CS 3513 - Programming Languages
Programming Project 01

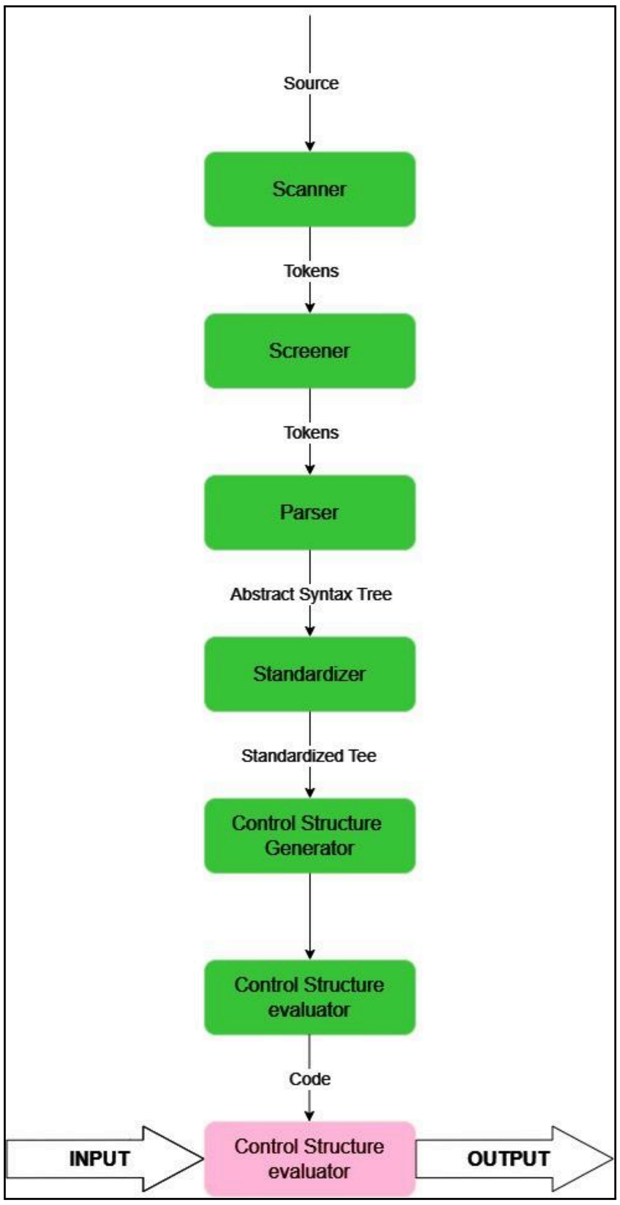
Group 17

Wickramasinghe J.J - 210706H
Chandrasena W.G.M.D - 200086U

1. Problem Description

The project requirement was to implement a lexical analyzer and a parser for the RPAL language. The output of the parser should be the Abstract Syntax Tree (AST) for the given input program. Then an algorithm must be implemented to convert the Abstract Syntax Tree (AST) into Standardize Tree (ST) and the CSE machine should be implemented. The program should be able to read an input file that contains an RPAL program. The output of the program should match the output of “rpal.exe“ for the relevant program.

2. Solution



Here’s an overview of each phase in a compiler.

1. Tokenizer (Scanner+Screener)	**Objective:** Implement a lexical analyzer to process an input file, generate tokens, identify reserved keywords, remove comments and whitespace, and return an array of tokens. 1. *Read Input File* 2. *Initialize Scanner* 3. *Token Generation* 4. *Reserved Keywords* 5. *Remove Comments, Whitespace* 6. *Return the array of Tokens*
2. Abstract SyntaxTree parser	**Objective:** Iterate through the token sequence and build an Abstract Syntax Tree (AST) using recursive descent parsing. 1. *Initialize Parser* 2. *Recursive Descent Parsing* 3. *AST Node Creation* 4. *Return AST*
3. Standardizer	**Objective:** Standardize the Abstract Syntax Tree using a given set of standardizing rules. 1. *Define Standardizing Rules* 2. *Traverse the AST and apply Rules* 3. *Return Standardized AST*
4. Control Structure Generation	**Objective:** Perform a pre-order traversal of the AST nodes while maintaining a FIFO queue to generate control structures. 1. *Pre-order Traversal* 2. *Use FIFO queue to manage the nodes during traversal* 3. *Generate Control Structures*
5. Control Structure Environment evaluation	**Objective:** Maintain a control structure array and a stack. Pop each element in the control structure array and execute a rule based on the stack top and the environment. 1. *Initialize Environment* 2. *Evaluation Loop* 3. *Stack-Based Execution* 4. *Return Results*

