FUNCTIONAL PROGRAMMING
MULTIREMBER

21297719

Table of Contents

Introduction

This assignment examines one of the functions utilised in Functional Programming. The function I will be analysing is called **multirember**

There are several IDE's such as Netbeans Eclipse etc. The IDE's used for this assignment is Dr. Racket. Hailing from the Lisp-Scheme family of programming language, Racket serves as a medium for several libraries and compilers, as well as its own language (scheme) to create functions.

Recursion

Recursion is one of the most important aspects of programming. It is a technique that calls itself when activated or when a condition is met. It will then repeat this process successively until the termination condition is met. When the terminate or end condition is enacted, each repetition is then processed in order of last to first.

Two basic principles:

Base case
- We need to finish looping somehow
- Termination based on some condition

Recursive case
- We need to make a call to our own function name
- We need to do something in that function call that helps us reach our goal

Multirember

The function Multirember is used to remove any number of atoms from a list, and return the final value of the list. It is very useful in locating **all** occurrences of the atom(s) you want to delete from the list. It uses anothers function called **Cons** which adds any atom to the front of the list, in the order in which you choose.

Code definition

```
#lang racket
(define multirember
 (lambda (a lat)
   (cond
    ((null? lat) (quote()))
   (else
     (cond
   ((eq? (car lat) a)
     (multirember a (cdr lat)))
   (else
     (cons (car lat)
  (multirember a (cdr lat)))))))))
```

High Level Code Explanation

Example – A is equal to '**man utd'**
Lat is (Liverpool man utd is man utd the best man utd)

The final value of lat will be (Liverpool is the best).

define multirember – name of the function to be used
lambda a lat – a collection of atoms or S expressions
condition – the first argument is given
null? – checks if the list is null. **False**
else – this statement is always **True**
condition – the second argument is given
eq? (car lat) – checks if the value of car (first atom in a list) is equal to lat. In the case, car equals **Liverpool**, so its **False**
else - this statement is always **True**
cons (car lat) – adds **Liverpool** to the front of the 'final list'
multirember a (cdr lat) – checks the rest of the list (cdr) for the S expression **man utd**.

The entire list will be inspected in the order in which they have been given until it is **null**. Recursion takes places

It will check the entire list until the list is null and return the final value.

Trace Execution

Trace execution is a very important facet of programming and an invaluable tool for all programmers. Simply put, it displays the current status and functionality of a program, highlighting any errors or unintended outcomes of a program. It systematically tests each line of code and displays the results. In that way, an error is easily located and remedied.

Below are a series of tests, to verify the following

- When the code functions correctly Test 1 and Test 2
- When the code functions incorrectly Test 3 and Test 4
- Test 5 is an example of some anomalies and features of the program.

TEST 1

A = 'hate'
Lat = (hate fate hate love hate and peace)

```
;;1 deletes the word 'hate' from list
(multirember 'hate '(hate fate hate love hate and peace))

;;2 'fate' is cons'ed to the final list
(cons 'fate (multirember 'hate '(hate love hate and peace)))
'(fate love and peace)

;;3 'hate' is removed from the list
(cons 'fate (multirember 'hate '(hate love and peace hate)))

;;4 'fate' 'love' are cons'ed to the final list
(cons 'fate (cons 'love (multirember 'hate '(and peace))))

;;5 'fate' 'love' 'and' are cons'ed to the final list
(cons 'fate (cons 'love (cons 'and (multirember 'fate '(peace)))))
```

```
;;6 'fate' 'love' 'and' 'peace' are added to final list
(cons 'fate (cons 'love (cons 'and (cons 'peace (multirember 'fate
'())))))

;;7 'fate' 'love' 'and 'peace' are cons'ed to the final list and quoted
(cons 'fate (cons 'love (cons 'and (cons 'peace (quote()))))))

;;8 final list
(cons 'fate (cons 'love (cons 'and (cons 'peace '()))))

;;9
(cons 'fate (cons 'love (cons 'and '(peace))))

;;10
'(fate love and peace)
```

Code explained

Line 1- removes first occurrences **(car lat)** of '**hate'** from the list and checks the rest of the list.
Lines 2 - 5 each atom or S expression will be inspected **(cdr lat)** and displayed in the order in which it appears.
Lines 6 – 7 the function ends when the list is **null.** It then quotes the final list and displays the results.

