Report

CS 3512: Programming Languages

Programming Project 01

Prepared by Group 22

- **200481C Prematilaka G.G.G.S.C.**
- **200614N Somarathna W.A.N.M.**
- **200629N Sumathipala H.E.S.**
- **200695K Warshamana W.I.S.N.**

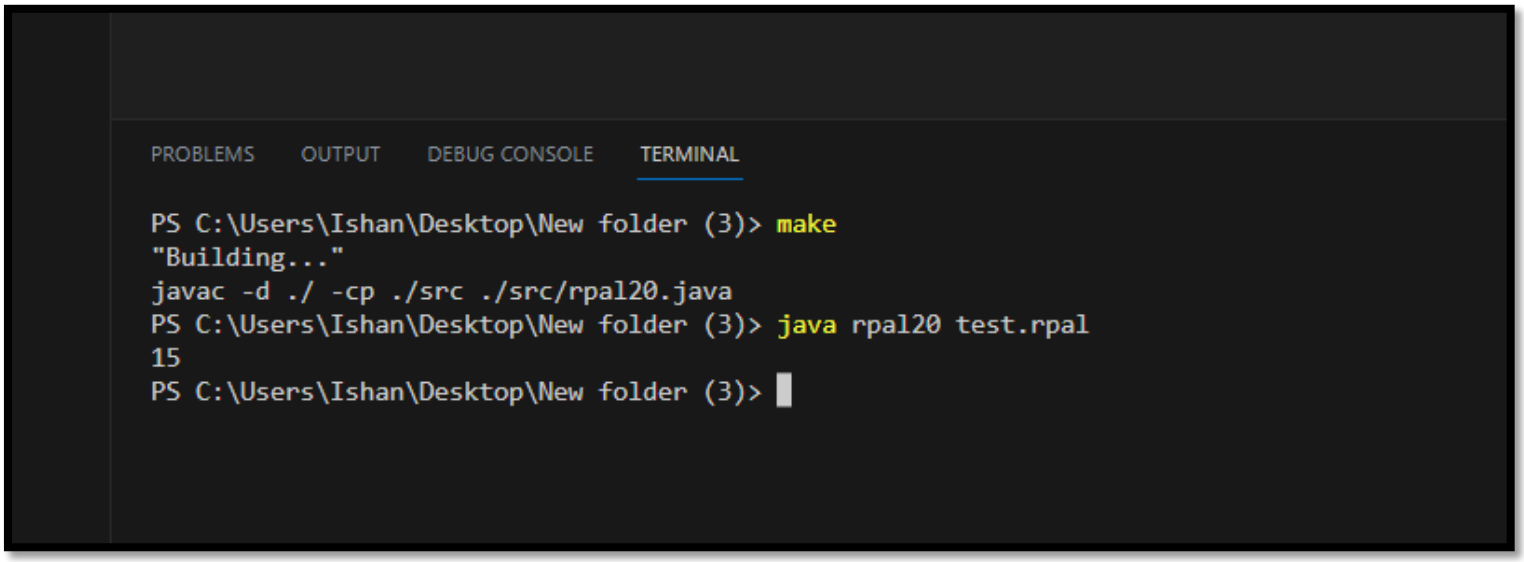
Date 2023/07/25

Table of content

Problem

You are required to implement a lexical analyzer and a parser for the RPAL language. Output of the parser should be the Abstract Syntax Tree (AST) for the given input program. Then you need to implement an algorithm to convert the Abstract Syntax Tree (AST) in to Standardize Tree (ST) and implement CSE machine. Your program should be able to read an input file which contains a RPAL program. Output of your program should match the output of “rpal.exe“ for the relevant program.

Run Programme



Solution

Lexical Analysis

- Implement a lexical analyzer (also known as a lexer or scanner) that reads the input RPAL program and converts it into a stream of tokens. Tokens are the smallest meaningful units in a programming language (e.g., keywords, identifiers, operators, etc.).
- Define rules to identify different types of tokens based on the RPAL language specifications.

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
package scanner;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;
import java.util.List;

public class Scanner {
   private BufferedReader buffer;
   private String extraCharRead;
   private final List<String> reservedIdentifiers = Arrays.asList("let", "in", "within", "fn", "where", "aug", "or", "not", "gr", "ge", "ls", "le", "eq", "ne", "true", "false", "nil", "dummy", "rec", "and");
   private int sourceLineNumber = 1;

   public Scanner(String var1) throws IOException {
      this.buffer = new BufferedReader(new InputStreamReader(new FileInputStream(new File(var1))));
   }

   public Token readNextToken() {
      Token var1 = null;
      String var2;
      if (this.extraCharRead != null) {
         var2 = this.extraCharRead;
         this.extraCharRead = null;
      } else {
         var2 = this.readNextChar();
      }

      if (var2 != null) {
         var1 = this.buildToken(var2);
      }

      return var1;
   }

   private String readNextChar() {
      String var1 = null;

      try {
         int var2 = this.buffer.read();
         if (var2 != -1) {
            var1 = Character.toString((char)var2);
            if (var1.equals("\n")) {
               ++this.sourceLineNumber;
            }
         } else {
            this.buffer.close();
         }
      } catch (IOException var3) {
      }
```

