**EXPERIMENT-14**

**Aim** : Design a lexical analyzer for the given language. The lexical analyzer should ignore redundant spaces, tabs and newlines, comments etc.

**Theory :** Lexical analysis is the first phase of a compiler. It takes 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.

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 punctuation symbols can be considered as tokens.

**Procedure :**

We make use of the following two functions in the process. look up() – it takes string as an argument and checks its presence in the symbol table. If the string is found then returns the address else it returns NULL. insert() – it takes string as its argument and the same is inserted into the symbol table and the corresponding address is returned.

1. Start

2. Declare an array of characters, an input file to store the input;

3. Read the character from the input file and put it into a character type of variable, say ‘c’.

4. If ‘c’ is blank then do nothing.

5. If ‘c’ is a new line character line=line+1.

6. If ‘c’ is digit, set token Val, the value assigned for a digit and return the ‘NUMBER’.

7. If ‘c’ is a proper token then assign the token value.

8. Print the complete table with Token entered by the user, Associated token value. 9. Stop

**Program :**

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

using namespace std;

int isKeyword(char buffer[]){

Char keywords[32][10] = {"auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef","union",

"unsigned","void","volatile","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main(){

char ch, buffer[15], operators[] = "+-\*/%=";

ifstream fin("program.txt");

int i,j=0;

if(!fin.is\_open()){

cout<<"error while opening the file\n";

exit(0);

}

while(!fin.eof()){

ch = fin.get();

for(i = 0; i < 6; ++i){

if(ch == operators[i])

cout<<ch<<" is operator\n";

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

cout<<buffer<<" is keyword\n";

else

cout<<buffer<<" is indentifier\n";

}

}

fin.close();

return 0;

}

**Output :**