Test 2

A = 'only'
Lat = (only 18 and only over can only vote)

```
;;1
(multirember 'only '(only 18 and only over can only vote))
;;2
(cons '18 (multirember 'only '(and only over can only vote)))
'(18 and over can vote)
```

```
;;3
(cons '18  (cons 'and (multirember 'only '(only over can only vote))))
;;4
(cons '18 (cons 'and (cons 'over (multirember 'only '(can only
vote)))))
;;5
(cons '18 (cons 'and (cons 'over (cons 'can (multirember 'only '(only
vote))))))
;;6
(cons '18 (cons 'and (cons 'over (cons 'can (cons 'vote (quote()))))))
;;7
(cons '18 (cons 'and (cons 'over (cons 'can (cons 'vote '())))))
;;8
(cons '18 (cons 'and (cons 'over (cons 'can (cons 'vote '())))))
;;9
'(18 and over can vote)
```

Code explained

Line 1: removes first occurrences **(car lat)** of '**only**' from the list and checks the rest of the list.
Lines 2 – 5: each atom or S expression will be inspected **(cdr lat)** and displayed in the order in which it appears.
Lines 6 – 7: the function ends when the list is **null.** It then quotes the final list and displays the results.

Test 3

A is '1'

Lat is (1 0 1 2 (1))

```
;;1
(multirember '1 '(1 0 1 2 (1)))
;;2
(cons '0 (multirember '1 '(2 (1))))

;; 3 function failure. 1 and (1) are both removed
(cons '0 (cons '2 (multirember '1 '(1))))
```

Code explained

Lines 1 – 2: function at this point is functioning correctly
Lines 3: function fails. Both the **multirember '1'** and **1(lat)** are removed from the list.

TEST 4

A = 'false'
Lat = (false this is false is (a list) false)

```
;1
(multirember 'false '(false this false is (a list) false))
;2
(cons 'this (multirember 'false '(false is (a list) false)))
'(this is (a list))
;3
(cons 'this (multirember 'false (cons 'is  (cons '(a list) '(false)))))

;4 '(a list) is in a list. Function ends when it encounters a list
(cons 'this (cons 'is (cons '(a list) (multirember 'false '(false)))))
```

Code explained

Lines 1 – 3: function at this point is functioning correctly
Lines 4: function fails when it encounters a list. The S expression (a list) caused this application to failure. To correct this, they have to be cons'ed before multirember is evoked.

TEST 5

A = 'wrong'

Lat = (wrong it wrong will wrong still work)

```
;;1
(multirember 'wrong '(wrong it wrong will wrong still work))
;;2
(cons 'it (multirember 'wrong '(wrong will still work)))
'(it will still work)
;;3
(cons 'it (multirember 'wrong '(wrong will still work)))
;;4
(cons 'it (cons 'will (cons 'still (cons 'work (multirember 'wrong
'(wrong))))))
;;5
(cons 'it (cons 'will (cons 'still (multirember 'no '(work)))))
;;6
(cons 'it (cons 'will (cons 'still (cons 'work (quote())))))
;;7
(cons 'it (cons 'will (cons 'still (cons 'work '()))))
;;8
(cons 'it (cons 'will (cons 'still '(work))))
;;9
'(it will still work)


;;10 'still' is inside a different list
(cons 'it (cons 'will (cons '(still) (cons 'work (multirember 'no '())))))

;;11 cons'ed ‘it’ before multirember is performed on the same atom
(cons 'it (multirember 'wrong (multirember 'it '(wrong it wrong will
wrong still work))))

;;12 cons'ed  ‘it’ after multirember is performed on the same atom
(multirember 'wrong (multirember 'it (cons 'it '(wrong it wrong will
wrong still work))))

;;13 multirember is also case sensitive
(multirember 'case '(case caSe caSE case Sensitive case))
(cons 'caSe (multirember 'case '(caSE case Sensitive case)))
```

Code explained

Line 10 – function still works despite the atom (still) residing in another list.

Line 11 – you can cons'ed an atom or S expression first then use the multirember function on the same word. The function will still work because it was cons'ed before multirember was used.

