# Check-in #1

Review: Regular expressions

## Write your name and answer the following on a piece of paper

- Explain the difference between the languages recognized by these regular expressions:
  1. `(cake)|(death)`
  2. `cake|death`

- Create a regular expression that is as short as possible (in terms of characters used to write down the regular expression) but matches the same language as:
  `a|(aa)|(a*)`

- The ? operator is sometimes used to denote "zero or one" repetitions of its operand. As as example `a((bc)?)` matches
  - `a` (0 repetitions of `bc`)
  - `abc` (1 repetition of `bc`).

  Using the operators listed previously, change the above regular expression so that it doesn't use the ? operator but specifies the same language of strings. *Hint: you may use the empty string symbol ε in your answer*

KU | EECS | Drew Davidson

# EECS 665 COMPILER CONSTRUCTION

## 1 – Overview

## Housekeeping
Administrivia & Announcements

**Assignments**

- Entry Survey out now
  - Due **tonight** at 11:59 PM
- Lab 1 out tonight
  - Due next Monday at 3:00 PM
- Lab 2 out by Friday
  - Due next next Monday at 3:00 PM
  - Will be the subject of in-person labs next week

# Today's Roadmap

Lecture Outline

- Orientation
  - About me
  - About you
  - About the course
- Overview the Compiler
- Lexical Specification

# About Me

A scientist

...and an administrator

...and also a teacher

**(Associate) Professor**
**Andrew "Drew" Davidson**

~~Pronouns: he/him/his~~

## What to call me

About Me

- **Preferred:** “Drew”
- **Ok:** “Professor Davidson”, “Dr. Davidson”
- **Never:** “Andy”, “Andrew”, “Mr. Davidson”, “Dr. Drew”

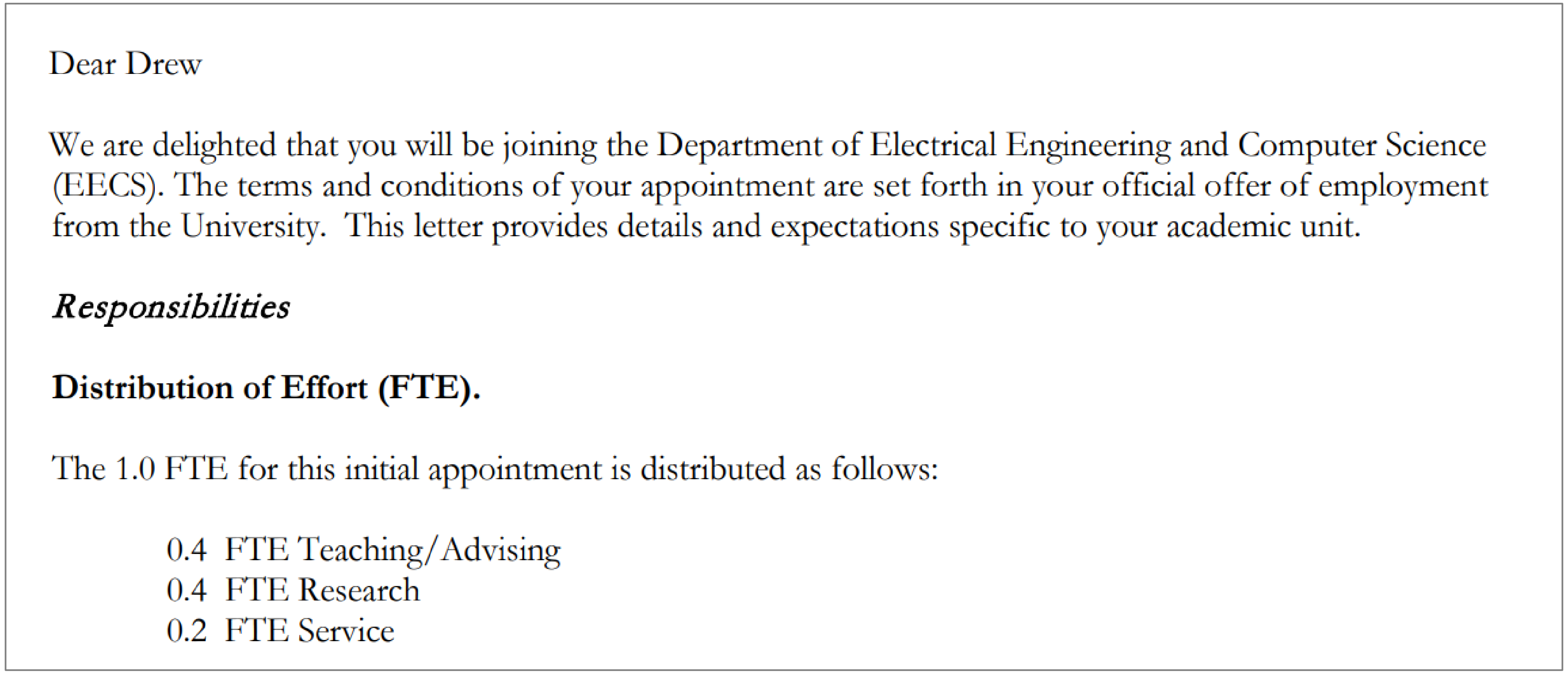
**Dr. Drew (Extremely not me) [1]**

**[1]: Credit: www.podcastone.com/Dr-Drew-Show**

# About Me: The Job of a Professor

About the class: FAQ

**The actual start of my job offer letter from KU:**



Dear Drew

We are delighted that you will be joining the Department of Electrical Engineering and Computer Science (EECS). The terms and conditions of your appointment are set forth in your official offer of employment from the University. This letter provides details and expectations specific to your academic unit.

