CS 3513: Programming Languages

Project Report

Group – 9

Gunapala S.A.C.H. **- 210190R**

Jayarathna G.V.H.H.B. **- 210242F**

Abstract:

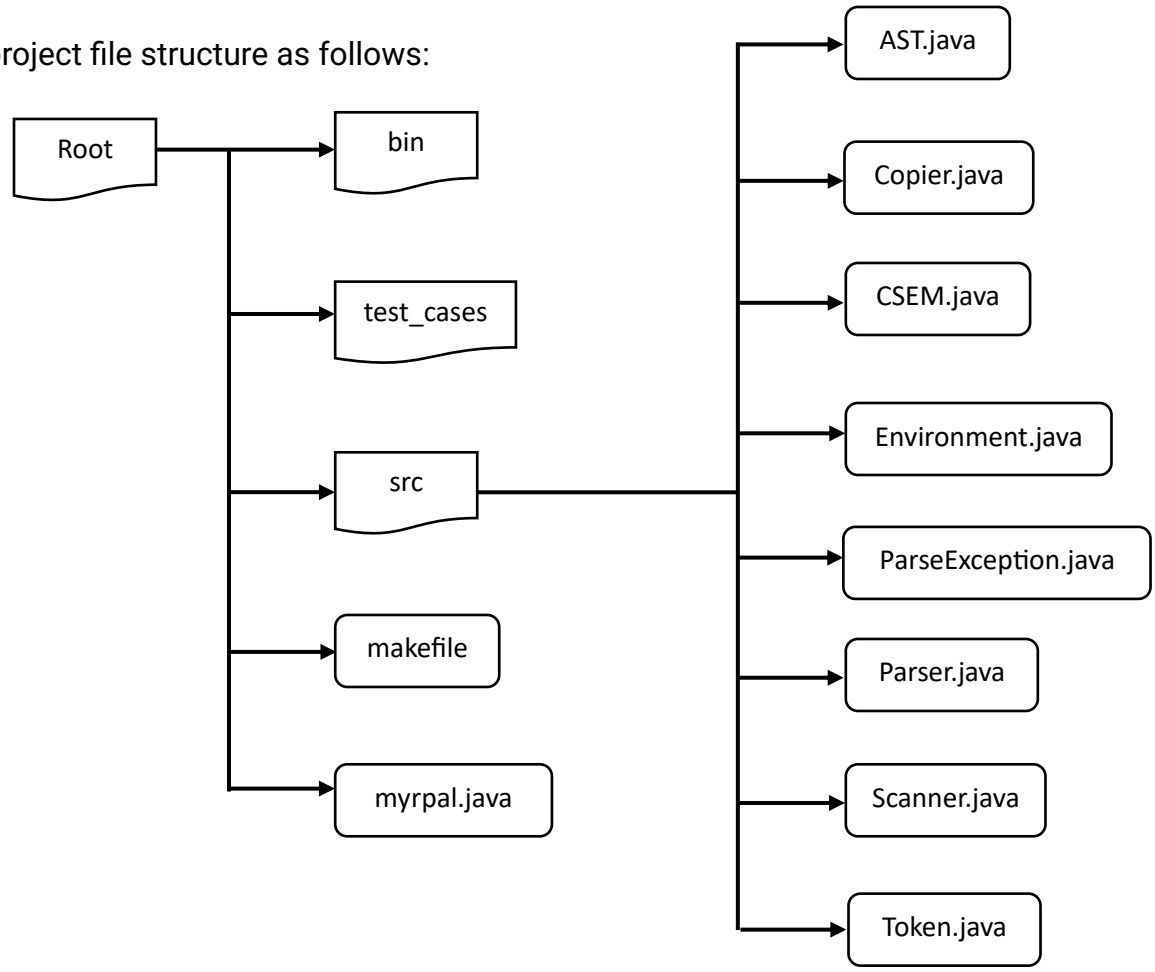
The project involves creating an interpreter for the RPAL (Right-reference Pedagogic Algorithmic Language) programming language. This interpreter will include several key components: a lexical analyzer, a parser, generation of an Abstract Syntax Tree (AST), conversion to a Standardized Tree (ST), and a Control Stack Environment (CSE) machine to execute RPAL programs. The main goal of the project is to build a working RPAL interpreter that can parse and run RPAL code, offering a deeper understanding of the language's structure and semantics.

