**THEORY OF PROGRAMMING LANGUAGES**

**A picture containing text, gear, metalware

Description automatically generated**

**Name: M. Hassan Sattar**

**Roll No: 22F-3773**

**Submitted to: Dr. Sajid Anwar**

**National University of Computer and Emerging Sciences**

**Chapter 15**

**(Problem set -9)**

**Solution:**

(define (y s lis)

(cond ( ( null? lis) ’( ) )

      ( ( equal? s (car lis )) lis)

(else ( y s (cdr lis))

 ) )

The above scheme function will perform as follows:

1. the first statement will define a function 'y' that has two parameters 's' (a value) and lis (a list)
2. The second statement says that the condition will be checked. Here three conditions are there to check.
3. The third statement checks if the list is empty, if it is true the empty list will be returned.
4. The fourth statement checks whether the value of the first element of the list is equal to s value. If it is true, then the list will be returned up to s.
5. The fifth statement will otherwise check rest of the list, by recurring over it.

**Chapter 15**

**(Problem set -10)**

**Solution:**

(define (x lis)

(cond ((null? lis) 0)

 ((not (list? (car lis))

 (cond ((eq? (car lis) #f) (x   (cdr lis)))

    (else (+ 1 (x (cdr lis)))))

      (else (+ (x (car lis)) (x (cdr lis))))

The above Scheme function will perform as follows:

1. The first statement defines the function 'x' with a parameter lis (a list)
2. The second statement says that the condition will be checked. Here two conditions are there to check.
3. The third statement checks if the list is empty and if it is true then 0 is returned.
4. fourth statement checks for the input pair and if its car (first element) is not a list than:
5. The fifth statement again says that conditions will be checked. Here two conditions are there to check.
6. The sixth statement will check that if the car is #f than list will recursed on cdr and cdr will be returned.
7. The seventh statement will otherwise count the car as 1 and add it to the result of the recursing on cdr.

The function 'x' will provide the total number of leaf nodes in the nested list structure by using recursive procedure and it will ignore #f.

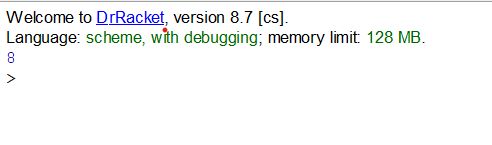
**Programming Exercise**

**Task-1**

(define (volume r) (\* r r r))

(volume 2)

**Screenshot:**



**Task-2**

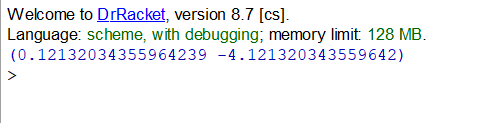
(define (quadratic\_roots a b c)

  (list (/ (+ (- b) (sqrt (- (\* b b) (\* 4 a c)))) (\* 2 a))

        (/ (- (- b) (sqrt (- (\* b b) (\* 4 a c)))) (\* 2 a))))

(quadratic\_roots 2 8 -1)

**Screenshot:**

****

**Task-4**

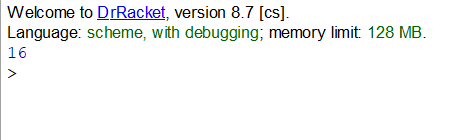
(define (pow a b)

  (cond ((= b 0) 1)

        (else (\* a (pow a (- b 1))))))

(pow 2 4)

**Screenshot:**



**Task-5**

  (define (zeros l)

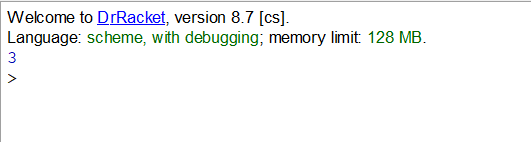
  (cond ((null? l) 0)

        ((zero? (car l)) (+ 1 (zeros (cdr l))))

        (else (zeros (cdr l)))))

(zeros '(1 2 3 0 5 6 0 7 0 3))

**Screenshot:**



**Task-6**

(define (min l)

  (cond ((null? l) '())

        ((null? (cdr l)) (car l))

        ((< (car l) (min (cdr l))) (car l))

        (else (min (cdr l)))))

(define (max l)

  (cond ((null? list) '())

