NOTRE DAME UNIVERSITY

BANGLADESH

CSE-3104 LAB Report-02

Submitted by:

Name: Istiak Alam

ID: 0692230005101005

Batch: CSE-20

Date: 15-09-24

Course Title: Compiler Design Lab

Course Code: CSE-3104

Lab Task Topic: Compiler Problem Solving using Lex

Submitted To:

Khorshed Alam
Lecturer, NDUB

Problem:

Write a Lexical Analyzer that will-

- Ignore White Space
- Match all numbers and insert it in symbol table
- Count line numbers
- Match all Identifiers (key words, variables)

  Where Keywords are - program, if, not, end, begin, else, then, do, while, function, Procedure, integer, int, real, var, oh, array, write .

  And variables can be start with a letter or underscore (_)

  Relational Operators are =, <> , < , <=, >, >=

  Addition Operator is +

  Subtraction Operator is -

  Multiplier Operators is *

  Division Operator is /

Print the corresponding token name and line number of occurring.

Solution:

Code : –

```
%{
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int num_lines = 1;            // Counter for line numbers
int symbol_table[100];        // Simple array to simulate symbol table for numbers
int sym_index = 0;            // Index for symbol table

                              // Function to check if a string is a keyword
int is_keyword(char* str) {
    char* keywords[] = {"program", "if", "not", "end", "begin", "else", "then",
                        "do", "while", "function", "Procedure", "integer",
                        "int", "real", "var", "oh", "array", "write"};
    int num_keywords = 18;    // Total number of keywords
    for (int i = 0; i < num_keywords; i++) {
        if (strcmp(str, keywords[i]) == 0) {
            return 1;         // It's a keyword
        }
    }
    return 0;                 // Not a keyword
}

%}

%%

                              //Ignoring whitespaces (space, tab, newline, etc.)
[ \t]+          { /* Ignore whitespaces */ }

       /*Count lines based on newline characters*/
\n     { num_lines++; }
```

```
        /*Matching keywords and identifiers (variables start with a letter or underscore)*/
[_a-zA-Z][_a-zA-Z0-9]* {
    if (is_keyword(yytext)) {
        printf("Keyword: %s at line %d\n", yytext, num_lines);
    } else {
        printf("Identifier: %s at line %d\n", yytext, num_lines);
    }
}

        /*Matching numbers and inserting them into the symbol table*/
[0-9]+        {
    symbol_table[sym_index++] = atoi(yytext);
        //atoi( ) converts an integer value from a string of characters
    printf("Number: %s inserted in symbol table at line %d\n", yytext, num_lines);
}

        /*Relational Operators*/
"="             { printf("Relational Operator: = at line %d\n", num_lines); }
"<>"            { printf("Relational Operator: <> at line %d\n", num_lines); }
"<="            { printf("Relational Operator: <= at line %d\n", num_lines); }
">="            { printf("Relational Operator: >= at line %d\n", num_lines); }
"<"             { printf("Relational Operator: < at line %d\n", num_lines); }
">"             { printf("Relational Operator: > at line %d\n", num_lines); }

        /*Addition, Subtraction, Multiplication, and Division operators*/
"+"             { printf("Addition Operator: + at line %d\n", num_lines); }
"-"             { printf("Subtraction Operator: - at line %d\n", num_lines); }
"*"             { printf("Multiplication Operator: * at line %d\n", num_lines); }
"/"             { printf("Division Operator: / at line %d\n", num_lines); }

        /*Catch-all for other characters (optional)*/
.               { printf("Unrecognized character: %c at line %d\n", yytext[0], num_lines); }

%%

int main( ) {
    printf("Starting lexical analysis...\n");
    yylex();  // Start the lexical analysis
    printf("Lexical analysis complete.\n");
    return 0;
}
```

Code Processing –

Step 1 : Open terminal in Linux and create a lex file named **analyzer.l**

⇒ **`touch analyzer.l`**

Step 2 : After Writing the code save it and type command in terminal :

⇒ **`lex analyzer.l`**

Step 3 : It will create a lex.yy.c file. Then next type command in terminal :

⇒ **`cc lex.yy.c -o analyzer -ll`**

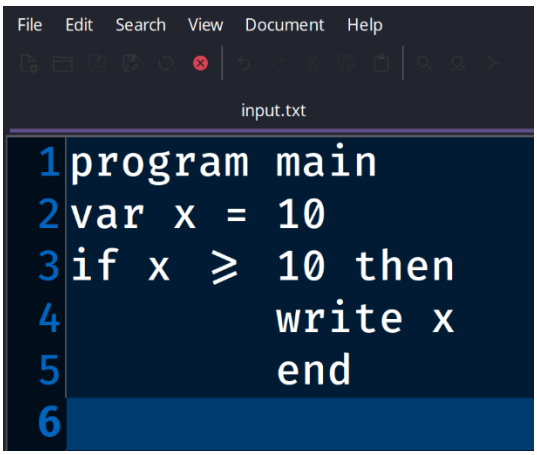
Step 4 : Then it will create an executable file named **analyzer.exe**.