```
54
55          return var1;
56      }
57
58      private Token buildToken(String var1) {
59          Token var2 = null;
60          if (LexicalRegexPatterns.LetterPattern.matcher(var1).matches()) {
61              var2 = this.buildIdentifierToken(var1);
62          } else if (LexicalRegexPatterns.DigitPattern.matcher(var1).matches()) {
63              var2 = this.buildIntegerToken(var1);
64          } else if (LexicalRegexPatterns.OpSymbolPattern.matcher(var1).matches()) {
65              var2 = this.buildOperatorToken(var1);
66          } else if (var1.equals("'")) {
67              var2 = this.buildStringToken(var1);
68          } else if (LexicalRegexPatterns.SpacePattern.matcher(var1).matches()) {
69              var2 = this.buildSpaceToken(var1);
70          } else if (LexicalRegexPatterns.PunctuationPattern.matcher(var1).matches()) {
71              var2 = this.buildPunctuationPattern(var1);
72          }
73
74          return var2;
75      }
76
77      private Token buildIdentifierToken(String var1) {
78          Token var2 = new Token();
79          var2.setType(TokenType.IDENTIFIER);
80          var2.setSourceLineNumber(this.sourceLineNumber);
81          StringBuilder var3 = new StringBuilder(var1);
82
83          for(String var4 = this.readNextChar(); var4 != null; var4 = this.readNextChar()) {
84              if (!LexicalRegexPatterns.IdentifierPattern.matcher(var4).matches()) {
85                  this.extraCharRead = var4;
86                  break;
87              }
88
89              var3.append(var4);
90          }
91
92          String var5 = var3.toString();
93          if (this.reservedIdentifiers.contains(var5)) {
94              var2.setType(TokenType.RESERVED);
95          }
96
97          var2.setValue(var5);
98          return var2;
99      }
100
```

```
101     private Token buildIntegerToken(String var1) {
102        Token var2 = new Token();
103        var2.setType(TokenType.INTEGER);
104        var2.setSourceLineNumber(this.sourceLineNumber);
105        StringBuilder var3 = new StringBuilder(var1);
106
107        for(String var4 = this.readNextChar(); var4 != null; var4 = this.readNextChar()) {
108           if (!LexicalRegexPatterns.DigitPattern.matcher(var4).matches()) {
109              this.extraCharRead = var4;
110              break;
111           }
112
113           var3.append(var4);
114        }
115
116        var2.setValue(var3.toString());
117        return var2;
118     }
119
120     private Token buildOperatorToken(String var1) {
121        Token var2 = new Token();
122        var2.setType(TokenType.OPERATOR);
123        var2.setSourceLineNumber(this.sourceLineNumber);
124        StringBuilder var3 = new StringBuilder(var1);
125        String var4 = this.readNextChar();
126        if (var1.equals("/") && var4.equals("/")) {
127           return this.buildCommentToken(var1 + var4);
128        } else {
129           while(var4 != null) {
130              if (!LexicalRegexPatterns.OpSymbolPattern.matcher(var4).matches()) {
131                 this.extraCharRead = var4;
132                 break;
133              }
134
135              var3.append(var4);
136              var4 = this.readNextChar();
137           }
138
139           var2.setValue(var3.toString());
140           return var2;
141        }
142     }
143
144     private Token buildStringToken(String var1) {
145        Token var2 = new Token();
146        var2.setType(TokenType.STRING);
147        var2.setSourceLineNumber(this.sourceLineNumber);
148        StringBuilder var3 = new StringBuilder("");
149        String var4 = this.readNextChar();
150
```

```
151         while(var4 != null) {
152            if (var4.equals("'")) {
153               var2.setValue(var3.toString());
154               return var2;
155            }
156
157            if (LexicalRegexPatterns.StringPattern.matcher(var4).matches()) {
158               var3.append(var4);
159               var4 = this.readNextChar();
160            }
161         }
162
163         return null;
164      }
165
166      private Token buildSpaceToken(String var1) {
167         Token var2 = new Token();
168         var2.setType(TokenType.DELETE);
169         var2.setSourceLineNumber(this.sourceLineNumber);
170         StringBuilder var3 = new StringBuilder(var1);
171
172         for(String var4 = this.readNextChar(); var4 != null; var4 = this.readNextChar()) {
173            if (!LexicalRegexPatterns.SpacePattern.matcher(var4).matches()) {
174               this.extraCharRead = var4;
175               break;
176            }
177
178            var3.append(var4);
179         }
180
181         var2.setValue(var3.toString());
182         return var2;
183      }
184
185      private Token buildCommentToken(String var1) {
186         Token var2 = new Token();
187         var2.setType(TokenType.DELETE);
188         var2.setSourceLineNumber(this.sourceLineNumber);
189         StringBuilder var3 = new StringBuilder(var1);
190         String var4 = this.readNextChar();
191
192         while(var4 != null) {
193            if (LexicalRegexPatterns.CommentPattern.matcher(var4).matches()) {
194               var3.append(var4);
195               var4 = this.readNextChar();
196            } else if (var4.equals("\n")) {
197               break;
198            }
199         }
200
201         var2.setValue(var3.toString());
202         return var2;
203      }
204
205      private Token buildPunctuationPattern(String var1) {
206         Token var2 = new Token();
207         var2.setSourceLineNumber(this.sourceLineNumber);
208         var2.setValue(var1);
209         if (var1.equals("(")) {
210            var2.setType(TokenType.L_PAREN);
211         } else if (var1.equals(")")) {
212            var2.setType(TokenType.R_PAREN);
213         } else if (var1.equals(";")) {
214            var2.setType(TokenType.SEMICOLON);
215         } else if (var1.equals(",")) {
216            var2.setType(TokenType.COMMA);
217         }
218
219         return var2;
220      }
221   }
```

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
package scanner;

public class Token {
   private TokenType type;
   private String value;
   private int sourceLineNumber;

   public Token() {
   }

   public TokenType getType() {
      return this.type;
   }

   public void setType(TokenType var1) {
      this.type = var1;
   }

   public String getValue() {
      return this.value;
   }

   public void setValue(String var1) {
      this.value = var1;
   }

   public int getSourceLineNumber() {
      return this.sourceLineNumber;
   }

   public void setSourceLineNumber(int var1) {
      this.sourceLineNumber = var1;
   }
}