Line 12 – however, if multirember on an atom is performed first, and then cons'ed to a list, that atom will not be added to the list

Line 13 – multirember is aslo case sensitive, which is obviously a very useful tool when trying to locate specific words and syntax.

Conclusion

Racket is easy to use IDE and the scheme syntax are relatively straightforward. Functions and variables are much easier to create and use on Racket. Although racket does not use arrays, the list function is more than adequate and versatile to use. However, when using recursion on a list, the final value will not be in the correct order unless the cons'ed function is used in the correct order. Unlike arrays where you can simply call its index, with scheme, you can not do this.
Without cons'ed being added, the nature of recursion would have meant that the atoms/expression would have been listed from last to first.

Appendix

```
(define multirember
  (lambda (a lat)
    (cond
     ((null? lat) (quote()))
    (else
      (cond
    ((eq? (car lat) a)
```

```
      (multirember a (cdr lat)))
    (else
      (cons (car lat)
  (multirember a (cdr lat)))))))))
```

Test 1

```
(multirember 'hate '(hate fate hate love hate and peace))
(multirember 'fate '(fate hate love hate and peace))
(cons 'fate (multirember 'hate '(hate love hate and peace)))
'(fate love and peace)
(cons 'fate (multirember 'hate '(love and peace hate)))
(cons 'fate (cons 'love (cons 'and (cons 'peace (multirember 'hate
'(hate))))))
(cons 'fate (cons 'love (cons 'and (cons 'peace (multirember 'why
'())))))
(cons 'fate (cons 'love (cons 'and (cons 'peace (quote())))))
(cons 'fate (cons 'love (cons 'and (cons 'peace '()))))
(cons 'fate (cons 'love (cons 'and '(peace))))
'(fate love and peace)
```

Test 2

```
(multirember 'only '(only 18 and only over can only vote))

(cons '18 (multirember 'only '(and only over can only vote)))
'(18 and over can vote)

(cons '18  (cons 'and (multirember 'only '(only over can only vote))))

(cons '18 (cons 'and (cons 'over (multirember 'only '(can only
vote)))))

(cons '18 (cons 'and (cons 'over (cons 'can (multirember 'only '(only
vote))))))

(cons '18 (cons 'and (cons 'over (cons 'can (cons 'vote (quote()))))))

(cons '18 (cons 'and (cons 'over (cons 'can (cons 'vote '())))))

(cons '18 (cons 'and (cons 'over (cons 'can (cons 'vote '())))))
```

'(18 and over can vote)

Test 3

(multirember '1 '(1 0 1 2 (1)))

(cons '0 (multirember '1 '(2 (1))))

;; function failure. 1 and (1) are both removed
(cons '0 (cons '2 (multirember '1 '(1))))

Test 4

(multirember 'false '(false this false is (a list) false))

(cons 'this (multirember 'false '(false is (a list) false)))
'(this is (a list))

(cons 'this (multirember 'false (cons 'is (cons '(a list) '(false)))))

(cons 'this (cons 'is (cons '(a list) (multirember 'false '(false)))))

Test 5

(multirember 'wrong '(wrong it wrong will wrong still work))

(cons 'it (multirember 'wrong '(wrong will still work)))
'(it will still work)

(cons 'it (multirember 'wrong '(wrong will still work)))

(cons 'it (cons 'will (cons 'still (cons 'work (multirember 'wrong
'(wrong))))))

(cons 'it (cons 'will (cons 'still (multirember 'no '(work)))))

(cons 'it (cons 'will (cons 'still (cons 'work (quote()))))))

(cons 'it (cons 'will (cons 'still (cons 'work '()))))

(cons 'it (cons 'will (cons 'still '(work))))

'(it will still work)

(cons 'it (cons 'will (cons '(still) (cons 'work (multirember 'no '()))))))

(cons 'it (multirember 'wrong (multirember 'it '(wrong it wrong will wrong still work))))

(multirember 'wrong (multirember 'it (cons 'it '(wrong it wrong will wrong still work))))

(multirember 'case '(case caSe caSE case Sensitive case))
(cons 'caSe (multirember 'case '(caSE case Sensitive case)))