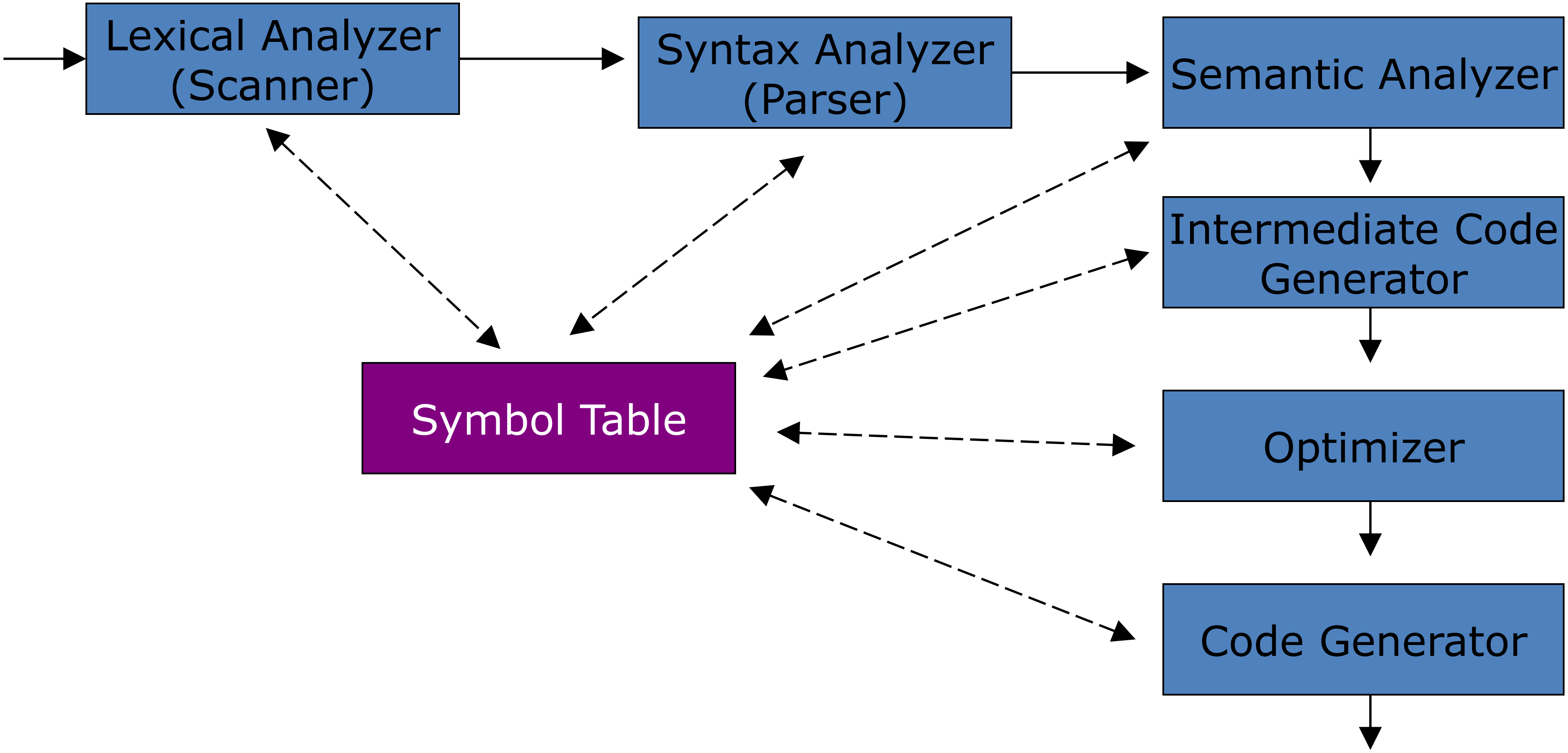
Introduction to Compiler Design

Lesson 3:

Scanners, Finite State Automata

Compilers Organization



The Scanner

- Input: characters from the source program.
- Groups characters into **lexemes**

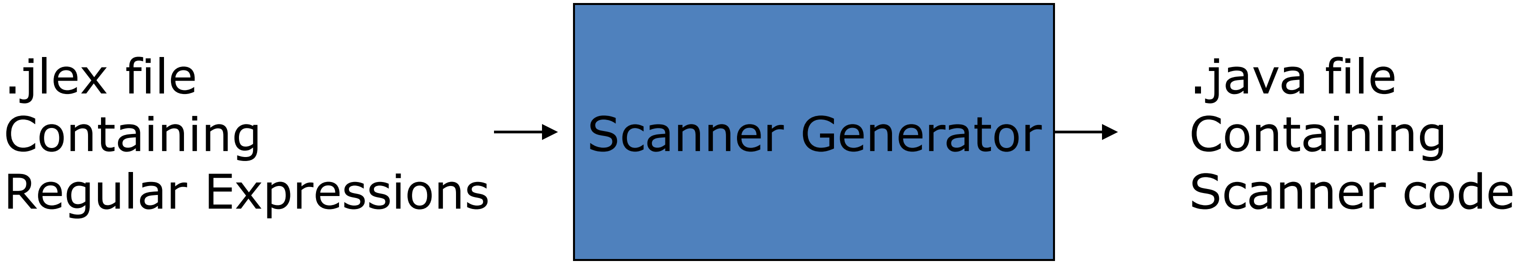
  sequences of characters that "go together"
- Output: **tokens**

  (plus maybe some additional information)
- Scanner also discovers lexical errors (e.g., erroneous characters such as # in java).
- each time scanner's **nextToken()** method is called: find *longest* sequence of characters in input stream, starting with the current character, that corresponds to a lexeme, and should return the corresponding token