```

Syntax Analysis (Parsing)

- Implement a parser that takes the token stream from the lexer and constructs the Abstract Syntax Tree (AST).
- The parser should follow the grammar rules of the RPAL language to validate the syntax and build the corresponding AST nodes.
- If there is no file name, an exception is thrown.
- If the file specified in the argument exists, first Read the text file mentioned above each by line, trimming any extra spaces, then create a list for each line.
- All of the following escape sequences can be found using a function.
    - BS
    - FF
    - NL
    - CR
    - TAB
    - double quotation
    - single quote
- To determine the depth of the root, we must count the dots at the beginning of each line.
- Use the CreateTree class to build a tree in accordance with the lines that were read.

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
package parser;

import ast.AST;
import ast.ASTNode;
import ast.ASTNodeType;
import java.util.Stack;
import scanner.Scanner;
import scanner.Token;
import scanner.TokenType;

public class Parser {
   private Scanner s;
   private Token currentToken;
   Stack<ASTNode> stack;

   public Parser(Scanner var1) {
      this.s = var1;
      this.stack = new Stack();
   }

   public AST buildAST() {
      this.startParse();
      return new AST((ASTNode)this.stack.pop());
   }

   public void startParse() {
      this.readNT();
      this.procE();
      if (this.currentToken != null) {
         throw new ParseException("Expected EOF.");
      }
   }

   private void readNT() {
      do {
         this.currentToken = this.s.readNextToken();
      } while(this.isCurrentTokenType(TokenType.DELETE));

      if (null != this.currentToken) {
         if (this.currentToken.getType() == TokenType.IDENTIFIER) {
            this.createTerminalASTNode(ASTNodeType.IDENTIFIER, this.currentToken.getValue());
         } else if (this.currentToken.getType() == TokenType.INTEGER) {
            this.createTerminalASTNode(ASTNodeType.INTEGER, this.currentToken.getValue());
         } else if (this.currentToken.getType() == TokenType.STRING) {
            this.createTerminalASTNode(ASTNodeType.STRING, this.currentToken.getValue());
         }
      }

   }

   private boolean isCurrentToken(TokenType var1, String var2) {
      if (this.currentToken == null) {
         return false;
      } else {
         return this.currentToken.getType() == var1 && this.currentToken.getValue().equals(var2);
      }
   }

```

```
60      private boolean isCurrentTokenType(TokenType var1) {
61         if (this.currentToken == null) {
62            return false;
63         } else {
64            return this.currentToken.getType() == var1;
65         }
66      }
67
68      private void buildNAryASTNode(ASTNodeType var1, int var2) {
69         ASTNode var3 = new ASTNode();
70         var3.setType(var1);
71
72         while(var2 > 0) {
73            ASTNode var4 = (ASTNode)this.stack.pop();
74            if (var3.getChild() != null) {
75               var4.setSibling(var3.getChild());
76            }
77
78            var3.setChild(var4);
79            var3.setSourceLineNumber(var4.getSourceLineNumber());
80            --var2;
81         }
82
83         this.stack.push(var3);
84      }
85
86      private void createTerminalASTNode(ASTNodeType var1, String var2) {
87         ASTNode var3 = new ASTNode();
88         var3.setType(var1);
89         var3.setValue(var2);
90         var3.setSourceLineNumber(this.currentToken.getSourceLineNumber());
91         this.stack.push(var3);
92      }
93
94      private void procE() {
95         if (this.isCurrentToken(TokenType.RESERVED, "let")) {
96            this.readNT();
97            this.procD();
98            if (!this.isCurrentToken(TokenType.RESERVED, "in")) {
99               throw new ParseException("E:  'in' expected");
100           }
101
102           this.readNT();
103           this.procE();
104           this.buildNAryASTNode(ASTNodeType.LET, 2);
105        } else if (this.isCurrentToken(TokenType.RESERVED, "fn")) {
106           int var1 = 0;
107           this.readNT();
108
109           while(this.isCurrentTokenType(TokenType.IDENTIFIER) || this.isCurrentTokenType(TokenType.L_PAREN)) {
110              this.procVB();
111              ++var1;
112           }
113
114           if (var1 == 0) {
115              throw new ParseException("E: at least one 'Vb' expected");
116           }
117
118           if (!this.isCurrentToken(TokenType.OPERATOR, ".")) {
119              throw new ParseException("E: '.' expected");
120           }
```

```
121
122             this.readNT();
123             this.procE();
124             this.buildNAryASTNode(ASTNodeType.LAMBDA, var1 + 1);
125          } else {
126             this.procEW();
127          }
128
129       }
130
131       private void procEW() {
132          this.procT();
133          if (this.isCurrentToken(TokenType.RESERVED, "where")) {
134             this.readNT();
135             this.procDR();
136             this.buildNAryASTNode(ASTNodeType.WHERE, 2);
137          }
138
139       }
140
141       private void procT() {
142          this.procTA();
143
144          int var1;
145          for(var1 = 0; this.isCurrentToken(TokenType.OPERATOR, ","); ++var1) {
146             this.readNT();
147             this.procTA();
148          }
149
150          if (var1 > 0) {
151             this.buildNAryASTNode(ASTNodeType.TAU, var1 + 1);
152          }
153
154       }
155
156       private void procTA() {
157          this.procTC();
158
159          while(this.isCurrentToken(TokenType.RESERVED, "aug")) {
160             this.readNT();
161             this.procTC();
162             this.buildNAryASTNode(ASTNodeType.AUG, 2);
163          }
164
165       }
166
167       private void procTC() {
168          this.procB();
169          if (this.isCurrentToken(TokenType.OPERATOR, "->")) {
170             this.readNT();
171             this.procTC();
172             if (!this.isCurrentToken(TokenType.OPERATOR, "|")) {
173                throw new ParseException("TC: '|' expected");
174             }
175
176             this.readNT();
177             this.procTC();
178             this.buildNAryASTNode(ASTNodeType.CONDITIONAL, 3);
179          }
180
181       }
```

