# Concordia University
# Department of Computer Science and Software Engineering

## Compiler Design (COMP 442/6421)
## Winter 2021

## Assignment 1, Lexical Analyzer

| | |
|---|---|
| **Deadline:** | Monday February 12, 2021 |
| **Evaluation:** | 10% of final grade |
| **Late submission:** | penalty of 50% for each late working day |

This assignment is about the design and implementation of 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 syntax of the language will be specified later in assignment #2. The assignment includes two grading source files used by the marker. These files should be used as-is and not be altered in any way. 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 in addition to what is in the grading source files provided.

### 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** |
| ***string*** | ::= | **“ character* “** |
| *character* | ::= | *alphanum* \| *space* |

### Operators, punctuation and reserved words

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| == | + | \| | ( | ; | if | public | read |
| <> | - | & | ) | , | then | private | write |
| < | * | ! | { | . | else | func | return |
| > | / | ? | } | : | integer | var | main |
| <= | = | | [ | :: | float | class | inherits |
| >= | | | ] | | string | while | break |
| | | | | | void | | continue |

### Comments

- Block comments start with `/*` and end with `*/` and may span over multiple lines.
- Inline comments start with `//` and end with the end of the line they appear in.

## Work to submit

### Document

You must provide a short document that includes the following sections:

Section 1. **Lexical specifications**: Identify the lexical specification, expressed as regular expressions, that you used in the design of the lexical analyzer. Highlight any changes that you may have applied to the original lexical specifications and justify your changes.

Section 2. **Finite state automaton**: Finite state machine diagram describing the operation of your lexical analyzer.

Section 3. **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.

Section 4. **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 token data structure containing the information about the next token identified in the source program file. The token data structure should contain information such as (1) the token **`type`** (2) the **`lexeme`** of the token and (3) the **`location`** of the token in the source code. When lexical errors are found, an error token should be produced, whose **`type`** is the specific error type, its **`lexeme`** is the erroneous character stream, and its **`location`** is where the error was found in the source code. When a token is identified from a file named, for example, **`originalfilename`**, it should be printed out into a file named ***`originalfilename`*****`.outlextokens`** that lists the token stream that corresponds to the original program. When lexical errors are found, error messages should be printed out in a file named ***`originalfilename`*****`.outlexerrors`**.
- **Test cases**: Include many test cases that test a wide variety of valid and invalid cases. Test cases are included in files that are read by the implementation. Your test files must include the test files provided with the assignment, which should not have been altered. Samples of expected output files are also given with the assignment as guidance.
- **Driver**: Include a driver that extracts the tokens from all your test files. For each test file, the corresponding **`outlexerrors`** and **`outlextokens`** files should be generated.

## 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 number:12.34e01] 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`<br>`[Invalid identifier:_][integer:1][id:abc]` |

## Assignment submission requirements and procedure

- Each submitted assignment should contain four components: (1) the source code, (2) a group of test files, (3) a brief report, and (4) an executable named **`lexdriver`**, that extracts the tokens from all your test files. For each test file, the corresponding **`outlexerrors`** and **`outlextokens`** files should be generated.
- The assignment statement provides test files (**`.src`**) and their corresponding output files.
- The source code should be separated into modules using a comprehensible coding style.
- The assignment should be submitted through moodle 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.
- You may use any language you want in the project and assignments but the only fully supported language during the lab is Java.
- You have to submit your assignment before midnight on the due date on moodle.
- The file submitted must be a **.zip** file

## Evaluation criteria and grading scheme

| | | |
|---|---|---|
| **Analysis:** | | |
| Lexical specifications as regular expressions – document Section 1. | ind 2.1 | 2 pts |
| Finite state automaton representing the implementation, and description of the method used to generate the automaton from the regular expressions – document Section 2. | ind 2.2 | 3 pts |
| **Design/implementation:** | | |
| Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution to the stated problem – document Section 3. | ind 4.3 | 2 pts |
| Correct implementation according to the stated problem. | ind 4.4 | 20 pts |
| Error reporting – Output of clear error messages (error description and location) in the **`outlexerrors`** file. | ind 4.4 | 3 pts |
| Output of token stream in the **`outlextokens`** file. | ind 4.4 | 3 pts |
| Error recovery – the lexical analyzer continues running after errors are found. | ind 4.4 | 2 pts |
| Completeness of test cases (in addition to the grading files). | ind 4.4 | 10 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 – document Section 4. | 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** |