*Responsibilities*

**Distribution of Effort (FTE).**

The 1.0 FTE for this initial appointment is distributed as follows:

0.4 FTE Teaching/Advising
0.4 FTE Research
0.2 FTE Service

# I’m a Busy Little Honeybee!

About Me

**I love my job!**

- But there is a lot of it
- I’d happily spend 40hrs/wk just on this class

**Takeaways**

- Delays in email/grading can happen
- ~~I’m too busy to help?~~
- Office hours are *just for you*
- I try to scale my help

No! I’m here for you!

This drives several course policies

# Interacting with Me

About Me

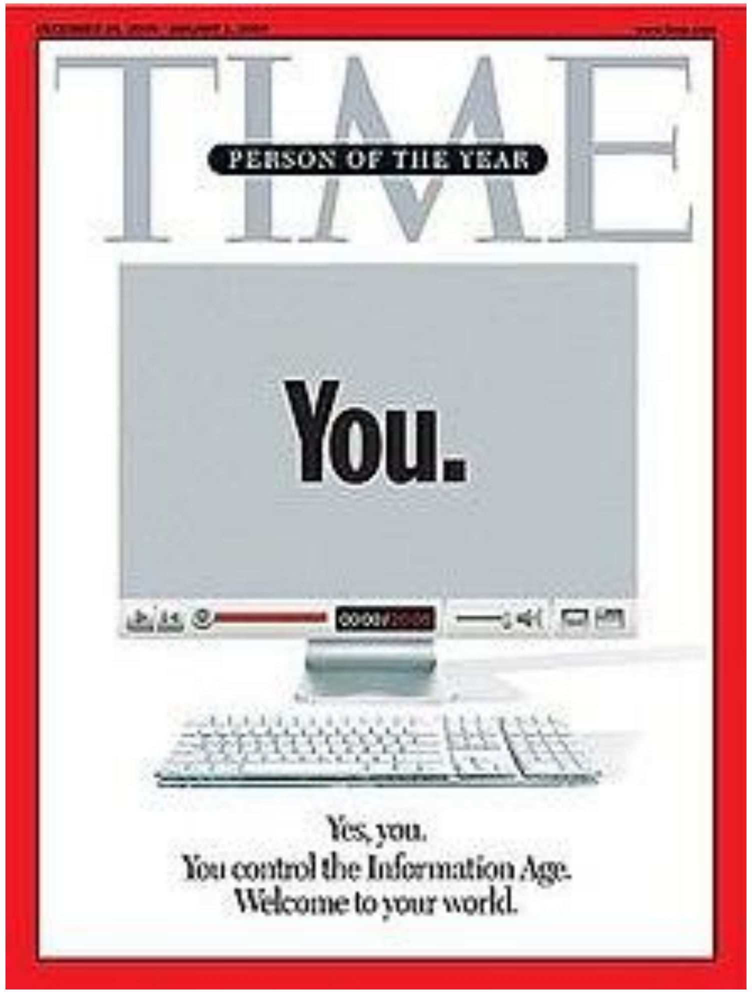
**I am pretty friendly** *(I think)*

- I'll make an effort to learn every student's name
- If you see me outside of class, feel free to say "hi!"

**I like when you visit office hours**

- Appreciate when you come with a specific question

# About You


TIME
PERSON OF THE YEAR
You.
Yes, you.
You control the Information Age.
Welcome to your world.

# Your Time is Valuable!

Orientation - About You

**There are a lot of assignments**

- Most of them are very quick

**You don’t have to come to class**

- You are rewarded for doing so

# One Small Favor

Orientation - About You

**Help me to make this class pleasant**

- If you come to class, try to engage
  - Frown when you are confused
  - Grin when you are amused
  - Ask questions if you have them
- If you have feedback, let me know!

# This course is built for y’all

Orientation - About You

**I value feedback**

- This course improves by matching your needs
- I encourage questions, comments, etc. (within reason)

**I’ve taught this course before**

…but I’ve never taught **YOU** this course before

## About The Class

COMPILER

CONSTRUCTION

# What I think you NEED to Know

About the class

**Read the syllabus:** [https://compilers.cool/syllabus.pdf](https://compilers.cool/syllabus.pdf)

# What I think you WANT to Know

About the class

# How 'bout that Covid, eh?

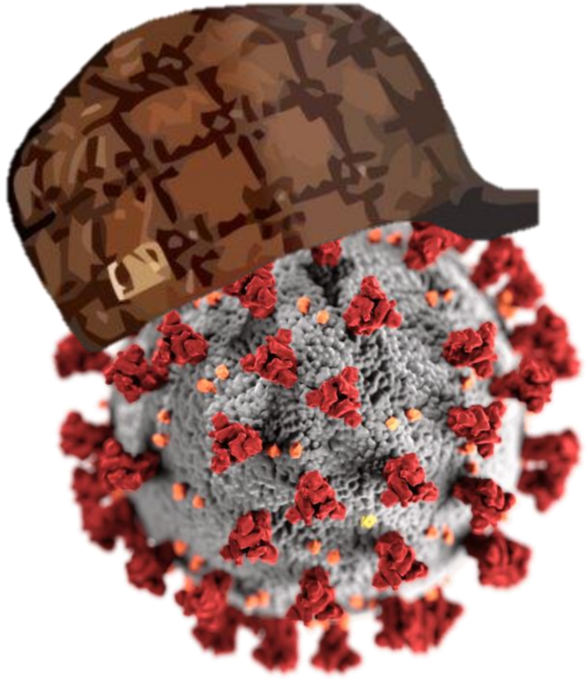
About the class

**Too Sick for Class?**

- You're never *required* to come to class (except for tests)
- If you're too sick for a test, we'll do a makeup

**Too Sick to work?**

- Homework should take way less time than you're given
- Projects can <u>collectively</u> be turned in 6 days late for no penalties



