Michael Burdi, Tristan Bui, Naoki Atkins

Cal state fullerton | cpsc 323 - summer 2020

RAT20SU

ASSIGNMENT 1: LEXICAL ANALYZER

# Problem Statement

A source code program is a file that contains text describing what the computer shall do. The purpose of a compiler is to take this source code program and make it into an executable program. In the compiler’s first phase, it goes through a process called lexical analysis. Lexical analysis uses a lexical analyzer to divide a program into tiny units called tokens. These tokens contain information about its type and the value that it holds.

A token is produced through the lexical analyzer by means of a Finite State Machine. The lexical analyzer will read each line of source code, character by character, feeding it into the Finite State Machine. The Finite State Machine will continuously update its state after reading each character. Upon reading a white space, or any character that marks the end of the current token, the lexical analyzer will return the token to the calling function.

Tokens must be categorized into one of the following categories:

* Identifier
* Integer
* Keywords
* Separator
* Operator
* Unknown
* None

# How to use the Program

* Unzip the folder.
* Run Lexer.exe (MacOS).
* Terminal will prompt to input filename. (Best to use absolute file path just in case.)
* Output will be printed to Terminal

# Design of the Program

The main part of this program is the Lexer class which overrides the function call operator allowing the class to be used as a function. A Lexer object is constructed with an input stream (i.e. file stream or string stream) where it will extract tokens from. Return values are type Lexer::OutputType which is a struct that contains the lexemes and the token enumeration type—one of NONE, IDENTIFIER, KEYWORD, SEPARATOR, OPERATOR, INTEGER, or UNKNOWN.

Inside the Lexer class are, we have 4 private helper functions:

* isoperator()
* isseparator()
* isunderscore()
* iskeyword()

Two additional library functions are used:

* isalpha()
* isnumber()

The Lexer function loops (transitions) until it reaches a break (a finished state). On each iteration of the loop there is a state transition. The change of the token TYPE signifies a transition to a new state. If the type does not change for any iteration, the state is not transitioning but goes back to itself and awaits more input.

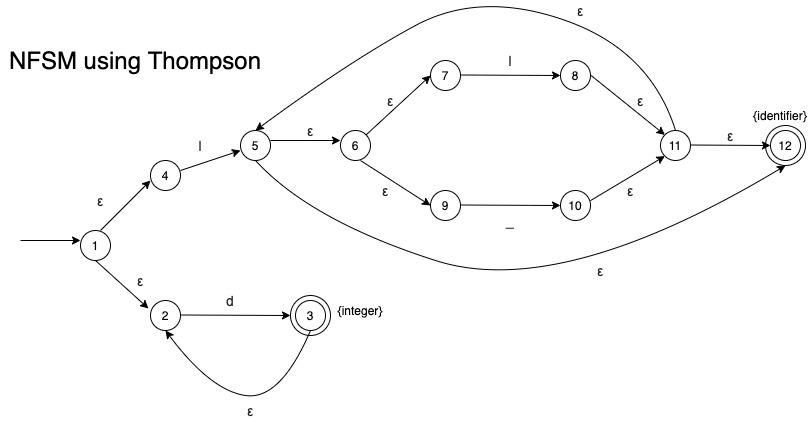
FSM

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  | a-zA-Z\_ | 0-9 | +-\*/<> | = | (); | SPACE |
| f0 = | 0 = none | 1 | 2 | 3 | 4 | 6 | 0 |
|  | 1 = identifier | 1 | 7 | - | - | - | - |
|  | 2 = integer | - | 2 | - | - | - | - |
|  | 3 = operator | - | - | - | - | - | - |
|  | 4 = assignment | - | - | - | 5 | - | - |
|  | 5 = equality | - | - | - | - | - | - |
|  | 6 - separator | - | - | - | - | - | - |
|  | 7 - unknown | - | - | - | - | - | - |
|  |  |  |  |  |  |  |  |
|  | Accepting = | 1, 2, 3, 4, 5, 6, 7 |  |  |  |  |  |

Regular Expressions

Identifier: l(l|\_)\*

Integer: d+



# Limitations

None.

# Shortcomings

None.