Course Wrap-Up

CS 331 Programming Languages
Lecture Slides
Monday, April 27, 2026

Glenn G. Chappell
Department of Computer Science
University of Alaska Fairbanks
ggchappell@alaska.edu

Course Wrap-Up

Course Wrap-Up
From the First Day of Class: Course Overview — Description

In this class, we study programming languages with a view toward the following.

- How programming languages are specified, and how these specifications are used.
- What different kinds of programming languages are like.
- How certain features differ between various programming languages.
- How to write code in various programming languages.

Course Wrap-Up
From the First Day of Class: Course Overview — Goals

Upon successful completion of CS 331, students are expected to:

- Understand the concepts of syntax and semantics, and how syntax can be specified.
- Understand, and have experience implementing, basic lexical analysis, parsing, and interpretation.
- Understand the various kinds of programming languages and the primary ways in which they differ.
- Understand standard programming language features and the forms these take in different programming languages.
- Be familiar with the impact (local, global, etc.) that choice of programming language has on programmers and users.
- Have a basic programming proficiency in multiple significantly different programming languages.

Course Wrap-Up
From the First Day of Class: Course Overview — Topics

The course material will be divided into eight units:

1. Formal Languages & Grammars
2. The Lua Programming Language
   - PL Feature: Compilation & Interpretation
3. Lexing & Parsing
4. The Haskell Programming Language
   - PL Feature: Type System
5. The Scheme Programming Language
   - PL Feature: Identifiers & Values
   - PL Feature: Reflection
6. Semantics & Interpretation
7. The Prolog Programming Language
   - PL Feature: Execution Model
8. Student Presentations on Programming Languages

Track 1: Syntax & Semantics of PLs.

Track 2: PL features & categories, specific PLs.

Course Wrap-Up
What We Covered — Specification [1/2]

We covered specifying the syntax of a programming language. Our primary tool was the (**phrase structure**) **grammar**, specified in practice using EBNF or something similar.

statement → `'while' '(' expr ')' '{' program '}'`

We also covered **regular expressions**.

We looked closely at two classes of formal languages—and the associated grammars that generate them—**regular languages** and **context-free languages**.

We briefly discussed formal methods for specifying semantics, but we did not actually use any of these.

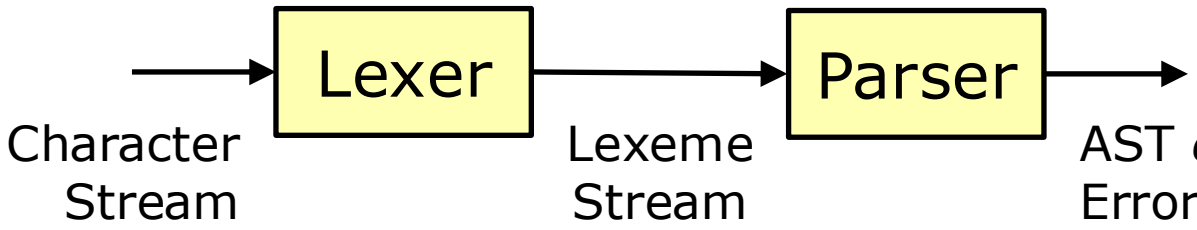
Course Wrap-Up
What We Covered — Specification [2/2]

Our 2 main classes of languages are in the **Chomsky Hierarchy**.

Language Category		**Generator**	**Recognizer**
Number	**Name**		
Type 3	**Regular**	Grammar in which each production has one of the following forms. • $A \to \varepsilon$ • $A \to b$ • $A \to bC$ Another kind of generator: *regular expressions*.	Deterministic Finite Automaton Think: Program that uses a small, fixed amount of memory.
Type 2	**Context-Free**	Grammar in which the left-hand side of each production consists of a single nonterminal. • $A \to$ [anything]	Nondeterministic Push-Down Automaton Think: Finite Automaton + Stack (roughly).
Type 1	Context-Sensitive	*Don't worry about it.*	*Don't worry about it.*
Type 0	Computably Enumerable	Grammar (no restrictions).	Turing Machine Think: Computer Program

Course Wrap-Up
What We Covered — Lexing, Parsing, Interpretation

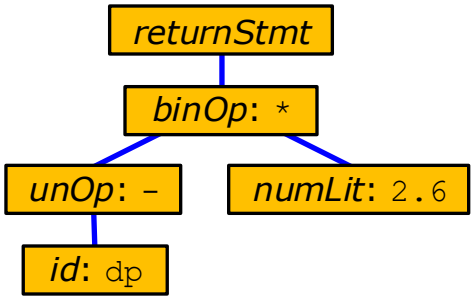
We looked at how to write a **lexical analyzer** (**lexer**) and a **syntax analyzer** (**parser**).




```
return (-dp * 2.6) // x
```

```
return (-dp * 2.6) // x
```

key, op, id, op, num lit, punct, punct



We also covered writing a simple **tree-walk interpreter**.

What We Covered — Programming-Language Features

We covered five programming-language features in depth:

Compilation & Interpretation

Runtime & runtime systems. Compiler & interpreter. JIT compilation.

