FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

System Programming and Compiler Construction

VI Semester (Computer) **Academic Year: 23-24**

Experiment No 2

- *Ivan Dsouza*
- *9601*
- *T.E. Comps A (Batch C)*

Aim: Write a program to implement Lexical analyzer

Learning Objective: Converting a sequence of characters into a sequence of tokens.

Theory:

THE ROLE OF LEXICAL ANALYZER

The lexical analyzer is the first phase of a compiler. Its main task is to read the input

characters and produce as output a sequence of tokens that the parser uses for syntax analysis. Upon receiving a “get next token” command from the parser, the lexical analyzer reads input characters until it can identify the next token.

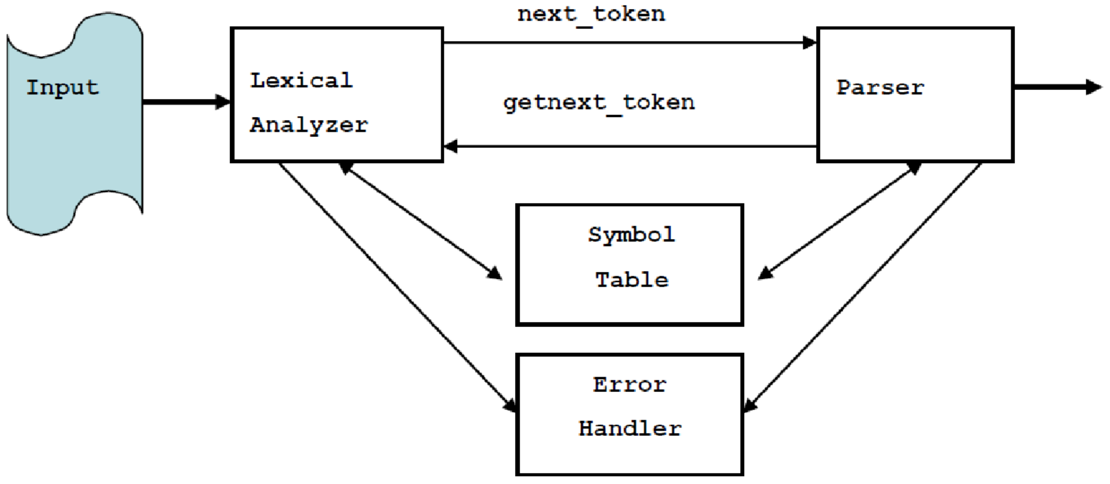


Figure 4.1 Interaction of Lexical Analyzer with Parser

Since the lexical analyzer is the part of the compiler that reads the source text, it may also perform certain secondary tasks at the user interface. One such task is stripping out from the source program comments and white spaces in the form of blank, tab, and new line characters. Another is correlating error messages from the compiler with the source program. Sometimes lexical analyzers are divided into a cascade of two phases first called "scanning" and the second "lexical analysis". The scanner is responsible for doing simple tasks, while the lexical analyzer proper does the more complex operations.

Implementation Details

1. Read the high level language as source program
2. Convert source program in to categories of tokens such as Identifiers, Keywords, Constants, Literals and Operators.

Test cases:

1. Input undefined token

Conclusion:

The role of Lexical Analyser in Compiler Construction was learnt. The conversion of a string of characters into a string of tokens was also performed and understood.

Post Lab Questions:

1. Explain the role of automata theory in compiler design.
2. What are the errors that are handled by Lexical analysis phase?

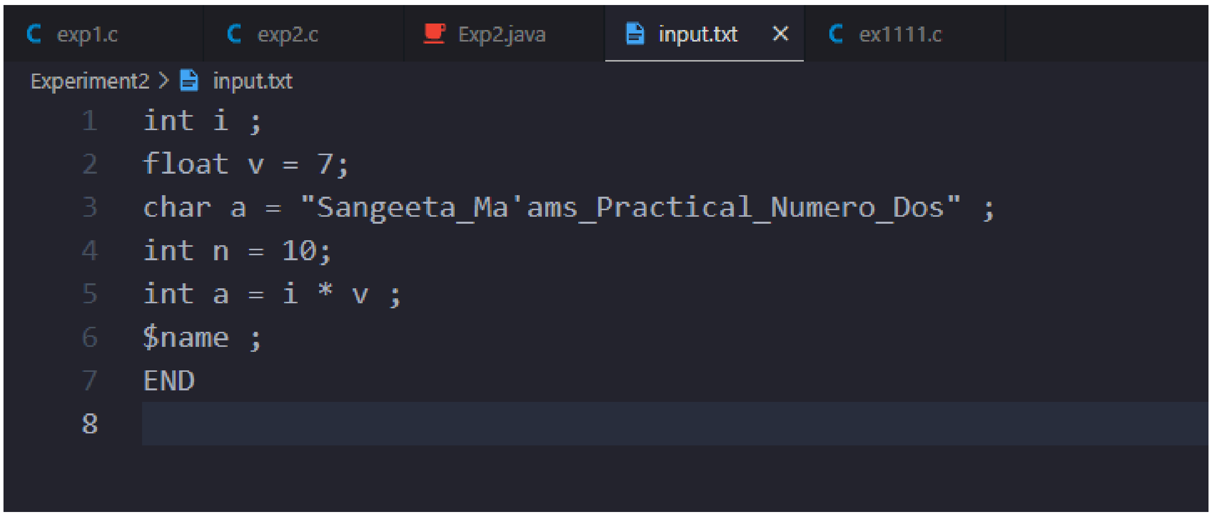
Code:

