Wentworth Institute of Technology

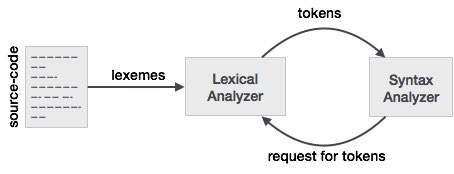
COMP3350 Programming Languages

Lab 4 Lexical Analysis – 15 points

An input program appears to the compiler as a stream of characters. The lexical analyzer collects characters into logical groupings and assigns internal codes to the groupings according to their structure. These logical groups are called lexemes and the internal codes for the categories of these groupings are called tokens.

A lexical analyzer is basically a pattern matcher. Lexemes are recognized by matching the input character string against character string patterns. Instead of representing tokens as integer values, they are often referenced through *named constants.*

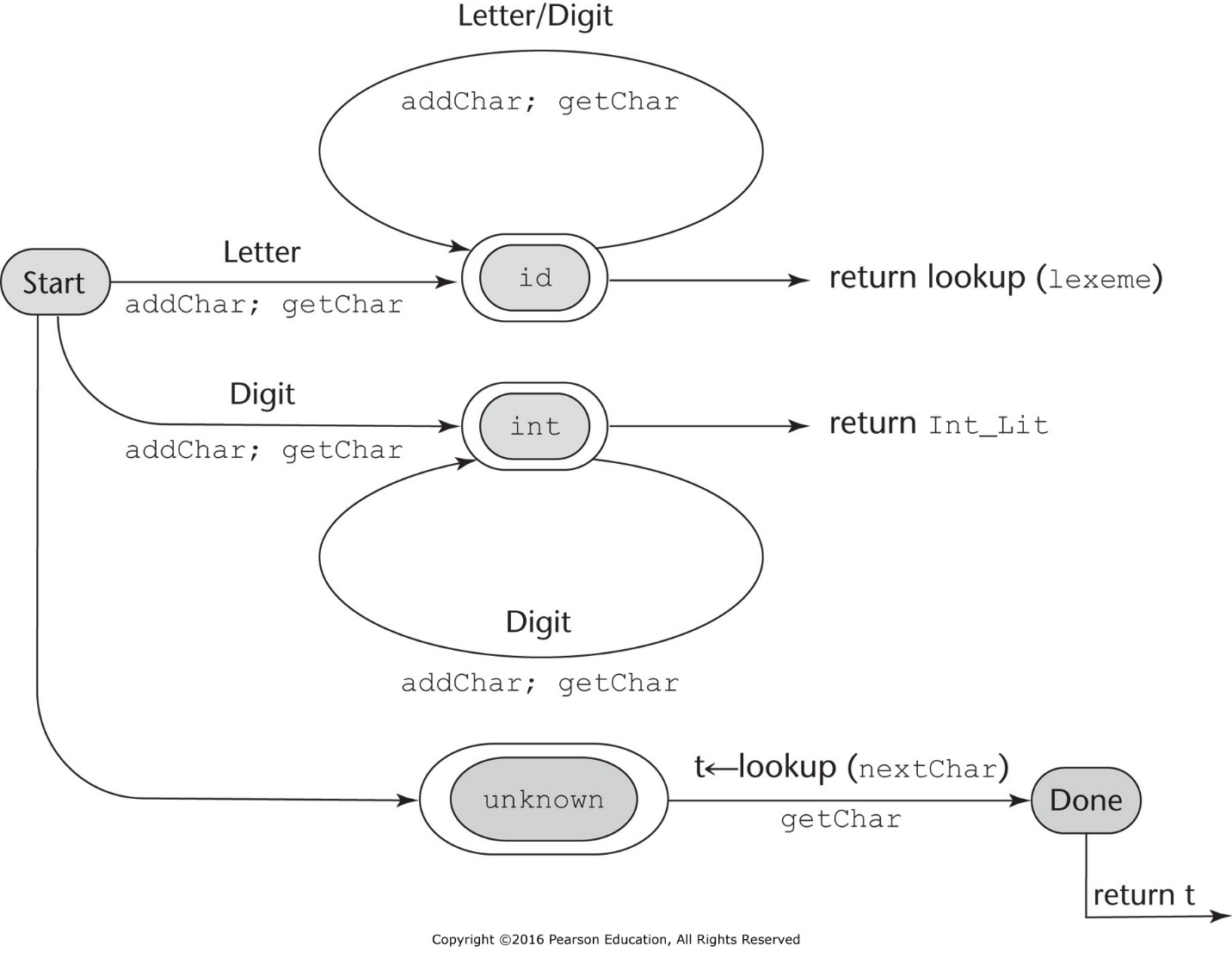
So the Lexical analyzers extract lexemes from a given input string and produces the corresponding tokens. The tokens are passed to the syntax analyzer.



The lexical-analysis process includes skipping comments and white space outside lexemes. Finally, lexical analyzers detect syntactic errors in tokens, such as ill-formed floating-point literals, and report such errors to the user.

One approach to building a lexical analyzer is to design a state transition diagram that describes the token patterns of the language and write a program that implements the diagram. This is the approach we will take in implementing a Lexical analyzer.

Here is the state diagram to recognize names, parentheses, and arithmetic operators.



Assumptions:

1. Variable names consist of strings of uppercase letters, lowercase letters, and digits but must begin with a letter.
2. Names have no length limitation.
3. The character class LETTER defines all 52 letters
4. The character class DIGIT defines all 10 digits.
5. The **Unknown** state is entered when parenthesis and operators are read.

For this Lab you will implement a simple Lexical Analyzer (Lexican) that recognizes identifiers, integers, parentheses, and arithmetic operator using the aforementioned state diagram. The lexical analyzer will read the statement,

(sum + 47) / total;

extract the tokens from the expression and display them. The lexemes and tokens of this statement are:

LEXEMES TOKENS

( RIGHT\_PAREN

sum IDENT

+ ADD\_OP

47 INT\_LIT

) LEFT\_PAREN

/ DIV\_OP

total IDENT

If the program works as expected, it should display the following output.

Next token is: 25, Next lexeme is **(**

Next token is: 11, Next lexeme is **sum**

Next token is: 21, Next lexeme is **+**

Next token is: 10, Next lexeme is **47**

Next token is: 26, Next lexeme is **)**

Next token is: 24, Next lexeme is **/**

Next token is: 11, Next lexeme is **total**

Next token is: -1, Next lexeme is **;**

Attached is pseudo code for a C program that implements the lexical analyzer specified by the state diagram, including the main driver function for testing the analyzer. The header file (.h) contains the definitions of the internal codes assigned to the tokens.

*Using the pseudo code as a reference, implement* Lexican *in Java* and generate tokens for the expression

(sum + 47) / total;

What to submit on Brightspace:

1. The Java source file (Lexican.java)
2. The output of Lexican