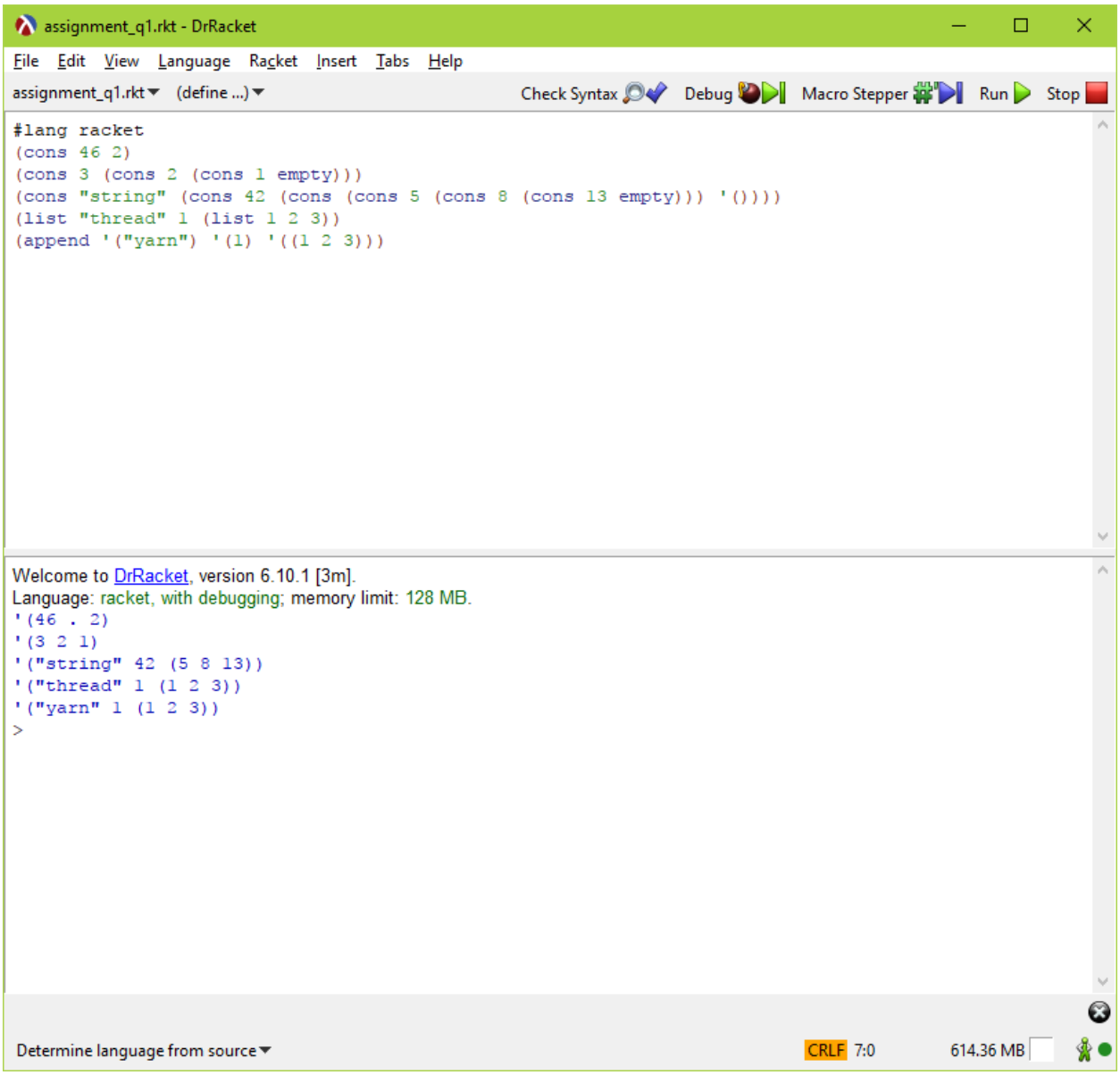
CT331 Assignment 2

Functional Programming with Scheme

Andrew Reid East
16280042
https://github.com/reideast/ct331_assignment2

Question 1

Code Editor and Command Line Output



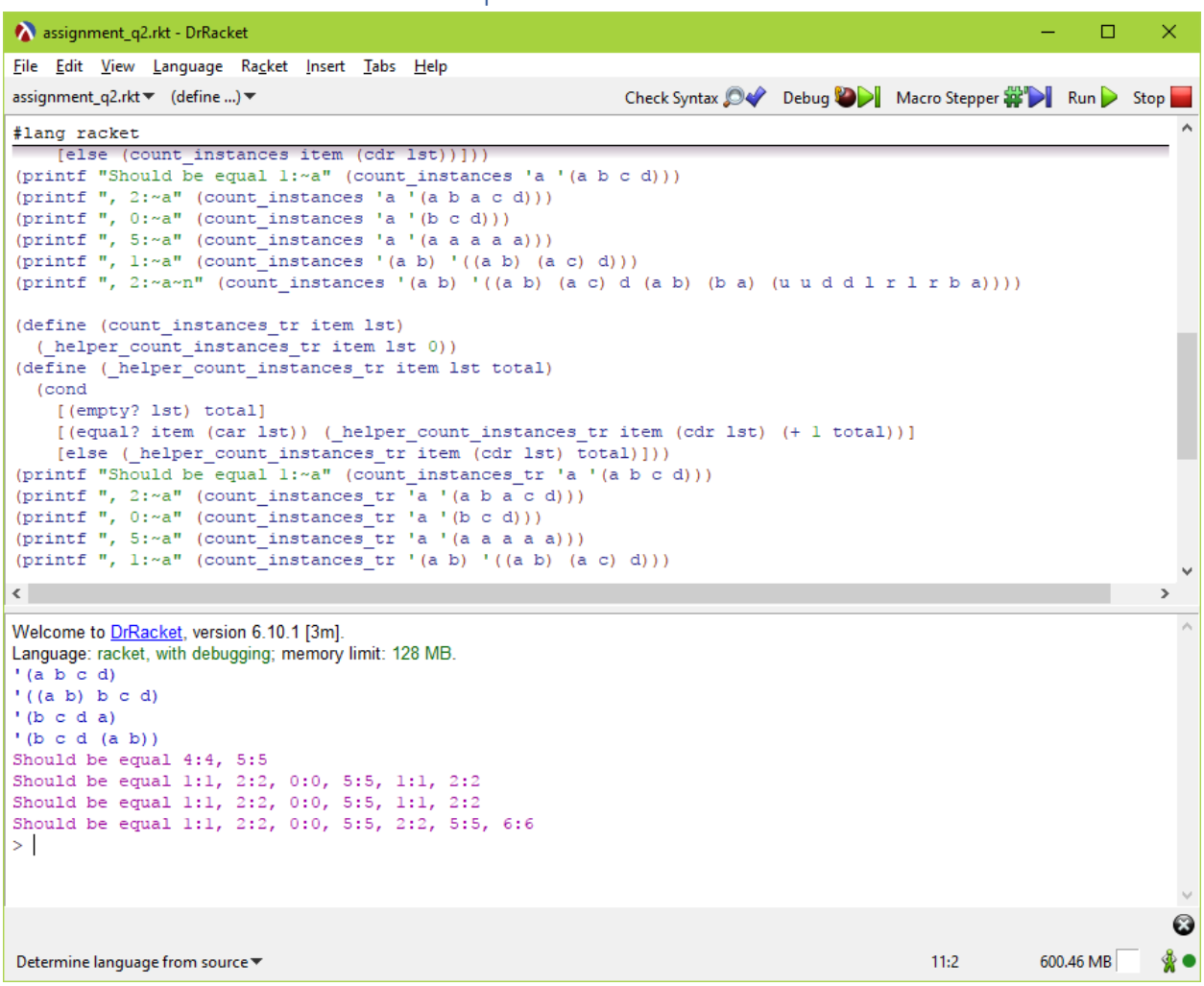
Comments

The lists being made up of nested cons pairs seems similar to recursion, and I'm curious if the Racket interpreter evaluates using order-of-operation rules in one stack frame, or actually just fires off each in a recursive manner. Considering that Lisp itself is so recursive, I wonder how the compiler's inner workings handle things.

Cons takes two s-expressions and combines then into a pair, which is in itself a single s-expression. The list function takes a series of zero or more s-expressions and combines them into a list, which is shown in the command line output as a single unit, but is really a shorthand for a more complex s-expression (a nested series of pairs, with the second item of the innermost pair being the empty value). The append function also builds a list, but each of its arguments must also be a list, and in contrast to the list function, each argument list is chained onto the previous, which makes one long list of all those elements. Finally, while the cons function always takes two values, both list and append can take any number of values.

