COVER PAGE

CS323 Programming Assignments

Name [1.Donghao Feng], (MW 1:00 pm - 2:15 pm)

[2. Yong Kim], (MW 5:00 pm - 6:45 pm)

[3. Haojie Pan], (Tue Thurs 4:00 - 5:15 pm)

Assignment Number [1]

Due Dates: Softcopy [2/26/2019], Hardcopy [ ]

Turn-In Dates: Softcopy [2/26/2019], Hardcopy [ ]

Executable FileName [lexer]

Lab Room [Computer Lab 202]

Operating System [Tuffix]

To be filled out by the Instructor:

GRADE:

COMMENTS:

CS323 Documentation

1. Problem Statement

To use FSM to write a lexical analyzer with procedure(Function) - lexer(), that returns a token when needed. The lexer() will return a record, one field for the token and another field the actual “value” of the token (lexeme).

The program will work as follow:

While not finished do
    Call the lexer for a token
    Print the token and lexeme
Endwhile

A sample test case

TOKENS		Lexemes
KEYWORD	=	int
IDENTIFIER	=	num1
SEPARATOR	=	,
IDENTIFIER	=	num2
SEPARATOR	=	,
IDENTIFIER	=	large$
KEYWORD	=	if
SEPARATOR	=	(
IDENTIFIER	=	num1
OPERATOR	=	>
IDENTIFIER	=	num2
SEPARATOR	=	)
SEPARATOR	=	{
IDENTIFIER	=	large
OPERATOR	=	=
IDENTIFIER	=	num1$
SEPARATOR	=	;

```
SEPARATOR    =    }
KEYWORD      =    else
SEPARATOR    =    {
IDENTIFIER   =    large
OPERATOR     =    =
IDENTIFIER   =    num2$
SEPARATOR    =    ;
SEPARATOR    =    }
```

2. How to use the program

Usage :

1. In terminal, move to the directory of executable file and input file
2. In command line “./lexer InputfileName.txt”
3. In the same folder, the program will make a new “output.txt” file

3. Design of the program

Library used implementing lexer and purpose:

<iostream> : grabbing every characters from the file and printing out the result

<fstream> : reading and opening a file

<map> : characters categorizing (to be modified)

Now, a container for every lexeme, but it will be changed two vectors, because later in this course, the sequence might have to be important.

<queue> : temporary container of current character

<array> : temporary container of keywords (to be modified)

<iomanip> : output file

Main role of each parts :

Read and write file(printing token) : Donghao Feng

Data Structures(storing token) : Haojie Pan

Flow Design : Yong Kim

Table Design : Donghao Feng / Haojie Pan / Yong Kim

Program Test : Donghao Feng

Currently, our state table will be like following:

```
// input l  d  ! sp ( ) [ ] { } ' , . : ; * +  - = / > < % \n Unknown

int fsa_table[ROW_SIZE][COL_SIZE] ={
    {1, 2, 9,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32},    // 0 start
    {1, 1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5},    // 1 identifier, keep going
    {31,2, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 20,7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7},    // 2 integer, keep going
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 3 regular end, will be used later
    {4, 4,10, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},    // 4 in comment, keep going
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 5 end string
    {8, 6, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8},    // 6 real, keep going
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 7 end integer
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 8 end real
    {4, 4,10, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},    // 9 start ! comment
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 10 end ! comment
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 11 found sp
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 12 found '('
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 13 found ')'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 14 found '['
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 15 found ']'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 16 found '{'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 17 found '}'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 18 found '\''
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 19 found ','
    {0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 20 found '.'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 21 found ':'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 22 found ';'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 23 found '*'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 24 found '+'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 25 found '-'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 26 found '='
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 27 found '/'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 28 found '>'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 29 found '<'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 30 found '%'
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},    // 31 found '\n'
    {32,32,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}     // 32 unknow TODO
};
```

4. Any Limitation

None.

5. Any shortcomings

None.