MALOUT INSTITUTE OF MANAGEMENT

AND

INFORMATION TECHNOLOGY

ASSIGNMENT NO 1:

**Name**: RISHAV SHANDILYA

**Class**: B. Tech CSE

**Semester**: 6th

**Reg No**: RA1911003010915

**Subject**: Compiler Design

1. **Introduction:**

**LEXICAL ANALYSIS** is the very first phase in the compiler designing. It takes the modified source code which is written in the form of sentences. In other words, it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyser breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.

Programs that perform lexical analysis are called lexical analysers. A lexical analyser contains tokenizer or scanner. If the lexical analyser detects that the token is invalid, it generates an error. It reads character streams from the source code, checks for legal tokens, and pass the data to the syntax analyser when it demands.

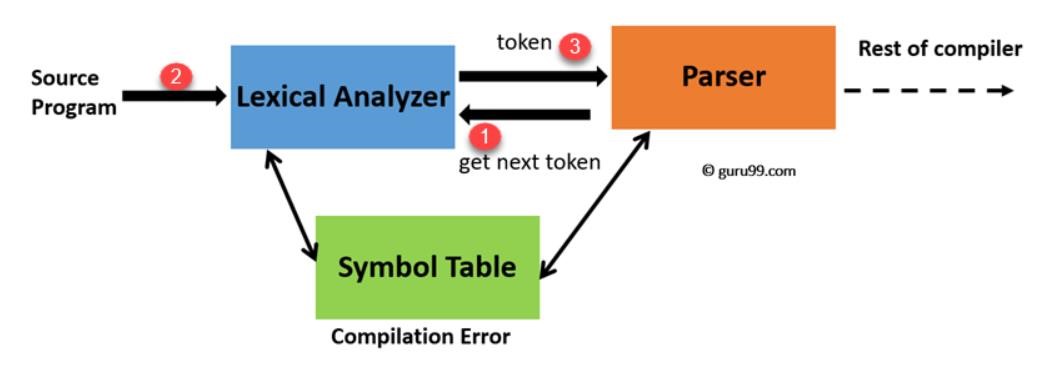
**A**

**rchitectu**

**re and**

**Working**

**:**



1. **My Assumptions**:

While designing the lexical analyser for a language L1, I have assumed the following assumptions.

**Note: My language L1 will ignore all single line comments ‘//’ ,’whitespaces’ and ‘\n’ while reading the tokens.**

**Special Symbol**: ; { } ( ) , #

**Keyword**: int, char, float, bool, cin, cout, main

**Pre-processor Directives**: include, define

**Library:** iostream, studio, string

**Operators:** \*, +, >>, <<,>,<

**Numbers/Integers:** All numbers Values from 0-9.

**Identifies/ Variables:** All alphabetic strings except the keywords, numbers, Pre-processor directive and library strings.

# 3. Lexical Analyzer (LEX.cpp)

**#include <bits/stdc++.h>**

**#include <regex>**

**#include <iterator>**

**#define deb(x) cout<<#x<<" = "<<x<<endl**

**using namespace std;**

**map<string,string> Make\_Regex\_Map(){**

**map<string,string> my\_map {**

**{ "\\;|\\{|\\}|\\(|\\)|\\,|\\#", "Special Symbol"},**

**{ "int|char|float|bool|cin|cout|main|using|namespace|std","Keywords"},**

**{ "\\include|define", "Pre-Processor Directive"},**

**{"\\iostream|\\stdio|\\string","Library"},**

**{ "\\\*|\\+|\\>>|\\<<|<|>", "Operator"},**

**{ "[0-9]+" , "Integer" },**

**{ "[^include][^iostream][^int][^main][^cin][^cout][^;][^>>][^,][^[B ;cin]][az]+" , "Identifier" },**

**{ "[A-Z]+", "Variable"},**

**{"[ ]",""},**

**}; return my\_map;**

**} map<size\_t,pair<string,string>> Match\_Language (map<string,string> patterns,string str){**

**map< size\_t, pair<string,string> > lang\_matches;**

**for ( auto i = patterns.begin(); i != patterns.end(); ++i )**

**{ regex compare(i->first);**

**auto words\_begin = sregex\_iterator( str.begin(), str.end(), compare ); auto words\_end = sregex\_iterator();**

**//MAKING PAIRS OF [STRING OF REGEX 'compare' : 'pattern'] for ( auto it = words\_begin; it != words\_end; ++it ) lang\_matches[ it->position() ] = make\_pair( it->str(), i->second );**

**}**

**return lang\_matches;**

**} string tell\_Lexeme(string op){ if(op=="\*") return "MUL"; else if(op=="+") return "ADD"; else if(op==">>") return "INS"; else if(op=="<<") return "EXTR"; else if(op==">") return "RSHFT"; else if(op=="<") return "LSHFT";**

**}**

## int main()

**{ ofstream fout; fout.open("OutputFile"); char c; fstream fin("SourceCode.txt", fstream::in); string str;**

**//Fetching Source Code in String type 'str' while(fin>> noskipws>>c) str=str+c;**

**//Making a map which which will define the regex in source code to its pattern in my l anguage.**

**map<string,string> patterns =Make\_Regex\_Map();**

**/\*DECLARING MAP 'lang\_matches' from 'patterns' map which will pair up the patterns from the ['Source Code':'Defined Pattern' via a Regex named 'compare'. \*/ map< size\_t, pair<string,string> > lang\_matches = Match\_Language(patterns,str);**

**// Writing matches in File ignoring 'spaces' and '\n'.**

**for ( auto match = lang\_matches.begin(); match != lang\_matches.end(); ++match ){**

**if(!(match->second.first==" ")&&!(match->second.first=="//")){ if(match->second.second=="Variable"||match->second.second=="Identifier") fout<<"Token ( '"<< match->second.first << "' " <<" -------> "<< match-**

**>second.second <<" , POINTER TO SYMBOL TABLE )"<<endl; else{ if(match->second.second=="Operator"){ string op=tell\_Lexeme(match->second.first); fout<<"Token ( "<< match->second.first << " " <<" -------> "<< match-**

**>second.second<<" , "<<op<<" )" <<endl;**

**} else fout<<"Token ( "<< match->second.first << " " <<" -------> "<< match-**

**>second.second<<" )" <<endl;**

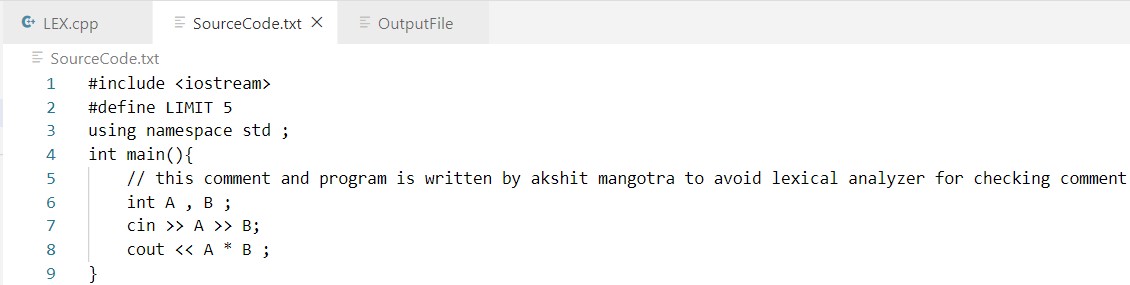
**}**

**} } return 0;**

**}**

**4.**

**Source File (SourceFile.txt)**



# 5. OUTPUT FILE CONTAINING TOKEN (OutputFile.txt)

