**Group Project (Part 1 – Lexical Analyzer)**

|  |  |
| --- | --- |
| **CLO: 3** | **SO: 2** |

* Identify lexemes and tokens from the source code.
* Truncate white spaces and comments from the source code.
* Differentiate between identifiers and reserved words.
* Generate an error after encountering some illegal character.

**Integrated FA for Identifiers, White Spaces, Integers, Relational Operators and Assignment Operator:**

return(Comma)

6

Error( )

5

Space, Tab, Newline

100

,

return(Colon)

return(Semi\_Colon)

;

7

:

0

?

return(Right\_Curly)

return(Left\_Curly)

return(Question\_Mark)

{

8

8

L , \_

D

}

else

9

L , D , \_

3

4

1

10

10

D

else

2

ungetc()

return (Int)

ungetc()

return (Id)

**Program for the above FA:**

//Add commands here to import all related packages

void tokenizer();

void error();

char lookahead;

//Write commands here to open “input.txt” file for reading and “output.txt”   
 //file for writing

// main function to start execution

void main()

{

// call to takenizer to generate tokens

tokenizer();

printf("Tokens have been identified successfully....!!!");

getch();

}

void tokenizer()

{

// Write statement(s) here to store all reserved words of C/Java into an array

// Create an array named “lexeme” to store lexemes

char lexeme[30];

int state=0; // integer variable representing states

char specifiers[]={'n','a','t','r', 'b',39,92};

// specifiers 39 represents ' and 92 represents \

int i=0,j=0,k=0, flag = 0;

if file “input.txt” does not exist, display some error message and quit

else

{

//Read one character from the input file and store it in lookahead   
 //variable

while (pointer does not reach End of File)

{

switch(state)

{

case 0:

Write the code for all the outgoing arrows from state 0 after constructing FA for Identifiers, Arithmetic Operators, Arithmetic Assignment Operators, Relational operators, Logical operators, Increment/Decrement Operators, Assignment operators, Integers, floating point numbers, character literals, string literals, Single Line, Multiline Comments, White Spaces and Delimeters/Punctuation Marks

…. ………………………………………………………………….

else

{

error();

state=0;

}

break;

case 1:

//Read the next character from the input file

//Write code for all outgoing arrows from this state

break;

case 2:

state=0;

//The following code will unget the last character   
 //read from the input file

lexeme[i]='\0'; //Storing null character at the end

for(j=0;j<32;j++)

{

if(strcmp(lexeme,reserveWords[j])==0)

{

flag = 1;

break;

}

}

//The following code is used to write the lexeme and its token   
 //in the output file

if( flag) //If it is reserved word

{

//Write statements here to print the lexeme   
 //and the reserved word

flag=0;

}

else //If it is identifier

{

//Write statements here to print the lexeme   
 //and its corresponding token “ID” }

i=0;

break;

case 3:

//Write statement to read one character from the   
 //input file and store it in lookahead variable

//Write corresponding code for digit here

break;

case 4:

state=0;

lexeme[i]='\0';

//Write statements here to print the lexeme   
 //and its corresponding token “INTEGER” i=0;

break;

Write code here for all the states which are present in your final integrated FA which recognizes

1. Arithmetic Operators (+ , -, \*, /, %)
2. Arithmetic Assignment Operators (+=, - =, \*=, /=, %=)
3. Relational Operators (<, <=, >, >=, ==, !=)
4. Logical Operators (&&, ||, !)
5. Increment/Decrement Operators (++, - -)
6. Single Line and Multi Line Comments
7. Character and String literals
8. Integer and float literals
9. Punctuation Marks ( [ | ] | ( | ) | ; | : | , ) etc

}

}

}

}

void error()

{

//Write statement here to display the error message   
 //“UNRECOGNIZED\_TOKEN”

//Write statement to read one character from the input file and store it in   
 //lookahead variable

}

**Assigned on: Monday, September 15, 2024**

**Due Date: Thursday, September 26, 2024 till 11:59 PM (Mid Night)**

**Important Note: Part 1 of Final Project (Lexical Analyzer):**

It is a **Group Project**. Its due date is **Thursday September 26, 2024 till 11:59 PM**. You can also upload the solution by **Friday September 27, 2024 till 11:59 PM with 25% deduction of marks or by Saturday September 28, 2023 till 11:59 PM with 50% deduction of marks**. **After this date, no submission will be accepted.**

**Hint:**

Construct individual FAs for

1. Identifiers
2. White Spaces
3. Arithmetic Operators (+ , -, \*, /, %)
4. Arithmetic Assignment Operators (+=, - =, \*=, /=, %=)
5. Relational Operators (<, <=, >, >=, ==, !=)
6. Logical Operators (&&, ||, !)
7. Increment/Decrement Operators (++, - -)
8. Assignment Operator
9. Single Line and Multi Line Comments
10. Integers and floating point/real numbers
11. Character and String literals
12. Punctuation Marks ( ; | : | , | ‘ | “ | { | } | [ | ] | ( | ) etc. )

Then combine these FAs into one integrated FA such that no state should have more than one outgoing edge with the same label. Finally write a program for implementing the integrated FA for lexical analyzer phase of compiler.

**Sample input** and **output** for this program is as follows:

**Input:**

void main(void)

{

int a=10; int b=20;

b = a++;

}

**Output:**

**Lexemes** **Tokens**

void void

main main

( Left\_Paren

void void

) Right\_Paren

{ Left\_Curly

int int

a id

= Assign\_Op

10 Int\_Literal

; Semi\_Colon

int int

b Id

= Assign\_Op

20 Int\_Literal

; Semi\_Colon

b id

= Assign\_Op

a id

++ Inc\_Op

; Semi\_Colon

} Right\_Curly

void main(void){int a=10;int b=20;b=a++;}