Before Starting

An objective CS 236 is to help you write complex programs by using mathematical concepts as the basis for solving real world problems through computation. Mathematical thinking leads to programs that are clear, organized, and verifiably correct. This first projects shows you how discrete math structures form the basis, not only for writing the code, but for the code itself. It thus also provides a clear guide to maintaining and extending the code by any other programmer who also understands the discrete math structures.

You are expected to write code from scratch that clearly emulates the discrete math structures on which the solution is built. You are expected to justify your design and your code as one that is maintainable and extendable by other programmers who understand and are conversant with discrete mathematical structures. You should also learn that for complex programs, if you *hack the code* or formulate your code off of the discrete math structures, it will take you longer to write, it is less likely to be correct.

Please be aware that the project standards apply to this project and all projects in this course, and in particular for this projects, **you may not use a regular expression library**.

Description

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 (e.g., a symbol, a numerical value, a string literal, or a keyword).

The project is to write a lexical analyzer for a subset of the Datalog language [http://en.wikipedia.org/wiki/Datalog]. The Lexer should output the **UNDEFINED** token if an undefined symbol is found. The input to your Lexer is a text file and the output is a print-out of formatted tokens explained below. Please refer to the project standards for details on how to indicate the program input and where the output should be directed.

A token object is comprised of:

- a string – extracted from the file text
- a number – the line the token started on
- a token type – listed below

The lexical analyzer must use finite state machines to recognize tokens in the input streams. It is recommended that you consider using enumerated types to enumerate the potential states of the machine.

Token Types

You are to generate tokens from a text input stream for a subset of Datalog. The list of all token types that you will be expected to identify is below. You need to define the token types in your code.

Token Type	Description	Examples	
COMMA	The ',' character	,	
PERIOD	The '.' character	.	
Q_MARK	The '?' character	?	
LEFT_PAREN	The '(' character	(	
RIGHT_PAREN	The ')' character	)	
COLON	The ':' character	:	
COLON_DASH	The string ":-"	:-	
MULTIPLY	The ' * ' character	*	
ADD	The ' + ' character	+	
SCHEMES	The string "Schemes"	Schemes	
FACTS	The string "Facts"	Facts	
RULES	The string "Rules"	Rules	
QUERIES	The string "Queries"	Queries	
ID	A letter followed by 0 or more letters or digits and is not a keyword (Schemes, Facts, Rules, Queries)	**valid**	**invalid**
		Identifier1	1stPerson
		Person	Person_Name
			Schemes
STRING	Any sequence of characters enclosed in single quotes. Two single quotes denote an apostrophe within the string. For line-number counts, count all \n's within a string. A string token's line number is the line where the string starts. If EOF is found before the end of the string, it is undefined (see UNDEFINED token below).	'This is a string'	
		' ' (*The empty string*)	
		'this isn''t two strings'	
COMMENT	A *line comment* starts with # and ends at the next newline or EOF.	# This is a comment	
	A *block comment* starts with a #| and ends with a |#. The comment's line number is the line where the comment started. Like STRING, if EOF is found before the end of the block comment, it is UNDEFINED (see UNDEFINED token below).	#| This is a tricky	
		multiline comment |#	
		#| This is an illegal block comment	
		because it ends with EOF	
WHITESPACE	Any character recognized by the **isspace()** function defined in **ctype.h**. *Though we give it a token type here, these tokens will be ignored and not saved.* Be sure to count the line numbers when skipping over white space.		
UNDEFINED	Any character not tokenized as a string, keyword, identifier, symbol, or white space is undefined. Additionally, any non-terminating string or non-terminating block comment is undefined. In both of the latter cases we reach EOF before finding the end of the string or the end of the block comment.	$&^ (*Three undefined tokens*)	
		'any string that does not end	
EOF	End of input file		

Output Format

For this and all subsequent labs, you MUST match the formatting requirements exactly. Automated testing systems are used to check your code (similar to those used by professional software design companies) and it is desired that you experience writing code that matches a given protocol.

The expected output is the representation of a token, printed one per line, followed by a single line that states how many tokens there were. You should NOT print any information for a WHITESPACE token. Format tokens as follows:

```
(Token_Type,"Value",Line_Number)
```

Notice there are no spaces on either side of the commas separating the three token's elements. The token type should appear in this list, spelled exactly as specified (all capital letters). The text is the value of the token surrounded by double quotes. You should have only one token per line, and the last line should look like:

```
Total Tokens = ###
```

where ### is replaced by the number of tokens found (excluding whitespace). ALL OUTPUT IS CASE SENSITIVE. See the example below:

Example 1

For input (the line numbers are **not part** of the input-they are to help clarify new lines)

```
01 .
02 ,
03 'a string'
04 #A comment
05 Schemes
06 FactsRules
07 ::-
08
```

The Lexer should output

```
 (PERIOD,".",1)
 (COMMA,",",2)
 (STRING,"'a string'",3)
 (COMMENT,"#A comment",4)
 (SCHEMES,"Schemes",5)
 (ID,"FactsRules",6)
 (COLON,":",7)
 (COLON_DASH,":-",7)
 (EOF,"",8)
 Total Tokens = 9
```

Example 2

Input

```
01 Queries:
02 IsInRoomAtDH('Snoopy',R,'M',H)
03 #SchemesFactsRules<
04 .
05 #|comment >=
06 wow|#
07
```

Output

```
(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 3

Input

```
01 Queries:
02 IsInRoomAtDH('Snoopy',R,'M'H)?
03 Rules Facts Schemes
04 &@Queries
05 :
06 'hello
07 I am' $ A 'this has a
08 Return
09 The end''s near
10
```

Output

```
(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
```

Submission

Review the project standards for creating a zip archive. **You submission cannot be scored if you fail to create the archive correct.**

Navigate to Learning Suite [http://learningsuite.byu.edu], select 'CS 236', and click on 'Assignments' on the course home page. Click on the link to the relevant project and at the bottom of the description click on 'View/Submit'. Use the resulting upload dialog to upload your zip archive.

Pass-off

Pass-off your project directly to a TA during normal TA hours after your archive is uploaded to learningsuite [http://learningsuite.byu.edu]. TAs help students on a first-come/first-serve basis and are under no obligation to stay later than designated hours so plan accordingly. Please review the syllabus for the pass-off requirements.

cs-236/lexical-analyzer.txt · Last modified: 2015/02/23 17:14 by egm