# Is This Class Hard?

About the class: FAQ

Not sure if class is hard...
or professor sucks
ICANHASCHEEZBURGER.COM

Definitely this option

# Is Drew a Good Teacher?

About the class: FAQ

**My core philosophy: teach the class I'd want to take**

# Is Drew a Good Teacher?

About the class: FAQ

**My couse design goal: teach the class I'd want to take**

- Put a lot of material in the course
- Only post assignments <u>after</u> material is covered
- Allow more time on assignments than needed
- Make myself available
  - Phone alerts for Piazza posts
  - Respect office hours
- Never require participation, always reward it
- Provide lots of status/understanding checks
  - The class is out of exactly 1000 points
  - Frequent assignments, exercises in the class readings
  - If you want to go above and beyond, extra assignments

# Is This Class Hard?

About the class: FAQ

Definitely this option

Let's go with "conceptually complex"

*may depend on definition of "hard"*

*The class <u>should be</u> hard, because constructing compilers <u>is</u> hard*

# Let's judge a book by it's cover

About the class: A brief aside on complexity

- Programming Languages

  Cute teddy bear!

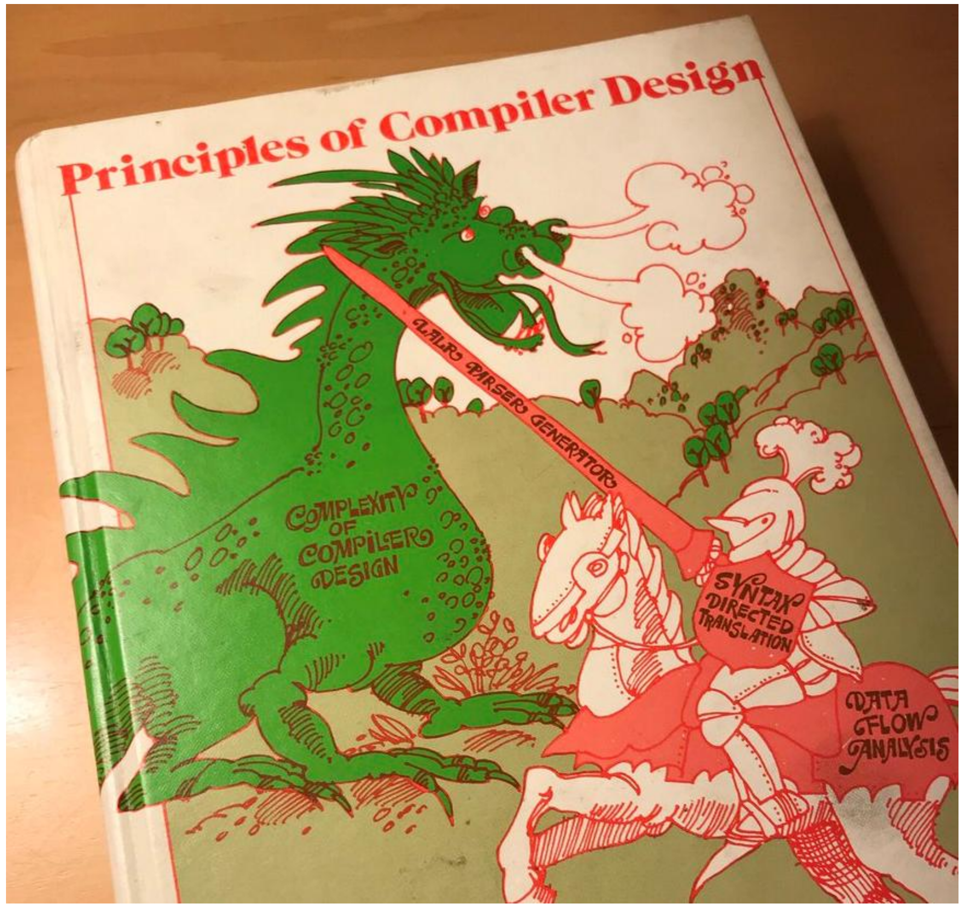
- Operating Systems

  Fun circus!

- Compilers…

  A dragon to murder

  (and the dragon is pissed)

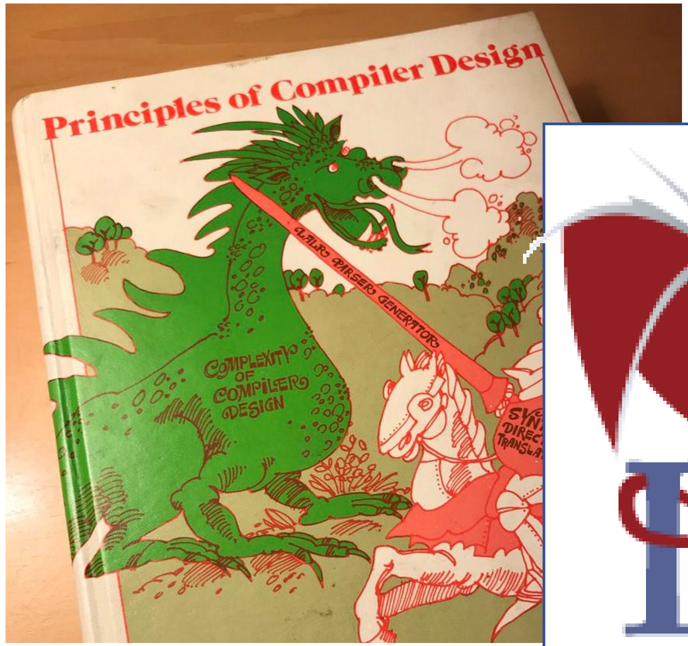

Principles of Compiler Design
LALR PARSER GENERATOR
COMPLEXITY OF COMPILER DESIGN
SYNTAX DIRECTED TRANSLATION
DATA FLOW ANALYSIS

