Daffodil International University
Department of Computer Science & Engineering
Semester: Fall 2022

COURSE TITLE: Compiler Design Lab
COURSE CODE: CSE332

Project Name: Compiler Phase –“ Lexical Analyzer ”

SUBMITTED TO:
Mr. Md. Aynul Hasan Nahid (Lecturer)
Daffodil International University
Department of Computer Science & Engineering

SUBMITTED BY:
Name: MD YASIN ARAFAT SHUVO
ID: 201-15-13706
Section: B-55
Daffodil International University
Department of Computer Science & Engineering

DATE OF SUBMISSION: 10-12-2022

Table of Contents

1. INTRODUCTION

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or code that a computer's processors use. The file used for writing a C-language contains what are called the source statements. The programmer then runs the appropriate language compiler, specifying the name of the file that contains the source statements. When executing, the compiler first parses all of the language statements syntactically one after the other and then, in one or more successive stages, builds the output code, making sure that statements that refer to other statements are referred to correctly in the final code. The output of the compilation is called object code or sometimes an object module.

Lexical analysis is the first phase of a compiler. It takes the modified source code from language pre-processors 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. Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, etc.

Symbol table is used by both the analysis and the synthesis parts of a compiler. We have designed a lexical analyzer for the C language using lex. It takes as input a C code and outputs a stream of tokens. The tokens displayed as part of the output include keywords, identifiers, signed/unsigned integer/floating point constants, operators, special characters, headers, data-type specifiers, array, single-line comment, multi-line comment, pre-processor directive, pre-defined functions (printf and scanf), user-defined functions and the main function. The token, the type of token and the line number of the token in the C code are being displayed. The line number is displayed so that it is easier to debug the code for errors. Errors in single-line comments, multi-line comments are displayed along with line numbers. The output also contains the symbol table which contains tokens and their type. The symbol table is generated using the hash organisation.

2. DESIGN STRATEGY

SYMBOL TABLE CREATION – “Lexical Analysis”

Lexical analysis is the process of converting a sequence of characters into a sequence of tokens (strings with an identified "meaning"). A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer). Such a lexer is generally combined with a parser, which together analyze the syntax of programming languages, web pages, and so forth. The script written by us is a computer program called the “lex” program, is the one that generates lexical analyzers ("scanners" or "lexers"). Lex reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language.

3. IMPLEMENTATION DETAILS

SYMBOL TABLE CREATION – "Lexical Analysis"

Lex reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language.

Hash tables and stacks were used to implement the Symbol table at this phase. The structure of the lex program consists of three sections:

{definition section}

%%

{rules section}

%%

{C code section}

The definition section defines macros and imports header files written in C. It is also possible to write any C code here, which will be copied verbatim into the generated source file.

The rules section associates regular expression patterns with C statements. When the lexer sees text in the input matching a given pattern, it will execute the associated C code.

The C code section contains C statements and functions that are copied verbatim to the generated source file. These statements presumably contain code called by the rules in the rules section. In large programs it is more convenient to place this code in a separate file linked in at compile time. The lex program, when compiled using the lex command, generates a file called lex.yy.c, which when executed recognizes the tokens present in the input C program.

4. Instructions on how to build and run program.

Lexical Analysis

```
D:\CD Project>flex lexer.l

D:\CD Project>gcc lex.yy.c -o output

D:\CD Project>.\output.exe
```

5. RESULTS

Lexical Analysis - Lexemes or tokens are generated after the code is passed through the lexer.l file. These tokens are then updated in the symbol table.

