Programming Languages & Translators

Instructor: Baishakhi Ray

COLUMBIA | ENGINEERING

The Fu Foundation School of Engineering and Applied Science

Instructor

Prof. Baishakhi Ray

Associate Professor

rayb@cs.columbia.edu

https://rayb.info

Office Hours: Tuesdays 2:00 pm - 3:00 pm

Location: CEPSR 6LE1

PLT 4115

- Lectures:
  - Tuesday and Thursday, 11:40 AM-12:55 PM @ CSB 451
  - September 3 – December 5
- Get the class updates in the website

- We will use Ed Discussion for class communication
  - See your coursework tab option

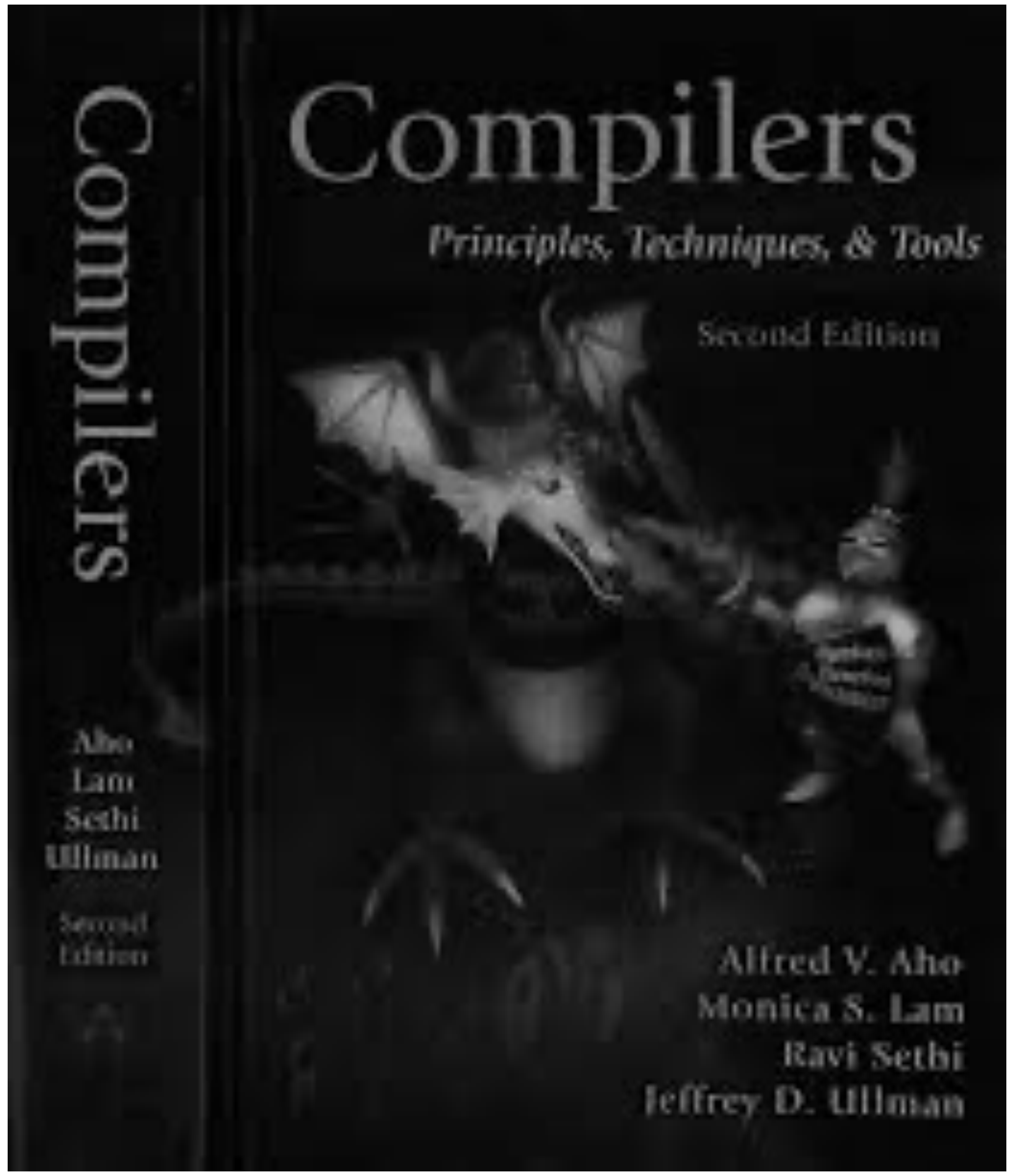
Programming Language & Translators

How can a computer program written in a high-level **programming language** (e.g., C, Python) be **translated** to a lower-level language (e.g., assembly language or machine code) to create an executable program?

