Date: 12th August 2013

#  Compilers Report

 Lexical Analyzer

Lexical Analyzer is a tool, which converts a given string of characters into a sequence of tokens. This forms an integral part of any Compiler.

* **Lexeme:** A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.
* **Token:** A token is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or sequence of input characters denoting an identifier. The token names are the input symbols that the parser processes.
* **Pattern:** A pattern is a description of the form that the lexemes of a token may take. In the case of a keyword as a token, the pattern is just the sequence of characters that form the keyword. For identifiers and some other tokens, the pattern is more complex structure that is matched by many strings.

**Stream of Characters from Input**

LEXICAL ANALYZER

**TOKENS**

PATTERNS

There are several ways to go about implementing the Lexical Analyzer.

# Serial Approach

The patterns to be recognized by the Lexical Analyzer are hard coded as separate machines (a machine here can be implemented by a simple function using switch cases). Use two pointers “Lexeme Begin” and “forward” while scanning the input. Start simulating machine mc1 starting from Lexeme Begin while advancing the forward pointer till a point where the machine accepts the string in-between or rejects it. Then simulate the next machine. Likewise, after simulating all the machines, the lexeme is the longest sequence of characters accepted by a machine.

Update Lexeme Begin pointer to the forward pointer.

MC1

Forward

Lexeme Begin

Simulate Machines one after another

MC2

**INPUT**

**x1 x2 x3………….**

MC3

# Parallel Approach

In this approach, all the machines are run parallel virtually sharing the CPU time. While the “forward” pointer keeps moving forward, some of the machines go into reject states. Proceed going forward until all the machines enter into either accept or reject states. This way, automatically the longest string, which is accepted by a machine, is obtained which is our given Lexeme and now we can update the Lexeme Begin pointer to the forward pointer.

MC1

**TOKEN**

**INPUT**

MC2

MC3

# LEX (Lexical Analyzer Generator):

lex.yy.c

LEX CONFIGURER

**PATTERN**

**(REGULAR EXPRESSION)**

C COMPILER

a.out + BLACK BOX

**TOKENS**

**Stream of Characters**

# Deterministic and Non-Deterministic Finite Automata:

These automate the process of recognizing tokens and they take as input the Regular Expressions of the Patterns to be recognized by the Lexical Analyzer. They have a distinguished advantage, as they are very easy to use. A Non-Deterministic finite automaton (NFA), is a finite state machine that includes at least one state such that, for a given input symbol, the automaton is capable of transitioning into one of two or more different states. A deterministic finite automaton (DFA) is a finite state machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string.

 **DFA D: (Q, Σ, δ, q0, F)**

Set of States

Input

Alphabet

Start State

Accept State

# Simulate a DFA:

**INPUT --- > String X**

 **D = {Q, Σ, δ, s, F}**

**0**

**OUTPUT -> Accept / Reject**

**S = s0;**

**C = read\_next\_char ( );**

**while (C != ‘\0’ )**

 **{**

 **S = δ ( S, C);**

 **C = read\_next\_char( );**

 **}**

**If ( S 🡪 F )**

 **ACCEPT**

**Else**

 **REJECT**