

PHASE 1: LEXICAL ANALYSIS

Definition:

Given a piece of code, this phase should analyze it and outputs a list of tokens found. This phase relies on a pre-made DFA transition table, which should be well-designed for the best result.

Design:

THE CLASS:

We made a class and called it `Lexical`. The constructor of the class takes only the path of the DFA table (as plain-text file).

A lexical analyzer requires a transition table to use. We wanted to make that table code-independent, which will make future additions and edits much easier. To solve this problem, we designed our system to deal with a specific-formatted plain-text file containing the transitions.

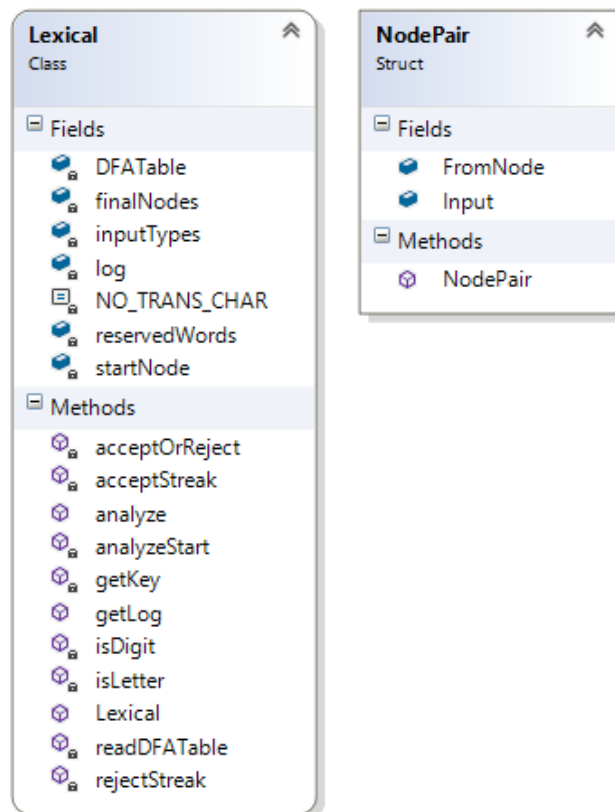
The system initially takes the path of the table once, and analyze any piece of code afterwards with the same constructed table using the function `analyze`.

`Lexical` main functions are:

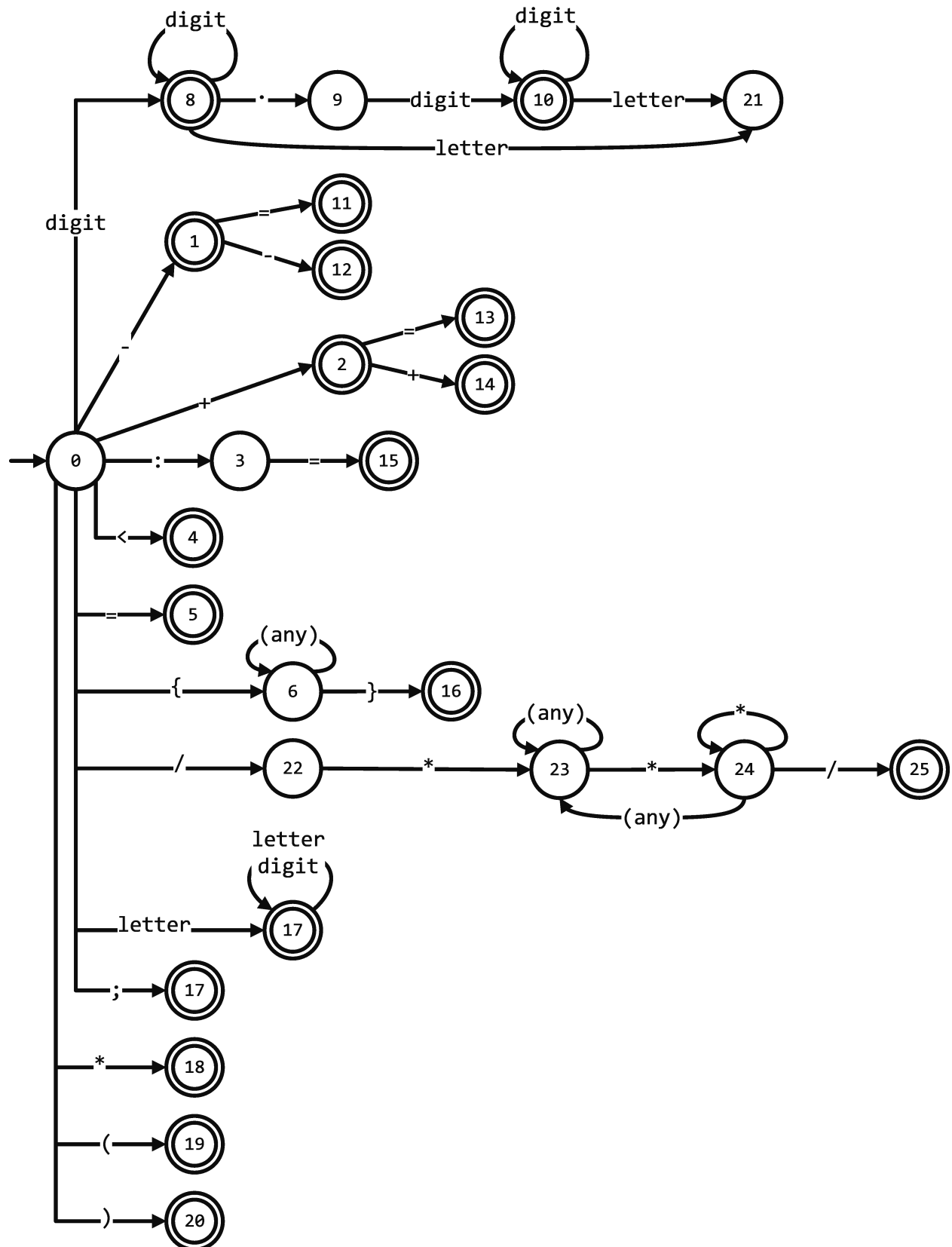
1. `readDFATable`: A private function that reads the content of the plain-text DFA transition table and store it a private data-structured object. This function is automatically called when a new object of `Lexical` is instantiated.

2. **analyze**: A public function that do the work of analyzing the code (converting the code into tokens). The code string is given as the only input parameter. Returns a **List** object of accepted tokens.
3. **getLog**: A public function that returns a summary (log history) of the last analyzed code by **analyze**. This is usually useful for testing the system. The log contains both accepted and rejected tokens along with their lexeme names.

CLASS DIAGRAM:



DFA DIAGRAM:



Output:

The expected output from this phase is a list of tokens found in the code. The **Lexical** class also offers log history showing the accepted tokens along with any error/unidentified tokens for testing purposes.

The system supports the following tokens:

- ID's
- Numbers
- Fraction numbers
- Minus operator (-)
- Plus operator (+)
- Less-than comparison (<)
- Equality comparison (=)
- Assign operator (:=)
- Augmented minus operation (-=)
- Augmented plus operation (+=)
- Decrement operator (--)
- Increment operator (++)
- C-like comments (/*comment*/)
- Curley braces comments ({comment})
- End of statement sign (;)
- Multiplication operator (*)
- Left parenthesis
- Right parenthesis

And the following reserved words (as defined by Tiny-C language description):

If, Then, Else, End, Write, Read, Repeat, Until.