ALGONQUIN COLLEGE

ASSIGNMENT 2.1 – LANGUAGE MODEL (to be used in Scanner)

General View

Due Date: prior or on **Oct 16th 2021 (midnight)**

- **2nd Due date** (until 23rd Oct) - 50% off.

Earnings: 5% of your course grade.

Development: Activity can be done **individually** or in teams (**only 2 students** allowed).

Purpose: Modelling the language using RE (Regular Expressions), TD (Transition Diagram = Finite Deterministic Automata) and TT (Transition Table) for your language.

- This is an important activity from front-end compiler that is based on the model definition for your language.
- Your next activity (*scanner implementation*) will require the correct solution for each model.
  - Start defining the **RE** for tokens that can recognize variables, (functions), literals (numerical) and strings.
  - Then, create the **TD** (automata) to these elements.
  - Finally, finish the model using the **TT** (table) that will be used to this implementation.
- Use the section **Model Definition** (**RETDTT**) to answer your assignment correctly.

Task 1: RE (1.0 mark)

See the **RETDTT_SOFIA** document that defines the SOFIA language. You need to create your own language model.

- Start defining each type of lexeme classes you can use to invoke the RE. For instance, due to different datatypes in SOFIA, several classes were defined: letters, digits, special chars (underscore, period, delimiters, etc.).

- Check your language specification to do this.

TIP: Your language can be reviewed / updated. What does matter is that you can define your own specification, that must be different from SOFIA.

- In this part you must write **regular expressions** for the your language elements:
  - How to define variables (identifiers);
    - They can be defined separately according to each type.
  - How to define literals (constants);
    - Remember to use definitions for INTEGERS, FLOAT and STRINGS.
  - How to express keywords.
    - Use your own keyword list.

TIP: Check the list of lexemes used partially by Sofia.

Lexeme Classes: Considering the following syntax:

- **L = [*A-Za-z*]** **(Letters)**
- **D = [*0-9*]** **(Digits)**
- **M = *&*** **(MVID delimitator)**
- **Q = “** **(SL delimitator)**
- **E = *EOFS*** **(End of file symbol)**
- **O = [^LDMQE]** **(Other chars)**

(Check other classes)…

Here is the RE description of some tokens in SOFIA:

Answer:

- **MVID = M(L|D)***
- **SL = Q[L|D|O]*Q = Q[^EQ]*Q**

TIP: Note that in SOFIA, we have several additional Lexeme classes: IVID, FVID, SVID, IL, FPL.

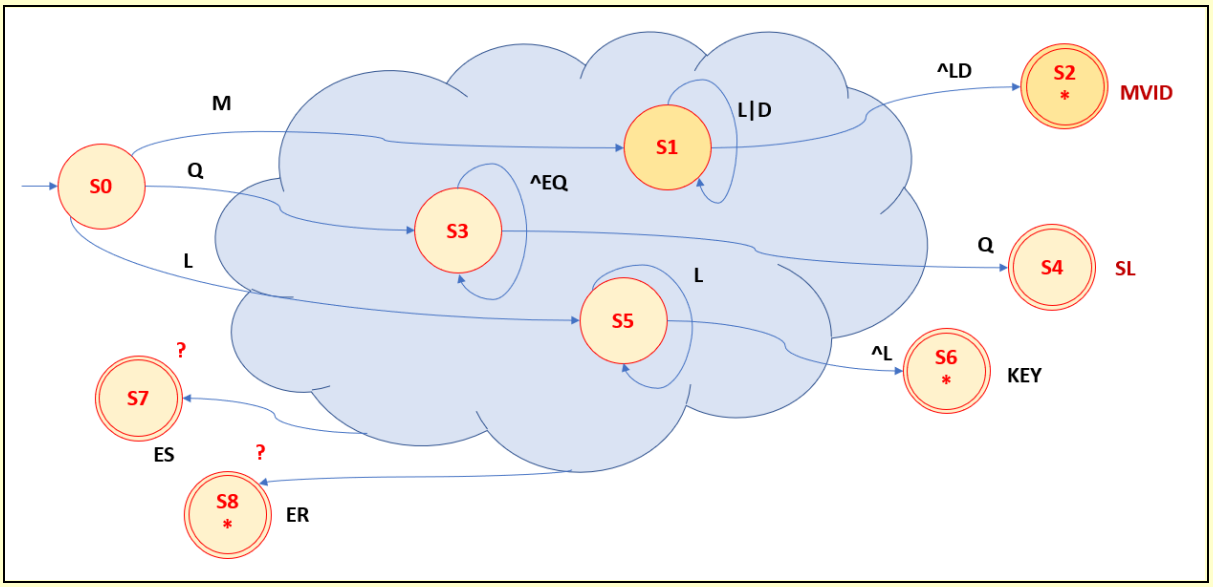
Task 2: TD (2.0 marks)

See again the **RETDTT_SOFIA** document that defines the automata for SOFIA. You need to have found the RE for this before (**Task 1**).

