

EECS4302 Compilers and Interpreters

Fall 2026

Digest of Lecture 1, September 10

Course Foundations, Learning Outcomes, and the Compiler Pipeline

Jackie Wang

HOW TO USE THIS DIGEST

Use this digest as a **roadmap and checklist**. It highlights the main ideas, important examples, common pitfalls, practice targets, and where to return in the course materials. It does **not** replace attending or reviewing the session, using the recording when needed, studying Jackie's annotated iPad notes and other official course materials, completing assigned work, or practising independently. Use it to identify what you understand, what you need to revisit, and what you should be able to do on your own.

If time is limited, start with the sections most relevant to your current review and return to the rest later; selective use is better than skipping the digest because it looks long.

This is an AI-assisted study roadmap reviewed by Jackie and grounded in the session transcript and Jackie's annotated iPad notes. Like any course document, it may still contain an error or omission. If something seems technically doubtful or inconsistent with the original course materials, return to those materials and ask Jackie for clarification rather than treating the digest as final authority.

TABLE OF CONTENTS

Main Ideas	2
Roadmap	2
Key Terms, Notation, and Structures	4
Key Examples and Demonstrations	5
Common Pitfalls and Misconceptions	6
What You Should Be Able to Do	6
Suggested Practice	7
Where to Review	8
Closing Reminder	8

Main Ideas

- **EECS4302 connects theory to a working compiler.** The course brings together theory of computation, Java and tree-based programming, and practical compiler construction. The project is intended to turn familiar foundations into an end-to-end software artifact.
- **The core technical picture is a source-to-target transformation.** Jackie's annotated iPad notes sketch the pipeline as an input program passing through *scanning* to produce tokens, then *parsing* to produce an abstract syntax tree (AST), followed by a recursive transformation implemented by you.
- **Scanning and parsing rest on formal-language foundations.** Regular languages can be represented using DFAs, NFAs, ϵ -NFAs, and regular expressions. Parsing will use context-free grammars, and the course will study both top-down and bottom-up parsing approaches.
- **The project requires specific prerequisite skills.** Jackie emphasized Java polymorphism and dynamic binding, inheritance, recursion, and trees. These ideas support the composite and visitor design patterns and the tree traversals used in compiler implementation.
- **Foundations still matter when AI tools are available.** AI can automate useful work, but it can also fabricate, misunderstand incomplete requirements, or return code that still requires testing and judgment. Jackie framed weak underlying understanding as “cognitive debt”: students must still be able to reason, test, debug, repair, and take responsibility for what they use.
- **Several course logistics already affect how you should prepare.** The Programming Test is scheduled for November 5 and the Written Test for November 19. Assignment 2, the Programming Test, and project submissions must be compatible with Eclipse. The exact attendance-bonus policy was still to be finalized after this lecture.

Roadmap

1. Course structure, resources, and expectations

Jackie introduced the course as a small, discussion-oriented fourth-year class with a substantial hands-on project. The main lecture site will hold study resources such as templates, Jackie's annotated iPad notes, lecture digests, recordings, and prerequisite-review material; eClass will carry announcements and additional instructions. There is no required textbook. *Engineering a Compiler* was mentioned as an optional reference for students who want additional reading.

The normal in-person meetings will end about five minutes early so Jackie can move to his immediately following class. He stated that the accumulated time will be recovered through useful tutorial help or asynchronous material rather than simply lost.

2. Prerequisite foundations and implementation tools

Jackie asked students to review Java before the programming-heavy parts of the course. The most important Java background named in class was polymorphism, dynamic binding, inheritance, recursion, and trees. He connected these ideas to the composite and visitor design patterns and to pre-order, in-order, and post-order tree traversal.

For theory of computation, the course will revisit DFAs, NFAs, regular expressions, DFA minimization, context-free grammars, and parsing. Eclipse is the required environment for Assignment 2 and the Programming Test, and for the submitted project format. Other editors or IDEs may be used during development if you can still produce the required Eclipse submission, but Jackie recommended becoming comfortable with Eclipse early.

