

CS3513 – Programming languages Project Report

Group 7

Group Members

Mamta N. 210365J

Madhushika K.A.H.M. 210350J

1. Abstract

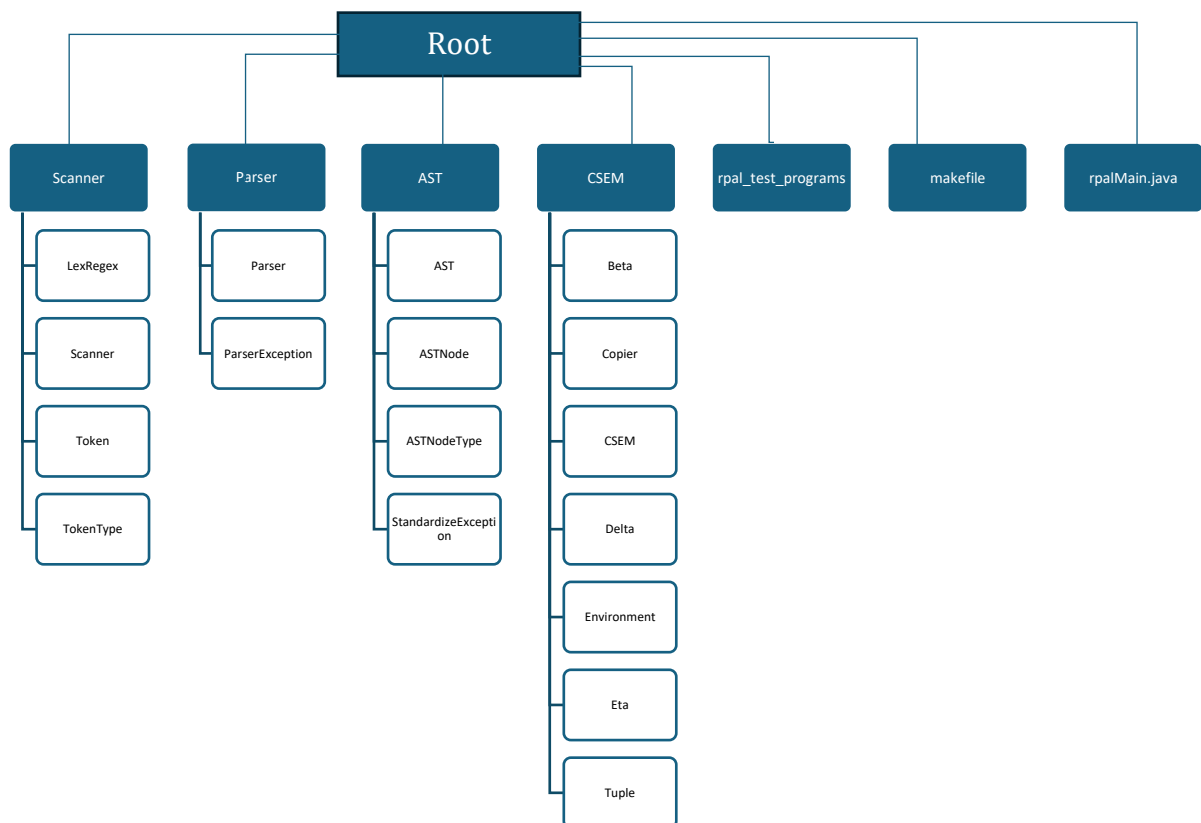
This project's objective is to develop an interpreter for RPAL(Right-reference Pedagogic Algorithmic Language). The interpreter is made up of several parts, such as lexical analyzer, parser, Abstract Syntax Tree(AST), Standardized tree(ST) and Control Stack Environment(CSE). The main goal of the project is to implement a working RPAL interpreter that can parse and run RPAL code, providing insights into the language's structure and semantics.

