ALGONQUIN COLLEGE

ASSIGNMENT 2.1 – LANGUAGE MODEL (for Scanner)

General View

Due Date: prior or on **16th Oct 2022 (Sun, midnight)**

- **2nd Due date** (until Oct 23rd 2022) - 50% off.

Earnings: 10% 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 – Regular Expressions (1.0 mark)

See the **RETDTT_BOA** document that defines the **BOA** language using a progressive model: **RE** – Regular Expression, **TD** – Transition Diagram and **TT** – Transition Table. 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 **BOA**, 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 BOA language.

- In this part you must write **regular expressions** for 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 NUMBERS (can be integers, floats or both) and STRINGS.
  - How to express **keywords**.
    - Use your own keyword list.

TIP: Check the list of lexemes used partially by BOA (see Appendixes in lecture notes).

Lexeme Classes: Considering the following syntax:

Answer:

- **L = [*A-Za-z*]** **(Letters)**
- **D = [*0-9*]** **(Digits)**
- **U = _** **(Underscore)**
- **M = *&*** **(MVID delimitator)**
- **Q = “** **(SL delimitator)**
- **O = [^LDUMQ]** **(Other chars)**

(Check other classes – Ex: EOF)

Tab 1 *– Lexeme Classes (partial) in* ***BOA***

Note 1: About Lexeme Classes

These classes are mapping specific chars in the **ASCII Table** (standard in C language compilers). You do not need to have 255 (or 127) different columns, but you can define “sets” for specific chars. Remember that if one char has a proper function in your language to identifiers, constants, etc., you probably will need to define the lexeme. For example, in the Tab 1, ‘&’ and ‘”’ are included because they must appear in some tokens (respectively MID – Method Identifier and SL – String Literal). The “**O**” (“Other chars”) class is mandatory because your language must be able to treat any kind of char that you are reading.

The above lexeme classes will be necessary to define **RE** (Regular Expressions) for your language. For instance, some tokens in **BOA** can be defined as:

Answer:

- **MID = L[L|D|U]*M** **// Method identifier**
- **SL = Q[^Q]*Q = Q[L|D|U|M|O]*Q** **// String Literal**

*Tab 2 – Some RE (partial) in **BOA***

TIP: Note that in **BOA**, we have several additional RE for different language elements: IVID, FVID, SVID, IL, FPL.

Note 2: About Tokens and Classes
All tokens from your language must be recognized. For this reason, it is also the proper time to perform some adjustments in your own language definition (**A11**). For instance, sometimes, the specification is so rich that the RE are very complex (see, for example, multiple kinds of literal formats. In some other situations, the use of scope can be removed (for instance, excluding some streams names and OO definitions). In these cases, the better strategy is simplification (remember: **KISS philosophy**).

Task 2: TD – Transition Diagram (2.0 marks)

To define the automata / **TD** (Transition Diagram), you need to have found the **RE** for this before (**Task 1**).

- The TD (from BOA language) is given to you in file **RETDTT_BOA** (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 BOA, we have:
  - **VIDs**: Variable identifiers:
    - **MVID**: For Methods (functions).
  - **L**: Literals (similar to variables):
    - **SL**: String literals.
  - **KEY**: Keywords.

Partial solution:

