SER 502 Project Deliverable 1

- Team 27

Team:

Srilakshmi Sravani Andaluri

Vamsi Krishna Somepalli

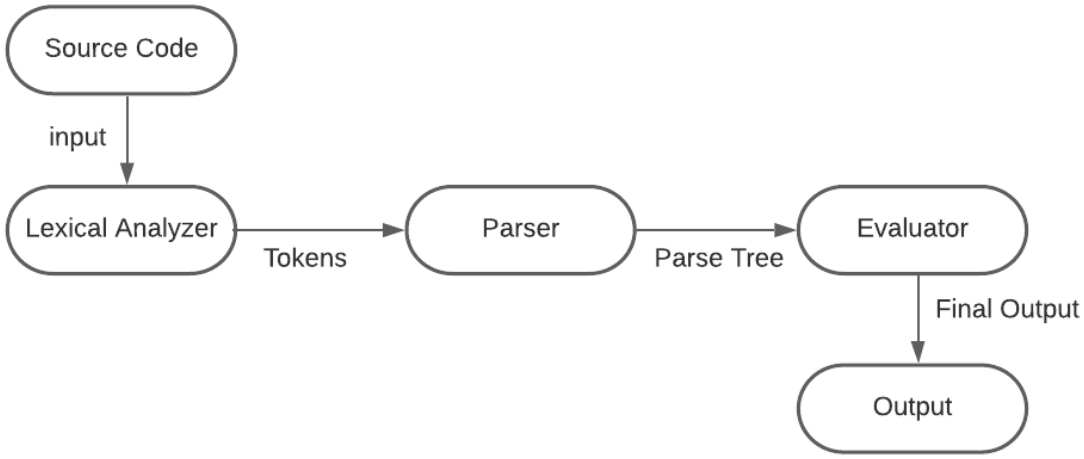
Sathwik Reddy Dontham

Saish Vemulapalli

Rohith Reddy Byreddy

Name: ALBITOR

Project Flow or Execution Outline:



Source Code: A program that we create is stored in a file known as the source code. We can run this program to obtain its output. The file containing the source code has an "albi" file extension.

Lexical Analyzer: The source code file is fed into the lexical analyzer, which divides the program into smaller units known as tokens through lexical analysis. During this phase, any trailing white spaces or unnecessary comments are eliminated, and the program is inspected for syntax errors or inconsistencies.

Parser: The parser receives the tokens generated by the lexical analyzer and performs syntax analysis on the program. It creates a parse tree or syntax tree that outlines the program's structure. Additionally, it identifies any grammatical or syntax errors present. lark parser in python is used. The design language employed in this process is Prolog. DCG grammar is used. Top Down parsing technique is used for parse trees. List data structure is used.

Evaluator: The parse tree produced by the parser is utilized to generate the program's output. The Evaluator verifies that the program is semantically accurate and executes the required calculations. List data structure and prolog is used. It assesses the entire program to generate the desired output, which is both accurate and expected.

Output: This is the final output generated after completion of the execution of our program in Albitor language.

Tokens:

The token that can be recognized by the Albitor are listed below:

1. **Data Types:**
    a. String
    b. Integer
    c. Boolean
2. **Variable/Identifier:**

    Variables are named starting with a lowercase letter and can include uppercase and underscores. They can also be assigned to a value with “=” token (assignment) between the expression and identifier

3. **Conditional Statements:**
    a. Ternary Operator : (Boolean?true:false)
    b. if-then-else (eg: if Boolean true else false)
4. **Loops:**
    a. For loop (for assignment ; boolean ; increment)
    b. While loop (while condition true)
    c. For in range loop (for identifier in range(expression,expression)
5. **Arithmetic Operators:**
    a. Addition (+)

b. Subtraction (-)
c. Multiplication (*)
d. Division (/)
e. Brackets (‘()’)
f. And (&&)
g. Or (||)
h. Mod (%)

6. **Comparison:**
    a. Equals (==)
    b. Not equals (!=)
    c. Greater than (>)
    d. Less than (<)
    e. Greater than or equal to (>=)
    f. Less than or equal to (<=)
7. **True/False:**
    a. true
    b. false
    c. Not (!)
8. **Print:**

    Print the given identifiers which includes boolean, integers and strings.

