Concordia University
Department of Computer Science
and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2017

Assignment 1, Lexical Analyzer

Deadline:	Monday January 30, 2017
Evaluation:	10% of final grade
Late submission:	penalty of 50% for each late working day

Design and implement a scanner for a programming language whose lexical specifications are given below. The scanner identifies and outputs tokens (valid words and punctuation) in the source program. Its output is a token that can thereafter be used by the syntactic analyzer to verify that the program is syntactically valid. When called, the lexical analyzer should extract the next token from the source program. The lexical analyzer should be able to output a token even if the input does not form a correct program. The syntax of the language will be specified later in assignment #2. Note that completeness of testing will be a major grading topic. You are responsible for providing appropriate test cases that test for a wide variety of valid and invalid cases.

Atomic lexical elements of the language

$$
\begin{aligned}
id &::= letter\ alphanum* \\
alphanum &::= letter\ |\ digit\ |\ \_ \\
num &::= integer\ |\ float \\
integer &::= nonzero\ digit*\ |\ \mathbf{0} \\
float &::= integer\ fraction \\
fraction &::= .digit*\ nonzero\ |\ \mathbf{.0} \\
letter &::= \mathbf{a..z\ |A..Z} \\
digit &::= \mathbf{0..9} \\
nonzero &::= \mathbf{1..9}
\end{aligned}
$$

Operators, punctuation and reserved words

==	+	(	if
<>	-	)	then
<	*	{	else
>	/	}	for
<=	=	[	class
>=	and	]	int
;	not	/*	float
,	or	*/	get
.		//	put
			return
			program

Note : It is up to you to analyze this lexical definition and figure out if there are ambiguities or other errors. Any changes to this definition, if any, must be justified and should not result in diminishing the expressive power of the language. Also, keep in mind that you have to design the lexical analyzer in a flexible way, so that it can easily be adapted if changes are needed when designing the syntactic analyzer.

Note : The tokens **//** and **/*** followed ***/** by denote comments in the source code.

Work to be done

- Identify the lexical specification you use in the design of the lexical analyzer, as well as any changes that you may have applied to the original lexical specifications.
- Include in your documentation a finite state machine describing the operation of your lexical analyzer.
- Identify all the possible lexical errors that the lexical analyzer might encounter. Identify and explain the error recovery technique that you implement.
- Write a lexical analyzer that recognizes the above mentioned tokens. It should be a function that returns a data structure containing the information about the next token identified in the source program file. The data structure should contain information such as (1) the token type (2) its value (or lexeme) when applicable and (3) its location in the source code. Fully describe this structure in your documentation.
- Give a brief overview of the overall structure of your solution, as well as a brief description of the role of each component of your implementation.
- Include a driver that repeatedly calls the lexical analysis function and prints the token type of each token until the end of the source program is reached. The lexical analyzer should optionally print the token stream to a file (for verification purposes). Another file (also for verification purposes) should contain representative error messages each time an error is encountered in the input program. The lexical analyzer should not stop after encountering an error. Error messages should be clear and should identify the location of the error in the source code.
- Include many test cases that test a wide variety of valid and invalid cases.
- Identify all the tools/libraries/techniques that you have used in your analysis or implementation and justify why you have used these particular ones as opposed to others.

Assignment submission requirements and procedure

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "*programming assignment 1*". The file submitted must be a **.zip** file containing:

• all your code
• a set of input files to be used for testing purpose, as well as a printout of the resulting output of the program for each input file
• a simple document containing the information requested above

You are also responsible to give proper compilation and execution instructions to the marker in a README file. If the marker cannot compile and execute your programs, you might have to have a meeting for a demonstration.

Evaluation criteria and grading scheme

Analysis:		
Lexical specifications as regular expressions (after changes)	ind 2.1	1 pt
Description of changes applied to the original specifications, methods used to apply the changes and to generate the finite state automaton	ind 2.2	2 pts
Finite state automaton representing the implementation	ind 2.2	2 pts
Design/implementation:		
Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution	ind 4.3	4 pts
Correct implementation according to assignment statement	ind 4.4	12 pts
Output of clear error messages (error description and location)	ind 4.4	3 pts
Output of token stream in a separate stream or file	ind 4.4	3 pts
Error recovery	ind 4.4	3 pts
Completeness of test cases	ind 4.4	15 pts
Use of tools:		
Description/justification of tools/libraries/techniques used in the analysis/implementation.	ind 5.2	2 pts
Successful/correct use of tools/libraries/techniques used in the analysis/implementation.	ind 5.1	3 pts
Total		**50 pts**