Question 2

Code Editor and Command Line Output



Question 3

Code Editor and Command Line Output

assignment_q3.rkt - DrRacket

File Edit View Language Racket Insert Tabs Help

assignment_q3.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang racket
(insert_item (insert_item (insert_item '() 2) 1) 3)

; Part D: Insert a list into a BST
(define (insert_list tree lst)
  (if (empty? lst)
     tree
     (insert_list (insert_item tree (car lst)) (cdr lst))))
; TODO make it tail recursive...WAIT it already is! cool.
(display "\nInsert list into empty tree: '(2 1 3). Should be ((() 1 ()) 2 (() 3 ())): ")
(insert_list '() '(2 1 3))
(display "Insert '(0 4) into that list. Should be ((() 0 ()) 1 ()) 2 (() 3 (() 4 ()))): ")
(insert_list two_tree '(0 4))
(display "Insert list that makes an unbalanced tree: '(1 2 3). Should be (() 1 (() 2 (() 3 ()))): ")
(insert_list '() '(1 2 3))
(display "Recreate example tree from a list. Should output #true: ")
(equal? example_tree (insert_list '() example_list_to_insert))

; Part E: Treesort
;Treesort to cout--I only left this one in here because I like how it's so svelte. The next block has my actual function.
```

```
Welcome to DrRacket, version 6.10.1 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Traverse & Print a tree with only root: 3
Traverse & Print a complete tree with height two: 1,2,3,
Traverse & Print the example tree: 1,3,4,6,7,8,10,13,14,

Search single item tree for value that exists (should be #t): #t
Search single item tree for value that does not exists (should be #f): #f
Search large tree for value that exists on left (should be #t): #t
Search large tree for value that exists on right (should be #t): #t
Search large tree for value that does not exists (should be #f): #f

Insert 4 into an empty/null tree. Should be (() 4 ()): '(() 4 ())
Insert 3 into a small tree where it already exists, should be (() 3 ()): '(() 3 ())
Insert 15 into a bigger tree where it already exists, should be '((() 3 ()) 5 (() 7 ())) 10 (() 15 ())): '((() 3 ()) 5 (() 7 ())) 10 (() 15 ()))
Insert 5 into the example tree, should be '((() 1 ()) 3 ((() 4 (() 5 ())) 6 (() 7 ()))) 8 (() 10 ((() 13 ()) 14 ())): '((() 1 ()) 3 ((() 4 (() 5 ())) 6 (() 7 ()))) 8 (() 10 ((() 13 ()) 14 ()))
Insert 9 into the result of the previous, should be '((() 1 ()) 3 ((() 4 (() 5 ())) 6 (() 7 ()))) 8 ((() 9 ()) 10 ((() 13 ()) 14 ())): '((() 1 ()) 3 ((() 4 (() 5 ())) 6 (() 7 ()))) 8 ((() 9 ()) 10 ((() 13 ()) 14 ()))
Insert first 2, then 1, then 3 into a null tree. Should be ((() 1 ()) 2 (() 3 ())): '((() 1 ()) 2 (() 3 ()))

Insert list into empty tree: '(2 1 3). Should be ((() 1 ()) 2 (() 3 ())): '((() 1 ()) 2 (() 3 ()))
Insert '(0 4) into that list. Should be ((() 0 ()) 1 ()) 2 (() 3 (() 4 ()))): '((() 0 ()) 1 ()) 2 (() 3 (() 4 ()))
Insert list that makes an unbalanced tree: '(1 2 3). Should be (() 1 (() 2 (() 3 ()))): '(() 1 (() 2 (() 3 ())))
Recreate example tree from a list. Should output #true: #t

Treesort on the example list (print). Should be 1,3,4,6,7,8,10,13,14,: 1,3,4,6,7,8,10,13,14,
Treesort on the example list (list-ify). Should be '(1 3 4 6 7 8 10 13 14): '(1 3 4 6 7 8 10 13 14)
Treesort of same data should be the same, even if inserted in different order: #t
Treesort of same num items but different data should be, of course, false: #f
Treesort 'make my CPU hurt': '(-2 0 2 4 5 6 8 9 11 12 21 22 27 33 34 42 43 46 60 64 65 67 76 77 90 97 244 345 435 677 678 2111 4444 60303)

Treesort on the example list: <. Should be '(1 3 4 6 7 8 10 13 14): '(1 3 4 6 7 8 10 13 14)
Treesort on the example list: >. Should be '(14 13 10 8 7 6 4 3 1): '(14 13 10 8 7 6 4 3 1)
Treesort on a list of strings with (string<?): . Should be '(Apple Banana Cumquat Duran Elderberry): '("Apple" "Banana" "Cumquat" "Duran" "Elderberry")
Reversed (string>?): . Should be '(Apple Banana Cumquat Duran Elderberry): '("Elderberry" "Duran" "Cumquat" "Banana" "Apple")
>
```

Determine language from source 114:41 817.38 MB