CS 3512 - Programming Languages

Programming Project 01 - REPORT

Name: ISHAD M. I. M.

Student ID: 190241X

Introduction

The project is to build an interpreter to evaluate RPAL programs from their Abstract Syntax Tree. For evaluate programs it standardizes the abstract syntax tree and use 13 rules of Control Stack Environment Machine.

Implementation is done using java version 18.

Executing the Program

Intellij IDE used for implementation. So, the configuration for run with intellij build system is in `.idea` folder.

- **Building with Makefile**

To compile the program run `make.` Then, it will compile the code in `src` folder to the root directory following the same structure in the src folder structure.

- **Executing the program**

To execute the program run

```
Java myrpal <file-name>
```

`<file-name>` needs to be replaced with the path to the file which contains the AST tree.

Project Structure

Following are the main directories of the project.

- `.idea` – contain the intellij IDE configurations.
- `samples` – contain sample RPAL programs used to test the implementation.
- `src` – contain java files and packages of the project.

In the project two independent tasks are identified. So, decomposed the java classes in to two packages

- `ast` - Include classes needed for parsing AST tree from the provided file and standardizing it.
    - ast.nodes – sub package of ast package include the classes to create nodes.
- `csemachine` - Include classes to evaluate the standardized tree.
    - csemachine.elements – sub package of cse machine to represent elements in the control, stack, environment of the CSE Machine.

Classes and Significant Functions

- myrpal.java

Contains the main function for the program. Accept filename from the terminal and use ast package to build the tree and standardize it. Then, use csemachine package to evaluate the standardized tree.

Classes in ast package

- ASTParser.java

```
public static ASTTree parse(String filename)
```

read the file and parse line by line to create an AST tree. Make use of NodeFactory class to create nodes and connection created among them according to number of periods preceding it.

- ASTTree.java

```
public void traverse()
```

Perform a depth first in-order traversal to print the tree. Used to verify the ast tree built/ standardized tree is correct. Make use of a private overloaded function as a helper function.

```
public Node standardize()
```

Perform a depth first in-order traversal to standardize the AST tree. Search for Standardizble nodes and standardize the accordingly. Make use of a private overloaded function as a helper function.

Classes in ast.nodes package

- Node.java

Abstract class for all nodes. Contain the constructor requesting name for the node.

- InnerNode.java

Abstract class used to distinguish the nodes which can hold children. Hold few key functions need to be implemented in sub classes.

```
public abstract void setChild(Node child);
```

Function to add a child to the node following the constrains.

```
public abstract List<Node> getChildren();
```

To get all the children of the node. Used when traversing through the AST tree.

```
public abstract void removeChildren();
```

To remove all the children from the node. Used clear and assign standardized children in standardizing process.

- LeafNode.java

Abstract class used to distinguish the nodes which cannot hold children.

- NodeFactory.java

```
public static Node createNode(String type)
```

Use factory design pattern to create the nodes for the ASTParser. Create node according to the type passed into the method.

- Standardizable.java

Interface to mark the standardizable nodes. Holds a function which need to implement by classes which implement this interface

```
Node standardize();
```

Returns the standardized node for a given node.

All other classes are to represent each node possible in AST tree along with the above abstract methods implemented according to the node.

Classes in csemachine package

- Machine.java

```
public void evaluate()
```

Function which applies the CSE Machine rules to evaluate the program given.

`public List<Delta> getControlStructures()`

To get the list of control structures which is obtained by flattening the Standardized tree. Make use of a helper function called flatten.

`private Element getElement(Node node, int tag)`

Create Elements to be used in CSE machine according to the Node found in Standardized tree.

`private void functionCall(String functionName)`

Apply the predefined functions in the RPAL namely,

- Print
- Conc
- Stern
- Stem
- Order
- Isstring

- MachineStack.java, MachineControl.java

Wrapper classes for Stack. Implements usual pop and push for the stack.

Key difference is MachineStack.java applies the values from the Environment for the variables upon push into stack.

- MachineEnvironment.java

Class is used for maintaining the environment tree for the program. Use an inner class called Node to support this.

`public void addNewEnvironment(Environment environment)`

Adds new environment node for the current environment tree. Adds as the child node for the environment node which is currently being used to execute the program.

`public Element findValue(String variableName)`

Used to find the values from the environment tree.

Classess in csemachine.elements package

- Element.java

  Abstract class for all the other classes in the package

- Bop.java, Uop.java

  `public Object apply(Object operand1 [, Object operand2])`

  used to apply the Binary or Urinary operation accordingly

All other classes are symbolic classes to represent Elements for the CSE machine.

Some have tags and children according to the requirement.

APPENDIX


NodeFactory
+ createNode(String): Node
FcnFormNode
+ standardize(): Node
+ removeChildren(): void
child: Node
children: List<Node>
AndNode
+ removeChildren(): void
+ standardize(): Node
child: Node
children: List<Node>
RecNode
+ standardize(): Node
+ removeChildren(): void
child: Node
children: List<Node>
WhereNode
+ standardize(): Node
+ removeChildren(): void
child: Node
children: List<Node>
LetNode
+ standardize(): Node
+ removeChildren(): void
child: Node
children: List<Node>
InfixNode
+ standardize(): Node
+ removeChildren(): void
child: Node
children: List<Node>
WithinNode
+ standardize(): Node
+ removeChildren(): void
child: Node
children: List<Node>
myrpal
+ main(String[]): void
PrimitiveNode
IDNode
CommaNode
+ removeChildren(): void
child: Node
children: List<Node>
TauNode
+ removeChildren(): void
child: Node
children: List<Node>
YNode
EqualNode
+ removeChildren(): void
child: Node
children: List<Node>
LambdaNode
+ removeChildren(): void
child: Node
children: List<Node>
<<interface>>
Standardizable
+ standardize(): Node
GammaNode
+ removeChildren(): void
child: Node
children: List<Node>
ArrowNode
+ removeChildren(): void
child: Node
children: List<Node>
UopNode
+ removeChildren(): void
child: Node
children: List<Node>
BopNode
+ removeChildren(): void
child: Node
children: List<Node>
ASTParser
+ parse(String): ASTTree
Machine
+ evaluate(): void
- flatten(int, Node, MachineControl, Queue<Node>): int
- controlStructureArrayToMap(List<Delta>): Map<Integer, Queue<Element>>
- getVariable(Node, int): Element
- functionCall(String): void
- pushToControl(Queue<Element>): void
- getElement(Node, int): Element
controlStructures: List<Delta>
LeafNode
InnerNode
+ removeChildren(): void
child: Node
children: List<Node>
ASTTree
+ standardize(): Node
- standardizeChildren(Node): void
- standardize(Node): Node
- traverse(Node, String): void
+ traverse(): void
MachineEnvironment
+ addNewEnvironment(Environment): void
# moveToParent(): void
+ findValue(String): Element
environmentTag: int
MachineStack
push(Element, MachineEnvironment): void
pop(): Element
Delta
Comma
Node
parent: Node
Node
- addChild(Environment): void
children: List<Environment>
parent: Node
Bop
+ apply(Object, Object): Object
MachineControl
push(Element): void
pop(): Element
empty: boolean
Variable
Environment
+ getValue(String): Element
+ addAssignment(String, Element): void
Lambda
Tor
Eta
Uop
+ apply(Object): Object
Gamma
Primitive
Y
Beta
Element
«create»
functionName
children
left
right
child
Expression
variables
astTree
environment
current
stack
control
assignments
boundedVariable