Introduction:

RPAL is a programming language created for educational purposes, aimed at exploring fundamental concepts in programming language design and implementation. The interpreter is built to enhance understanding of parsing, abstract syntax trees, and interpretation algorithms. Java is used for its implementation.

Project Structure:

The project file structure as follows:



Parser:

The parser processes the token stream generated by the lexical analyzer and constructs a parse tree based on RPAL grammar rules outlined in "RPAL_Grammar.pdf." This grammar defines the language's structure. The parser uses recursive descent parsing to implement RPAL's production rules. The relevant Java classes for the parser implementation include **Parser.java**, **AST.java**, **Token.java**, and **ParseException.java**.

Abstract Syntax Tree (AST):

During the AST generation phase, the parse tree produced by the parser is transformed into a more compact and hierarchical representation. The AST removes unnecessary syntactic details while maintaining the program's structure and semantics. AST creation is handled within **Parser.java**, and the resulting AST can be printed using the **print()** function in **AST.java**. This implementation utilizes a first child-next sibling representation for the AST.

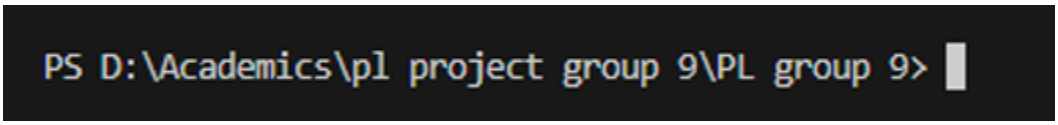
Instructions:

Main directory of the project folder is where the ‘rpal20.java’ file is located

Main directory includes the following files –

- **bin** – Output source file.
- **test_cases** – Rpal files which are to be tested using the program
- **src** – Source folder of the source code
- **myrpal.java** – Main file to run the program
- **makefile** – Make file instructions

Using command prompt/ terminal or in VSCode, should make sure to have the main directory set. For example –



Since this is our main directory of the program.

Using ‘make’ or ‘make all’ should be able to compile the program.

To run a test program,

Use the terminal and the format goes as follows –

‘make run file=(file_name)’

‘java -cp bin myrpal test_cases/(file_name)’

For example –

```
PS C:\Users\Hasitha\Downloads\PL group 9\PL group 9> make run file=5-t3
java -cp bin myrpal test_cases/5-t3
Palindrome
```

Instructions to print the Abstract Syntax Tree(AST) –

Use the command:

‘make ast file=(file_name)’

‘java -cp bin myrpal test_cases/(file_name) -ast’

For example –

```
PS C:\Users\Hasitha\Downloads\PL group 9\PL group 9> make ast file=5-t3
java -cp bin myrpal test_cases/5-t3 -ast
let
.function_form
..<ID:isPalindrome>
..<ID:x>
..where
...->
....eq
.....gamma
......<ID:getReverse>
......tau
.......<ID:x>
.......<INT:0>
.....<ID:x>
....<STR:'Palindrome'>
....<STR:'Not a palindrome'>
...rec
```

Instructions to print the Standardize Tree(ST) –

Use the command:

‘make st file=(file_name)’

‘java -cp bin myrpal test_cases/(file_name) -st‘

For example –

```
PS C:\Users\Hasitha\Downloads\PL group 9\PL group 9> make st file=5-t3
java -cp bin myrpal test_cases/5-t3 -st
gamma
.lambda
..<ID:isPalindrome>
..gamma
...<ID:Print>
...gamma
....<ID:isPalindrome>
....<INT:1001>
.lambda
..<ID:x>
..gamma
...lambda
....<ID:getReverse>
....->
.....eq
......gamma
.......<ID:getReverse>
.......tau
```

File names can be found inside the ‘test_cases’ folder.