3. Assessments, project structure, and calendar commitments

There will be three programming assignments. Jackie described them as learning-oriented: an on-time best honest attempt earns full assignment credit, but the actual learning from the work matters because the same skills feed into the tests and project. Assignment 1 is planned for Week 3; Assignment 2 and the project are planned around the end of Week 5, before reading week.

The project is worth 24% under the scheme discussed in class and will be divided across milestones and a final submission rather than treated as one all-or-nothing deliverable. Jackie also discussed a possible report and presentation component. The Written Test is scheduled for November 19 and is paper-and-pen. The Programming Test is scheduled for November 5, uses Java and Eclipse, and was described as an 80-minute task in which students build a small compiler. The final exam was described as a three-hour, cumulative, paper-and-pen exam worth 50% under the current scheme. Jackie also said that, because the class is small, he may shift some test/exam weight later in a way that benefits students based on performance; the exact mechanism was not fixed in this lecture.

Attendance is intended to be recognized through a bonus rather than a penalty-based participation component, but the exact bonus policy was still pending confirmation. Students were asked to set up iClicker for course 4302, with a trial attendance check planned for the next Tuesday and real tracking beginning the following Thursday.

4. Why foundations matter in the age of AI

Jackie argued that AI does not remove the need for technical understanding. A compiler component generated with AI still has to satisfy the intended requirements and must be tested carefully before it can be trusted. The course will therefore emphasize manual reasoning, testing and debugging, and the ability to judge, repair, and adapt generated work. This was the setting for the warning about accumulating “cognitive debt” by outsourcing reasoning that you never learn yourself.

5. Course learning outcomes: languages, scanners, parsers, and ASTs

The second page of Jackie’s annotated iPad notes organizes the technical introduction around five course learning outcomes. The first distinguishes general-purpose programming lan-

guages (GPLs) from domain-specific languages (DSLs). Java, C++, and Python were used as mainstream GPL examples; SQL was used as a DSL for storage/query work, and propositions/predicates were discussed as a language for expressing constraints.

The next outcomes move through the compiler front end: understand the theory and tools behind a lexical scanner, then a grammatical parser, then construct an AST and perform semantic operations on it. The final outcome emphasizes communicating the design and implementation of a DSL and its associated tools, which motivates the planned report/presentation aspect of the project.

6. First compiler pipeline sketch

The lecture ended with the first concrete compiler architecture. An input program is scanned into a sequence of tokens. Those tokens are parsed into an AST. The AST is then recursively transformed into the required output. Jackie distinguished the theoretical understanding of scanning and parsing from the practical automation provided by ANTLR4: if the token patterns and grammar are specified, the tool can generate the front-end skeleton so that the project can focus on the later transformation logic.

Key Terms, Notation, and Structures

General-Purpose Language (GPL)

A language intended for many problem domains. Java, C++, and Python were examples discussed in class.

Domain-Specific Language (DSL)

A language designed for a narrower domain or kind of task. SQL was used as an example for storage and queries; propositions and predicates were discussed as a notation for constraints.

Regular language representations

The lecture named DFAs, NFAs, ϵ -NFAs, and regular expressions as equivalent ways the course will use to describe regular languages.

Context-free grammar (CFG)

The grammar formalism used in the parsing part of the course. Jackie emphasized that CFGs have recursive structure and support the construction of parse/abstract syntax structures from input.

Scanner / lexical scanner

The front-end phase that performs a linear scan of the input and produces tokens.

Token

A lexical unit produced by scanning, such as a keyword like `if` or an identifier. This use of “token” is unrelated to AI-service token accounting.

Parser

The phase that consumes tokens and organizes the input according to a grammar. The course will study both top-down and bottom-up parsing algorithms.