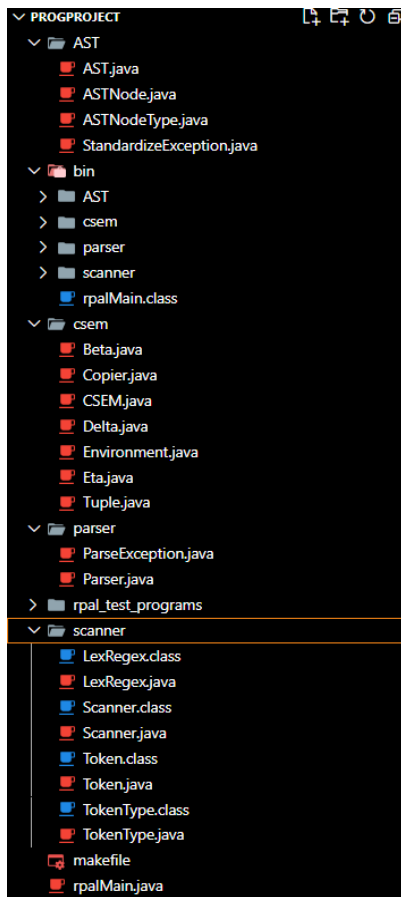
2. RPAL Language

RPAL is a pedagogic programming language designed for educational purposes and exploring fundamental concepts in programming language design and implementation. The interpreter is being built to acquire a deeper comprehension of interpreting algorithms, abstract syntax trees, and parsing. Here, the implementation is done in Java.

3. Project Structure

Img1: flowchart description





img2: visual description

4.1 Lexical Analyzer

Constructing a lexical analyzer is the first stage in implementing an RPAL interpreter. The input RPAL code is tokenized by the lexical analyzer, which divides it into meaningful components (tokens) such operators, literals, keywords, and identifiers. Four Java files namely Scanner.java, LexRegex.java, Token.java and TokenType.java have been created to implement the Lexical Analyzer.

4.2 Parser

The parser creates a parse tree using the RPAL grammar rules and the stream of tokens that the lexical analyzer produces. The RPAL grammar details, which are given in "RPAL_Grammar.pdf," establish the language's structure. The parsing process is based on recursive descent parsing, implementing the production rules of RPAL. Parser.java, ParseException.java are the Java classes which are associated with the implemented parser.

4.3 Abstract syntax Tree (AST)

The parse tree produced by the parser is converted into a more condensed and hierarchical representation during the AST generation phase. The AST maintains the structure and semantics of the program while abstracting away useless syntactic details. The Parser.java file is used to create ASTs. Through the use of the print() function included in AST.java, created AST can be printed. The AST in this implementation uses the first child - next sibling representation.

4.4 Standardized Tree (ST)

The AST is further standardized during the ST conversion process, which further streamlines the representation for simpler understanding. To make the control flow easier to understand, the procedure removes unnecessary nodes and adds the proper structures. In the AST.java, the definitions of "let," "where," "fcnform," "at," "within," "rec," "lambda," and simultaneous are standardized. CS optimization rules are used in the CSE machine to handle other types of nodes.

4.5 CSE Machine

The primary component of the RPAL interpreter, the CSE machine, is in charge of interpreting the standardized tree. To execute RPAL applications, it applies a stack-based methodology, putting dynamic scoping rules and function calls into action. The Java classes CSEM.java, Environment.java, Beta.java, Delta.java, Eta.java, Copier.java, and Tuple.java are associated with the CSE Machine implementation. CSEM.java is where all evaluations, keyword processing, and CSE Optimization rule application are completed. The assessments regarding Eta, Delta, and Beta are completed independently in the corresponding files, Beta.java, Delta.java, and Eta.java. Environment.java is where environmental functions are implemented. Copier.java is a tool for copying various nodes.

5. Input and Output

The program can be run with the following sequence.

To compile the source program:

> make all

To run the program:(here in example we have taken file as hw)