Lexical Analyser Code:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
// I1 Dsouza - 9601
// Exp2
public class Exp2 {

    private static final String[] KEYWORDS = {"auto", "double",
"int", "struct", "break", "else", "long", "switch",
            "case", "enum", "register", "typedef", "char",
"extern", "return", "union", "const", "float", "short",
            "unsigned", "continue", "for", "signed", "void",
"default", "goto", "sizeof", "volatile", "do", "if",
            "static", "while"};

    private static final String[] OPERATORS = {"+", ">", "~",
"%=", "-", "<", "&", "<<=", "*", ">=", "^", ">>=",
            "/", "<=", "|", "&=", "%", "&&", "=", "^=", "++",
"||", "+=", "|=", "--", "!", "-=", "==", "<<", "*=",
            "!=", ">>", "/="};

    private static final String[] SPECIAL_SYMBOLS = {"[", "]",
"{", "}", ",", ";", ":", "(", ")"};

    private static boolean keyword(String word) {
        for (String keyword : KEYWORDS) {
            if (keyword.equals(word)) {
                return true;
            }
        }
        return false;
    }

    private static boolean operator(String word) {
        for (String operator : OPERATORS) {
            if (operator.equals(word)) {
```

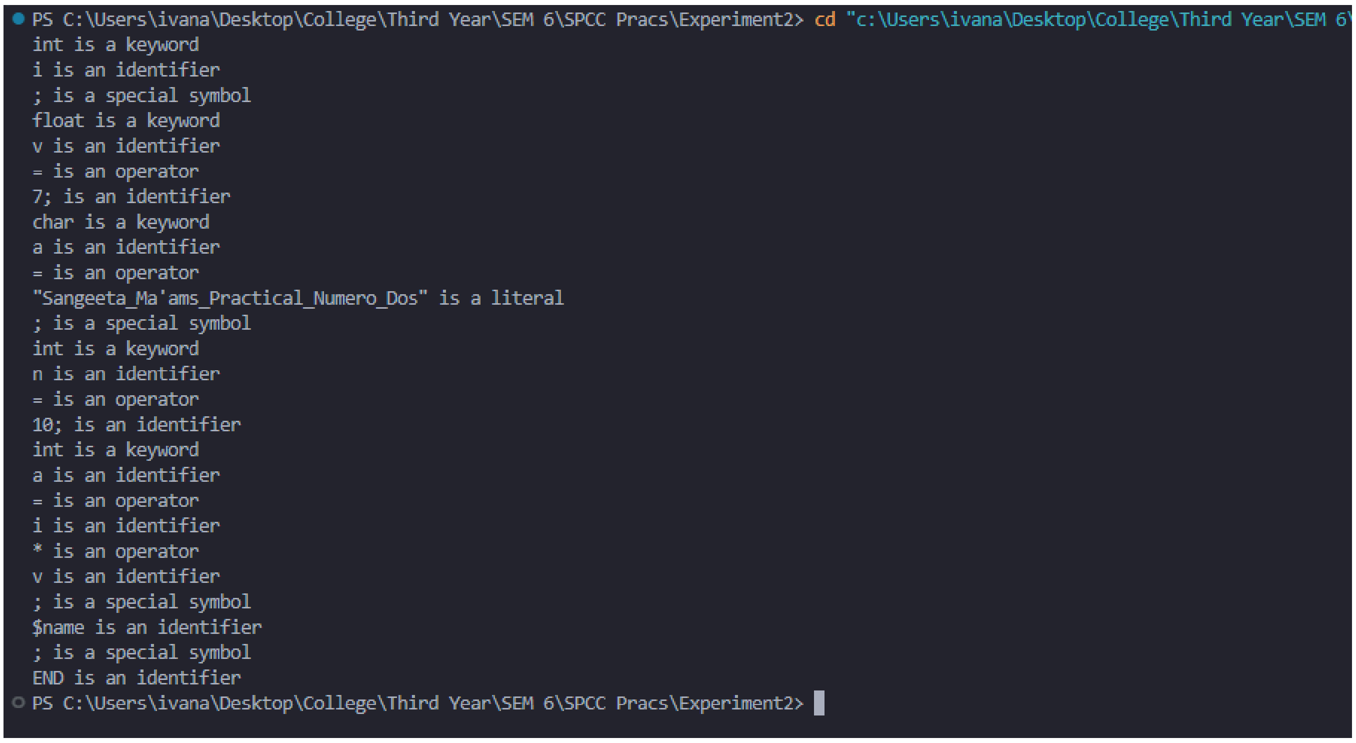
```java
                return true;
            }
        }
        return false;
    }