```
182
183     private void procB() {
184         this.procBT();
185
186         while(this.isCurrentToken(TokenType.RESERVED, "or")) {
187             this.readNT();
188             this.procBT();
189             this.buildNAryASTNode(ASTNodeType.OR, 2);
190         }
191
192     }
193
194     private void procBT() {
195         this.procBS();
196
197         while(this.isCurrentToken(TokenType.OPERATOR, "&")) {
198             this.readNT();
199             this.procBS();
200             this.buildNAryASTNode(ASTNodeType.AND, 2);
201         }
202
203     }
204
205     private void procBS() {
206         if (this.isCurrentToken(TokenType.RESERVED, "not")) {
207             this.readNT();
208             this.procBP();
209             this.buildNAryASTNode(ASTNodeType.NOT, 1);
210         } else {
211             this.procBP();
212         }
213
214     }
215
216     private void procBP() {
217         this.procA();
218         if (!this.isCurrentToken(TokenType.RESERVED, "gr") && !this.isCurrentToken(TokenType.OPERATOR, ">")) {
219             if (!this.isCurrentToken(TokenType.RESERVED, "ge") && !this.isCurrentToken(TokenType.OPERATOR, ">=")) {
220                 if (!this.isCurrentToken(TokenType.RESERVED, "ls") && !this.isCurrentToken(TokenType.OPERATOR, "<")) {
221                     if (!this.isCurrentToken(TokenType.RESERVED, "le") && !this.isCurrentToken(TokenType.OPERATOR, "<=")) {
222                         if (this.isCurrentToken(TokenType.RESERVED, "eq")) {
223                             this.readNT();
224                             this.procA();
225                             this.buildNAryASTNode(ASTNodeType.EQ, 2);
226                         } else if (this.isCurrentToken(TokenType.RESERVED, "ne")) {
227                             this.readNT();
228                             this.procA();
229                             this.buildNAryASTNode(ASTNodeType.NE, 2);
230                         }
231                     } else {
232                         this.readNT();
233                         this.procA();
234                         this.buildNAryASTNode(ASTNodeType.LE, 2);
235                     }
236                 } else {
237                     this.readNT();
238                     this.procA();
239                     this.buildNAryASTNode(ASTNodeType.LS, 2);
240                 }
241             } else {
242                 this.readNT();
243                 this.procA();
244                 this.buildNAryASTNode(ASTNodeType.GE, 2);
245             }
246         } else {
```

```
247             this.readNT();
248             this.procA();
249             this.buildNAryASTNode(ASTNodeType.GR, 2);
250          }
251
252       }
253
254       private void procA() {
255          if (this.isCurrentToken(TokenType.OPERATOR, "+")) {
256             this.readNT();
257             this.procAT();
258          } else if (this.isCurrentToken(TokenType.OPERATOR, "-")) {
259             this.readNT();
260             this.procAT();
261             this.buildNAryASTNode(ASTNodeType.NEG, 1);
262          } else {
263             this.procAT();
264          }
265
266          boolean var1 = true;
267
268          while(this.isCurrentToken(TokenType.OPERATOR, "+") || this.isCurrentToken(TokenType.OPERATOR, "-")) {
269             if (this.currentToken.getValue().equals("+")) {
270                var1 = true;
271             } else if (this.currentToken.getValue().equals("-")) {
272                var1 = false;
273             }
274
275             this.readNT();
276             this.procAT();
277             if (var1) {
278                this.buildNAryASTNode(ASTNodeType.PLUS, 2);
279             } else {
280                this.buildNAryASTNode(ASTNodeType.MINUS, 2);
281             }
282          }
283
284       }
285
286       private void procAT() {
287          this.procAF();
288          boolean var1 = true;
289
290          while(this.isCurrentToken(TokenType.OPERATOR, "*") || this.isCurrentToken(TokenType.OPERATOR, "/")) {
291             if (this.currentToken.getValue().equals("*")) {
292                var1 = true;
293             } else if (this.currentToken.getValue().equals("/")) {
294                var1 = false;
295             }
296
297             this.readNT();
298             this.procAF();
299             if (var1) {
300                this.buildNAryASTNode(ASTNodeType.MULT, 2);
301             } else {
302                this.buildNAryASTNode(ASTNodeType.DIV, 2);
303             }
304          }
305
306       }
```

