CS323 Documentation

About 2 pages

1. Problem Statement

Our project seeks to address the issue of constructing a lexical analyzer for the programming language, RAT24S. The process of compiling a program begins with lexical analysis, which employs a lexical analyzer, AKA, lexer. The lexer has the function of dividing the source code into their token categories and identifying their respective lexemes. This is a multi-step process which begins with identifying the regular expressions for the tokens, utilizing Thompson's method to create an accompanying NFSM, then converting the NFSM to a DFSM through the subset method, and establishing a transition table for possible inputs, states, and accepting states. In implementing these structures into our program, we allow the lexer function to call the DFSMs of integer, int, and reals, which return whether the inputted token is valid or not. This allows the lexer to then identify the token and its corresponding lexeme. In addition, the established keywords, separators, and operators are identified and processed, and written to an output file, along with the results from the DFSM processing. The study and practice of lexical analysis is significant as it is the first step of the compilation process and understanding the breakdown of the process of identifying and validating tokens establishes a foundation upon which the rest of the understanding and implementation of the compilation process is built. The objective of this assignment is to gain a better understanding as to the mechanisms of a lexical analyzer deployed during the compilation process. By manually implementing one, it allows for further understanding of how lexers process input, segment the input, and call the corresponding FSMs to validate and identify them.

2. How to use your program

1. *First make sure the main.py plus all the test case files are in the same directory.*

2. *open up terminal in that directory, type: python main.py*
3. *it should export the output into a text file called 'output.txt", read that to acquire the output.*
4. *you can modify the input of the test case by directly modifying the text file itself or replacing it with your own file with the same file name.*
5. *The python version we are using is python 3.12*

3. Design of your program

< write major components of your program. Also, data structures you are utilizing, particular algorithms you have chosen etc. >

Left Recursion

R25. <Expression> ::= <Expression> + <Term> | <Expression> - <Term> | <Term>
E -> E + T | E - T | T
Left-Recursive Productions: E + T, E - T
Non-Left-Recursion Productions: T
Introduce New Non-terminal: E'

E -> TE'
E' -> +TE' | -TE' | ε

R26. <Term> ::= <Term> * <Factor> | <Term> / <Factor> | <Factor>
T -> T * F | T / F | F
Left-Recursive Productions: T * F, T / F
Non Left-Recursion Productions: F
Introduce New Non-terminal: T'

T -> FT'
T' -> *FT' | /FT' | ε

Backtracking

R3. R3. <Function Definitions> ::= <Function> | <Function> <Function Definitions>
D -> F | FD
Re-worked:
D -> FD'
D' -> D | ε

R6. <Parameter List> ::= <Parameter> | <Parameter> , <Parameter List>
L -> P | P,L
Re-worked:
L -> PL’
L’ -> L | ε

R11. <Declaration List> := <Declaration> ; | <Declaration> ; <Declaration List>
L -> D | D;L
Re-worked:
L -> DL’
L’ -> L | ε

R13. <IDs> ::= <Identifier> | <Identifier>, <IDs>
D -> I | I, D
Re-worked:
D -> ID’
D’ -> I | ε

R14. <Statement List> ::= <Statement> | <Statement> <Statement List>
L -> S | SL
Re-worked:
L -> SL’
L’ -> L | ε

R18. <If> ::= if (<Condition>) <Statement> endif |
if (<Condition>) <Statement> else <Statement> endif

I -> iCSf | iCSeSf
Re-worked:
I -> iCSI’
I’ -> eSf | f

I = If, i = if, C = Condition, S = Statement, I’ = If Prime, e = else, f = endif

R19. <Return> ::= return; | return <Expression>;
R -> r | rE
Re-worked:
R -> rR’
R’ -> E | ε

List of Non-terminals (Functions to be Implemented)

1. Rat24S
2. Optional Function Definitions
3. Optional Declaration List
4. Statement List
5. Function Definitions
6. Empty
7. Function
8. Optional Parameter List
9. Body
10. Parameter List
11. Parameter
12. IDs
13. Qualifier
14. Declaration List
15. Declaration
16. Statement
17. Compound
18. Assign
19. If
20. If Prime
21. Return
22. Print
23. Scan
24. While
25. Expression
26. Expression Prime
27. Condition
28. Relop
29. Term
30. Term Prime
31. Factor
32. Primary
33. Function Definition Prime
34. Parameter List Prime
35. Declaration List Prime
36. IDs Prime
37. Statement List Prime

38. Return Prime

4. Any Limitation

<All features are running according to the assignment but you limit your program due to resource limitations, such as Maximum number of lines in the source code, size of the identifier, integer etc. ***Say 'None' if there is no limitation****>*

5. Any shortcomings

<Anything you could NOT implement although that is required by the Assignment. ***Say 'None' if there is no shortcoming****>*

didn't finish