Concordia University
Department of Computer Science
and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2019

Assignment 1, Lexical Analyzer

Deadline:	Monday January 28, 2019
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 is 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

```
      id ::= letter alphanum*
alphanum ::= letter | digit | _
 integer ::= nonzero digit* | 0
   float ::= integer fraction [e[+|–] integer]
fraction ::= .digit* nonzero | .0
  letter ::= a..z |A..Z
   digit ::= 0..9
 nonzero ::= 1..9
```

Operators, punctuation and reserved words

```
==      +       (       if
<>      -       )       then
<       *       {       else
>       /       }       for
<=      =       [       class
>=      &&      ]       integer
;       !       /*      float
,       ||      */      read
.               //      write
:                       return
::                      main
```

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 submit

Document

- **Lexical specifications**: Identify the lexical specification you used in the design of the lexical analyzer, as well as any changes that you may have applied to the original lexical specifications.
- **Finite state automaton**: Finite state machine describing the operation of your lexical analyzer.
- **Error reporting and recovery**: Identify all the possible lexical errors that the lexical analyzer might encounter. Identify and explain the error recovery technique that you implement.
- **Design**: 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.
- **Use of tools**: 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.

Implementation

- **Lexical analyzer**: 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.
- **Driver**: 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 in the AtoCC format (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.
- **Test cases**: Include many test cases that test a wide variety of valid and invalid cases.

Selected examples

0123	[Invalid number:0123] OR [integer:0][integer:123]
01.23	[Invalid number:01.23] OR [integer:0][float:1.23]
12.340	[Invalid number:12.340] OR [float:12.34][integer:0]
012.340	[Invalid number:012.340] OR [integer:0][float:12.34][integer:0]
12.34e01	[Invalid number12.34e01] OR [12.34e0][integer:1]
12345.6789e-123	[float:12345.6789e-123]
12345	[integer:12345]
abc	[id:abc]
abc1	[id:abc1]
abc_1	[id:abc_1]
abc1_	[id:abc1_]
_abc1	[Invalid identifier:_abc1]
1abc	[Invalid identifier:1abc] OR [integer:1][id:abc]
_1abc	[Invalid identifier:_1abc] OR [Invalid identifier:_][integer:1][id:abc]

Assignment submission requirements and procedure

- Each submitted assignment should contain two components: the source code and a brief report;
- The source code should be separated into modules using a comprehensible coding style and naming rules.
- The report should contain instructions about how to run your code. If any extra environment needs to be setup for your code please give full instruction in your report.
- The report should list all your test cases, including the expected result for each test case.
- the assignment should be submitted through EAS in a file named: "A#_student id" (e.g. A1_1234567, for Assignment #1, student ID 1234567) and the report must in a PDF format.
- If the marker cannot run your code, a demo will be required and you will be notified through email.
- You can use any language you want in the project and assignments but the only full supported language during the lab is Java.

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

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 file in the AtoCC format	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 of tools/libraries/techniques used in the analysis/implementation. Description of other tools that might have been used. Justification of why the chosen tools were selected.	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**