**CS 164 Programming Languages and Compilers** **Spring 2019**

# Discussion 1/28: Lexical Analysis

Instructor: Paul N. Hilfinger GSIs: Nikhil Athreya, Vivant Sakore

## 1 Stages of Compilation

1. Compilers are conceptually broken down into multiple phases, each of which carry out transformations on a source program towards the compiler's ultimate goal; the generation of a target program. The result of each phase is an intermediate representation that facilitates the implementation of subsequent phases (or, if it is the last phase, the result is the target program). The first phase in a compiler, the lexical analyzer or *lexer*, reads an input stream of characters and converts it into a stream of *tokens* to be sent to the *parser* for syntactic analysis.

   (a) Convert the following Java source code into a sequence of tokens.

   ```
   // n! = 1 * 2 * 3 * .... * (n-1) * n
   public static int factorial(int n) {
       int result = 1;
       for(int i = 2; i <= n; i++)
           result *= i;
       return result;
   }
   ```

2. What are the rest of the stages of compilation? List them in order and write a brief description about what they and what they take as input and what they output.

## 2 Regular Expressions

The lexer needs to scan and identify finite character sequences that match a pattern corresponding to a particular token (the matching strings are also called *lexemes*). Patterns of finite strings can be codified using *regular expressions*, each of which specify what is called a *regular language*. Kleene's theorem states that any regular language can be recognized by a finite state automaton (FSA) and any language that is recognized by an FSA is regular.

1. What language is denoted by each of the following regular expressions? Try writing down a few simple strings and give a concise description of the language.
   (a) `[_a-zA-Z][_a-zA-Z0-9]*`
   (b) `((`$\epsilon$`|a)b*)*` (Can you simplify this expression?)
2. Write a regular expression for the following languages:
   (a) All strings of uppercase letters, where the letters are in ascending lexicographic order (empty string allowed).
   (b) The following four strings: October 8th, October 8, Oct 8th, Oct 8 (Be as concise as possible with your regex).
   (c) Even binary numbers without leading 0's. (Read from left to right; i.e.,"1101" is 13 and odd).
   (d) Non-empty binary numbers differing in the first and last bits.
3. Check whether a string of nested open- and close- parentheses are balanced.

## 3 Regular Expressions with Backreferences

1. Suppose you have a large spreadsheet, each row of which has a `firstname`, `lastname`, and `email` column. You want to find all people who use an email address of the form `firstname+lastname@gmail.com`. Write a regular expression using backreferences that describes this pattern.