>make run file=filename or

>java -cp bin rpalMain rpal_test_programs/filename

To output the ast

>java -cp bin rpalMain rpal_test_programs/filename -ast

To output the st

>java -cp bin rpalMain rpal_test_programs/filename -st

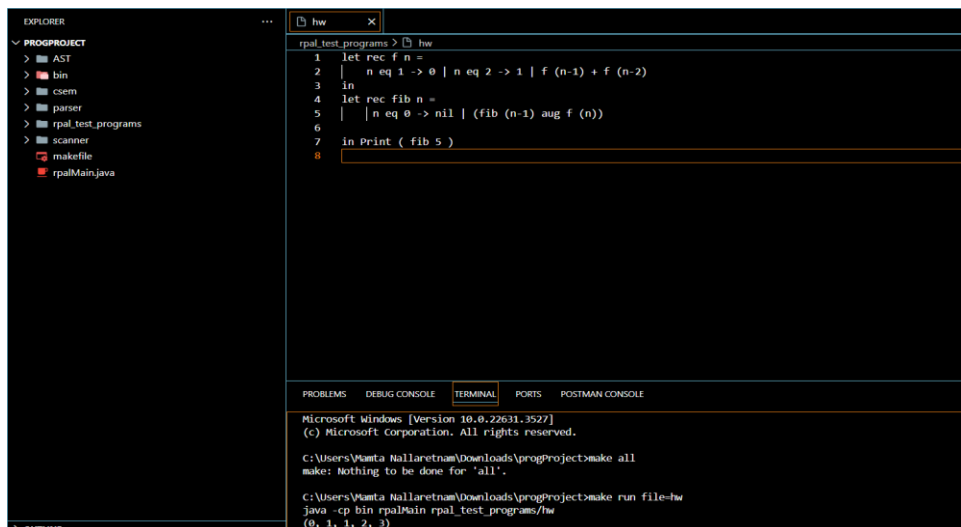
6. Testing & Validation

To ensure the correctness a sample test program is run and the snapshot of its working properly is shown here.

The sample program:

```
1  √ let rec f n =  
2  |   n eq 1 -> 0 | n eq 2 -> 1 | f (n-1) + f (n-2)  
3  in  
4  √ let rec fib n =  
5  |   | n eq 0 -> nil | (fib (n-1) aug f (n))  
6  
7  in Print ( fib 5 )
```

The output:



The screenshot shows an IDE with a dark theme. On the left is the Explorer pane showing a project structure with folders like 'bin', 'csem', 'parser', 'rpal_test_programs', 'scanner', and files like 'makefile' and 'rpalMain.java'. The main editor window displays a Haskell program file named 'hw'. The code in the editor is the same as shown in the previous block. Below the editor is a terminal pane with tabs for 'PROBLEMS', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'POSTMAN CONSOLE'. The 'TERMINAL' tab is active, showing the command prompt output. The output shows the command 'make' being run, which returns 'Nothing to be done for 'all''. Then, the command 'make run file=hw' is run, which executes 'java -cp bin rpalMain rpal_test_programs/hw' and produces the output '(0, 1, 1, 2, 3)'.

```
Microsoft Windows [Version 10.0.22631.3527]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\Wanta Nallaretnam\Downloads\progproject>make all  
make: Nothing to be done for 'all'.  
  
C:\Users\Wanta Nallaretnam\Downloads\progproject>make run file=hw  
java -cp bin rpalMain rpal_test_programs/hw  
(0, 1, 1, 2, 3)
```

AST:

```
C:\Users\Mamta Nallaretnam\Downloads\progProject>make ast file=hw
java -cp bin rpalMain rpal_test_programs/hw -ast
let
  .rec
  ..function_form
  ...<ID:f>
  ...<ID:n>
  ....>
  ....eq
  .....<ID:n>
  .....<INT:1>
  .....<INT:0>
  ....>
  ....eq
  .....<ID:n>
  .....<INT:2>
  .....<INT:1>
  .....+
  .....gamma
  .....<ID:f>
  .....-
  .....<ID:n>
  .....<INT:1>
  .....gamma
  .....<ID:f>
  .....-
  .....<ID:n>
  .....<INT:2>
  .let
  .rec
  ..function_form
  ...<ID:fib>
  ...<ID:n>
  ....>
  ....eq
  .....<ID:n>
  .....<INT:0>
  .....<nil>
  .....aug
  .....gamma
  .....<ID:fib>
  .....-
  .....<ID:n>
  .....<INT:1>
  .....gamma
  .....<ID:f>
  .....<ID:n>
  ..gamma
```

ST:

```
C:\Users\Mamta Nallaretnam\Downloads\progProject>make st file=hw
java -cp bin rpalMain rpal_test_programs/hw -st
gamma
  lambda
  ..<ID:f>
  ..gamma
  ...lambda
  ....<ID:fib>
  ....gamma
  .....<ID:Print>
  ....gamma
  .....<ID:fib>
  .....<INT:5>
  ...gamma
  ....<Y*>
  ...lambda
  ....<ID:fib>
  ...lambda
  ....<ID:n>
  ....>
  .....eq
  .....<ID:n>
  .....<INT:0>
  .....<nil>
  .....aug
  .....gamma
  .....<ID:fib>
  .....-
  .....<ID:n>
  .....<INT:1>
  .....gamma
  .....<ID:f>
  .....<ID:n>
  .gamma
  ..<Y*>
  ..lambda
  ...<ID:f>
  ...lambda
  ....<ID:n>
  ....>
  .....eq
  .....<ID:n>
  .....<INT:1>
  .....<INT:0>
  .....>
  .....eq
  .....<ID:n>
```