```
307
308     private void procAF() {
309        this.procAP();
310        if (this.isCurrentToken(TokenType.OPERATOR, "**")) {
311           this.readNT();
312           this.procAF();
313           this.buildNAryASTNode(ASTNodeType.EXP, 2);
314        }
315
316     }
317
318     private void procAP() {
319        this.procR();
320
321        while(this.isCurrentToken(TokenType.OPERATOR, "@")) {
322           this.readNT();
323           if (!this.isCurrentTokenType(TokenType.IDENTIFIER)) {
324              throw new ParseException("AP: expected Identifier");
325           }
326
327           this.readNT();
328           this.procR();
329           this.buildNAryASTNode(ASTNodeType.AT, 3);
330        }
331
332     }
333
334     private void procR() {
335        this.procRN();
336        this.readNT();
337
338        while(this.isCurrentTokenType(TokenType.INTEGER) || this.isCurrentTokenType(TokenType.STRING) || this.isCurrentTokenType(TokenType.IDENTIFIER) || this.isCurrentToken(TokenType.RESERVED, "true")
339           this.procRN();
340           this.buildNAryASTNode(ASTNodeType.GAMMA, 2);
341           this.readNT();
342        }
343
344     }
345
346     private void procRN() {
347        if (!this.isCurrentTokenType(TokenType.IDENTIFIER) && !this.isCurrentTokenType(TokenType.INTEGER) && !this.isCurrentTokenType(TokenType.STRING)) {
348           if (this.isCurrentToken(TokenType.RESERVED, "true")) {
349              this.createTerminalASTNode(ASTNodeType.TRUE, "true");
350           } else if (this.isCurrentToken(TokenType.RESERVED, "false")) {
351              this.createTerminalASTNode(ASTNodeType.FALSE, "false");
352           } else if (this.isCurrentToken(TokenType.RESERVED, "nil")) {
353              this.createTerminalASTNode(ASTNodeType.NIL, "nil");
354           } else if (this.isCurrentTokenType(TokenType.L_PAREN)) {
355              this.readNT();
356              this.procE();
357              if (!this.isCurrentTokenType(TokenType.R_PAREN)) {
358                 throw new ParseException("RN: ')' expected");
359              }
360           } else if (this.isCurrentToken(TokenType.RESERVED, "dummy")) {
361              this.createTerminalASTNode(ASTNodeType.DUMMY, "dummy");
362           }
363        }
364
365     }
```

```
366
367     private void procD() {
368        this.procDA();
369        if (this.isCurrentToken(TokenType.RESERVED, "within")) {
370           this.readNT();
371           this.procD();
372           this.buildNAryASTNode(ASTNodeType.WITHIN, 2);
373        }
374
375     }
376
377     private void procDA() {
378        this.procDR();
379
380        int var1;
381        for(var1 = 0; this.isCurrentToken(TokenType.RESERVED, "and"); ++var1) {
382           this.readNT();
383           this.procDR();
384        }
385
386        if (var1 > 0) {
387           this.buildNAryASTNode(ASTNodeType.SIMULTDEF, var1 + 1);
388        }
389
390     }
391
392     private void procDR() {
393        if (this.isCurrentToken(TokenType.RESERVED, "rec")) {
394           this.readNT();
395           this.procDB();
396           this.buildNAryASTNode(ASTNodeType.REC, 1);
397        } else {
398           this.procDB();
399        }
400
401     }
402
403     private void procDB() {
404        if (this.isCurrentTokenType(TokenType.L_PAREN)) {
405           this.procD();
406           this.readNT();
407           if (!this.isCurrentTokenType(TokenType.R_PAREN)) {
408              throw new ParseException("DB: ')' expected");
409           }
410
411           this.readNT();
412        } else if (this.isCurrentTokenType(TokenType.IDENTIFIER)) {
413           this.readNT();
414           if (this.isCurrentToken(TokenType.OPERATOR, ",")) {
415              this.readNT();
416              this.procVL();
417              if (!this.isCurrentToken(TokenType.OPERATOR, "=")) {
418                 throw new ParseException("DB: = expected.");
419              }
420
421              this.buildNAryASTNode(ASTNodeType.COMMA, 2);
422              this.readNT();
423              this.procE();
424              this.buildNAryASTNode(ASTNodeType.EQUAL, 2);
425           } else if (this.isCurrentToken(TokenType.OPERATOR, "=")) {
426              this.readNT();
427              this.procE();
```

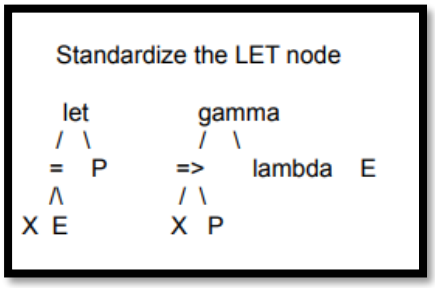
Create the ST to the provided AST using a node-based standardization tree

- Implementing the tree node is necessary for creating a ST or AST tree. Create a node class containing the parent, children, value, and label for that job.
- There are functions to clone a node, add a child, obtain a child, see if there are any children, clear the node, and add a child to it.

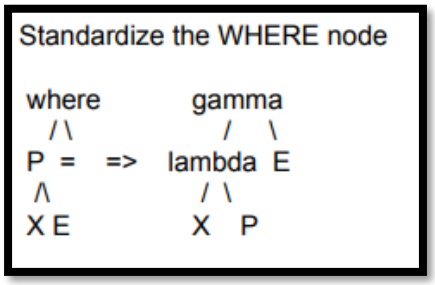
```java
public class Node {
    /**
     * Represents a node in the ast and st.
     *main two types they are child and parent
     * Label represents the type of Node.
     * EleValue represents the value of node
     */
    private final ArrayList<Node> children;
    private Node parent;
    private String label;
    private String value;
    /**
     *node with one argument called label
     */
    public Node(String label) {
        this.label = label;
        this.children = new ArrayList<>();
    }

