RPAL Project Report

Name: O.L.I Athukorala, K.S.A Silva

Index Numbers: 220052G, 220612B

Function Prototypes and Program Structure

1. LexicalAnalyzer

Files: LexicalAnalyzer/LexicalAnalyser.java, Token.java, TokenEnum.java

```
public class LexicalAnalyser {
    public LexicalAnalyser(String inputFileName);
    public List<Token> scan() throws CustomException;
    public static List<Token> screener(List<Token> inputTokens);
}

public class Token {
    public Token(TokenEnum type, String value);
    public TokenEnum getType();
}

public enum TokenEnum {
    KEYWORD, IDENTIFIER, INTEGER, OPERATOR, STRING, PUNCTUATION, DELETE, HEAD
}
```

2. Parser

Files: Parser/Parser.java, Node.java, NodeEnum.java

```
public class Parser {
    public Parser(List<Token> tokens);
    public List<Node> parse();
    public ArrayList<String> convertAST_toStringAST();
}

public class Node {
    public Node(NodeEnum type, String value, int children);
}

public enum NodeEnum {
    let, fcn_form, identifier, integer, string, where, gamma, lambda, tau, rec, aug, conditional, op_or, op_
}
```

3. Standardizer

Files: Standardizer/ASTFactory.java, AST.java, NodeFactory.java, Node.java

```
public class ASTFactory {
    public ASTFactory();
    public AST getAbstractSyntaxTree(ArrayList<String> data);
}

public class AST {
    public AST(Node root);
    public void setRoot(Node root);
    public Node getRoot();
    public void standardize();
    public void printAst();
}

public class NodeFactory {
    public NodeFactory();
    public static Node getNode(String data, int depth);
    public static Node getNode(String data, int depth, Node parent, ArrayList<Node> children, boolean isStan
}

public class Node {
    public Node();
    public void setData(String data);
    public String getData();
    public int getDegree();
    public void setDepth(int depth);
    public int getDepth();
    public void setParent(Node parent);
    public Node getParent();
    public void standardize();
}
```

4. CSEMachine

Files: CSEMachine/CSEMachineFactory.java, CSEMachine.java

```
public class CSEMachineFactory {
    public CSEMachineFactory();
    public Symbol getSymbol(Node node);
    public B getB(Node node);
    public Lambda getLambda(Node node);
    public Delta getDelta(Node node);
    public ArrayList<Symbol> getControl(AST ast);
    public ArrayList<Symbol> getStack();
    public ArrayList<Environment> getEnvironment();
    public CSEMachine getCSEMachine(AST ast);
}

public class CSEMachine {
    public CSEMachine(ArrayList<Symbol> control, ArrayList<Symbol> stack, ArrayList<Environment> environment
    public void setControl(ArrayList<Symbol> control);
    public void setStack(ArrayList<Symbol> stack);
    public void setEnvironment(ArrayList<Environment> environment);
    public void execute();
    public void printControl();
    public void printStack();
    public void printEnvironment();
    public Symbol applyUnaryOperation(Symbol rator, Symbol rand);
    public Symbol applyBinaryOperation(Symbol rator, Symbol rand1, Symbol rand2);
    public String getTupleValue(Tup tup);
    public String getAnswer();
}
```

Hierarchical Call Structure

```
myrpal.main(String[] args) : void
|
└── Evaluator.evaluate(String filePath, boolean isPrintAST, boolean isPrintST) : String
    |
    ├── LexicalAnalyser.<init>(String inputFileName)
    ├── LexicalAnalyser.scan() : List<Token>
    ├── LexicalAnalyser.screener(List<Token>) : List<Token>
    ├── Parser.<init>(List<Token>)
    ├── Parser.parse() : List<Node>
    ├── Parser.convertAST_toStringAST() : ArrayList<String>
    ├── ASTFactory.<init>()
    ├── ASTFactory.getAbstractSyntaxTree(ArrayList<String>) : AST
    ├── AST.standardize() : void
    ├── AST.printAst() : void
    ├── CSEMachineFactory.<init>()
    ├── CSEMachineFactory.getCSEMachine(AST) : CSEMachine
    │   ├── CSEMachineFactory.getControl(AST) : ArrayList<Symbol>
    │   ├── CSEMachineFactory.getStack() : ArrayList<Symbol>
    │   └── CSEMachineFactory.getEnvironment() : ArrayList<Environment>
    └── CSEMachine.getAnswer() : String
        └── CSEMachine.execute() : void
```

Detailed Call Trees for Core Methods

LexicalAnalyser.scan()

```
LexicalAnalyser.scan() : List<Token>
│
├── BufferedReader(FileReader)
│   └── for each line:
│       └── LexicalAnalyser.tokenizeLine(String line) : void
│           ├── Pattern/Matcher for comments, whitespace, identifiers, integers, operators, strings, punctuation
│           └── tokens.add(new Token(...))
└── returns: List<Token>
```

Parser.parse()

```
Parser.parse() : List<Node>
│
├── tokens.add(new Token(TokenEnum.HEAD, ""))
├── Parser.E() : void
│   ├── Parser.D(), Parser.Ew(), etc. (recursive descent parsing)
│   └── AST.add(new Node(...))
├── if (tokens.get(0).type == TokenEnum.HEAD)
│   └── returns: AST (List<Node>)
└── else
    └── returns: null
```

Parser.convertAST_toStringAST()

```
Parser.convertAST_toStringAST() : ArrayList<String>
│
├── while (!AST.isEmpty())
│   ├── stack-based traversal of AST
│   └── Parser.addStrings(String dots, Node node) : void
│       └── stringAST.add(...)
└── Collections.reverse(stringAST)
└── returns: ArrayList<String>
```

ASTFactory.getAbstractSyntaxTree(ArrayList data)

```
ASTFactory.getAbstractSyntaxTree(ArrayList<String> data) : AST
│
```

```
├── NodeFactory.getNode(String, int) : Node
├── for each string in data:
│   ├── parse depth and label
│   ├── NodeFactory.getNode(...)
│   └── build tree by setting parent/children
└── returns: new AST(root)
```

AST.standardize()

```
AST.standardize() : void
│
├── if (!root.isStandardized)
│   └── Node.standardize() : void
│       └── recursively standardizes children and transforms tree structure
└── returns: void (in-place modification)
```

CSEMachineFactory.getCSEMachine(AST)

```
CSEMachineFactory.getCSEMachine(AST ast) : CSEMachine
│
├── CSEMachineFactory.getControl(AST) : ArrayList<Symbol>
│   ├── getDelta(AST.getRoot())
│   └── e0 (initial environment)
├── CSEMachineFactory.getStack() : ArrayList<Symbol>
│   └── e0
├── CSEMachineFactory.getEnvironment() : ArrayList<Environment>
│   └── e0
└── returns: new CSEMachine(control, stack, environment)
```

CSEMachine.getAnswer()

```
CSEMachine.getAnswer() : String
│
├── CSEMachine.execute() : void
│   └── while (!control.isEmpty())
│       └── manipulates stack, control, environment (core interpreter loop)
├── if (stack.get(0) instanceof Tup)
│   └── CSEMachine.getTupleValue(Tup) : String
└── else
    └── stack.get(0).getData()
└── returns: String (final result)
```

Summary

- The main entry is `myrpal.main`, which calls `Evaluator.evaluate`.
- `Evaluator.evaluate` orchestrates the entire process: lexical analysis, parsing, AST standardization, and evaluation.
- The final result is produced by the CSE machine and returned as a string, which is printed by the main method.