Step 5 : Next type command in terminal and run the exe file :

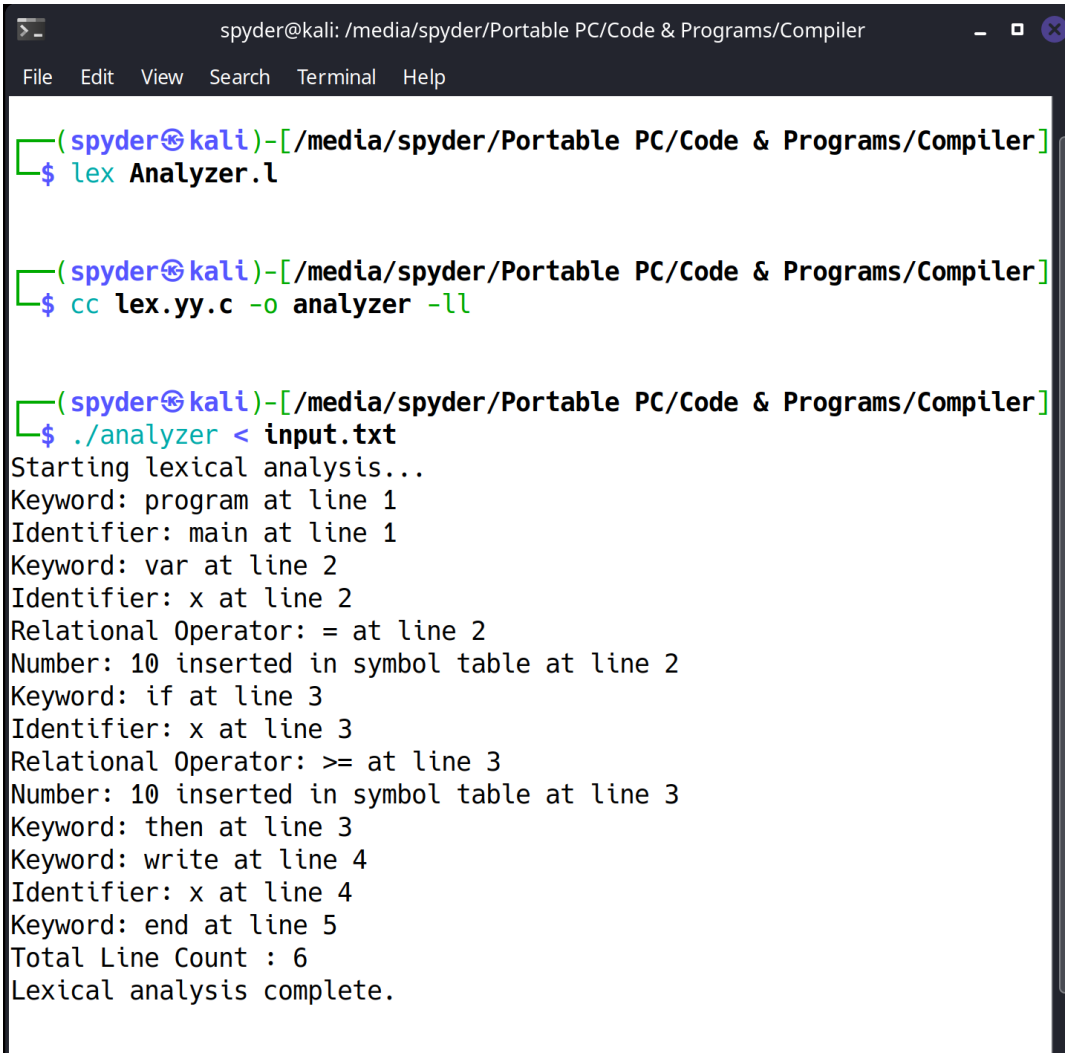
⇒ `./analyzer < input.txt`

Input and Output:

1. We are using **input.txt** file for checking out inputs in this lexical analyzer.
2. In the **input.txt** file there are a simple program language –



3. After injecting the input, the output is –



Output