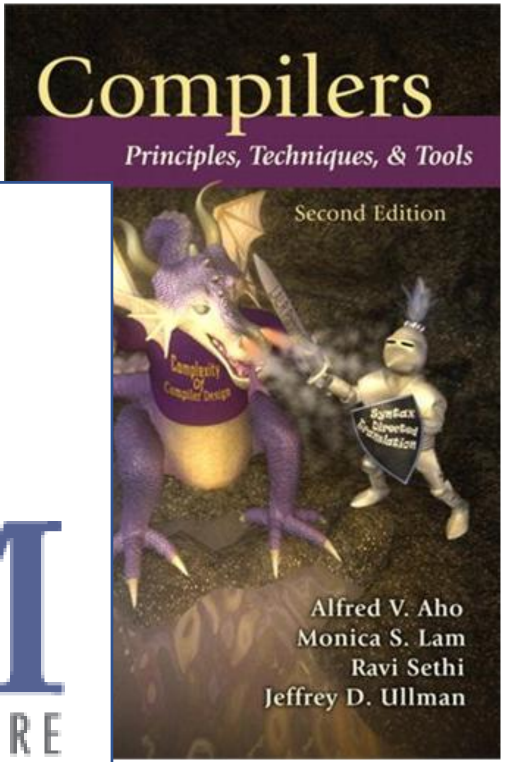
# That's just one book, right?

About the class: A brief aside on complexity

Uhh, actually dragons are like a whole thing

But why dragons?

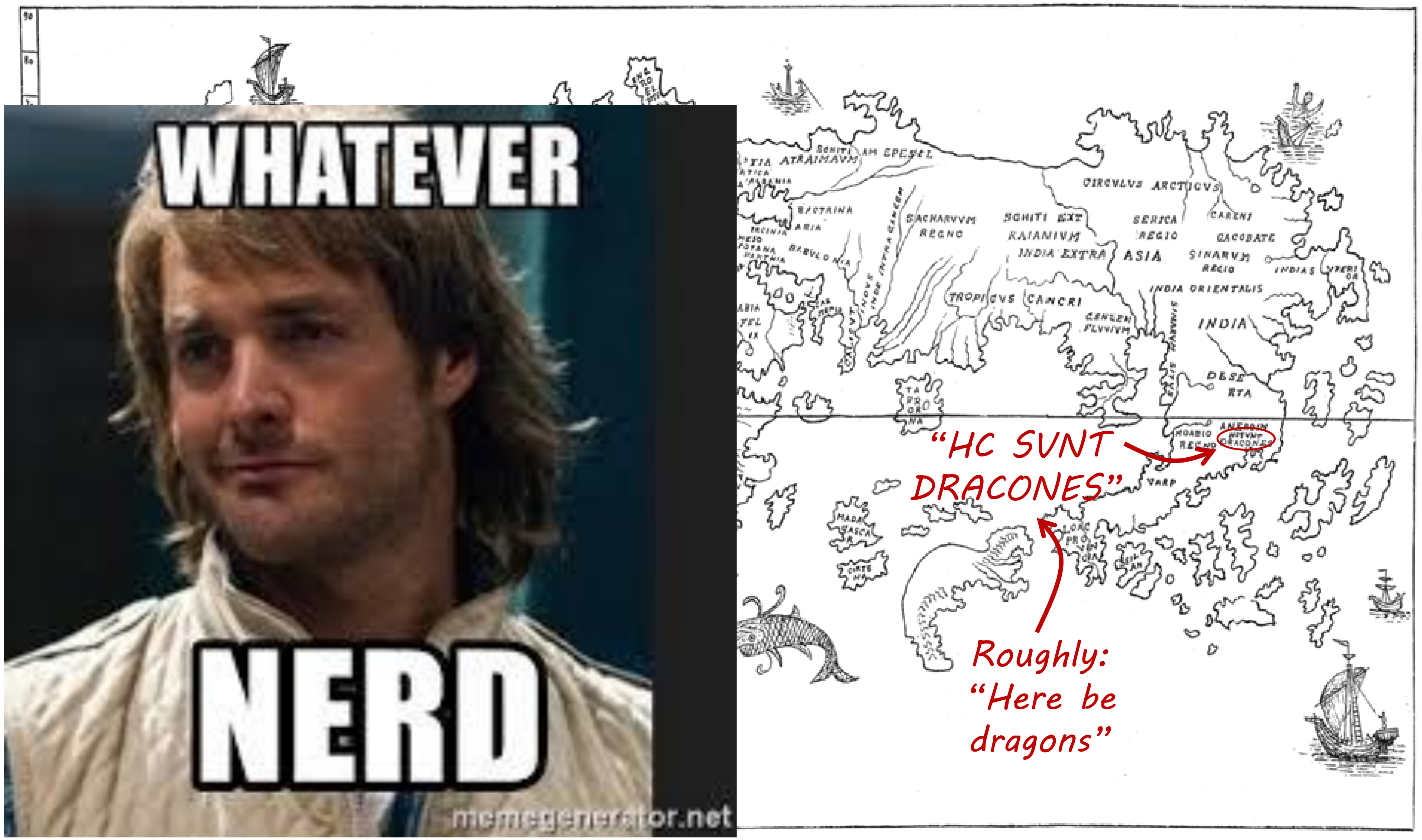



Compilers
Principles, Techniques, & Tools
Second Edition
Complexity of Compiler Design
Syntax Directed Translation
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman

# Dragons: symbols of the unknowable

About the class: A brief aside on complexity

THE·LENOX·GLOBE



# This Class is About Complexity of Design

We'll wield the classic tools to combat complexity:

- Formalisms
- Abstractions
- Modularity
- Disciplined software design

## Explore Design Complexity through Implementation

About the class

**Let's Build a Compiler!**

- Seems like a good thing to do in a class called "Compiler Construction"
- Regardless of your interest in compilers, you'll get to do some non-trivial code development