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

- We will follow this book
  - but not line-by-line/section-by-section



This Class

- Theory: Learn different phases of a compiler design (50%)

- Practice: Implement a compiler (50%)
  - Implement different phases of compiler

This Class

Theory deliverables:

- Written assignments
- Midterm
- Final

	Lectures	Programming
1	Introduction	
2	Lexical Analysis	Prog-1
3	Syntax Analysis	Prog-2
4	Semantic Analysis	Prog-3
5	Run-Time Environment	
6	Code Generation	Prog-4
7	Optimization	Prog-5, Prog-6

Programming deliverables:
5-6 prog assignments

Default: 2 member team

Assignments and Grading

- Programming assignments are most important, but most students do well on it. Grades for tests often vary more.

Extra Credit:

- 10% of earned (extra credit/total extra credit) will be added with the original 100% from other assignments/exams
  - If you earn 50 out of 100 in extra credit, 5 will be added with your total (100%) achievement.

Assignments and Grading

• Programming Assignments	50%
• Written Assignments	10%
• Midterm	20%
• Final	20%
• Extra Credit	10%

Assignments Policy

- Hard Deadline
  - There will be no extension unless you produce medical certificate or permission from school authorities
  - The instructor or TAs will not reply to such email requests.
  - Plan ahead so that you can finish the assignments on time.
    - There can be challenges that you have not anticipated

- Written Assignments will be submitted through Gradescope
  - We will share Gradescope entry code
  - Type your submission

- Programming Assignments will be submitted through Github Classroom
  - TAs will send you detailed instructions

Assignments Policy

- Programming assignments: work in a 2 member team.
  - You can discuss with TAs/Instructor/Classmate
- Written assignments: do by yourself.
  - No discussion
  - Only clarification questions are allowed on Ed Discussion
  - TAs/Instructors will not respond to individual email
- DO NOT USE AI-Assisted Tool.
  - You will not learn
  - We will check for plagiarism

Submission Policy

- Read the CS Department’s Academic Honesty Policy: https://www.cs.columbia.edu/education/honesty/
- **OK**: Discussing lecture content
- **Not OK**: Solving a homework problem with classmates
- **OK**: Doing programming assignments together
- **Not OK**: Copying from others’ solutions.
- **Not OK**: Posting any homework questions or solutions.
- **Not OK**: Use AI-assisted tools to find the answers.

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

Exam Policy

- Exams: Open book
  - Follow CU honor code.
  - No internet
  - In-class exam

- In-Class Participations
  - Class participation is important
  - **There will be in-class quiz**
    - Quiz marks will go towards extra credit

Prerequisites

1. Advanced Programming on C/C++
2. Computer Science Theory
   1. Regular languages and expressions
   2. Context-free grammars
   3. Finite automata (NFAs and DFAs)
3. Fundamentals Of Computer Systems
   1. Memory layout
   2. Register
   3. Instruction Set
   4. Performance Analysis

Exam Schedule

- Midterm: October 22nd
- Final: December 5th

Submission Links

- **Written Assignments** : gradescope

  Entry Code will be posted in Coursework

- **Programming Assignments** : github classroom

  Details will be posted in Coursework

Q&A

Team Project

The Team Project

- Design and implement your little language.

- Six deliverables:
  1. A proposal describing your language
  2. A language reference manual defining it formally
  3. An intermediate milestone: compiling simple program like “Hello World.”
  4. A compiler for it, running sample programs
  5. Running a small optimization pass.
  6. A final project report & presentation

Teams

- Immediately start forming two-person teams
- Each team will develop its own language
- Each team member should participate in design, coding, testing, and documentation
- Tasks include:

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

Project Due Dates (Tentative)

Section	Author
1. Proposal	September 17 (soon)
2. Language Reference Manual	October 1st
3. Parser & Semantic Analysis	October 15th
4. Code Generation	November 7th
5. Demonstrate Simple program	November 26th
5. Enhancement (complex feature, optimization)	December 10th (extra credit)
7. Final Report & Demo	December 15th

