**CPSC 3740: Programming Languages**

**Final Project(**Spring 2019)

Davison Mathew, Zach Nelson,

Taranjot Kaur

Distribution Of Work

For our project, we divided the work up as evenly as possible and each worked on our separate parts before bringing them all together. Our work assignments were as follows:

* Davison Mathew’s part was to work with lists, conditionals and applying lambda expression to arguments.
* Zach Nelson's portion was working with Constants and variables , Arithmetic operators and Relational operators.
* Taranjot Kaur's part was to work with lambda expressions and local bindings.
* After we had each finished our parts, we came back together and worked as a group to bring the entire project together by combining all of our portions.

Proceedings of the Program

We implemented an interpreter for a subset of Racket inside Racket. The main goal of our project was the implementation of the function startEval which takes list1 into consideration and apply various constructs like constants and variables , arithmetic operations(eg:- +,-,\*,/) , relational operations(eg:- =,<=,>=,<,>) , conditionals(eg:- if), list operations(eg:- car,cdr,cons,pair?) , lambda expressions and applying lambda expression to an argument and local binding(eg:- let, letrec).

Key Data Structures

* Pair- pair is used with cons, and the val­ues can be retrieved with car and cdr.
* Lists- Used with car and cdr. List is also the fundamental data structure in racket.

Limitations

Test Cases

1. Arithmetic Operations

* Addition

(startEval '(+ 4 3)) :- This returns simple addition of two numbers and the answer will be 7.

* Subtraction

(startEval '(- 4 2)) :- This returns simple subtraction of two numbers and the answer will be 2.

* Multiplication

(startEval '(\* 3 5)) :- This returns simple multiplication of two numbers and the answer will be 15.

* Division

(startEval '(/ 4 2)) :- This returns simple quotient of two numbers and the answer will be 2 .

1. Relational Operations

* equal?

(startEval '(yes yes)

|  |
| --- |
| ([equal?](https://docs.racket-lang.org/reference/booleans.html#%28def._%28%28quote._~23~25kernel%29._equal~3f%29%29) yes yes) |
| )  This returns #t after checking the equality of the two arguments. |

* <=

(startEval '(2 2)

|  |
| --- |
| ([<=](https://docs.racket-lang.org/reference/booleans.html#%28def._%28%28quote._~23~25kernel%29._equal~3f%29%29) 2 2)) |
| This returns #t because 2 is less than or equal to 2.   * >=   (startEval '(2 3)   |  | | --- | | ([>=](https://docs.racket-lang.org/reference/booleans.html#%28def._%28%28quote._~23~25kernel%29._equal~3f%29%29) 2 3)) | | This returns #f because 2 is not greater than or equal to 3.   * >   (startEval '(2 3)   |  | | --- | | (> 2 3)) | | This returns #f because 2 is not greater than 3.   * <   (startEval '(2 3)   |  | | --- | | (< 2 3)) | | This returns #t because 2 is less than 3.   1. Lists  * pair?   (startEval '(1 2)  (pair? '(1 2)))  This returns #t because 1 and 2 form a pair of the list.   * car   (startEval '(1 2)  (car '(1 2)))  This returns 1 because 1 is the first element of the list.   * cdr   (startEval '(1 2)  (cdr '(1 2)))  This returns 2 because 2 is the second element of the list.   1. Lambda   startEval (lambda (x y)  (- y x)))  (subtract 4 9)  This returns 5 after the evaluation of lambda expression.   1. Letrec   (letrec ((even?  (lambda (n)  (if (zero? n)  #t  (odd? (- n 1)))))  (odd?  (lambda (n)  (if (zero? n)  #f  (even? (- n 1))))))  (even? 10))  This returns #t after the evaluation of letrec.     |  | | --- | |  | |  |  | | |  | | |  | | |  |  | | | | | |
|  |