Lexical Analysis

Zakia Zinat Choudhury
Lecturer
Department of Computer Science & Engineering
University of Rajshahi

The Role of Lexical Analyzer

Lexical analyzer is the first phase of a compiler. It is also known as a scanner. So, it's main job is to read the input characters of the source program and group them into lexemes, and produce a sequence of tokens for each lexeme in the source program as an output.

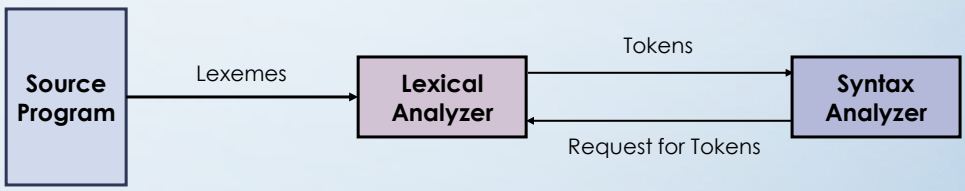

Figure: Interaction between the Lexical analyzer and the Syntax analyzer

The Role of Lexical Analyzer

Lexical analyzers sometimes are divided into two processes:

a) **Scanning** consists of the simple processes that do not require tokenization of the input, such as deletion of comments and compaction of consecutive whitespace characters into one.

b) **Lexical analysis** proper is the more complex portion, where the scanner produces the sequence of tokens as output.

Why lexical analysis and syntax analysis phases are separated?

- Simplicity of design is the most important consideration. The separation of lexical and syntax analysis often allows us to simplify at least one of these tasks.
- Compiler efficiency is improved. A separate lexical analyzer allows us to apply specialized techniques that serve only the lexical task, not the job of parsing.
- Compiler portability is enhanced. Input-device-specific peculiarities can be restricted to the lexical analyzer.

Functions of Lexical Analyzer

- Convert the source code into stream of tokens
- Removing white spaces
- Removing the comments
- Recognizing identifiers and keywords
- Recognizing of constants
- Show error when the lexeme does not match any patterns

Lexemes, Patterns and Tokens

Lexeme:
A lexeme is a sequence of characters in the source program that is matched by the pattern for a token.

Pattern:
A pattern is a description of the form or rule that describes the set of strings.

Token:
A token is a set of strings over source alphabets. Also a token is a pair consisting of a token name and an optional attribute value.
Typical tokens are,
1) Identifiers 2) keywords 3) operators 4) special symbols 5)constants

Attributes for Tokens

When more than one lexeme can match a pattern, the lexical analyzer must provide the subsequent compiler phases additional information about the particular lexeme that matched.

For each lexeme, the lexical analyzer produces as output a token of the form

(token-name, attribute-value)

- token-name is an abstract symbol that is used during syntax analysis
- attribute-value points to an entry in the symbol table for this token.

Specification of Tokens

- An **alphabet** is any finite set of symbols.

  Typical examples of symbols are letters, digits, and punctuation. The set {0,1} is the binary alphabet.

- A **string** over an alphabet is a finite sequence of symbols drawn from that alphabet.

  For example, banana is a string of length six. The empty string, denoted ε, is the string of length zero.

- A **language** is any countable set of strings over some fixed alphabet.

  Abstract languages like Ø, the empty set, or {ε}, the set containing only the empty string, are languages under this definition.

Terms for Parts of Strings

- Prefix
- Suffix
- Substring
- Proper prefixes, suffixes, substrings
- Subsequence

Operations on Languages

- Union of two languages L and M is written as

  L U M = {s | s is in L or s is in M}

- Concatenation of two languages L and M is written as

  LM = {st | s is in L and t is in M}

- The Kleene Closure of a language L is written as

  L* = Zero or more occurrence of language L

- The Positive Closure of a language L is written as

  L+ = One or more occurrence of language L

Regular Expressions

Regular expressions have the capability to express finite languages by defining a pattern for finite strings of symbols.

The grammar defined by regular expressions is known as **regular grammar**.

The language defined by regular grammar is known as **regular language**.

Regular Expressions' Operations

If r and s are regular expressions denoting the languages L(r) and L(s), then

Union : (r) | (s) is a regular expression denoting L(r) U L(s)

Concatenation : (r)(s) is a regular expression denoting L(r)L(s)

Kleene closure : (r)* is a regular expression denoting (L(r))*

(r) is a **regular expression** denoting L(r)

Example of Lexical Analyzer

int position , rate, initial

position = rate + initial *60;

Stream of token

<id,1> <=> <id,2> <+> <id,3> <*> <60>

Symbol Table Manager

Serial no	Variable Name	Variable Type
1	position	int
2	rate	int
3	initial	int

Example of Lexical Analyzer

1. int x1;
   x=23;

2. /*find the total value x and y*/
   int x, y, sum;
   sum = x + y ;
   printf("Total = %d\n", sum);

Lexical Error

Lexical error is a sequence of characters that does not match the pattern of any token.

Lexical phase error can be:

- × **Spelling error.**
- × **Exceeding length of identifier or numeric constants.**
- × **Appearance of illegal characters.**
- × **To remove the character that should be present.**
- × **To replace a character with an incorrect character.**
- × **Transposition of two characters.**

Transition Diagrams

- As an intermediate step in the construction of a lexical analyzer, patterns are converted into stylized flowcharts, called "transition diagrams".
- Transition diagrams have a collection of nodes or circles, called **states**. Each state represents a condition that could occur during the process of scanning the input looking for a lexeme that matches one of several patterns.
- **Edges** are directed from one state of the transition diagram to another.

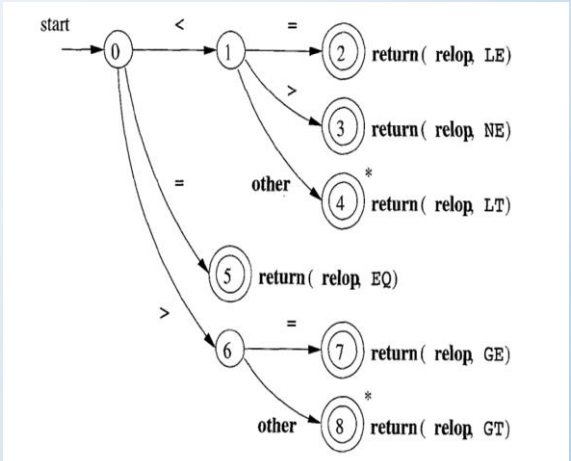

Figure: Transition Diagram of Relation Operator

Finite Automata

- Finite Automata(FA) is the simplest machine to recognize patterns.
- Finite automata come in two flavors:

  (a) Nondeterministic finite automata (NFA) have no restrictions on the labels of their edges. A symbol can label several edges out of the same state, and ε, the empty string, is a possible label.

  (b) Deterministic finite automata (DFA) have, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state.

Thank You