Assignment No : A3

1 Title:

Lexical Analyzer.

2 Problem Definition:

Lexical analyzer for sample language using LEX.

3 Learning Objectives:

1. To understand concept of lexical analyzer.
2. To use LEX as a lexical analyzer generator.

4 S/W and H/W requirements:

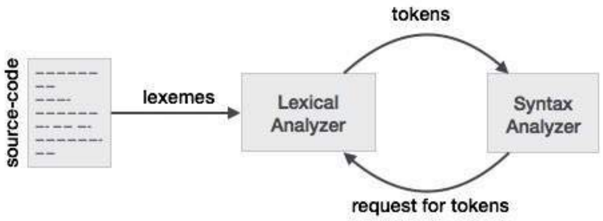
1. Open source 64 bit OS.
2. Gedit text editor.
3. flex.

5 Theory

Lexical Analysis:

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.



Tokens:

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.
In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuations symbols can be considered as tokens.

Symbol Table:

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.
A symbol table may serve the following purposes depending upon the language in hand:

1. To store the names of all entities in a structured form at one place.
2. To verify if a variable has been declared.
3. To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
4. To determine the scope of a name (scope resolution).

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the following format.
¡symbol name, type, attribute¿

LEX:

Lex is a program generator designed for lexical processing of character input streams. It accepts a high-level, problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions. At the boundaries between strings program sections provided by the user are executed. The Lex source file associates the regular expressions and the program fragments. As each expression appears in the input to the program written by Lex, the corresponding fragment is executed.

The user supplies the additional code beyond expression matching needed to complete his tasks, possibly including code written by other generators. The program that recognizes the expressions is generated in the general purpose programming language employed for the user's program fragments. Thus, a high level expression language is provided to write the string expressions to be matched while the user's freedom to write actions is unimpaired. This avoids forcing the user who wishes to use a string manipulation language for input analysis to write processing programs in the same and often inappropriate string handling language.

6 Related Mathematics

Let S be the solution perspective of the given problem.
The set S is defined as:
$S = \{\ s, e, X, Y, F, DD, NDD|_s\}$
Where,
s= Start point
e= End point
F= Set of main functions
DD= set of deterministic data
NDD= set of non deterministic data

X= Input Set.
X = $\{sourceprogramcodeinhighlevellanguage, matchingrules\}$

Y = $\{tokens, symboltable\}$

s = sample language ready into read buffer.
e = tokens created and symbol table made.

F = $\{f_{read}, f_{match}, f_{st}\}$

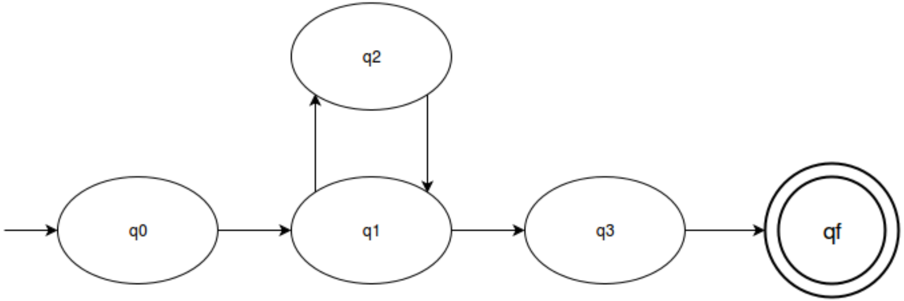
f_{read} :function to read the sample language.

f_{match} :function to match the strings in sample language with the token rules.

f_{st} :function to add token information into symbol table.

DD = $\{samplecode, matchingrules\}$
NDD = ϕ

7 State Diagram



q0 = read input code
q1 = token generation state
q2 = sysmbol table maintainance state
q3 = display symbol table state
qf = final state