Abstract Syntax Tree (AST)

The tree produced after parsing in the introductory pipeline. It becomes the structure on which the later compiler transformation operates.

Recursive / tree-to-tree transformation

The project-oriented stage after the AST in Jackie's sketch. Students will implement recursive operations that transform the source representation into the required target representation.

ANTLR4

The parser-generator tool Jackie plans to use. After students specify token patterns and a grammar, it can generate the scanning and parsing skeleton for the compiler.

Key Examples and Demonstrations

GPLs versus DSLs

Setup. Jackie contrasted mainstream programming languages with languages built for a narrower purpose.

Main point. Java, C++, and Python can be used across many domains, whereas SQL targets storage/query work and propositions/predicates can express constraints. The important distinction is not that a DSL is "less useful," but that it is intentionally specialized.

What to practise. Be able to explain why one of the lecture examples is general-purpose or domain-specific and identify the narrower domain served by the DSL example.

Equivalent descriptions of regular languages

Setup. Jackie asked the class to recall ways of describing regular languages.

Main point. The list developed in class was DFA, NFA, ε -NFA, and regular expression. The course will revisit how these representations relate and how one can be converted into another.

What to practise. Refresh the meaning of each notation before the scanner material begins; do not worry yet about reproducing conversions that were not worked out in this lecture.

The compiler pipeline in Jackie's annotated iPad notes

Setup. Page 2 contains the first hand-drawn pipeline for the course.

Main point.

input program $\longrightarrow$ scanning $\longrightarrow$ tokens $\longrightarrow$ parsing $\longrightarrow$ AST
 $\longrightarrow$ recursive transformation $\longrightarrow$ output.

The scanner and parser are grounded in formal-language theory; the later AST transformation is the part Jackie described as the “intelligent” transformation that students will implement.

What to practise. Redraw the pipeline from memory and explain, in one sentence each, what information is produced by scanning, parsing, and the final recursive transformation.

Finding prerequisite review material

Jackie demonstrated the course website and pointed students to the Fall 2021 EECS2030 material, especially Week 7 on polymorphism and dynamic binding. This was not just website navigation: it was a concrete instruction to begin repairing Java background early, before the course reaches the project and Programming Test.

Common Pitfalls and Misconceptions

- **Do not treat AI-generated code as automatically correct or safe.** The lecture’s AI discussion emphasized incomplete requirements, fabrication, and the need for testing and professional judgment. Using AI does not transfer responsibility for the result.
- **Do not postpone prerequisite review until the project arrives.** Polymorphism, dynamic binding, inheritance, recursion, and trees were identified on day one because they are needed for the design patterns and recursive tree transformations later in the course.
- **Do not confuse compiler tokens with AI tokens.** In this course, a token is a lexical unit produced by scanning, such as a keyword or identifier.
- **Do not interpret full assignment credit as evidence that the underlying skill has been mastered.** The on-time best-attempt policy is designed to encourage learning; the programming skills will still be assessed later in the Programming Test and project.
- **Do not assume your preferred IDE removes the Eclipse requirement.** You may develop elsewhere, but the required Assignment 2/Programming Test/project workflow is based on Eclipse, and submitted project work must be in the expected Eclipse form.
- **Do not treat the attendance-bonus details as final yet.** Jackie indicated the direction and discussed a bonus of at least 5%, but said the exact policy would be confirmed after this lecture and added to the syllabus.

What You Should Be Able to Do

After this session, you should be able to:

- explain the difference between a GPL and a DSL using examples from the lecture;
- name the four regular-language representations discussed in class and distinguish them from the context-free-grammar foundation used for parsing;

- sketch the introductory compiler pipeline from input program through scanning, tokens, parsing, AST, recursive transformation, and output;
- explain the role of the scanner, parser, AST, and student-implemented transformation at the level introduced in Lecture 1;
- explain why Java polymorphism/dynamic binding, inheritance, recursion, and trees are prerequisite skills for the implementation part of the course;
- describe why ANTLR4 is useful in the course and what students still have to understand and implement themselves;
- explain Jackie's "cognitive debt" warning and why testing/debugging remain necessary when AI tools are used;
- identify the fixed test dates discussed in class and the Eclipse requirement that affects preparation for Assignment 2, the Programming Test, and the project.