        ((null? (cdr l)) (car l))

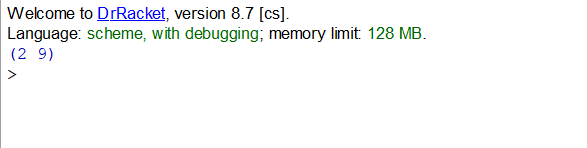
        ((> (car l) (max (cdr l))) (car l))

        (else (max (cdr l)))))

(define (min\_max l) (list (min l) (max l)))

(min\_max '(2 7 8 9 5 4 3))

**Screenshot:**

****

**Task-7**

(define (removal l s)

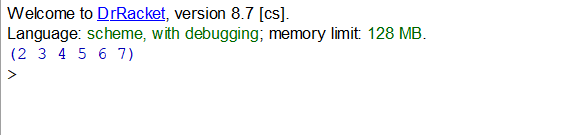
  (cond ((null? l) '())

        ((eq? (car l) s) (removal (cdr l) s))

        (else (cons (car l) (removal (cdr l) s)))))

(removal '(1 2 3 1 4 1 5 6 7) 1)

**Screenshot:**

****

**PROBLEMS**

**Problem 1:**

(define (flatten l)

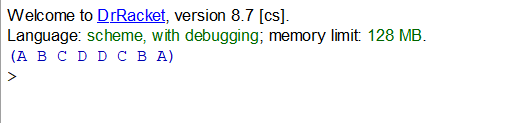
  (cond ((null? l) null)

        ((list? (car l)) (append (flatten (car l)) (flatten (cdr l))))

        (else (cons (car l) (flatten (cdr l))))))

(flatten '(A B (C (D D) C) B A))

**Screenshot:**

****

**Problem 2:**

(define (slice i j l)

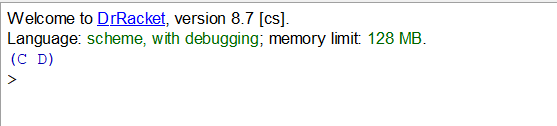
  (cond ((eq? i j) '())

        ((> i 0) (slice (- i 1) (- j 1) (cdr l)))

        ((> j 0) (cons (car l) (slice i (- j 1) (cdr l))))))

(slice 2 4 '(A B C D E))

**Screenshot:**

****

**Problem 3:**

(define (insertToSortedList element lst)

 (cond

  [(or (empty? lst) (<= element (first lst))) (cons element lst)]

  [else (cons (first lst) (insertToSortedList element (rest lst)))]))

(define (sort-demo lst)

 (cond

 [(empty? lst) empty]

 [else (insertToSortedList (first lst) (sort-demo (rest lst)))]))

**Problem 4:**

(define (gcd m n)

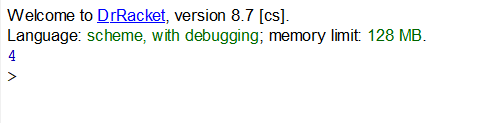
  (cond ((eq? m n) m)

        ((> m n) (gcd (- m n) n))

        (else (gcd m (- n m)))))

(gcd 52 108)

 **Screenshot:**



**Problem 5:**

(define (prime-factors n)

(if (or (not (integer? n)) (< n 2))

      '()

(let loop ((n n) (f 2) (factors '()))

 (if (= n 1) (reverse factors)

  (if (zero? (modulo n f))

  (loop (/ n f) f (cons f factors))

  (loop n (+ f 1) factors))

        )))

)

**Problem 6:**

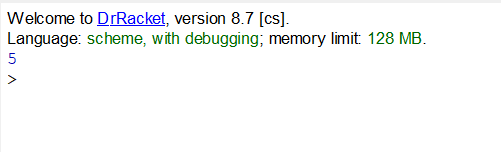
(define (length l)

  (cond ((null? l) 0)

        (else (+ 1 (length (cdr l))))))

(length '(A B (C (D D) C) B A))

**Screenshot:**

****

**Problem 7:**

(define (sumsquare l)

  (cond ((null? l) 0)

        (else (+ (\* (car l) (car l)) (sumsquare (cdr l))))))

(sumsquare '(1 2 4 8))

**Screenshot:**