    /**
     * node with two argument both lable and value
     */
    public Node(String label, String value) {
        this.label = label;
        this.value = value;
        this.children = new ArrayList<>();
    }
    Node copy() {
        Node copied = new Node(label, value);
        for (int i = 0; i < getNumChild(); i++) {
            copied.addChild(getChild(i).copy());
        }
        return copied;
    }
    //get parent node
    Node getParent() {return parent;    }
    public String getLabel() {
        return label;
    }
    public String getValue() {
        return value;
    }
    public int getNumChild() {
        return children.size();
    }
    boolean hasChildren(int n) {
        return children.size() == n;
    }
    public boolean isLabel(String label) {
        return getLabel().equals(label);
    }
    public Node getChild(int i) {
        return children.get(i);
    }
    public void forEachChild(Consumer<? super Node> action) {
        children.forEach(action);
    }
    void setLabel(String label) {this.label = label;this.value = null;    }
    void clearChildren() {children.forEach(child -> child.parent = null);children.clear();}
    void addChild(Node child) {children.add(child);child.parent = this;}
}
```

Using the AST to ST class, the raw AST from the text file is transformed into the ST. The converters for the AST node labels listed below need first be implemented.

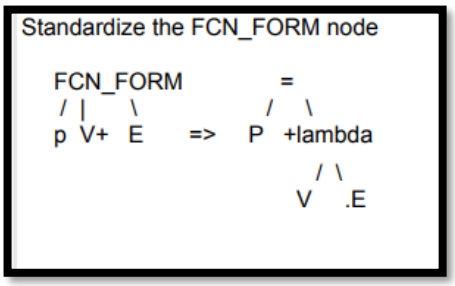
Let



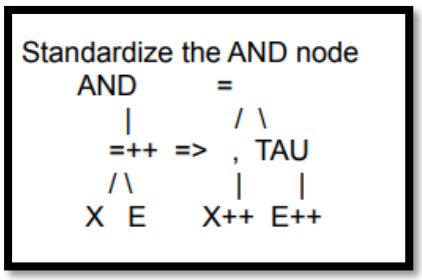
Where



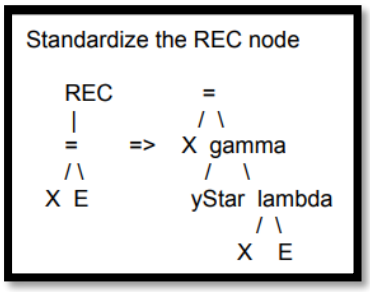
Function_form



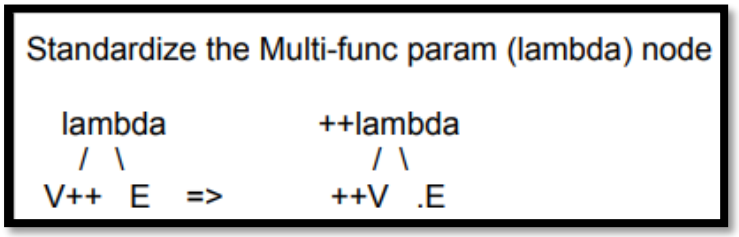
And



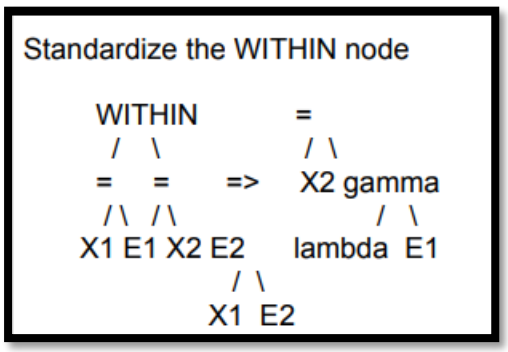
Rec



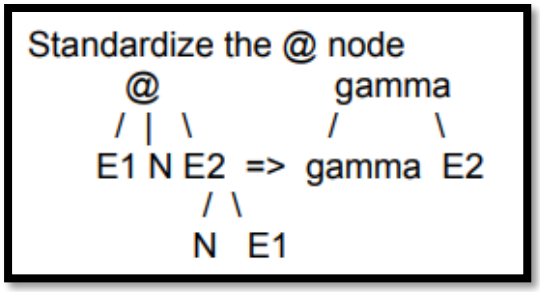
Lambda



Within



At



Operation Handling

- Three interfaces in the OperationHandler class distinguish boolean, arithmetic, and array type operations. This class includes implementations of every form of tree implementation.

Make a stack with all the components

- Create the next stack class to store the components.

```java
public class Stack<T extends EleValueOrTuple> implements Iterable<T> {
    protected final java.util.Stack<T> stack;

    Stack() {
        stack = new java.util.Stack<>();
    }

    void push(T element) {
        stack.push(element);
    }

    T pop() {
        return stack.pop();
    }

    boolean isEmpty() {
        return stack.isEmpty();
    }

    int size() {
        return stack.size();
    }

    @Override
    public String toString() {
        return stack.toString();
    }

    @Override
    public Iterator<T> iterator() {
        return stack.iterator();
    }
}
```

- Implement operations in the operation handler class

Generate the CSE machine for created ST

```java
package cse;

import java.util.ArrayList;
import Interpreter.OperationHandler;
import cse.ele.EleTuple;
import cse.ele.EleValue;
import cse.ele.EleValueOrTuple;

/**
 * CSE machine to evaluate the traversed tree
 */
public class CSEMachine {
    private final Stack<EleValue> eleValues;
    private final Stack<EleValueOrTuple> eleValueOrTuples;
    private final OperationHandler operationHandler;
    private final ArrayList<Environment> environments;
    private final ArrayList<Stack<EleValue>> CS;

    public CSEMachine(ArrayList<Stack<EleValue>> controlStructures) {
        this.CS = controlStructures;
        this.eleValueOrTuples = new Stack<>();
        this.operationHandler = new OperationHandler();

        eleValues = new Stack<>();
        eleValues.push(new EleValue("environment", "0"));
        eleValues.push(new EleValue("delta", "0"));
        eleValueOrTuples.push(new EleValue("environment", "0"));
        environments = new ArrayList<>();
        environments.add(new Environment());
    }