Suggested Practice

1. **Refresh Java now.** Revisit polymorphism, dynamic binding, and inheritance in the EECS2030 Fall 2021 material Jackie identified, especially Week 7. Also refresh recursion and trees from your data-structures background.
2. **Refresh theory of computation.** Review the meanings of DFA, NFA, ϵ -NFA, regular expression, and context-free grammar so the coming scanner/parser lectures can focus on the compiler connection rather than basic recall.
3. **Reproduce the page-2 pipeline.** Without looking, draw input $\rightarrow$ scanning $\rightarrow$ tokens $\rightarrow$ parsing $\rightarrow$ AST $\rightarrow$ recursive transformation $\rightarrow$ output. Then compare your drawing with Jackie's annotated iPad notes.
4. **Get comfortable with Eclipse.** If Eclipse is not your usual environment, begin using it before Assignment 2, the Programming Test, or the project forces you to learn it under time pressure.
5. **Set up the course logistics that affect participation.** Join iClicker course 4302, monitor eClass announcements, bookmark the main lecture site, and record the November 5 Programming Test and November 19 Written Test dates.
6. **Begin thinking about a DSL/compiler idea.** Jackie encouraged students to brainstorm what kind of compiler they might want to build. At this stage, focus on the source domain and target/output you would want rather than implementation detail.

Where to Review

- **Jackie's annotated iPad notes, page 1:** the session's three framing themes – foundations in the age of AI, semantic domains, and compilation as source-to-target transformation.
- **Jackie's annotated iPad notes, page 2:** the five course learning outcomes, the hand-written GPL/DSL examples, the regular-language and CFG annotations, and especially the hand-drawn compiler pipeline at the bottom of the page.
- **Lecture recording:** return to the early course-site walkthrough for resources and prerequisite material; the middle portion for assessment structure, Eclipse requirements, and fixed test dates; and the final portion for the AI/foundations discussion, CLOs, and compiler pipeline explanation.
- **EECS2030 Fall 2021 review material:** Jackie specifically pointed to Week 7 for polymorphism and dynamic binding/inheritance background.
- **Course lecture site and eClass:** use the lecture site for the evolving study resources and recordings, and eClass for official announcements and any logistical updates. The exact attendance-bonus policy was still pending after Lecture 1, so use the later written update rather than this digest as the final policy source.
- **Semester calendar in the course materials:** use it to track the Programming Test on November 5, the Written Test on November 19, and later assignment/project milestones. Jackie stated that test dates are intended to remain fixed, while some non-test deadlines may be adjusted explicitly in writing.

Closing Reminder

Use this digest as a roadmap for active review, not as a substitute for the learning process itself. Attend Jackie's in-person sessions and pay attention to how he develops ideas and works through the important reasoning step by step. If a transition, explanation, example, or reasoning step remains unclear afterwards, return to the recording to fill that gap.

For structured review, start with Jackie's annotated iPad notes because they most closely reflect the actual flow and development of the class. Work through them in manageable logical units. After each logical unit, study the corresponding part of the presentation slides alongside it. Jackie does not normally teach by reading the slides line by line, so the slides may contain relevant supporting material that was not explicitly read aloud.

Finally, choose a suitable selection of the important reasoning from this session and work through it again on your own. For Lecture 1, that can include the GPL-versus-DSL distinctions, the regular-language representations, and the scanner–parser–AST–transformation pipeline. Redraw and explain these from your own understanding rather than memorizing Jackie's recorded steps. The goal is to be able to explain why the stages and distinctions matter before the course adds more detailed grammars, parsing traces, compiler algorithms, and implementation work.