Problem Set 1 Solutions

COMP301 Fall 2023

Mehmet Murat Budak ID:78940

Problem 1

```
Welcome to DrRacket, version 8.10 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
> (+ 25 9 16)
50
> (/ 24 4)
6
> (+ (* 3 28) (- 2 2))
84
> (define a 8)
> (define b (+ a 7))
> (+ a b (* a b))
143
> (= a b)
#f
> (if (and (> b a) (< b (* a b)))
b
a)
15
> (cond ((= a 9) 6)
((= b 3) (+ 6 7 a))
(else 25))
25
> (+ 10 (if (> b a) b a))
25
> (* (cond ((> a b) a)
((< a b) b)
(else -1))
     (+ a 15))
345
>
```

Problem 2

Part A

```
;----- PART A -----;
(define idx_getter
  (lambda (L i) ; L: list, i: index
    (cond
      ((null? L) '())
      ((eq? i 0) (car L)) ;Found i'th index, return its element
      (else (idx_getter (cdr L) (- i 1))) ;Iterate through the list untill we reach the index
    )
  )
)

;(display(idx_getter '(5 8 6 91 87 12) 2))

;Returning range from i to j: Assuming the list structure in eopl is similar to linked lists,
; we would first have to truncate the list to i'th index,
; which can be done by modifying idx_getter to return the `List` instead of (car `List`)
; After getting the list truncated from the start, we would need to truncate its end (up to index j).
; For this I think we can iterate through the list untill we find the j'th index and set its cdr value to '().
; Apperently, the function set-cdr! which is used to set cdr is not allowed in eopl.
;
; Considering this constraint, we could create a new list and pop the values from i to j to the new list,
; but we haven't learned that functionality yet.

; IMPLEMENTATION:
(define initial-truncate
  (lambda (L i)
    (cond
      ((null? L) '())
      ((eq? i 0) L) ;Found i'th index, return the list
      (else (initial-truncate (cdr L) (- i 1) ) ) ;Iterate through the list untill we reach the index
    )
  )
)

(define latter-truncate
  (lambda (L j) ; L: list, i: index
    (cond
      ((null? L) '())
      ;((eq? j 0) (set-cdr!  )) ;Found j'th index, set its cdr to '() this is the problematic part
      (else (latter-truncate (cdr L) (- j 1))) ;Iterate through the list untill we reach the index
    )
  )
)

(define truncate-range
  (lambda (L i j)
    (latter-truncate (initial-truncate L i) (- j i))
  )
)
; (display (initial-truncate '(84 65 8 615 489 65 7) 4))

;----- END OF PART A -----;
```

```
> (idx_getter '(1 2 3 4 5 6) 0)
1
> (idx_getter '(1 1 2 3 5 8 13 21) 4)
5
> (idx_getter '() 0)
()
>
```

Part B

```
56 ;----- PART B -----;
57 (define part_b
58   (lambda (n)
59     (cond
60       ((< n 0) 0)
61       ((= n 0) 1)
62       ((> n 0) (+ 4(* (part_b (- n 1)) (part_b (- n 1))))) ; Series calculation
63     )
64   )
65 )
66
67 ;(display (part_b 3))
68
69 ;----- END OF PART B -----;
```

```
> (part_b 0)
1
> (part_b 1)
5
> (part_b 2)
29
> (part_b 3)
845
> (part_b 4)
714029
```

Part C

```scheme
;----- PART C -----;
(define is-prime?
  ;Stupid implementation: Check all remainders from n to sqrt(n)
  (lambda (n)
    (if (<= n 1)
        #f ;Return f
        (check-divisibility n 2) ;Start checking for divisibility starting from 2
    )
  )
)
(define check-divisibility
  (lambda (n div)
    (cond
      ((> div (sqrt n)) #t) ;No divisor found up to sqrt(n), so n is prime
       ((is-divisible? n div) #f) ;Found a divisor, so n is not prime
       (else (check-divisibility n (+ div 1))) ;Check for div+1
    )
  )
)

(define is-divisible?
  (lambda (n div) ;Check if n is divisible by div
    (= (remainder n div) 0)
  )
)

;(display (is-prime? 116239))

;----- END OF PART C -----;
```

```
> (is-prime? 2)
#t
> (is-prime? 0)
#f
> (is-prime? 7)
#t
> (is-prime? 11)
#t
> (is-prime? 12)
#f
```