# COMS W4115
# Programming Languages and Translators
# Lecture 3: January 30, 2013
# Structure of a Compiler

## Overview

1. Language processing tools
2. Structure of a compiler
3. The lexical analyzer
4. Language theory background
5. Regular expressions
6. Tokens/patterns/lexemes/attributes

## 1. Language Processing Tools

- Compiler
- Interpreter
- C compiler
- Java compiler
- Just-in-time compiler

## 2. Structure of a Compiler

- Front end: analysis
- Back end: synthesis
- IR: Intermediate representation(s)
- Phases
  - lexical analyzer (scanner)
  - syntax analyzer (parser)
  - semantic analyzer
  - intermediate code generator
  - code optimizer
  - code generator
  - machine-specific code optimizer
- Symbol table
- Error handler
- Compiler component generators
  - lexical analyzer generators: lex, flex
  - syntax analyzer generator: yacc, bison
  - front-end generator: antlr

## 3. The Lexical Analyzer

- The first phase of the compiler is the lexical analyzer, often called a lexer or scanner.
- The lexer reads the stream of characters making up the source program and groups the characters into logically meaningful sequences called lexemes.
- Many lexers use a leftmost-longest rule. For example, a+++++b would be partitioned into the lexemes a `++` `++` `+` b, not a `++` `+` `++` b.
- For each lexeme the lexer sends to the parser a token of the form <token-name, attribute-value>.
- For a token such as an identifier, the lexer will make an entry into the symbol table in which it stores attributes such as the lexeme and type associated with the token.
- The lexer will also strip out whitespace (blanks, horizontal and vertical tabs, newlines, formfeeds, comments).
- Tokens in C
  - identifiers
  - keywords
  - constants
  - string literals
  - operators
  - separators
- Issues in the design of a lexical analyzer
  - efficiency: buffered reads

    - portability and character sets
    - need for lookahead
- Coping with lexical errors
    - types of lexical errors
        - insertion/deletion/replacement/transposition errors
    - edit distance
    - panic mode of error recovery

## 4. Language Theory Background

- Symbol (character, letter)
- Alphabet: a finite nonempty set of characters
    - Examples: {0, 1}, ASCII, Unicode
- String (sentence, word): a finite sequence of characters, possibly empty.
- Language: a (countable) set of strings, possibly empty.
- Operations on strings
    - concatenation
    - exponentiation
        - $x^0$ is the empty string Îµ.
        - $x^i = x^{i-1}x$, for $i > 0$
    - prefix, suffix, substring, subsequence
- Operations on languages
    - union
    - concatenation
    - exponentiation
        - $L^0$ is { Îµ }, even when $L$ is the empty set.
        - $L^i = L^{i-1}L$, for $i > 0$
    - Kleene closure
        - $L^* = L^0$ âˆª $L^1$ âˆª â€¦
        - Note that $L^*$ always contains the empty string.

## 5. Regular Expressions

- A regular expression is a notation for specifying a set of strings.
- Many of today's programming languages use regular expressions to match patterns in strings.
    - E.g., awk, flex, lex, java, javascript, perl, python
- Definition of a regular expression and the language it denotes
    - Basis
        - Îµ is a regular expression that denotes { Îµ }.
        - A single character *a* is a regular expression that denotes { *a* }.
    - Induction: suppose *r* and *s* are regular expressions that denote the languages L(*r*) and L(*s*).
        - (*r*)|(*s*) is a regular expression that denotes L(*r*) âˆª L(*s*).
        - (*r*)(*s*) is a regular expression that denotes L(*r*)L(*s*).
        - (*r*)* is a regular expression that denotes L(*r*)*.
        - (*r*) is a regular expression that denotes L(*r*).
    - We can drop redundant parenthesis by assuming
        - the Kleene star operator * has the highest precedence and is left associative
        - concatenation has the next highest precedence and is left associative
        - the union operator | has the lowest precedence and is left associative
    - E.g., under these rules r|s*t is interpreted as (r)|((s)*(t)).
    - Extensions of regular expressions
        - Positive closure: *r*+ = *rr**
        - Zero or one instance: *r*? = Îµ | *r*
        - Character classes:
            - [abc] = a | b | c
            - [0-9] = 0 | 1 | 2 | â€¦ | 9
- Today regular expressions come many different forms.
    - The earliest and simplest are the Kleene regular expressions: See ALSU, Sect. 3.3.3.
    - Awk and egrep extended grep's regular expressions with union and parentheses.
    - POSIX has a standard for Unix regular expressions.

    - Perl has an amazingly rich set of regular expression operators.
    - Python uses pcre regular expressions.
- Lex regular expressions
    - The lexical analyzer generators flex and lex use extended regular expressions to specify lexeme patterns making up tokens: See ALSU, Fig. 3.8, p. 127.

## 6. Tokens/Patterns/Lexemes/Attributes

- a *token* is a pair consisting of a token name and an optional attribute value.
    - e.g., <id, ptr to symbol table>, <=>
- a *pattern* is a description of the form that the lexemes making up a token in a source program may have.
    - We will use regular expressions to denote patterns.
    - e.g., identifiers in C: `[_A-Za-z][_A-Za-z0-9]*`
- a lexeme is a sequence of characters that matches the pattern for a token, e.g.,
    - identifiers: `count, x1, i, position`
    - keywords: `if`
    - operators: `=, ==, !=, +=`
- an *attribute* of a token is usually a pointer to the symbol table entry that gives additional information about the token, such as its type, value, line number, etc.

## 7. Practice Problems

1. What language is denoted by the following regular expressions?
    a. `(a*b*)*`
    b. `a(a|b)*a`
    c. `(aa|bb)*((ab|ba)(aa|bb)*(ab|ba)(aa|bb)*)*`
    d. `a(ba|a)*`
    e. `ab(a|b*c)*bb*a`
2. Construct Lex-style regular expressions for the following patterns.
    a. All lowercase English words with the five vowels in order.
    b. All lowercase English words with exactly one vowel.
    c. All lowercase English words beginning and ending with the substring "ad".
    d. All lowercase English words in which the letters are in strictly increasing alphabetic order.
    e. Strings of the form `abxba` where `x` is a string of aâ€™s, bâ€™s, and câ€™s that does not contain `ba` as a substring.

## 8. Reading Assignment

- ALSU: Ch. 1, Sects. 3.1-3.3
- See The Lex & Yacc Page for lex, flex, yacc and bison tutorials and manuals.
- See ANTLR 3.x for an antlr video tutorial.

---

aho@cs.columbia.edu