Type System

Types & type systems. What type systems are for. Static vs. dynamic. Manifest vs. implicit. Nominal vs. structural. Type safety, soundness.

Identifiers & Values

Identifier, namespace, overloading. Value, variable. Scope & lifetime. Implementation & lazy evaluation. Thunks.

Reflection

The ability of a program to deal with its own code at runtime: examining it, looking at its properties, and possibly modifying it.

Execution Model

Different tasks drive execution in different PLs. Unification.

What We Covered — Programming Languages

We covered four programming languages:

Lua

Dynamic PL. Fixed set of types. Small versatile feature set.

Haskell

Pure functional PL. Static typing. Type inference. Write expressions & functions. Execution is evaluation. I/O: return side effect descriptions.

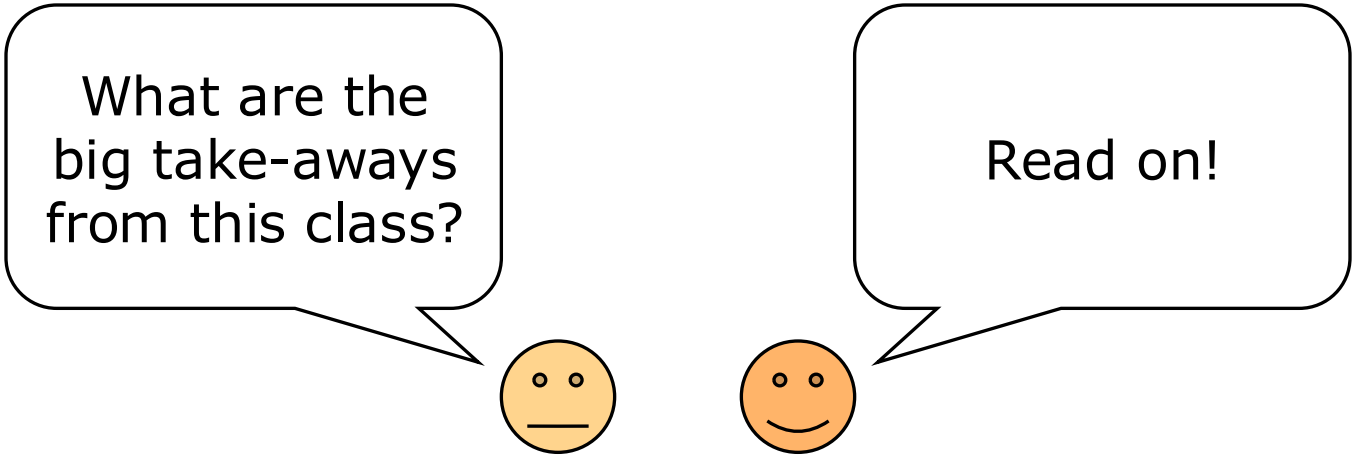
Scheme

Lisp-family PL. Dynamic typing. Fixed set of types. Write expressions & procedures. Execution is evaluation. Macros.

Prolog

Logic PL. Dynamic typing. Predicates. Write facts & rules. Do queries. Execution is unification with goals handled via backtracking search.

Course Wrap-Up
Take-Aways [1/4]



Course Wrap-Up
Take-Aways [2/4]

Take-Away #1. There are many different ways to think about code and programs. Some of these, though they may be unfamiliar, can be quite worthwhile.

Take-Away #2. Learning new programming languages can help you be a better programmer even in more familiar programming languages.

Take-Away #3. Cross-pollination is constantly occurring between programming languages. Many features being added to more mainstream PLs today are taken from the Lisp family or functional PLs like Haskell.

Course Wrap-Up
Take-Aways [3/4]

Take-Away #4. *Formal syntax specification* is well understood and often used.

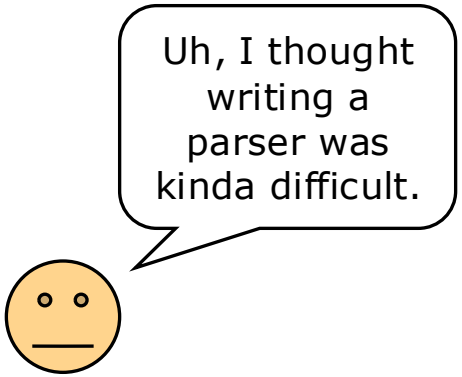
Take-Away #5. *Declarative programming*, including *functional* and *logic* programming, can improve code clarity and prevent bugs.

Take-Away #6. PL support + knowledge of how a programming pattern works can lead to a number of useful tools:

- *Regular expressions.*
- *Closures.*
- *Coroutines.*
- *Comprehensions.*
- *State machines.*
- *Parsers.*

Furthermore, none of these are particularly difficult to write/use.

Course Wrap-Up
Take-Aways [4/4]



When I had you write a parser:

- You had probably never written one before.
- You only got about a week to do it.
- You were coding in a relatively unfamiliar PL.
- I did not allow you to use third-party libraries.

If you write a parser in the future, it is likely that none of these will be true.

Similar comments apply to other tools.

Course Wrap-Up
THE END