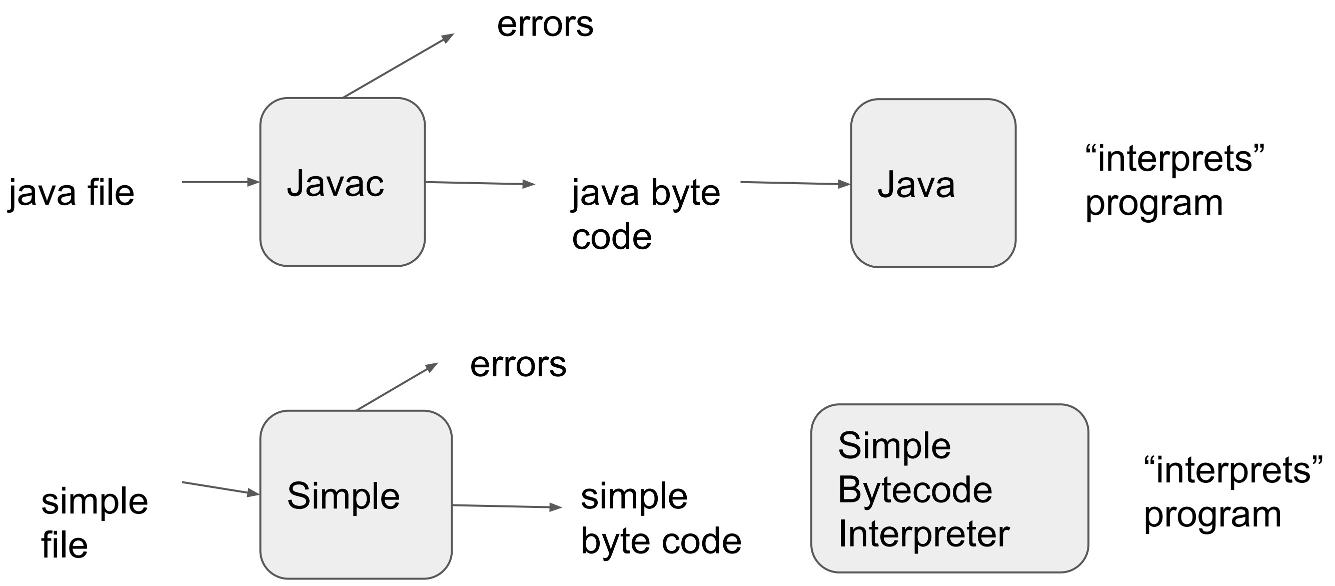
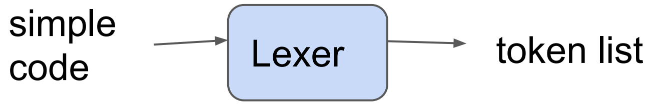
Interpreter Project

You’ll build tools similar to javac and java

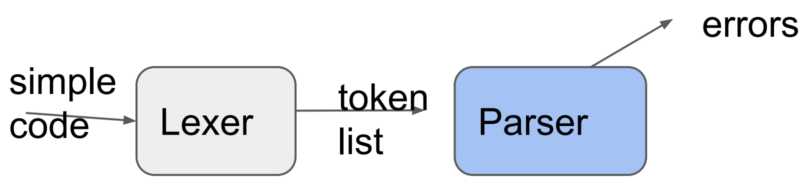


but for a language that only has simple assignment statements

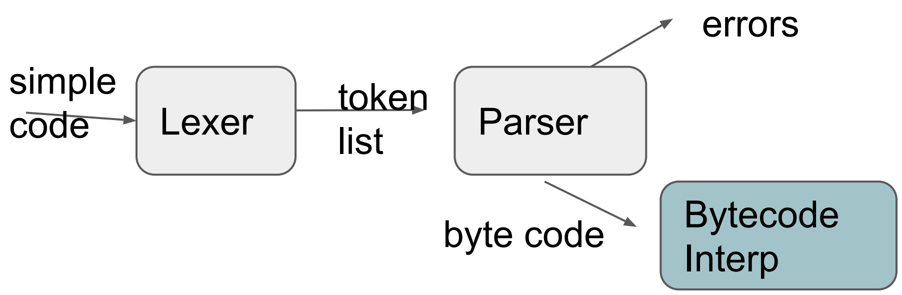
3 Parts: Lexer, Parser, Codegen/Interp



Part 1. Lexer-- break code text into tokens. Find the words in sentence



Part 2. Parser-- analyze tokens to see if valid statement. If not, print error



Part 3. Generate byte code, and interpret it

Backus-Naur Form Describes Language Grammar

Backus Naur Form (BNF)

BNF is a notation for describing context-free grammars, often used to describe the syntax of a programming language.

