The Lexical Analyzer

CS236 Project One

Introduction to Lexical Analyzers

A lexical analyzer groups characters in an input stream into tokens. A token is a sequence of one or more characters that form a single element of a language, like a symbol, string, keyword, identifier, or white space.

Project Description

Write a LexicalAnalyzer to convert a sequence of characters in an input stream into a sequence of tokens. The sequence of tokens will be used in project 2. The Java API is full of methods that can be used to simplify your code and save time for testing them. For this project in particular, the Character class will prove to be most useful. The following lists and describes the different types of tokens.

Varieties	Description	Example
<String>	Any sequence of characters enclosed in single quotes. (Two single quotes imply an apostrophe in the string. Please note that this will only occur within a string.) Spaces, '\t', and '\n' should be appended to the string value but not '\r' carriage return. Always count the '\n'. A string token's line number is the line where the string started.	**`'quoted string'`** **`'this isn''t two strings'`** **`'' (empty string)`** **`'don''t forget about multi-line strings'`**
<Keyword>	One of the following four character sequences: **`Schemes`**, **`Facts`**, **`Rules`**, **`Queries`**. These keywords are case sensitive!	Example: **`Schemesa`** is a single identifier and **not** a keyword and an identifier.
<Identifier>	An identifier is a letter followed by a sequence of zero or more letters or numbers. No underscores.	Legal identifiers: **`Identifier1`** **`Person`** Invalid identifiers: **`1stPerson`** **`Person_Name`**
<Symbol>	One of the following character sequences: **`: , < > = ( * ?`** **`:- . <= >= != ) +`**	**`<=('a','b')`** **`( + ()`** **`::- ???`**
White Space	Always ignore white space. Whether or not a character, c, is whitespace is defined by invoking the method: Character.isWhitespace(c). Remember, the character '\n' is whitespace but also marks the end of a line.	
<Undefined>	Any character not tokenized as a string, keyword, identifier, symbol, or white space. Any non-terminating string or non-terminating comment is undefined. In both of the latter two cases we reached EOF before finding the end of string or end of comment.	**`$&^`** (Three individual tokens.) **`'any string that doesn''t end`**
<Comment>	A line comment starts with # and ends at newline. A block comment starts at #| and ends with |#. The comment's line number is the line where the comment started.	**`#this is a comment`** **`#|this is a multiline comment|#`**
<EOF>	End of input file.	

Output one token per line with the TokenType, quoted value, and line number, all enclosed within parentheses. When finished, output "Total Tokens = x" (where x is the number of tokens).

For example, input: **`Facts:`** would parse to:

```
(FACTS,"Facts",1)
(COLON,":",1)
(EOF,"",2)
Total Tokens = 3
```

The following are the possible TokenTypes:

Type	Token	TokenType
<EOF>	eof marker	**EOF**
<Identifier>	Identifier	**ID**
<Keyword>	Schemes	**SCHEMES**
<Keyword>	Facts	**FACTS**
<Keyword>	Rules	**RULES**
<Keyword>	Queries	**QUERIES**
<String>	quoted string	**STRING**
<Symbol>	:	**COLON**
<Symbol>	:-	**COLON_DASH**
<Symbol>	,	**COMMA**
<Symbol>	(	**LEFT_PAREN**
<Symbol>	.	**PERIOD**

Type	Token	TokenType
<Symbol>	?	**Q_MARK**
<Symbol>	)	**RIGHT_PAREN**
<Symbol>	=	**EQ**
<Symbol>	!=	**NE**
<Symbol>	>	**GT**
<Symbol>	>=	**GE**
<Symbol>	<	**LT**
<Symbol>	<=	**LE**
<Symbol>	*	**MULTIPLY**
<Symbol>	+	**ADD**
<Undefined>	characters	**UNDEFINED**
<Comment>	#|block|# # to \n	**COMMENT**

Examples

These are not sufficient to completely test your program. You must be able to process any file containing any sequence of characters. Your program must have terminal/console output formatted exactly like the example outputs below:

Example 10 Input: ex10.txt

```
Queries:
     IsInRoomAtDH('Snoopy',R,'M',H)
#SchemesFactsRules
.
#|comment >=
wow|#
```