- The SOFIA TD is given to you in file **RETDTT_SOFIA** (model task). You need to understand it in order to complete your own diagram based on the following states:
- The initial state must be unique.
- The end / final states must map token classes from your language. For instance, in SOFIA, we have:
  - **VIDs**: Variable identifiers:

        - **MVID**: For Methods (functions).
    - **Ls**: Literals (similar to variables):
        - **SL**: String literals.
    - **KEY**: Keywords.
- Note that some intermediate states are used to take lexemes from initial state to one specific final state.
    - **Note:** This automaton must be DETERMINISTIC (no multiple destinations, neither empty symbol allowed).
- **TIPS:**
    - **TIP 1:** Since your language can contain different datatypes and formats, just focus on the minimum: arithmetic and string variables and constants.
    - **TIP 2:** You do not need to have different numerical ranges (such as "short", "long", etc.) or modifiers ("signed", "unsigned").

Partial solution:



Task 3: TT (2.0 mark)

See again the **RETDTT_SOFIA** document that defines the transition table for SOFIA.

- Part of the Scanner will be implemented using a Transition Table (TT) that comes from Deterministic Finite Automaton (DFA) (see **Task 2**).
- The TT of SOFIA will be composed by a table where:
    - **Columns** represent the lexeme classes (see task 1).

- **Rows** represent the states (see task 2).
- The **content** of the table is also a state given by the transition diagram (what will happen when, in one specific state, you read a symbol from a class).

- The states must be identified as:
  - NOAS: Internal states (non-acceptable states).
  - ASWR: Accepting (final) states with retract.
  - ASNR: Accepting (final) states without (no) retract.
- Note that in the end, this table must be implemented in your code (it will happen on Assignment A22).

Example:

Input	Input Symbol						Output
State	0	1	2	3	4	5	Type
	L(A-Z)	D(0-9)	M(&)	Q(")	E	O	
0	5	7	1	3	ER	7	NOAS
1	1	1	2	2	ER	2	NOAS
2	IS	IS	IS	IS	IS	IS	ASWR (MVID)
3	3	3	3	4	ER	3	NOAS
4	IS	IS	IS	IS	IS	IS	ASNR (SL)
5	5	6	6	6	ES	6	NOAS
6	IS	IS	IS	IS	IS	IS	ASWR (KEY)
7	IS	IS	IS	IS	IS	IS	ASNR (ERR)
8	IS	IS	IS	IS	IS	IS	ASWR (ERR)

Submission Details

- **Digital Submission: Compress** into a **zip** file with **ALL files** that you are using in this model – essentially, DOC file, but you can eventually include pictures. Also include a cover page.
- The submission must follow the course submission standards. You will find the Assignment Submission Standard as well as the Assignment Marking Guide (**CST8152_ASSAMG.pdf)** for the Compilers course on the Brightspace.
- Upload the **zip** file on Brightspace. The file must be submitted prior or on the due date as indicated in the assignment.
- ***IMPORTANT NOTE:*** The name of the file must be **Your Last Name** followed by the last three digits of your student number followed by your lab **section number**. For example: **Sousa123_s10.zip**.

- If you are working in teams, please, include also your partner. For instance, something like: **Sousa123_Melo456_s10.zip**.
- **Remember:** Since we have just one lab professor, students from the **different sections** can constitute a team.

❖ **How to Proceed:** You need to demonstrate your progress to your Professor in **private Zoom Sections** during Lab sessions.

- If you are working in teams, you and your partner must do it together, otherwise, only the student that has presented can get the bonus marks.
- Eventual questions can be posed by the Lab professor for any explanation about the code developed.
- Each demo is related to a **specific lab** in one specific week. If it is not presented, no marks will be given later (even if the activity has been done).

Marking Rubric

Maximum Deduction (%)	Deduction Event
CRICTICAL	**Severe Errors**
100%	Late submission (1 week after due date)
Up to 100%	Plagiarism detection (remember languages are different)
Task 1 – RE	**Regular Expression**
Up to 20%	Missing lexeme classes
Up to 20%	Wrong RE (remember to match with IDs and literals)
Task 2 – TD	**Transition Diagram (automata)**
Up to 40%	Compliance with RE
Up to 40%	Wrong TD (remember deterministic behavior)
Task 3 – TT	**Transition Table**
Up to 40%	Compliance with TD
Up to 40%	Wrong TT (bad table definition)
ADDITIONAL	**Small problems**
Up to 20%	Language adaptation (missing elements – ex: datatypes / constants)
Up to 20%	Unjustified modification (if you changed the language, explain why)
up to 10%	Other minor errors
up to 10%	Bonus: discretionary ideas about language.
Final Mark	**Formula:** 5*((100- ∑ penalties + bonus)/100), max score 5%.

File update: Oct 3rd 2021.

Good luck with A21!