    @Override
    public String toString() {
        return eleValues + "\n" + eleValueOrTuples + "\n" + currentEnvironment() + "\n";
    }

    private int currentEnvironmentIndex() {
        int closestEnvironment = 0;
        for (EleValueOrTuple element : eleValues) {
            if (element instanceof EleValue && element.isLabel("environment")) {
                String closestEnvironmentStr = ((EleValue) element).getValue();
                closestEnvironment = Integer.parseInt(closestEnvironmentStr);
            }
        }
        return closestEnvironment;
    }

    /**
     * Get current environment
     */
```

```
51      private Environment currentEnvironment() {
52          return environments.get(currentEnvironmentIndex());
53      }
54
55      /**
56       * Start processing the control stack to evaluate result.
57       */
58      public void evaluateTree() {
59          while (!eleValues.isEmpty()) {
60              EleValue currentElement = eleValues.pop();
61
62              if (currentElement.isLabel("gamma")) {
63                  EleValueOrTuple firstElem = eleValueOrTuples.pop();
64                  EleValueOrTuple secondElem = eleValueOrTuples.pop();
65                  if (firstElem.isLabel("yStar")) {
66                      Rule12(secondElem);
67                  } else if (firstElem.isLabel("eta")) {
68                      eleValueOrTuples.push(secondElem);
69                      Rule13(currentElement, firstElem);
70                  } else if (firstElem.isLabel("lambda")) {
71                      EleValue firstValue = (EleValue) firstElem;
72                      if (firstValue.getValue().contains(",")) {
73                          Rule11(firstElem, secondElem);
74                      } else {
75                          Rule4(firstElem, secondElem);
76                      }
77                  } else if (firstElem.isLabel("tau")) {
78                      Rule10(firstElem, secondElem);
79                  } else {
80                      Rule3(firstElem, secondElem);
81                  }
82              } else if (currentElement.isLabel("delta")) {
83                  int controlIndex = Integer.parseInt(currentElement.getValue());
84                  extractDelta(controlIndex);
85              } else if (currentElement.isLabel("id")) {
86                  Rule1(currentElement);
87              } else if (currentElement.isLabel("lambda")) {
88                  Rule2(currentElement);
89              } else if (currentElement.isLabel("environment")) {
90                  Rule5(currentElement);
91              } else if (currentElement.isLabel("beta")) {
92                  Rule8();
93              } else if (currentElement.isLabel("tau")) {
94                  Rule9(currentElement);
95              } else if (!Rule6_7(currentElement)) {
96                  eleValueOrTuples.push(currentElement);
97              }
98          }
99      }
100
```

```
101     private void extractDelta(int controlIndex) {
102         Stack<EleValue> control = CS.get(controlIndex);
103         for (EleValue controlElem : control) {
104             this.eleValues.push(controlElem);
105         }
106     }
107 
108     // RULE 1
109     private void Rule1(EleValue name) {
110         String id = name.getValue();
111         EleValueOrTuple value = currentEnvironment().lookup(id);
112         if (value == null) {
113             value = new EleValue(id);
114         }
115         eleValueOrTuples.push(value);
116     }
117 
118     // RULE 2
119     private void Rule2(EleValue lambda) {
120         String[] kAndX = lambda.getValue().split(" ");
121         String c = Integer.toString(currentEnvironmentIndex());
122         String[] newValues = { kAndX[0], kAndX[1], c };
123         EleValueOrTuple newLambda = new EleValue("lambda", String.join(" ", newValues));
124         eleValueOrTuples.push(newLambda);
125     }
126 
127     // RULE 3
128     private void Rule3(EleValueOrTuple rator, EleValueOrTuple rand) {
129         EleValueOrTuple result = operationHandler.apply(rator, rand);
130         eleValueOrTuples.push(result);
131     }
132 
133     // RULE 4
134     private void Rule4(EleValueOrTuple lambda, EleValueOrTuple rand) {
135         if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
136             String[] kAndXAndC = ((EleValue) lambda).getValue().split(" ");
137             String k = kAndXAndC[0];
138             String x = kAndXAndC[1];
139             String c = kAndXAndC[2];
140             Environment envC = environments.get(Integer.parseInt(c));
141 
142             Environment newEnvironment = new Environment(envC, x, rand);
143             String newEnvIndex = Integer.toString(environments.size());
144             environments.add(newEnvironment);
145             eleValues.push(new EleValue("environment", newEnvIndex));
146             eleValues.push(new EleValue("delta", k));
147             eleValueOrTuples.push(new EleValue("environment", newEnvIndex));
148             return;
149         }
150         throw new ExceptionHandlerOfCSE("Expected lambda element but found: " + lambda);
```