Scanner Generators

- Scanner Generators make Scanners (don’t need to hand code a scanner)
- Lex and Flex create C source code for scanner
- JLex creates Java source code for scanner
- Input to Scanner Generator is a file containing (among other things) Regular Expressions

Scanner Generator



.jlex file
Containing
Regular Expressions

Scanner Generator

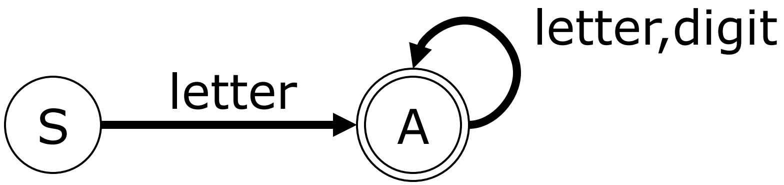
.java file
Containing
Scanner code

To understand Regular Expressions
you need to understand Finite-State Automata

Finite State Automata

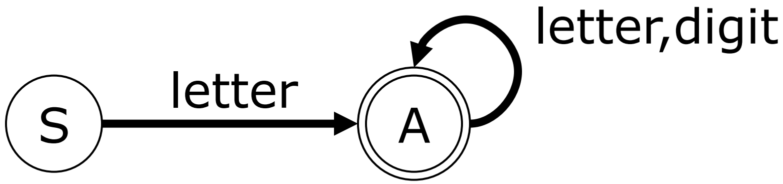
- A compiler recognizes legal *programs* in some (source) language.
- A finite-state machine recognizes legal *strings* in some language.
- The input is a sequence of characters.
- The output is to *accept* or *reject* input

Example FSA



- Nodes are *states*.
- Edges (arrows) are *transitions,* labeled with a single character. My single edge labeled "letter“ stands for 52 edges labeled 'a', 'b', ..., 'z', 'A', ..., 'Z'.(Similarly for “digit“)
- S is the *start state*; every FSA has exactly one (a standard convention is to label the start state "S").
- A is a *final state*. By convention, final states are drawn using a double circle, and non-final states are drawn using single circles. A FSA may have more than one final state.

Applying FSA to Input



aX23
Y1aBss
c

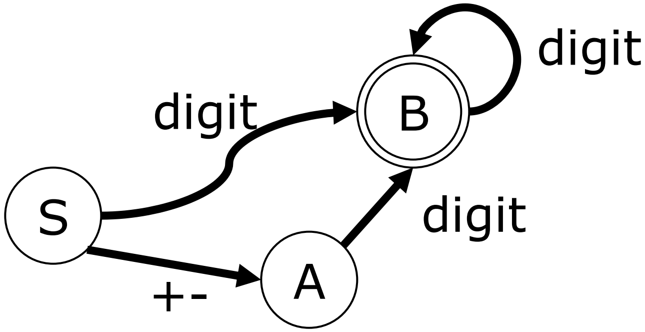
1AbeR6
343
A?

- The FSA starts in its start state.
- If there is a edge out of the current state whose label matches the current input character, then the FSA moves to the state pointed to by that edge, and "consumes" that character; otherwise, it gets stuck.
- The finite-state automata stops when it gets stuck or when it has consumed all of the input characters.
- An input string is *accepted* by a FSA if:
  - The entire string is consumed (the machine did not get stuck)
  - the machine ends in a final state.
- The *language defined by a FSA* is the set of strings accepted by the FSA.

Try It

- **Question 1:** Write a finite-state automata that accepts Java identifiers (one or more letters, digits, underscores, or dollar signs, not starting with a digit).
- **Question 2:** Write a finite-state automata that accepts only Java identifiers that do **not** end with an underscore.

Another Example FSA



FSA accepts integers with optional plus or minus

FSA Formal Definition (5-tuple)

Q – a finite set of states

Σ – The alphabet of the automata
(finite set of characters to label edges)

δ – state transition function
$\delta(\text{state}_i,\text{character}) \rightarrow \text{state}_j$

q – The start state

F – The set of final states

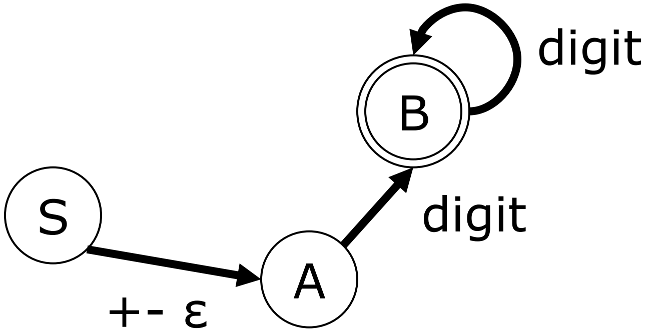
Transition Table for $\delta(\text{state}_i,\text{character}) \rightarrow \text{state}_j$

		Characters		
		+	-	Digit
States	S	A	A	B
	A			B
	B			B

Types of FSA

- Deterministic (DFA)
  - No State has more than one outgoing edge with the same label
- Non-Deterministic (NFA)
  - States *may* have more than one outgoing edge with same label.
  - Edges may be labeled with ε, the empty string. The FSA can take an epsilon transition *without* looking at the current input character.

Example NFA



Consider Scanning +75

After Scanning	Can be in State	
(nothing)	S	A
+	A	-stuck-
+7	B	-stuck-
+75	B	-stuck-

Accept Input

NFA accepts integers with optional plus or minus

A string is accepted by a NFA if there exists a sequence of moves starting in the start state, ending in a final state, that consumes the entire string

NFA, DFA equivalence

For every non-deterministic finite-state automata M, there exists a *deterministic* automata M' such that M and M' accept the *same* language.