# Today's Lecture Roadmap

Lesson Outline

- Orientation
  - About you
  - About me
  - About the course
- Overview the Compiler
- Lexical Specification

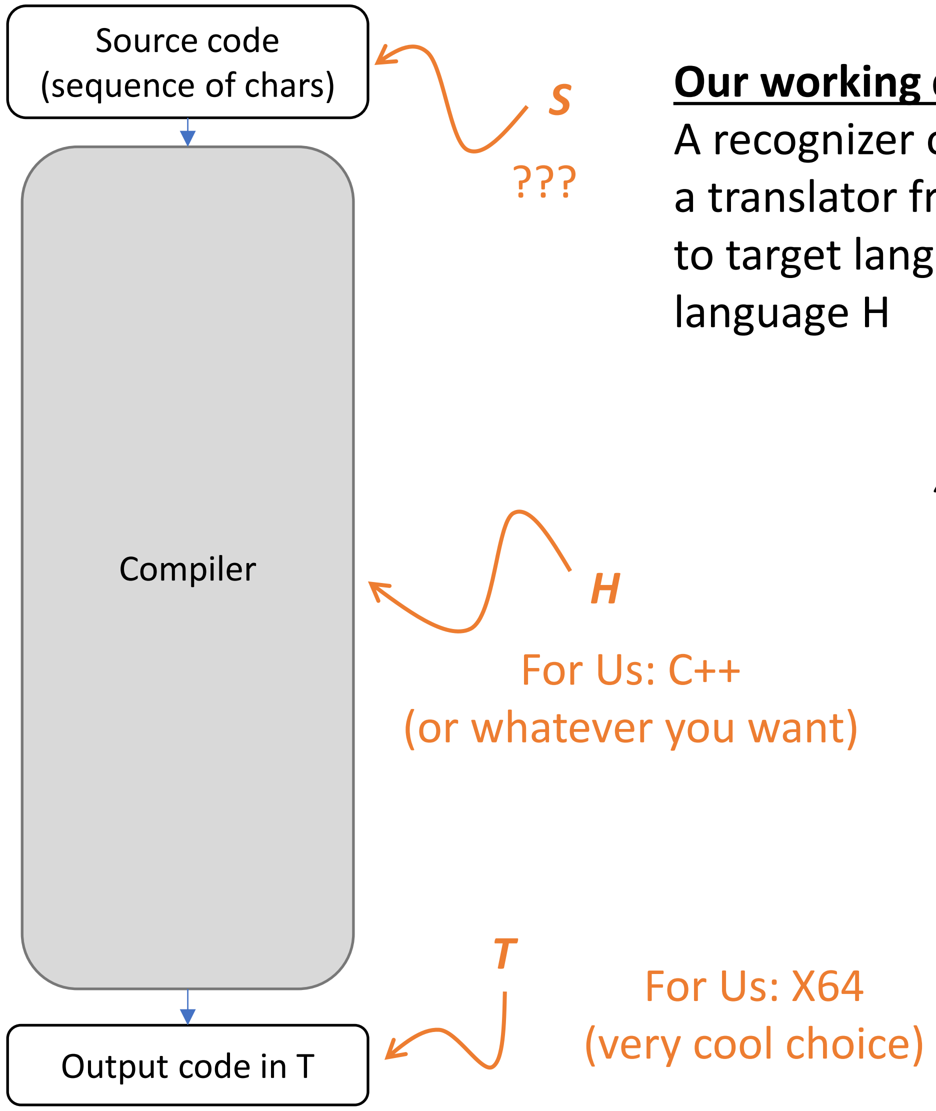
# What is a Compiler?

Overview the Compiler

***(Audience participation)***

# What is a Compiler?

Overview the Compiler



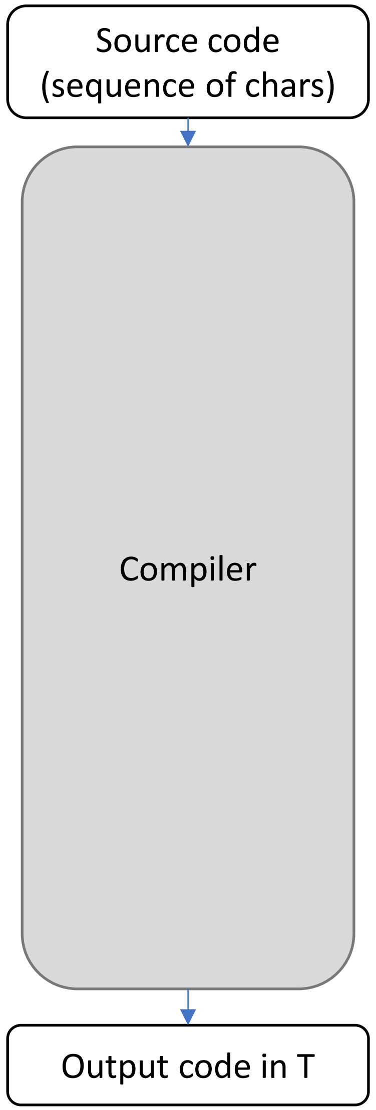
**Our working definition of a compiler**
A recognizer of source language S and a translator from source language S to target language T written in host language H