Grammar:

program → [begin] ,block, [end].

block → [{], command, [}].

command → statement,command | statement.

statement → print_statement , [;] |

assignment_statement , [;] |

declaration_statement , [;] |

for_loop |

while_loop |

if_condition.

print_statement → [print(],expression,[);],

expression →variable |

[“], string_literal , [“] |

artimetic_operation |

compare_operations.

arthimetic_operation → arthimetic_operation1,[+],arthimetic_operation |

arthimetic_operation1,[-],arthimetic_operation.

arithmetic_operation → arithmetic_operation1.

arithmetic_operation1 → arithmetic_operation2, [*], arithmetic_operation1 |

arthimetic_operation2, [/], arthimetic_operation1 |

arthimetic_operation2, [%], arthimetic_operation1.

arithmetic_operation1 → arithmetic_operation2.

arithmetic_operation2 → [(], arithmetic_operation, [)].

integer |

variable.

compare_operations →

compare_statement, [compare_operator], compare_operations |

compare_statement.

compare_statement → variable |

Integer |

[(],arthimetic_operation,[)].

compare_operator → [<] | [<=] | [>] | [>=] | [==] | [!=] | [&&] | [||] | [!].

assignment_statement → declaration_statement,[=],expressions1|

variable,[=],expressions1.

expressions1 → variable | boolean |

arthimetic_operation |

compare_operations |

[“], string_literal ,[“] |

compare_operations , [?] , block , [:] , block .

boolean → true | false.

declaration_statement → datatype, variable.

for_loop → [for], [(], assignment_statement, [;], compare_operations, [;], assignment, [)], block. |

[for], variable, [in range (], integer, [,], integer, [)], block.

while_loop → [while], [(], compare_operations, [)], block.

if_condition → [if], [(], compare_operations, [)], block |

[if], [(], compare_operations], [)], block, [then], [(], compare_operations, [)], block |

[if], [(], compare_operations], [)], block, [else], [(], compare_operations, [)], block |

[if], [(], compare_operations], [)], block, [then], [(], compare_operations, [)], block, [else], [(], compare_operations, [)], block.

integer → [0],Z | [1],Z | [2],Z | [3],Z | [4],Z | [5],Z | [6],Z | [7],Z | [8],Z | [9],Z

Z → [0],Z | [1],Z | [2],Z | [3],Z | [4],Z | [5],Z | [6],Z | [7],Z | [8],Z | [9],Z | []

datatype → [int],[bool],[string].

variable → [a], Y | [b], Y | [c], Y | [d], Y | [e], Y | [f], Y | [g], Y | [h], Y | [i], Y | [j], Y | [k], Y | [l], Y | [m], Y | [n], Y | [o], Y | [p], Y | [q], Y | [r], Y | [s], Y | [t], Y | [u], Y | [v], Y | [w], Y | [x], Y | [y], Y | [z], Y

Y → [a], Y | [b], Y | [c], Y | [d], Y | [e], Y | [f], Y | [g], Y | [h], Y | [i], Y | [j], Y | [k], Y | [l], Y | [m], Y | [n], Y | [o], Y | [p], Y | [q], Y | [r], Y | [s], Y | [t], Y | [u], Y | [v], Y | [w], Y | [x], Y | [y], Y | [z], Y, | [_], Y | [A], Y | [B], Y | [C], Y | [D], Y | [E], Y | [F], Y | [G], Y | [H], Y | [I], Y | [J], Y | [K], Y | [L], Y | [M], Y | [N], Y | [O], Y | [P], Y | [Q], Y | [R], Y | [S], Y | [T], Y | [U], Y | [V], Y | [W], Y | [X], Y | [Y], Y | [Z], Y. | []

https://github.com/RohithReddyByreddy/SER502-Spring2023-Team27