Sample Projects

1. Simple Calculator (arithmetic expression evaluator)
    - Parse and evaluate simple arithmetic expressions like "2 + 3 * 4”
    - Support basic operators: +, -, *, /
    - Handle parentheses for precedence

2. Tiny general-purpose programming language interpreter
    - Design a minimal language with variables, if statements, and loops
    - Implement a lexer, parser, and interpreter for the language

3. JSON parser
    - Create a parser for a subset of JSON
    - Convert JSON strings into an internal data structure

Sample Projects

4. Regular expression engine
   - Implement a simple regex engine supporting basic patterns
   - Include features like character classes, repetition, and alternation

5. Markdown to HTML converter
   - Parse a subset of Markdown syntax
   - Generate corresponding HTML output

6. Simple query language for CSV files
   - Design a basic query language to filter and select data from CSV files
   - Implement a parser and executor for the queries

Sample Project: PromptLang Compiler

- Create a compiler that translates natural language prompts into a formal, structured query language designed for interacting with LLMs.
  - Combines NLP + Compiler techniques

- The structured query language, **PromptLang**, will allow for:
  - more precise control over LLMs
  - specify intents, contexts, constraints
  - expected outputs in a formalized way.

Key Components

1. **Lexer (Tokenizer):** Tokenizes natural language input into words, phrases, and operators, identifying key elements like intents, entities, actions, and constraints.

2. **Parser:** Converts the tokens into an Abstract Syntax Tree (AST) that represents the structure of the prompt in terms of intent, context, and expected outcomes.

3. **Semantic Analyzer:** Ensures that the parsed prompt is valid within the context of PromptLang, checking for consistency and ensuring that all necessary elements (like intents and constraints) are present.

4. **Intermediate Representation (IR) Generation:** Translates the AST into an intermediate representation (IR) that captures the essential elements of the prompt in a more structured form.

5. **Code Generator:** Converts the IR into a structured PromptLang query, which can be used to interact with LLMs more effectively.

6. **Interpreter:** Executes the PromptLang query by interacting with an LLM API, such as GPT, and returning the results to the user.

Sample Project: TinySQL Compiler

- A miniaturized version of SQL designed for basic database queries.

- The goal of this project is to create a compiler that translates TinySQL queries into a simple query execution plan that a rudimentary database engine can execute.

- This project involves parsing SQL-like syntax, generating execution plans, and interpreting those plans to retrieve data from a simulated database.

- SELECT name, age FROM users WHERE age > 18;
- SELECT users.name, orders.amount FROM users JOIN orders ON users.id = orders.user_id;

Project Structure

1. **Lexer:** Identify SQL tokens such as SELECT, INSERT, FROM, WHERE, operators (=, >, <), and literals (strings, numbers).
2. **Parser:**Build a parse tree or AST that captures the structure of the SQL query. For example, the query SELECT name FROM users WHERE age > 18 would result in an AST that includes nodes for the SELECT clause, FROM clause, and WHERE clause.
3. **Semantic Analyzer:** Validate that tables and columns referenced in the query exist and that operations are type-correct (e.g., comparing integers with integers).
4. **Query Planner:** Create a simple execution plan from the AST. For instance, the execution plan for SELECT might involve scanning a table, applying filters, and projecting columns.
5. **Query Executor:** Execute the plan by scanning data from the in-memory tables, applying filters, and returning the results as a list or table.
6. **Database Engine:** Implement basic functionality to store tables and handle data operations such as insertions, updates, and simple indexing for faster lookups.

Sample Project: ExprLang Compiler

- A small language designed specifically for mathematical expressions and simple control flow.

- The language supports arithmetic operations, variables, conditionals, and functions.

- This project aims to create a compiler that translates ExprLang code into a simple stack-based bytecode, which will be executed by a custom virtual machine (VM).
  - x = 5;
  - y = x + 10;
  - z = (x + y) * 2;
  - if (x > 10) {   y = y + 1; } else { y = y - 1; }

Cooler Sample Projects

- Emoji programming language
- Music notation compiler
- ASCII art generator language
- Cellular automata simulator
- Procedural story generator
- Code obfuscator
- SVG generation language
- Chatbot scripting language
- Meme generator language
- Puzzle game-level compiler
- Choreography notation compiler
- Network protocol simulator

Q&A