**Our compiler**
Host Language H = C++
Target language T = X64
Source language = ???

***Audience Participation:***
***What should we name our language?***

# What is a Compiler?

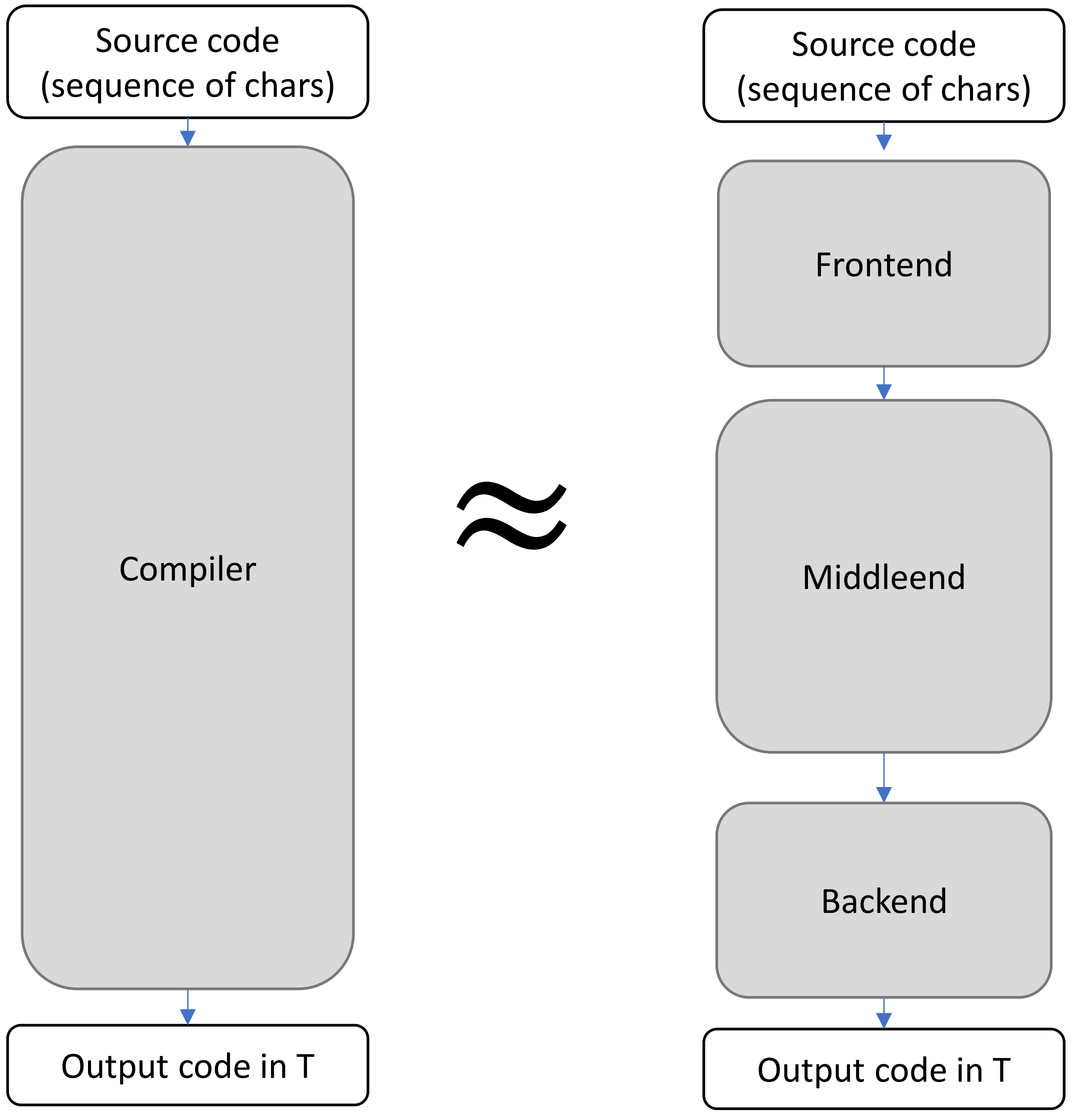
## Overview the Compiler



***Great! With our language defined, we can resume exploring the compiler's structure***

## What is a Compiler?

### Overview the Compiler



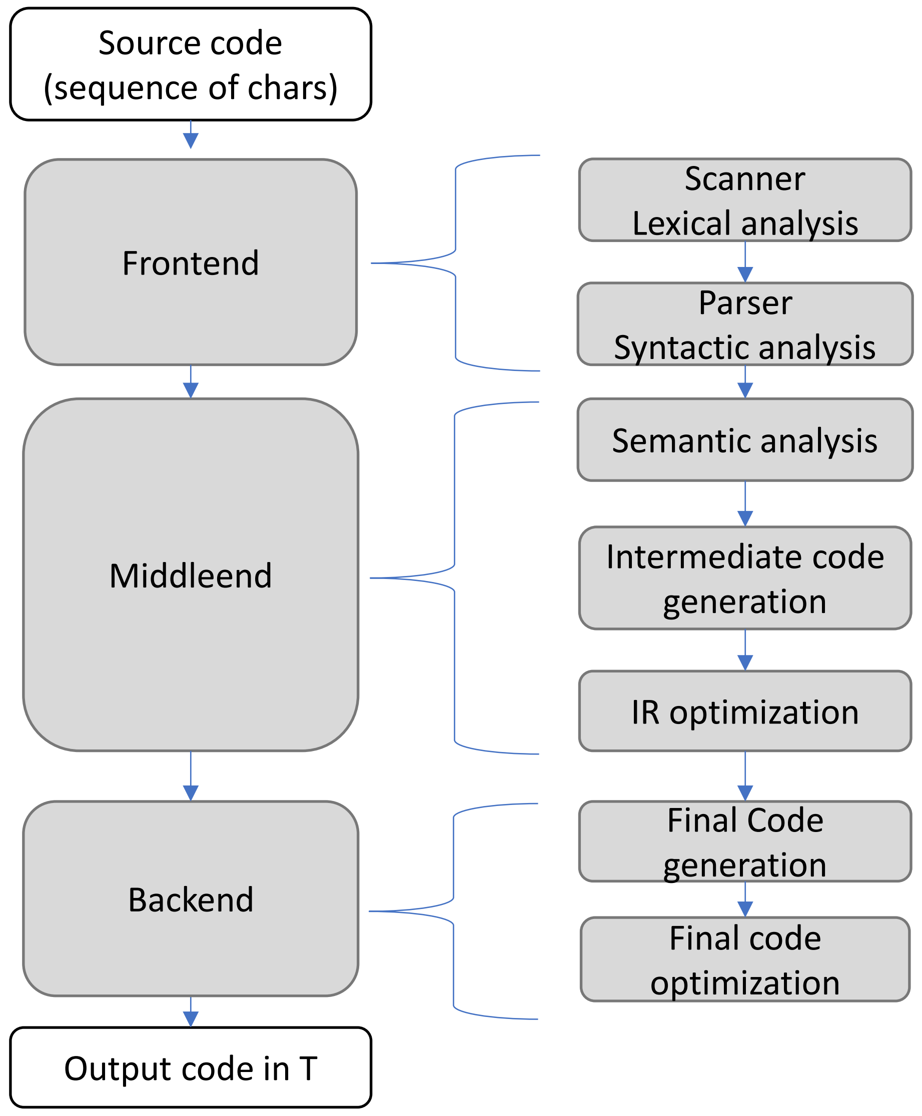
**<u>Traditional compilers</u>**

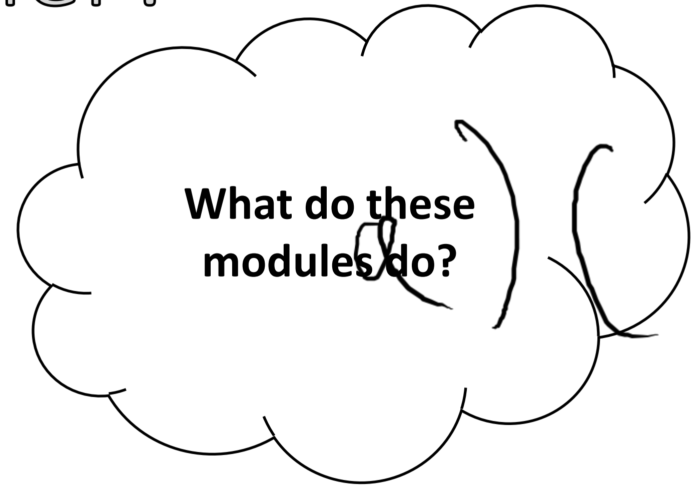
Divided into phases

- Frontend: input handling
- Middleend: program reasoning
- Backend: output handling

# What is a Compiler?

Overview the Compiler





**Traditional compilers**

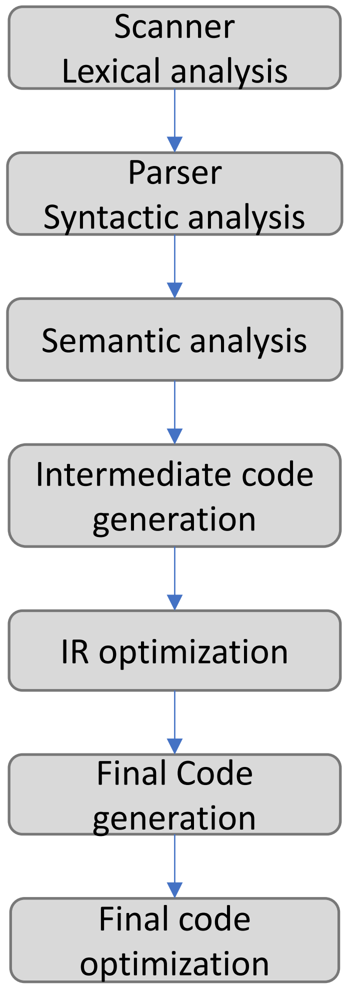