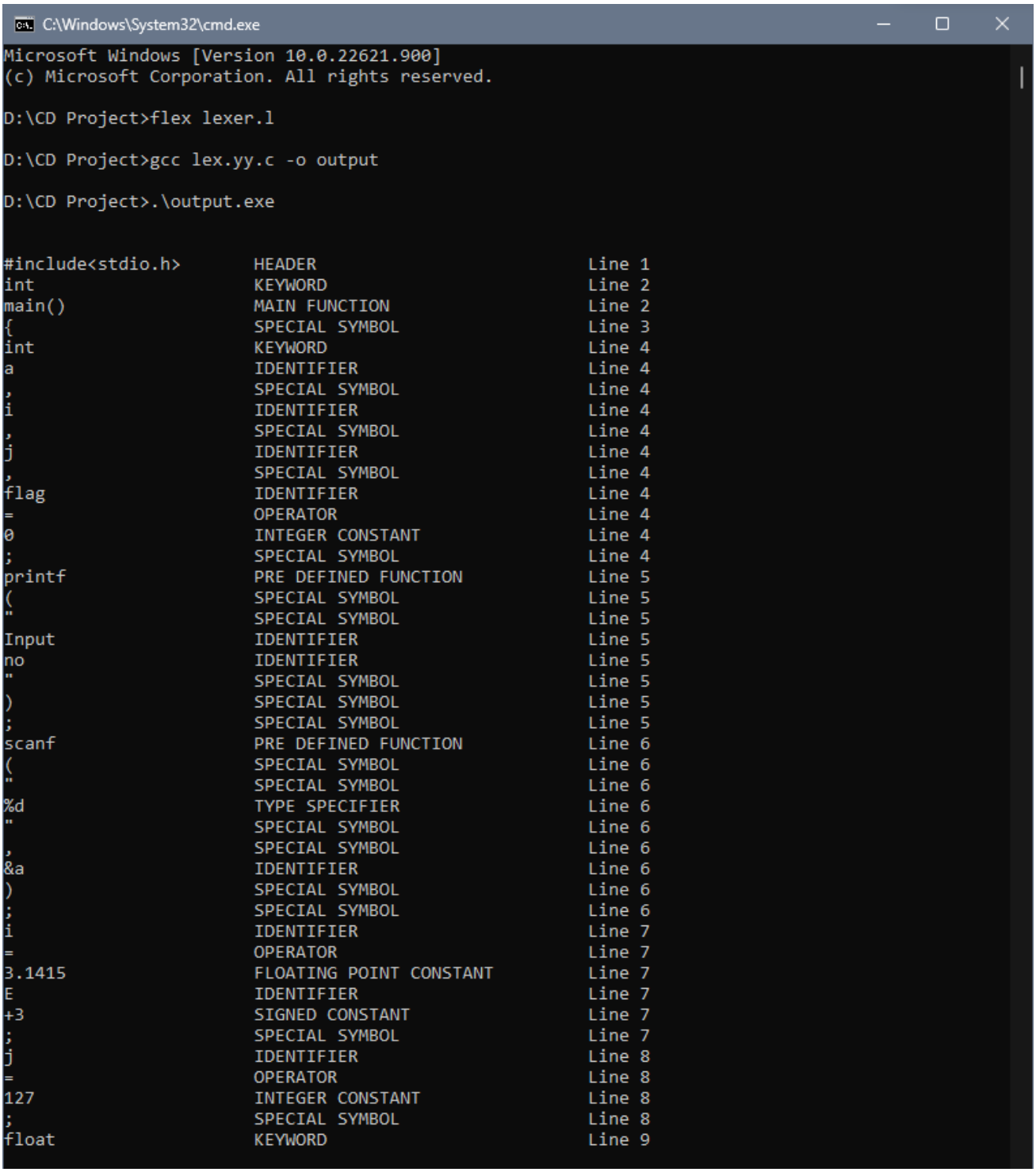
6. SNAPSHOTS

Lexical analysis - Test case: isPrime.c

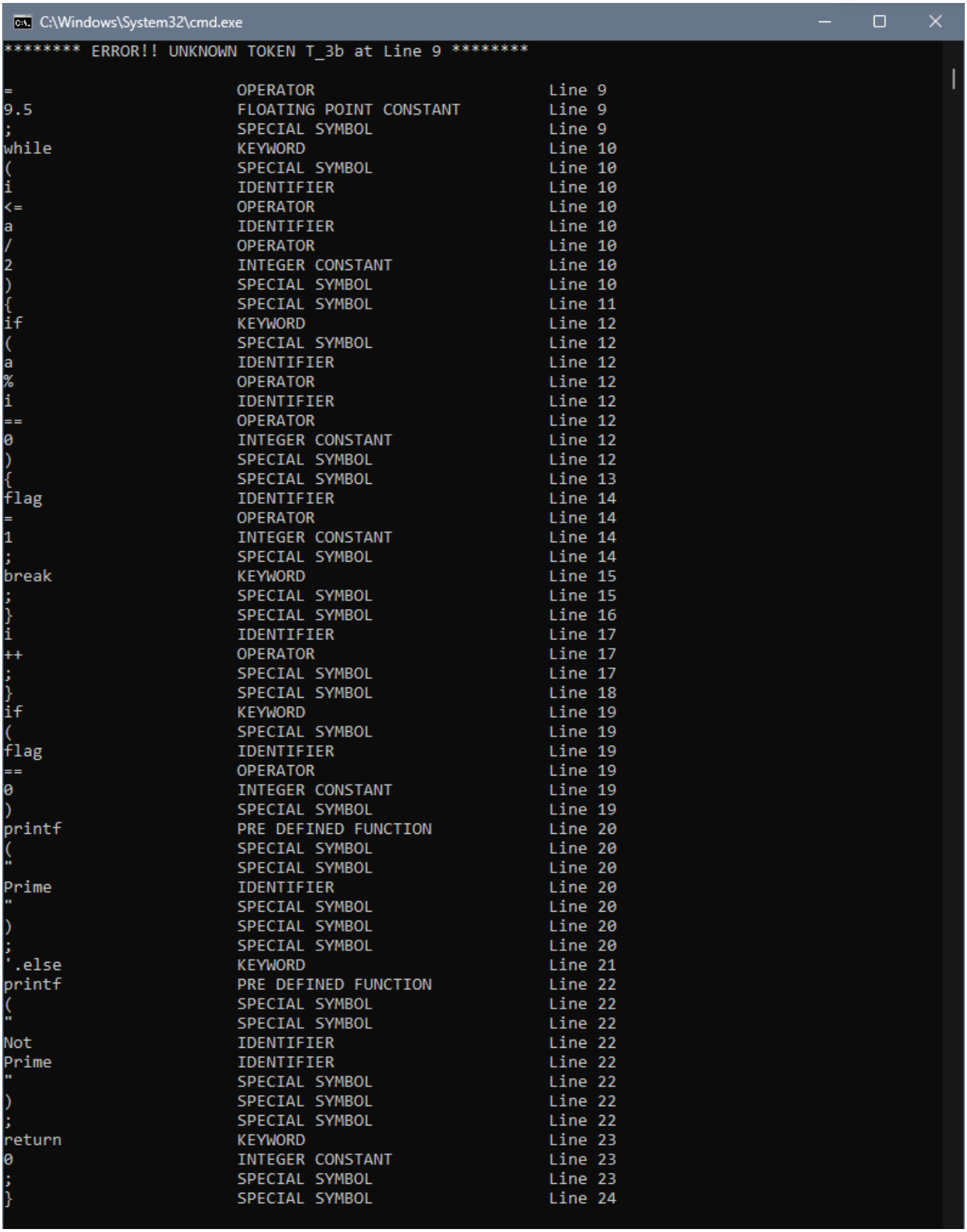
D: > CD Project > C isPrime.c

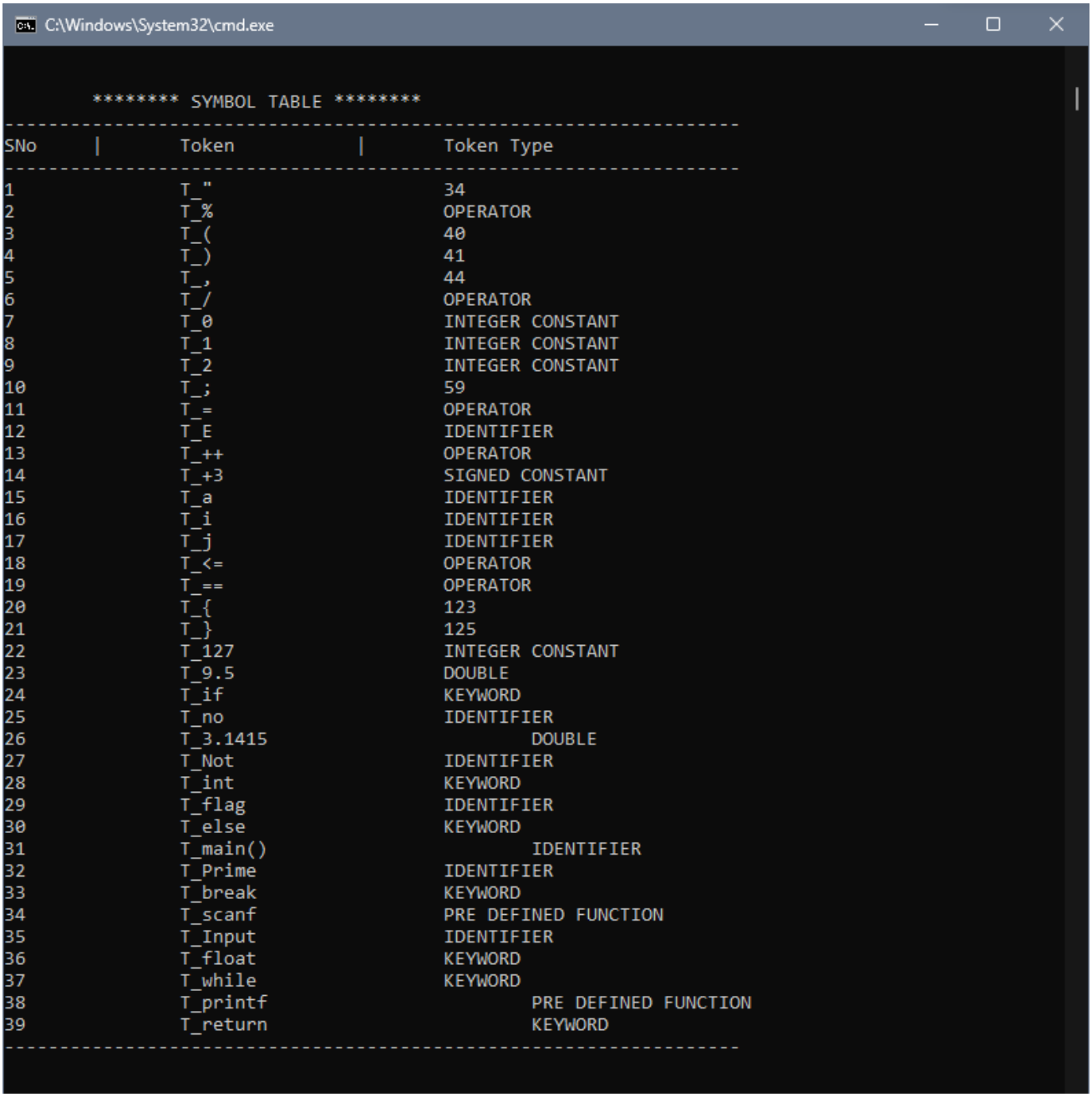
```c
#include<stdio.h>
int main()
{
    int a,i,j,flag=0;
    printf("Input no"); //Input
    scanf("%d",&a);
    i=3.1415E+3;
    j=127;
    float 3b = 9.5;
    while(i <= a/2)
    {
        if(a%i == 0)
        {
            flag=1;
            break;
        }
        i++;
    }
    if(flag==0)
        printf("Prime"); // It's a prime number.
    else
        printf("Not Prime");
    return 0;
}

```

Output:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22621.900]
(c) Microsoft Corporation. All rights reserved.