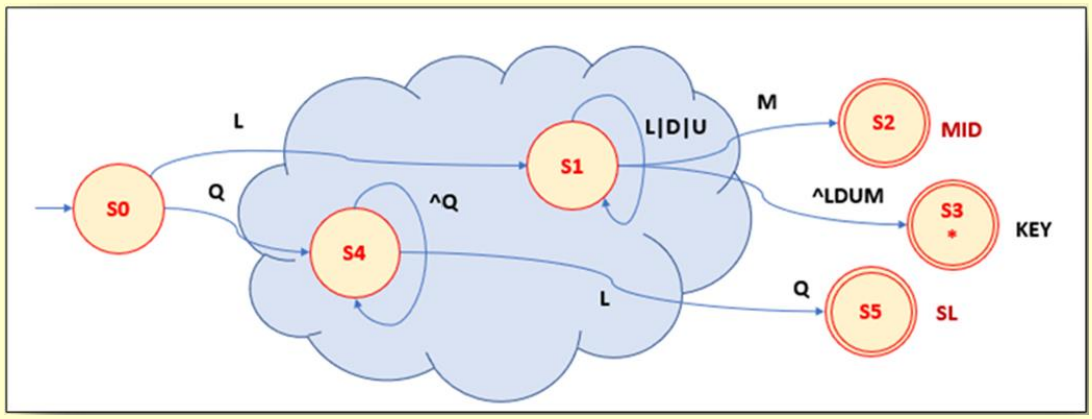

Fig 1 – Partial automaton used by RE in BOA

- 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).

Note 3: About Automata
The **automata** (or **TD** = Transition Diagram) is a visual representation that describes how the **buffer** (**A11** / **A12**) is used to classify some tokens. Note that several tokens can be checked not using the TD. You can realize if one specific RE must be recognized using automata is given by the **variability** and **application** (you will see in Lectures). Basically "atomic" tokens (ex: '+' or '(') do not need specific actions – so they can be checked immediately in one specific part of the code. However, the case is completely different when you need to recognize literals (ex: strings) or identifiers (ex: method names), when we need to develop specific functions to treat them, as it will be shown in **A22**.

- **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").

Task 3: TT – Transition Table (2.0 mark)

See again the **RETDTT_BOA** document that defines the transition table for **BOA**.

- Part of the Scanner will be implemented using a Transition Table (TT) that comes from Deterministic Finite Automaton (DFA) (see **Task 2**).

Example:

Input	Input Symbol						Output
State	0	1	2	3	4	5	Type
	L(A-Z)	D(0-9)	U(_)	M(&)	Q(")	O	-
0	1	ES	ES	ES	4	ES	NOAS [0]
1	1	1	1	2	ES	3	NOAS [1]
2	IS	IS	IS	IS	IS	IS	ASNR (MNID) [2]
3	IS	IS	IS	IS	IS	IS	ASWR (KEY) [3]
4	4	4	4	4	5	4	NOAS [4]
5	IS	IS	IS	IS	IS	IS	ASNR (SL) [5]

***Fig 2** – Transition Table (partial) that represents the TD used in BOA*

- The TT of **BOA** 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).

Note 4: About Transition Table
The table is one simple way to implement an automaton without changing the code for any new token to be recognized. The logic is based on the idea that, you **move** from one state to another (rows) according the symbols that you are reading (columns), until one specific final state is reached (last column). This is the way that you will implement the **automaton** in the code. It means that some states are final (ASWR / ASNR), and, therefore, must be able to be matched with specific **functions**, while most of them are not final (NOAS).

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** (**CST8152_Compilers_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: **A21_Sousa123.zip**.
  - If you are working in teams, please, include also your partner. For instance, something like: **A21_Sousa123_Sousa456.zip**.
  - ***Note:*** *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**
10 pt	Late submission (after 1 week / 2 weeks)
10 pt	Plagiarism detection (remember languages are different)
Task 1 – RE	**Regular Expression**
5 pt	Missing lexeme classes
5 pt	Wrong RE (remember to match with IDs and literals)
Task 2 – TD	**Transition Diagram (automata)**
3 pt	Compliance with RE
3 pt	Wrong TD (remember deterministic behavior)
Task 3 – TT	**Transition Table**
2 pt	Compliance with TD
2 pt	Wrong TT (bad table definition)
ADDITIONAL	**Small problems**
1 pt	Language adaptation (missing elements – ex: datatypes / constants)
1 pt	Unjustified modification (if you changed the language, explain why)
0.5 pt	Other minor errors
1 pt	Bonus: GitHub utilization
1 pt	Bonus: discretionary ideas about language.
Final Mark	**Formula: 10***((100- ∑ penalties + bonus)/100), max score 12%**.**

File update: Oct 2nd 2022.

Good luck with A21!