Divided into phases

- Frontend: input handling
- Middleend: program reasoning
- Backend: output handling

Phases further divided into modules

# What is a Compiler?

Overview the Compiler



**Scanner:** Transform input characters into tokens (the “words” of the language)

**Parser:** arrange tokens into syntax (the “sentences” and “paragraphs” of the language)

**Semantic Analysis:** Check properties of the AST and add metadata

- **Name analysis:** bind identifiers to their symbols
- **Type analysis:** associate types with operations

**Intermediate codegen:** Transform AST into a more “operational” internal representation

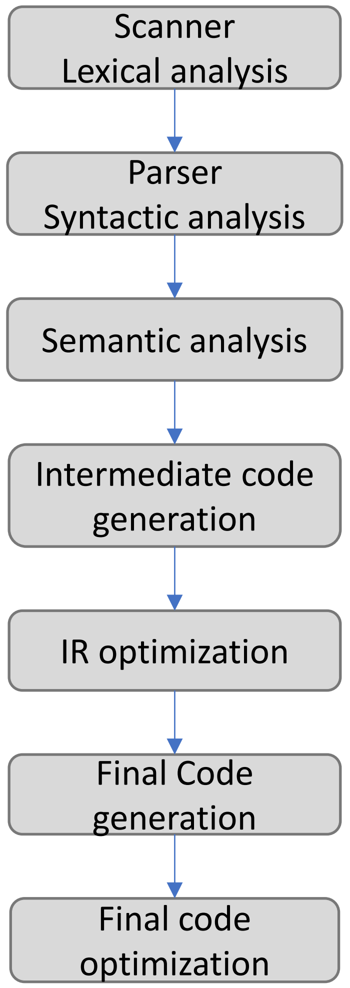
**Intermediate Representation Optimization:** Structural improvements