D:\CD Project>flex lexer.l

D:\CD Project>gcc lex.yy.c -o output

D:\CD Project>.\output.exe


#include<stdio.h>       HEADER                      Line 1
int                     KEYWORD                     Line 2
main()                  MAIN FUNCTION               Line 2
{                       SPECIAL SYMBOL              Line 3
int                     KEYWORD                     Line 4
a                       IDENTIFIER                  Line 4
,                       SPECIAL SYMBOL              Line 4
i                       IDENTIFIER                  Line 4
,                       SPECIAL SYMBOL              Line 4
j                       IDENTIFIER                  Line 4
,                       SPECIAL SYMBOL              Line 4
flag                    IDENTIFIER                  Line 4
=                       OPERATOR                    Line 4
0                       INTEGER CONSTANT            Line 4
;                       SPECIAL SYMBOL              Line 4
printf                  PRE DEFINED FUNCTION        Line 5
(                       SPECIAL SYMBOL              Line 5
"                       SPECIAL SYMBOL              Line 5
Input                   IDENTIFIER                  Line 5
no                      IDENTIFIER                  Line 5
"                       SPECIAL SYMBOL              Line 5
)                       SPECIAL SYMBOL              Line 5
;                       SPECIAL SYMBOL              Line 5
scanf                   PRE DEFINED FUNCTION        Line 6
(                       SPECIAL SYMBOL              Line 6
"                       SPECIAL SYMBOL              Line 6
%d                      TYPE SPECIFIER              Line 6
"                       SPECIAL SYMBOL              Line 6
,                       SPECIAL SYMBOL              Line 6
&a                      IDENTIFIER                  Line 6
)                       SPECIAL SYMBOL              Line 6
;                       SPECIAL SYMBOL              Line 6
i                       IDENTIFIER                  Line 7
=                       OPERATOR                    Line 7
3.1415                  FLOATING POINT CONSTANT     Line 7
E                       IDENTIFIER                  Line 7
+3                      SIGNED CONSTANT             Line 7
;                       SPECIAL SYMBOL              Line 7
j                       IDENTIFIER                  Line 8
=                       OPERATOR                    Line 8
127                     INTEGER CONSTANT            Line 8
;                       SPECIAL SYMBOL              Line 8
float                   KEYWORD                     Line 9
```


C:\Windows\System32\cmd.exe
******** ERROR!! UNKNOWN TOKEN T_3b at Line 9 ********
= OPERATOR Line 9
9.5 FLOATING POINT CONSTANT Line 9
; SPECIAL SYMBOL Line 9
while KEYWORD Line 10
(SPECIAL SYMBOL Line 10
i IDENTIFIER Line 10
<= OPERATOR Line 10
a IDENTIFIER Line 10
/ OPERATOR Line 10
2 INTEGER CONSTANT Line 10
) SPECIAL SYMBOL Line 10
{ SPECIAL SYMBOL Line 11
if KEYWORD Line 12
(SPECIAL SYMBOL Line 12
a IDENTIFIER Line 12
% OPERATOR Line 12
i IDENTIFIER Line 12
== OPERATOR Line 12
0 INTEGER CONSTANT Line 12
) SPECIAL SYMBOL Line 12
{ SPECIAL SYMBOL Line 13
flag IDENTIFIER Line 14
= OPERATOR Line 14
1 INTEGER CONSTANT Line 14
; SPECIAL SYMBOL Line 14
break KEYWORD Line 15
; SPECIAL SYMBOL Line 15
} SPECIAL SYMBOL Line 16
i IDENTIFIER Line 17
++ OPERATOR Line 17
; SPECIAL SYMBOL Line 17
} SPECIAL SYMBOL Line 18
if KEYWORD Line 19
(SPECIAL SYMBOL Line 19
flag IDENTIFIER Line 19
== OPERATOR Line 19
0 INTEGER CONSTANT Line 19
) SPECIAL SYMBOL Line 19
printf PRE DEFINED FUNCTION Line 20
(SPECIAL SYMBOL Line 20
" SPECIAL SYMBOL Line 20
Prime IDENTIFIER Line 20
" SPECIAL SYMBOL Line 20
) SPECIAL SYMBOL Line 20
; SPECIAL SYMBOL Line 20
'.else KEYWORD Line 21
printf PRE DEFINED FUNCTION Line 22
(SPECIAL SYMBOL Line 22
" SPECIAL SYMBOL Line 22
Not IDENTIFIER Line 22
Prime IDENTIFIER Line 22
" SPECIAL SYMBOL Line 22
) SPECIAL SYMBOL Line 22
; SPECIAL SYMBOL Line 22
return KEYWORD Line 23
0 INTEGER CONSTANT Line 23
; SPECIAL SYMBOL Line 23
} SPECIAL SYMBOL Line 24


C:\Windows\System32\cmd.exe
******** SYMBOL TABLE ********
SNo | Token | Token Type
1 T_" 34
2 T_% OPERATOR
3 T_(40
4 T_) 41
5 T_, 44
6 T_/ OPERATOR
7 T_0 INTEGER CONSTANT
8 T_1 INTEGER CONSTANT
9 T_2 INTEGER CONSTANT
10 T_; 59
11 T_= OPERATOR
12 T_E IDENTIFIER
13 T_++ OPERATOR
14 T_+3 SIGNED CONSTANT
15 T_a IDENTIFIER
16 T_i IDENTIFIER
17 T_j IDENTIFIER
18 T_<= OPERATOR
19 T_== OPERATOR
20 T_{ 123
21 T_} 125
22 T_127 INTEGER CONSTANT
23 T_9.5 DOUBLE
24 T_if KEYWORD
25 T_no IDENTIFIER
26 T_3.1415 DOUBLE
27 T_Not IDENTIFIER
28 T_int KEYWORD
29 T_flag IDENTIFIER
30 T_else KEYWORD
31 T_main() IDENTIFIER
32 T_Prime IDENTIFIER
33 T_break KEYWORD
34 T_scanf PRE DEFINED FUNCTION
35 T_Input IDENTIFIER
36 T_float KEYWORD
37 T_while KEYWORD
38 T_printf PRE DEFINED FUNCTION
39 T_return KEYWORD

7. CONCLUSION

The biggest drawback of using Lexical analyzer is that it needs additional runtime overhead is required to generate the lexer tables and construct the tokens. It eases the process of lexical analysis and the syntax analysis by eliminating unwanted tokens

8. REFERENCES

1. https://www.guru99.com/compiler-design-lexical-analysis.html
2. http://web.cs.wpi.edu/~kal/courses/compilers/
3. https://www.jigsawacademy.com/blogs/business-analytics/lexical-analysis/