Here is the BNF for the SIMPLLE language we'll define and interpret for this project

```
<assignment-stmt> ::=   <identifier> = <arithmetic-expr>

<arithmetic-expr>   ::=   <term> | <arithmetic-expr> + <term>

<term>  ::= <identifier> | <integer>
```

Example legal statements: x12=4   y= x12+ 5

Lexer Sample

Lexer is short for Lexical Analysis, which identifies the “words in a sentence”

Find each word and say what type it is, e.g., Identifier, Assmt, Integer, Plus

What tokens would be identified for: x12= 3 + 43

What tokens would be identified for: 345xyz543

Lexer Code

- Code file read into a buffer.
- Index through buffer
- An id is a letter followed by any number of letters/digits
- An integer is a digit followed by any number of digits
- When lexer sees: x12= 3 + 43
    - it identifies that first token has type “ID” and value, “x12”, index=3 after first token done
- getNextToken() gets the next token based on the Lexer’s index.

What is the definition of an identifier-- what does it consist of?

What is the definition of an integer-- what does it consist of?

Parser Sample

Given the tokens from the Lexer, the parser determines if there is a valid program, or an error.

> <id> <assmt> <int> <plus> <int>   is valid,   e.g., x12= 3 + 43
>
> <int> id> <int> is not. Error is “expecting id”   e.g., 345xyz543

Parser Code

- You’ll use a “recursive descent” parser.
- Define a parseX function for each construct you need to identify
- parseProgram, parseAssmt, parseExpression, parseId, parseInt
- The parse functions get the next token from the Lexer (or the Token List Lexer created)
- Lexer does the dirty char-level work, Parser works with tokens

Simple Byte Code

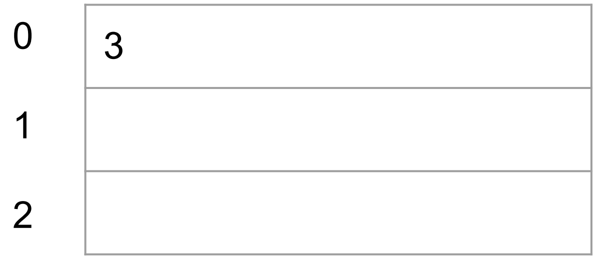
OPERATION	OP CODE	DESCRIPTION
LOAD	0	Add value at address in operand to Accumulator
LOADI	1	Add value of operand to Accumulator
STORE	2	Store value in Accumulator to address of operand Set Accumulator to 0

each address holds an int

For x=3, what byte code will be generated?

Simple Byte Code

OPERATION	OP CODE	DESCRIPTION
LOAD	0	Add value at address in operand to Accumulator
LOADI	1	Add value of operand to Accumulator
STORE	2	Store value in Accumulator to address of operand Set Accumulator to 0

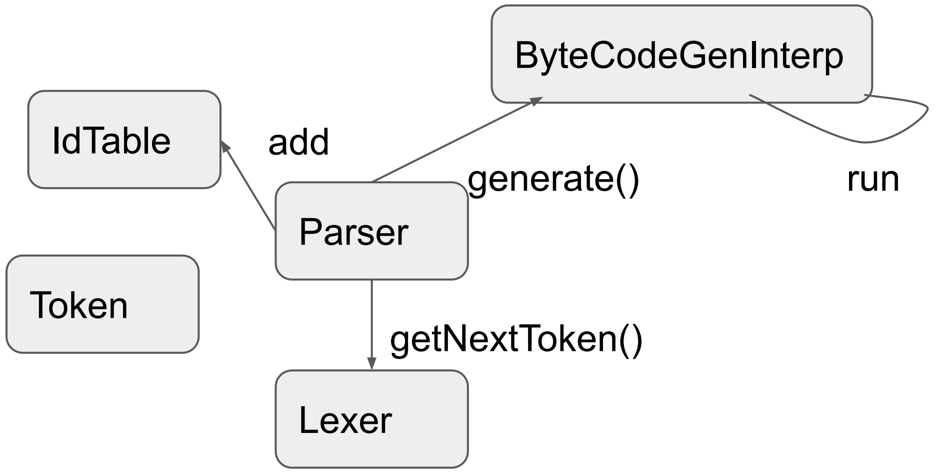


each address holds an int

For x=3, what byte code will be generated?

loadi 3 store 0

An object-oriented program interacting objects



Getting Started

- Project statements and large programs can be daunting-- how do i start???

- Make a plan of small steps
    - 1. Identify the token type of the first token in buffer
    - 2. code getNextToken to get the first token only and return it
    - 3. ...

- The key is to ignore details, temporarily
- and ask questions!