**Final Code Generation:** Translate IR into target language representation

**Final Code Optimization:** target-specific optimizations

# Compiler's Recognizer Duties

Overview the Compiler



**Scanner:** Transform input characters into tokens (the "words" of the language)

*Lexical errors*

**Parser:** arrange tokens into syntax (the "sentences" and "paragraphs" of the language)

*Syntactic errors*

**Semantic Analysis:** Check properties of the AST and add metadata

*Naming errors*

**Name analysis:** bind identifiers to their symbols

**Type analysis:** associate types with operations

*Type errors*

**Intermediate codegen:** Transform AST into a more "operational" internal representation

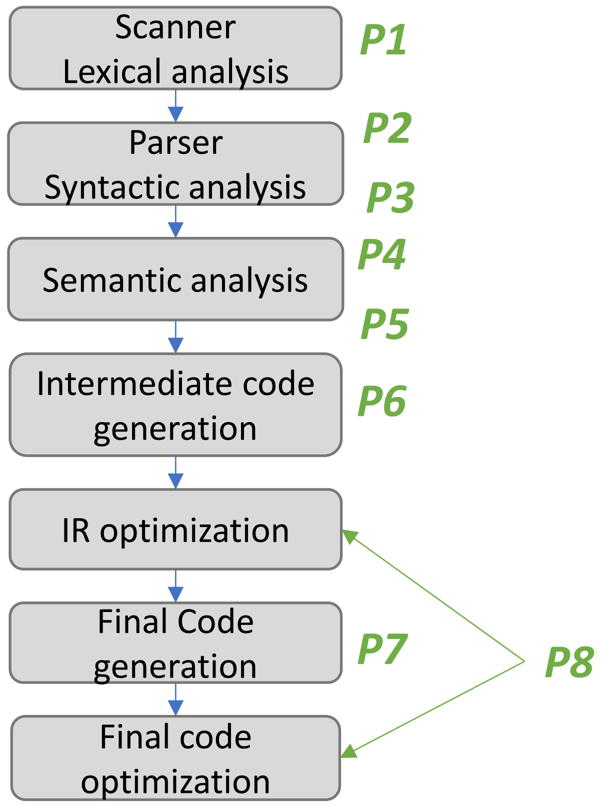
**Intermediate Representation Optimization:** Structural improvements

**Final Code Generation:** Translate IR into target language representation

**Final Code Optimization:** target-specific optimizations

## Our Class Workflow

Overview the Compiler



**We'll work through the compiler front-to-back**

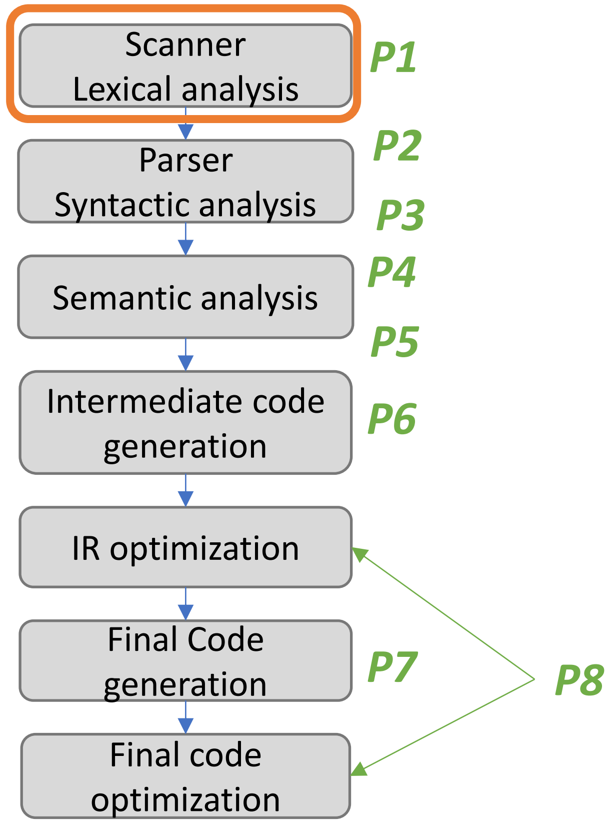
- Pause on background information as needed
- Review underlying theory, implementation details, and techniques as needed

**Often need to…**

- Precisely define / express some language concept
- Build a recognizer of that concept
- Build a translator for that concept

# Exploring Lexical Analysis Design

Lexicial Analysis



**We'll work through the compiler front-to-back**

- Pause on background information as needed
- Review underlying theory, implementation details, and techniques as needed

**Often need to…**

- Precisely define / express some language concept
- Build a recognizer of that concept
- Build a translator for that concept

# Exploring Lexical Analysis Design

Overview the Compiler

Scanner
Lexical analysis

**Often need to…**

- Precisely define / express some language concept
- Build a recognizer of that concept
- Build a translator for that concept

**We'll use some (hopefully) familiar theory techniques in building the scanner:**

- Regular Languages / Regular Expressions
- Deterministic Finite Automata
- Nondeterministic Finite Automata

*These would be good concepts to review if you're shaky on them*

# Lexical Definition

Overview the Compiler

**Describe the tokens (i.e. the "words") of the language using regular expressions**

| **Token** | **Examples** | **RegEx** |
|---|---|---|
| Integer Literal | 1 230 | 0\|(1\|2\|3\|4\|5\|6\|7\|8\|9)(0\|1\|2\|3\|4\|5\|6\|7\|8\|9)* |
| star | * | "*" |

# Lecture Wrap-Up

Goodbye for now!

**Summary**

- Working definition of a compiler
- Compiler overview
  - Phases of the compiler
  - Modules of the phases

**Next Lecture**

- Describe how we can build a token recognizer from the specification

**Your ToDos:**

- Survey due at midnight tonight
- If you missed class, C1 is due Sunday at midnight
- Familiarize yourself with https://compilers.cool
- Sign up for Piazza
- If you need some theory review, check out https://compilers.cool/theory_review/