3. Structure of the Project:

myrpal.py	The main executable script for the interpreter which contains the parser as well.
Tokenizer.py	Handles the tokenization of the input RPAL program into meaningful units (tokens).
ASTNode.py	Handles the creation ,manipulation and Standarizationof nodes within the Abstract Syntax Tree (AST).
cseMachine.py	Implements the Control structure environment machine, crucial for executing the Standardized Tree.
Environment.py	Manages environment settings such as variable bindings, essential for the execution phase.
controlStructure.py	Generates and Manages the control structures within the interpreter.

4. Detailed Functionality Analysis

4.1 Tokenization Process

File: myrpal.py

Initialization: The program begins by processing command line arguments, initializing a token list, and iteratively calling the get_next_token() method from Tokenizer.py to append tokens to the list.

File: Tokenizer.py

4.2 Scanner

The lexical analyzer breaks down the RPAL program text into tokens, categorizing each token by type (e.g., keyword, identifier) using the TokenType Enum.

4.3 Screening

- The Screener class is responsible for screening the tokens.
- The merge_tokens() method combines tokens like **, ->, =>, and =<.
- The remove_comments() method removes any comments.
- The screen_reserved_keywords() method identifies reserved keywords.

4.4 Parsing and AST Generation

File: myrpal.py

4.4.1 Parsing:

- Uses recursive descent parsing to create an Abstract Syntax Tree (AST) from the tokens.
- The parsing process begins with the procE() method in the ASTParser class.

4.5 AST to Standardized Tree Transformation

File: ASTNode.py

Standardisation:

- The standardize() method in the ASTNode class transforms the AST.
- Nodes standardized include:

1. let
2. where
3. within
4. function_form
5. and
6. rec
7. @

4.6 Control Structure Generation

File: controlStructure.py

Data Structures:

- ❖ Utilizes a binary AST Node and a FIFO Queue.

Control Structure Generation:

- ❖ The ControlStructureGenerator class generates control structures, starting with the generate_control_structures() method.
- ❖ Initially, it traverses the standardized tree in a pre-order fashion, treating lambda nodes as leaves and pushing the right child of the lambda to the FIFO queue.
- ❖ After the initial traversal, it iterates through the queue to generate control structures, storing them in the map_ctrl_structs dictionary with control structure IDs as keys and node lists as values.

4.7 CSE Machine Execution

File: cseMachine.py

- ❖ The CSEMachine class evaluates control structures starting with the execute() method.
- ❖ It begins by pushing the e0 environment variable with empty key values to both the control structure list and the stack.
- ❖ Then, it pushes the contents of the 0th control structure to the control structure list.
- ❖ The CSEMachine performs operations based on the top of the control structure and stack, following specific CSEMachine rules.

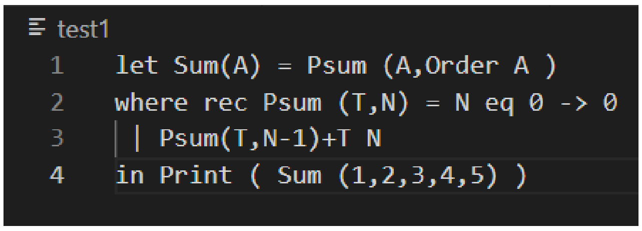
5. Program Execution

Execution Instructions:

- ❖ In order to get the output only,
  - ➢ *Python .\myrpal.py file_name*

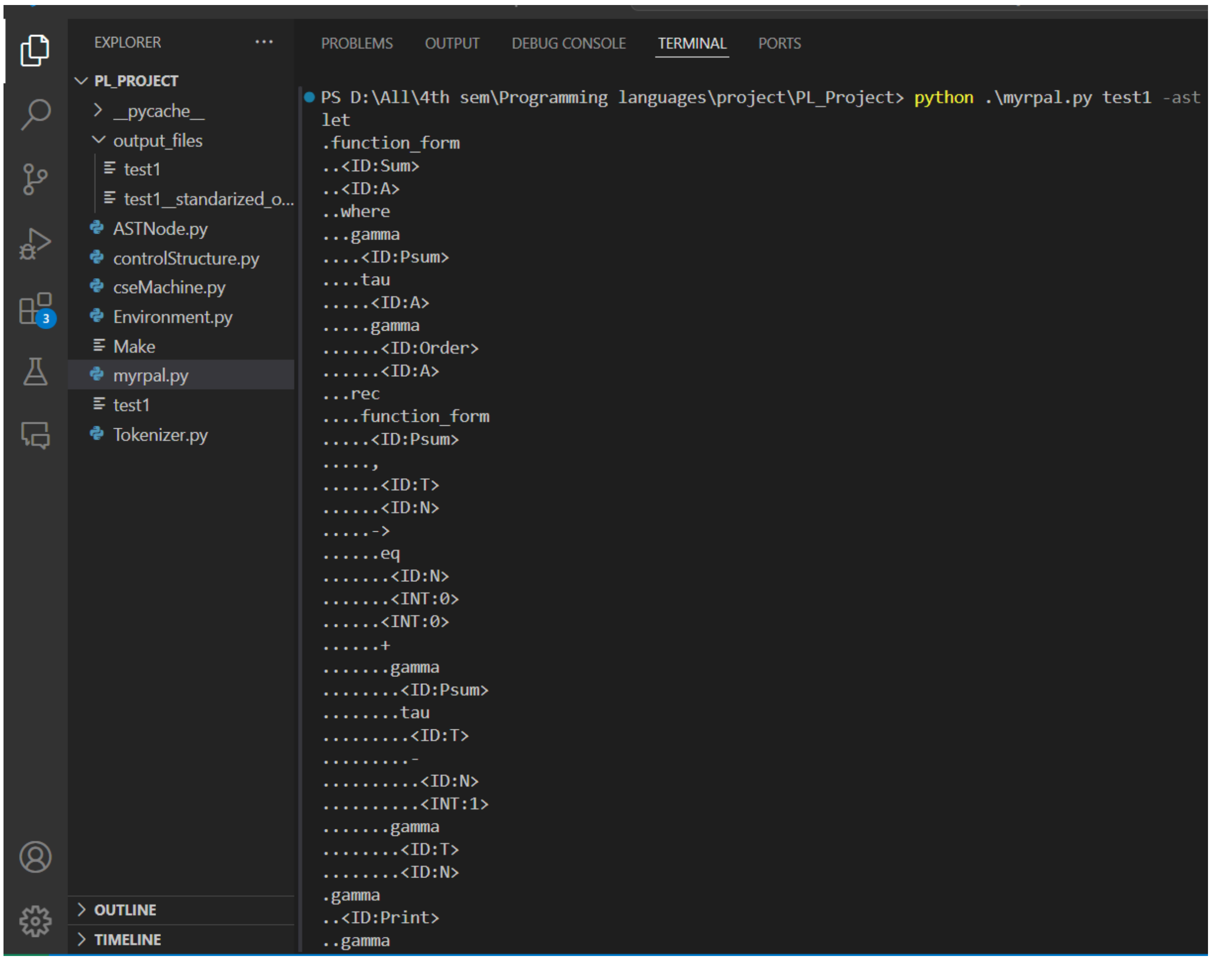
- ❖ In order to get the AST tree printed in the command line
  - ➢ *Python .\myrpal.py file_name -ast*

Input :



```
test1
let Sum(A) = Psum (A,Order A )
where rec Psum (T,N) = N eq 0 -> 0
  | Psum(T,N-1)+T N
in Print ( Sum (1,2,3,4,5) )
```

Output- Generated AST :



```
PS D:\All\4th sem\Programming languages\project\PL_Project> python .\myrpal.py test1 -ast
let
.function_form
..<ID:Sum>
..<ID:A>
..where
...gamma
....<ID:Psum>
....tau
.....<ID:A>
.....gamma
......<ID:Order>
......<ID:A>
...rec
....function_form
.....<ID:Psum>
.....,
......<ID:T>
......<ID:N>
.....->
......eq
.......<ID:N>
.......<INT:0>
......<INT:0>
......+
.......gamma
........<ID:Psum>
........tau
.........<ID:T>
.........-
..........<ID:N>
..........<INT:1>
.......gamma
........<ID:T>
........<ID:N>
.gamma
..<ID:Print>
..gamma
```

6. Conclusion

❖ In conclusion, this project has provided invaluable insights into compiler construction, demonstrating the synergy between theoretical concepts and practical implementation.