    private static boolean constant(String word) {
        for (char ch : word.toCharArray()) {
            if (ch == '.') continue;
            if (!Character.isDigit(ch)) {
                return false;
            }
        }
        return true;
    }

    private static boolean special(String word) {
        for (String symbol : SPECIAL_SYMBOLS) {
            if (symbol.equals(word)) {
                return true;
            }
        }
        return false;
    }

    private static boolean literal(String word) {
        int length = word.length();
        return (length > 1 && ((word.charAt(0) == '"' &&
word.charAt(length - 1) == '"') ||
                (word.charAt(0) == '\'' && word.charAt(length - 1)
== '\'')));
    }

    public static void main(String[] args) {
        String fileName = "input.txt";
        try (Scanner scanner = new Scanner(new File(fileName))) {
            while (scanner.hasNext()) {
                String word = scanner.next();
                if (keyword(word)) {
                    System.out.println(word + " is a keyword");
                } else if (operator(word)) {
                    System.out.println(word + " is an operator");
                } else if (constant(word)) {
                    System.out.println(word + " is a constant");
                } else if (special(word)) {
                    System.out.println(word + " is a special
symbol");
                } else if (literal(word)) {
                    System.out.println(word + " is a literal");
                } else {
                    System.out.println(word + " is an
identifier");
                }
            }
        } catch (FileNotFoundException e) {
            System.err.println("Can't open " + fileName + " for
reading.");
        }
    }
}
```

Input.txt file



```
int i ;
float v = 7;
char a = "Sangeeta_Ma'ams_Practical_Numero_Dos" ;
int n = 10;
int a = i * v ;
$name ;
END
```

Output:



```
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\SPCC Pracs\Experiment2> cd "c:\Users\ivana\Desktop\College\Third Year\SEM 6\
int is a keyword
i is an identifier
; is a special symbol
float is a keyword
v is an identifier
= is an operator
7; is an identifier
char is a keyword
a is an identifier
= is an operator
"Sangeeta_Ma'ams_Practical_Numero_Dos" is a literal
; is a special symbol
int is a keyword
n is an identifier
= is an operator
10; is an identifier
int is a keyword
a is an identifier
= is an operator
i is an identifier
* is an operator
v is an identifier
; is a special symbol
$name is an identifier
; is a special symbol
END is an identifier
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\SPCC Pracs\Experiment2>
```

Postlab:

9601 Ivan Dsouza
9601
T.E.-COMPS A

Experiment 2 - Postlab SPCC

1) Role of Automata Theory in Compiler Design

Ans Automata theory plays a crucial role in compiler design, particularly in the lexical analysis phase.

Lexical Analysis involves recognizing and tokenizing the source code into meaningful units. Finite Automata (FA) and Regular Expressions are fundamental concepts ~~are~~ derived from Automata theory and are extensively used in designing lexical Analyzers. Regular expressions describe the patterns of tokens and finite automata help efficiently recognize and match these patterns. By Employing Automata Theory, compiler designers can create robust lexical Analyzers that efficiently process source code.

2) Errors Handled by LA phase:

Ans i) Illegal Characters: Lexical Analyzers (LA) identify and report characters that do not conform to the programming languages syntax

ii) Incorrect Identifiers and keywords:

Detection of invalid identifiers or keywords that violate language rules.

iii) Unterminated Strings or Comments:
Recognition of unterminated string literals or comments in the source code.

iv) Unexpected End-of-file: Detection of situation where the source code ends unexpectedly without completing a valid construct.