University of Central Florida

**Department Computer Science**

**COP3402: Systems Software**

**Summer 2022**

**Homework #2 (Lexical Analyzer)**

This is a team project (same team as in project one)

**Goal:**

In this assignment you must implement a lexical analyzer for the programming language PL/0. Your program must be capable to read in a source program written in PL/0, identify some errors, and produce, as output, lexeme table and token list for the program. ***For an example of input and output refer to Appendix A****.*  You may have seen the grammar for the programming language PL/0 using the Extended Backus-Naur Form (EBNF), however, input programs will not necessarily be written following this grammar. An error free input program will be written using tokens from PL/0, but it may not be a valid program. Your program should not assume or try to check for grammatical validity; it should only look for lexical validity. Thus, we do not include a full grammar

**Limited Grammar:**

number ::= digit {digit}**.**

ident ::= letter {letter | digit}**.**

digit ;;= "**0**" | "**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**“**.**

letter ::= "**a**" | "**b**" | … | "**y**" | "**z**" | "**A**" | "**B**" | ... | "**Y**" | "**Z**"**.**

**Example of a program written in PL/0:**

**var** x, w;

**begin**

**read** w;

x:= 4;

**if** w > x **then**

w:= w + 1

**else**

w:= x;

**write** w;

**end.**

**Lexical Conventions for PL/0:**

***A numerical value is assigned to each token (internal representation) as follows:***

elsesym = 1, identsym = 2, numbersym = 3, plussym = 4, minussym = 5, multsym = 6, slashsym = 7, modsym = 8, eqlsym = 9, neqsym = 10, lessym = 11, leqsym = 12,

gtrsym = 13, geqsym = 14, lparentsym = 15, rparentsym = 16, commasym = 17, semicolonsym = 18, periodsym = 19, becomessym = 20, beginsym = 21, endsym = 22, ifsym = 23, thensym = 24, whilesym = 25, dosym = 26, callsym = 27, constsym = 28, varsym = 29, procsym = 30, writesym = 31, readsym = 32.

***Reserved Words:*** const, var, procedure, call, begin, end, if, then, else, while, do, read, write.

***Special Symbols:*** ‘**+**’, ‘**-**‘, ‘\*’, ‘**/**’, ‘%’, ‘**(**‘, ‘**)**’, ‘:**=**’, ’**,**’ , ‘**.’,**  ‘**;**’ ,

’=’ , ‘ **<**’, ‘>’, ‘<=‘, ‘>=‘, ‘!=‘.

***Identifiers:*** identsym = letter (letter | digit)\*

***Numbers:*** numbersym = (digit)**+**

***Invisible Characters:*** tab, white spaces, newline

***Comments denoted by:*** /\* . . . \*/

*Refer to* ***Appendix B*** *for a declaration of the token symbols that may be useful.*

**Constraints:**

***Input:***

1. Identifiers can be a maximum of 11 characters in length.
2. Numbers can be a maximum of 5 digits in length.
3. Comments should be ignored and not tokenized.
4. Invisible Characters should be ignored and not tokenized.
5. Comments should always have a closing symbol (\*/).

**Important Note: Input files may NOT be grammatically valid PL/0 code.**

***Output:***

1. In your output's Lexeme List, identifiers must show the token number and the identifier name separated by a space.
2. In your output's Lexeme List, numbers must show the token number and the value separated by a space. The value must be transformed from the char representation.
3. The token representation of the Lexeme List will be used in the Parser (Project 3). So, PLAN FOR IT!

**Detect the Following Lexical Errors:**

1. Variable identifier does not start with letter.
2. Number length exceeds 5 digits.
3. Identifier length exceeds 11 characters.
4. Invalid symbol.
5. Neverending comment.

Hint: You could create a transition diagram (DFS) to recognize each lexeme on the source program and once accepted generate the token, otherwise emit an error message.

**Submission Instructions**:

***Submit to Webcourse:***

1. Source code. lex.c
2. Instructions to use the program in a readme text file.
3. This is a team assignment (the same team members who worked together in HW1).
4. Only one submission per team.
5. The name of all team members must be written at the beginning of the program.
6. Include comments in your program..
7. Same policy on late submission as in HW1. If there is an extension, no late submissions accepted

**Appendix A:**

***If the input is:***

var x, y;

begin

y := 3;

x := y + 56;

end.

***The output will be:***

Lexeme Table:

lexeme token type

var 29

x 2

, 17

y 2

; 18

begin 21

y 2

:= 20

3 3

; 18

x 2

:= 20

y 2

+ 4

56 3

; 18

end 22

. 19

Token List:

29 2 x 17 2 y 18 21 2 y 20 3 3 18 2 x 20 2 y 4 3 56 18 22 19

**Appendix B:**

***Declaration of Token Types:***

typedef enum {

elsesym = 1, identsym, numbersym, plussym, minussym,

multsym, slashsym, modsym, eqlsym, neqsym, lessym, leqsym,

gtrsym, geqsym, lparentsym, rparentsym, commasym, semicolonsym,

periodsym, becomessym, beginsym, endsym, ifsym, thensym,

whilesym, dosym, callsym, constsym, varsym, procsym, writesym,

readsym } token\_type;

***Token Types:***

elsesym 1 “else”

identsym 2 identifier (see grammar definition)

numbersym 3 number (see grammar definition)

plussym 4 +

minussym 5 -

multsym 6 \*

slashsym 7 /

modsym 8 %

eqlsym 9 =

neqsym 10 !=

lessym 11 <

leqsym 12 <=

gtrsym 13 >

geqsym 14 >=

lparentsym 15 (

rparentsym 16 )

commasym 17 ,

semicolonsym 18 ;

periodsym 19 .

becomessym 20 :=

beginsym 21 “begin”

endsym 22 “end”

ifsym 23 “if”

thensym 24 “then”

whilesym 25 “while”

dosym 26 “do”

callsym 27 “call”

constsym 28 “const”

varsym 29 “var”

procsym 30 “procedure”

writesym 31 “write”

readsym 32 “read”

**Appendix C:**

***Example of a PL/0 program:***

**const** m := 7, n := 85;

**var** i,x,y,z,q,r;

**procedure** mult;

**var** a, b;

**begin**

a := x; b := y; z := 0;

**while** b > 0 **do**

**begin**

**if** x % 2 != 0 **then** z := z+a;

a := 2\*a;

b := b/2;

**end**

**end**;

**begin**

x := m;

y := n;

**call** mult;

**end**.