Example 10 Output: out10.txt

```
(QUERIES,"Queries",1)
(COLON,":",1)
(ID,"IsInRoomAtDH",2)
(LEFT_PAREN,"(",2)
(STRING,"Snoopy",2)
(COMMA,",",2)
(ID,"R",2)
(COMMA,",",2)
(STRING,"M",2)
(COMMA,",",2)
(ID,"H",2)
(RIGHT_PAREN,")",2)
(COMMENT,"#SchemesFactsRules",3)
(PERIOD,".",4)
(COMMENT,"#|comment >=
wow|#",5)
(EOF,"",7)
Total Tokens = 16
```

Example 11 Input: ex11.txt	Example 11 Output: out11.txt

Queries:
  IsInRoomAtDH('Snoopy',R,'M'H)?
Rules Facts Schemes
&@Queries
:
  'hello
I am' $ A 'this has a
Return
  The end''s near
	(QUERIES,"Queries",1)
	(COLON,":",1)
	(ID,"IsInRoomAtDH",2)
	(LEFT_PAREN,"(",2)
	(STRING,"Snoopy",2)
	(COMMA,",",2)
	(ID,"R",2)
	(COMMA,",",2)
	(STRING,"M",2)
	(ID,"H",2)
	(RIGHT_PAREN,")",2)
	(Q_MARK,"?",2)
	(RULES,"Rules",3)
	(FACTS,"Facts",3)
	(SCHEMES,"Schemes",3)
	(UNDEFINED,"&",4)
	(UNDEFINED,"@",4)
	(QUERIES,"Queries",4)
	(COLON,":",5)
	(STRING,"hello
	I am",6)
	(UNDEFINED,"$",7)
	(ID,"A",7)
	(UNDEFINED,"'this has a
	Return
	  The end''s near
	",7)
	(EOF,"",10)
	Total Tokens = 24

Requirements (Get the Design Right)

Ninety percent (90%) of your grade will be based on correctness. The percent of test cases you pass measures correctness. Ten percent (10%) of your grade will be based on getting the design right.

Your program must have the following six classes:

1. A class containing a single *main* method. This class contains no attributes.
2. A class that contains a sequence of characters and allows the user to access those characters.
3. A class, often called **Lex**, that contains a sequence of tokens. There is a method in this class that converts sequences of characters into sequences of tokens. This method contains the implementation of the finite state machine for this project.
4. A class representing a token (see example below). Each token should have a token type, value, and line number.
5. An enumerated type defining token types (see example below).
6. An enumerated type defining states.

The design of your conversion algorithm must be based on a finite state machine and implemented as a single method in the **Lex** class. This method may be very long. Your implementation of the finite state machine must use a switch statement or set of if statements where each case in the switch statements or each if statement represents a state in the finite state machine.

Requirements for Passing It Off

Build your project using the cs236Bundle. Be sure to read about it. Your source code goes in the *src/project1* directory. All classes must be in the package *project1*. The class containing the main method must be called **Project1**. Its only task is to construct your lexical analyzer object and generate the output. The input for your program comes from an input file. The name of the file will be the first parameter on the command line. An example invocation might be: java Project1 testCase1. Other requirements for all cs236 projects can be found on the policies page.

Example Token Class Definition

Token.java

```
public class Token {
//Domain
    private String value;
    private TokenType type;
    private int lineNumber;

//Constructors
    public Token(){}
    public Token(TokenType t, String val, int lineNum){}

//Access Methods
    public String getValue(){}
    public TokenType getType(){}
    public int getLineNumber(){}

//Update Methods
    public void setValue(String val){}
    public void setType(TokenType t){}
    public void setLineNumber(int lineNum){}

//Print Method
    public String toString(){}
};
```

Notice the “toString()” method in the Token class. In this case this is used to generate most of the required output. For debug purposes it is suggested that you include “toString()” methods in all other non-enum classes.

Example TokenType Class Definition

TokenType.java

```
public enum TokenType {
    COLON, COLON_DASH, COMMA, LEFT_PAREN, PERIOD, Q_MARK, RIGHT_PAREN, EQ, NE,
    GT, GE, LT, LE, MULTIPLY, ADD, STRING, SCHEMES, FACTS, RULES, QUERIES, ID,
    EOF, UNDEFINED, COMMENT;
};
```