Ø

Problem Bank Style Rules Design Recipes Language Glossarv Discussion Progress

Course > 10: Accumulators > Tail Recursion > Question 11-12

















Next >

Question 11-12

☐ Bookmark this page

Question 11

1/1 point (graded)

You are asked to refactor the following function, product to make it tail recursive:

```
;; (listof Number) -> Number
;; produce the product of all the numbers in lon
(check-expect (product empty) 1)
(check-expect (product (list 1 2 3)) 6)
(check-expect (product (list 2.5 1 -4)) -10)
(define (product lon)
  (cond [(empty? lon) 1]
         [else
(* (first lon)
              (product (rest lon)))]))
```

What type of accumulator will we need to make the function tail recursive?

- C We don't need an accumulator the function is already tail recursive
- C Number; the previous number in the list
- Number; the product of the elements seen so far
- C Number; the current position in the list



Explanation

In order to make product tail recursive, we will need a result-so-far accumulator that keeps track of the product of the numbers already seen.

1 Answers are displayed within the problem

Question 12

1/1 point (graded)

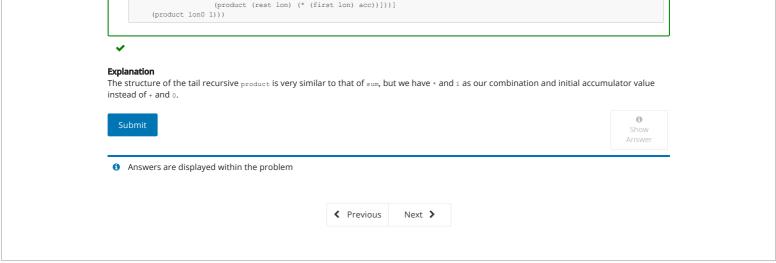
Which is the correct function body for the tail recursive version of product?

```
(define (product lon)
 (cond [(empty? lon) 1]
       [else
         (* (first lon)
            (product (rest lon)))]))
```

```
(define (product lon0)
 ;; acc: Number; product of the numbers seen so far
 (local [(define (product lon acc)
           (cond [(empty? lon) 1]
                 [else
                  (* (first lon)
                     (product (rest lon) acc))]))]
   (product lon0 1)))
```

```
(define (product lon0)
 ;; acc: Number; product of the numbers seen so far
 (local [(define (product lon acc)
           (cond [(empty? lon) 0]
                 [else
                  (product (rest lon) (* (first lon) acc))]))]
    (product lon0 1)))
```

```
(define (product lon0)
 ;; acc: Number; product of the numbers seen so far
 (local [(define (product lon acc)
           (cond [(empty? lon) acc]
              [else
```



⊚⊕§⊚ Some Rights Reserved



edX

About edX for Business Legal

Terms of Service & Honor Code Privacy Policy Accessibility Policy

Connect

Blog Contact Us Help Center













© 2020 edX Inc. All rights reserved. | 深圳市恒宇博科技有限公司 粤ICP备17044299 号-2