```
151     }
152
153     // RULE 5
154     private void Rule5(EleValue env) {
155         EleValueOrTuple value = eleValueOrTuples.pop();
156         EleValueOrTuple envS = eleValueOrTuples.pop();
157         if (envS instanceof EleValue && envS.isLabel("environment")) {
158             if (env.equals(envS)) {
159                 eleValueOrTuples.push(value);
160                 return;
161             }
162             throw new ExceptionHandlerOfCSE(String.format("Environment element mismatch: %s and %s", env, envS));
163         }
164         throw new ExceptionHandlerOfCSE("Expected environment element but found: " + envS);
165     }
166
167     // RULE 7
168     private boolean Rule6_7(EleValue element) {
169         if (operationHandler.checkMathematicalOperation(element)) {
170             EleValueOrTuple rator = eleValueOrTuples.pop();
171             EleValueOrTuple rand = eleValueOrTuples.pop();
172             EleValueOrTuple result = operationHandler.applyOperations(element, rator, rand);
173             eleValueOrTuples.push(result);
174         } else if (operationHandler.checkArrayOperation(element)) {
175             EleValueOrTuple rand = eleValueOrTuples.pop();
176             EleValueOrTuple result = operationHandler.apply(element, rand);
177             eleValueOrTuples.push(result);
178         } else {
179             return false;
180         }
181         return true;
182     }
183
184     // RULE 8
185     private void Rule8() {
186         EleValue deltaElse = eleValues.pop();
187         EleValue deltaThen = eleValues.pop();
188         EleValueOrTuple condition = eleValueOrTuples.pop();
189
190         if (deltaElse.isLabel("delta") && deltaThen.isLabel("delta")) {
191             if (condition.isLabel("true")) {
192                 eleValues.push(deltaThen);
193                 return;
194             } else if (condition.isLabel("false")) {
195                 eleValues.push(deltaElse);
196                 return;
197             }
198             throw new RuntimeException("If condition must evaluate to a truth value.");
199         }
200         throw new ExceptionHandlerOfCSE("Expected delta elements.");
201     }
```

```
202
203     // RULE 9
204     private void Rule9(EleValue tau) {
205         int elements = Integer.parseInt(tau.getValue());
206         EleValueOrTuple[] tupleElements = new EleValueOrTuple[elements];
207         for (int i = 0; i < elements; i++) {
208             tupleElements[i] = eleValueOrTuples.pop();
209         }
210         EleValueOrTuple tuple = new EleTuple(tupleElements);
211         eleValueOrTuples.push(tuple);
212     }
213
214     // RULE 10
215     private void Rule10(EleValueOrTuple tuple, EleValueOrTuple index) {
216         if (tuple instanceof EleTuple) {
217             if (index instanceof EleValue && index.isLabel("int")) {
218                 int ind = Integer.parseInt(((EleValue) index).getValue());
219                 EleValueOrTuple value = ((EleTuple) tuple).getValue()[ind];
220                 eleValueOrTuples.push(value);
221                 return;
222             }
223             throw new ExceptionHandlerOfCSE("Expected integer index but found: " + index);
224         }
225         throw new ExceptionHandlerOfCSE("Expected tuple but found: " + tuple);
226     }
227
228     // RULE 11
229     private void Rule11(EleValueOrTuple lambda, EleValueOrTuple rand) {
230         if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
231             if (rand instanceof EleTuple) {
232                 String[] kAndVAndC = ((EleValue) lambda).getValue().split(" ");
233                 String k = kAndVAndC[0];
234                 String[] v = kAndVAndC[1].split(",");
235                 String c = kAndVAndC[2];
236                 Environment envC = environments.get(Integer.parseInt(c));
237
238                 Environment newEnvironment = new Environment(envC);
239                 for (int i = 0; i < v.length; i++) {
240                     newEnvironment.remember(v[i], ((EleTuple) rand).getValue()[i]);
241                 }
242                 String newEnvIndex = Integer.toString(environments.size());
243                 environments.add(newEnvironment);
244                 eleValues.push(new EleValue("environment", newEnvIndex));
245                 eleValues.push(new EleValue("delta", k));
246                 eleValueOrTuples.push(new EleValue("environment", newEnvIndex));
247                 return;
248             }
249             throw new ExceptionHandlerOfCSE("Expected tuple but found: " + rand);
250         }
251         throw new ExceptionHandlerOfCSE("Expected lambda element but found: " + lambda);
252     }
253
254     // RULE 12
255     private void Rule12(EleValueOrTuple lambda) {
256         if (lambda instanceof EleValue && lambda.isLabel("lambda")) {
257             String iAndVAndC = ((EleValue) lambda).getValue();
258             EleValueOrTuple etaElement = new EleValue("eta", iAndVAndC);
259             eleValueOrTuples.push(etaElement);
260             return;
261         }
262         throw new ExceptionHandlerOfCSE("Expected lambda element but found: " + lambda);
263     }
264
265     // RULE 13
266     private void Rule13(EleValue gamma, EleValueOrTuple eta) {
267         if (eta instanceof EleValue && eta.isLabel("eta")) {
268             String iAndVAndC = ((EleValue) eta).getValue();
269             EleValue lambda = new EleValue("lambda", iAndVAndC);
270             EleValue newGamma = new EleValue("gamma");
271
272             eleValueOrTuples.push(eta);
273             eleValueOrTuples.push(lambda);
274
275             eleValues.push(gamma);
276             eleValues.push(newGamma);
277             return;
278         }
279         throw new ExceptionHandlerOfCSE("Expected eta element but found: " + eta);
280     }
281 }
282
```

By using the executeTree function, we can obtain the definitive response to the AST

Exception handling

- The command line parameter is checked first; if there are any invalid arguments, an exception is thrown.
- The exceptionHandlerOfCSE handles any exception that arises during the construction and evaluation of the CSE machine.
- The exceptionHandlerOfAST handles any exception that arises when producing an AST and ST.

References

[1] Wikipedia contributors, “Abstract syntax tree,” Wikipedia, The Free Encyclopedia, 10-Aug-2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Abstract_syntax_tree&oldid=1103 626323.

[2] “Abstract syntax tree (AST) in java,” GeeksforGeeks, 11-Aug-2021. [Online]. Available: https://www.geeksforgeeks.org/abstract-syntax-tree-ast-in-java/. [Accessed: 08-Dec-2022].

[3] “What is Syntax Tree?,” Tutorialspoint.com. [Online]. Available: https://www.tutorialspoint.com/what-is-syntax-tree. [Accessed: 08-Dec-2022].