EXPERIMENT 3 - LEXICAL ANALYSER

AIM:

To create a lexical analyser and apply it on a program.

ALGORITHM:

1.The program consists of a class called lexical analyser that encapsulates all the methods required to perform lexical analysis of a cpp program.
2.The constructor creates a set named keywords consisting of all the keywords in cpp.
3.The analyse function takes a program and classifies its tokens into identifiers, constants, operators and keywords.
4.It starts off by splitting the program into separate lines and removing all one line comments.
5.After removing comments it uses a set of regular expressions to obtain various types of tokens.

[a-zA-Z_][a-zA-Z0-9_]* - Matches any string that starts with an alphabet or an underscore followed by alphanumeric characters or underscores. This regex will capture all the identifiers and keywords.

[*\+\-\%\/\=] - Matches operators.

[{}\[\],\(\)] - Matches punctuations

[0-9]+[\.]?[0-9]* - Matches any integer, floating point or double values.

\".*\" - Matches any values enclosed within double quotes(Strings)

\'.*\' - Matches any values enclosed within single quotes(Characters)

6.The strings, characters and numerical values(float,int double etc.) are stored into a list named constants.
7.The tokens list contains identifiers and keywords. The next for loop segregates them by iterating through each value and verifying if its present in the list of keywords in cpp or not.
8.Finally any value within strings that is classified as an identifier is removed from the set of identifiers .
9.The result contains four iterables each containing constants,keywords,operators and identifiers respectively.
10.To test the lexical analyser a program is read from a file and is supplied to the lexical analyser and the result is printed.

PROGRAM:

```
import re


class LexicalAnalyser:
    def __init__(self):
        self.keywords = {
            "include", "auto", "double", "int", "struct", "break", "else",
            "long", "switch", "case", "enum", "register", "typedef", "char",
```

```
        "extern", "return", "union", "continue", "for", "signed", "void",
        "do", "if", "static", "while", "default", "goto", "sizeof",
        "volatile", "const", "float", "short", "unsigned", "using",
        "namespace"
    }

def isKeyword(self, word):
    return word in self.keywords

def getOperators(self, line):
    operators = []
    matches = re.findall("[\*\+\-\%\/]", line)
    operators.extend(matches)
    prev = ""
    ops = {"<", ">", "=", "!"}
    for char in line:
        if (char in ops):
            if (prev == ""):
                prev = char
            else:
                operators.append(prev + char)
                prev = ""
        elif (prev != ""):
            operators.append(prev)
            prev = ""
    return operators

def analyse(self, program: str):
    lines = program.split("\n")
    commentLines = []
    for i in range(len(lines)):
        comments = lines[i].find("//")
        if (comments != -1):
            commentLines.append(lines[i][comments + 2:])
            lines[i] = lines[i][:comments]
    tokens = []
    operators = []
    constants = []
    punctuations = []
    characters = 0
    whiteSpaces = 0
```

```
for line in lines:
    if (line[0] == "#"):
        continue
    whiteSpaces += len(line.split(" ")) - 1
    characters += len(line)
    matches = re.findall("[a-zA-Z_][a-zA-Z0-9_]*", line)
    tokens.extend(matches)
    operators.extend(self.getOperators(line))
    matches = re.findall("[{}\[\],\(\);]", line)
    punctuations.extend(matches)
    numbers = re.findall("[0-9]+[\.]?[0-9]*", line)
    constants.extend(numbers)
    stringLiterals = re.findall("\".*\"", line)
    constants.extend(stringLiterals)
    characterLiterals = re.findall("\'.*\'", line)
    constants.extend(characterLiterals)
identifiers = set()
keywords = set()
for token in tokens:
    if (self.isKeyword(token)):
        keywords.add(token)
    else:
        identifiers.add(token)
for i in range(len(constants)):
    item = constants[i]
    if (item[0] == "'" and item[-1] == "'"):
        if (item[1:-1] in identifiers):
            identifiers.remove(item[1:-1])
        """ if (item[1:-1] in keywords):
            keywords.remove(item[1:-1]) """
tokenCount = len(identifiers) + len(operators) + len(keywords) + len(
    constants)
return {
    "punctuations": punctuations,
    "identifiers": identifiers,
    "operators": operators,
    "keywords": keywords,
    "constants": constants,
    "characters": characters,
    "lines": len(lines),
```

```
            "tokenCount": tokenCount,
            "commentLines": commentLines,
            "whiteSpaces": whiteSpaces
        }


program = ""
with open("file.cpp") as file:
    program = file.read()
analyser = LexicalAnalyser()
result = analyser.analyse(program)
for item in result:
    if hasattr(result[item], '__iter__'):
        for element in result[item]:
            print(item[:-1].ljust(15) + " " + element)
    else:
        print(item.ljust(15) + " " + str(result[item]))
```

File given as input:

```
#include <string>
using namespace std;
int main()
{
    int a = 15;
    float b = 10.2;
    if (a >= b)
        b = a;
    char c = 'a';
    string s = "int";
    double x = a + b; //this is a comment
}
```

OUTPUT:

```
punctuation     ;
punctuation     (
punctuation     )
punctuation     {
punctuation     ;
punctuation     ;
punctuation     (
punctuation     )
punctuation     ;
punctuation     ;
punctuation     ;
punctuation     ;
punctuation     }
identifier      std
identifier      b
identifier      main
identifier      s
identifier      x
identifier      a
identifier      string
identifier      c
operator        =
operator        =
operator        >=
operator        =
operator        =
operator        =
operator        +
operator        =
keyword         int
keyword         char
keyword         namespace
keyword         using
keyword         float
keyword         double
keyword         if
constant        15
constant        10.2
constant        'a'
constant        "int"
characters      155
lines           12
tokenCount      27
commentLine     this is a comment
whiteSpaces     58
```

RESULT:

A lexical analyser is created an applied on a program.