you to the dean.

TEAM PROJECT

The Team Project (Same as Section 1)

- Design and implement your own little language.
- Seven deliverables:
  1. A proposal describing your language
  2. A language reference manual defining it formally
  3. An intermediate milestone: compiling “Hello World.”
  4. A compiler for it, running sample programs
  5. Running a small optimization pass.
  6. A final project report
  7. A final project presentation

Teams

- Immediately start forming four-person teams
- Each team will develop its own language
- Each team member should participate in design, coding, testing, and documentation
- Choose one team member to head specific tasks:

Role	Responsibilities
Manager	Timely completion of deliverables
Language Guru	Language design
System Architect	Compiler architecture, development environment
Compiler Architect	Architect the optimization plan

- Cover for flaky teammates.
  - They will thank you later by completely reforming their behavior, making up for all the times you did their work for them.
  - Assign the least qualified team member to each task.
- Avoid leadership
  - include every feature and make all decisions by arguing.
  - Never let anybody take responsibility for anything.
  - Write software communally so nobody is ever at fault.
- Never tell the instructor or a TA that something is wrong with your group. It will only lower your grade.

Start Early!!

How Do You Work In a Team?

- Address problems sooner rather than later
  - If you think your teammate's a flake, you're right
- Complain to me or your TA as early as possible
  - Alerting me a day before the project is due isn't helpful
- Not every member of a team will get the same grade
  - Remind your slacking teammates of this early and often

First Three Tasks

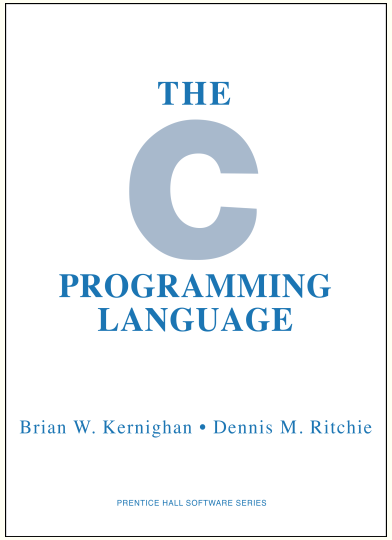
- Decide who you will work with
  - You'll be stuck with them for the term; choose wisely.
- Assign a role to each member
- Select a weekly meeting time

Project Proposal

- Describe the language that you plan to implement.
- Explain what sorts of programs are meant to be written in your language
- Explain the parts of your language and what they do
- Include the source code for an interesting program in your language
- 2–4 pages

Language Reference Manual

- A careful definition of the syntax and semantics of your language.
- Follow the style of the C language reference manual (Appendix A of Kernighan and Ritchie, The C Programming Language; see the class website).



THE C PROGRAMMING LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

PRENTICE HALL SOFTWARE SERIES

Final Report Sections

Section	Author
Introduction	Team
Tutorial	Team
Reference Manual	Team
Project Plan	Manager
Language Evolution	Language Guru
Translator Architecture	System Architect
Optimizer	Compiler Architect
Conclusions	Team
Full Code Listing	Team

Project Due Dates

Section	Author
Proposal	September 19 (soon)
Language Reference Manual and parser	October 15
Hello World Demo	November 14
Final Report	December 19

Design a Language?

- A domain-specific language: awk or PHP, not Java or C++.
- Examples from earlier terms:
  - Matlab-like array manipulation language
  - Geometric figure drawing language
  - Music manipulation language
  - Mathematical function manipulator
  - Simple scripting language (à lá Tcl)

Three Common Mistakes to Avoid

- Configuration File Syndrome
  - Must be able to express algorithms, not just data
  - E.g., a program like “a bird and a turtle and a pond and grass and a rock,” is just data, not an algorithm
- Standard Library Syndrome
  - Good languages express lots by a combining few things
  - Write a standard library in your language
  - Aim for Legos, not Microsoft Word
- Java-to-Java Translator Syndrome
  - A compiler adds implementation details to code
  - Your compiler’s output should not look like its input
  - Try your best not to re-invent Java

What I am Looking for

- Your language must be able to express different algorithms
  - Avoid Configuration File Syndrome. Most languages should be able to express, e.g., the GCD algorithm.
- Your language should consist of pieces that can mix freely
  - Avoid Standard Library Syndrome. For anything you provide in the language, ask yourself whether you can express it using other primitives in your language.
- Your compiler must lower the level of abstraction
  - Don’t write a Java-to-Java translator. Make sure your compiler adds details to the output such as registers, evaluation order of expressions, stack management instructions, etc.