Final Project Report

Racket Interpreter for Arithmetic Expressions

Qiuyuan (Sophie) Zhang

2016/12/6

1. Overview

This project implements an interpreter for a subset of Racket Language in C, namingly two argument mathematical operations of plus, minus, multiplication and division, also includes parenthesis checking function.

2. Requirement Satisfaction

The project satisfies all the requirement made in the proposal.

3. Example

```
> ./interp
Enter Racket Expression (two argument operations):
(* (* 3 2) (+ 1 4))
------------------
Lexer Output:
L->(
O->*
L->(
O->*
I->3
I->2
R->)
L->(
O->+
I->1
I->4
R->)
R->)
------------------
Arithmetic Tree:
(* (* 3 2) (+ 1 4))
answer-> 30
```

The “interp” program takes in an arithmetic expression written in Racket, performs lexical analysis, splits expression into recognizable tokens, and parses these tokens into an arithmetic expression tree. The program recursively evaluates the tree until an result is finalized.

4. Results

The program performs as expected with simple or embedded two arguments integer operations of plus, minus, multiplication, and division.

5. Lessons Learned

Most of all, I have fulfilled my curiosity on general interpreters/ compilers :).

While implementing Lexer, Parser, and Eval module, I have gained deeper understanding on data structures like Queue and Tree, as well as confidence in implementation of recursion algorithms.

As of the user-define-function part, the high level idea seems to be create a tree of such function name, with variables embedded in the tree. Then later transverse the tree to replace variable with value. Due to the time constraints, I was not able to successfully implement it. So there is a lot of improvement to be done on managing project timeline.

I have also gained an appreciation for interpretive language like Racket. Though not as run-time friendly as C, interpretive language offers efficiency in developing time. And Racket as a Student Learning Language is indeed helpful for fostering structural understanding of programing in general.