TRABAJO PRÁCTICO N° 7

Interprete TLC LISP en LISP

Fecha de entrega: Fechas previas al final.
Formato: Documento digital en PDF, que incluya las consignas, los datos del alumno, licencia Creative Commons, con permisos de modificación e impresión, cuyo nombre debe ser "Nombre_Apellido–TP7.pdf" enviado a consultas@profmatiasgarcia.com.ar el cual debe tener como asunto "SEMINARIO TN - tu Nombre y Apellido – TP 7".
Ayuda: En caso de no comprender alguna consigna o tener dudas, puede solicitarse asistencia enviando un email a consultas@profmatiasgarcia.com.ar con el asunto "SEMINARIO TN Nombre y Apellido TP 7 CONSULTA".
Modalidad: Individual

Interprete de TLC Lisp en Lisp

Desarrollar en Lisp un intérprete para poder ejecutar programas desarrollados en el mismo lenguaje.

Ejemplos para realizar pruebas:

;numeros

(eval '2 nil) `> ------> 2`

;true false

(eval nil nil) `> ------> nil`

(eval 'nil nil) `> ------> nil`

(eval 't nil) `> ------> t`

;asociaciones en el ambiente

(eval 'A '(A 2)) `> ------> 2`

(eval 'B '(A 2 B 10) `> ------> 10`

;quote

(eval '(quote A) nil) `> ------> A`

(eval '(quote 1) nil) `> ------> 1`

(eval '(quote (car a)) nil) `> ------> (car a)`

(eval '(quote ((2 3) (4 5)))) `> ------> ((2 3) (4 5))`

;and y or

(eval '(and (or t nil) t) nil) `> ------> t`

(eval '(and (or t nil) (or nil nil)) nil) `> ------> nil`

(eval '(or (or t nil) (or nil nil)) nil) `> ------> t`

;car + ambiente

(eval '(car (list a 2 3)) '(a 100)) `> ------> 100`

;cdr + ambiente

(eval '(cdr (list a b c)) '(a 100 b 99 c 98)) `> ------> (99 98)`

;lambda

(eval '((lambda (x) (* x 2)) 2) nil) `> ------> 4`

(eval '((lambda (x y) (+ (* x 2) y)) 2 4) nil) `> ------> 8`

(eval '(lambda (x) (* x 2)) nil) `> ------> (lambda (x) (* x 2))`

(eval '(mapcar (lambda (x) (cons x (cdr '(3 4 5)))) '(1 2 3)) nil) `> ------> ((1 4 5) (2 4 5) (3 4 5))`

;mapcar

(eval '(mapcar 'numberp (quote (4)))) '(t))

(eval '(mapcar 'numberp (quote (4 5 6 nil)))) `> ------> (t t t nil)`

(eval '(mapcar 'car (quote ((2 3) (4 5))))) `> ------> (2 4)`

```
;reverse
(eval '(reverse (quote (4 5 6 7))) nil) > ------> (7 6 5 4)

;funciones
(eval
    '(my_fun 10)
    '( my_fun (lambda (x) (* x 2)))
)

> ------> 2

(eval
    '(mapcar 'my_fun (quote (10 2 3 4)))
    '( my_fun (lambda (x) (* x 2)) )
)

> ------> (20 4 6 8 )

(eval
    '(mapcar 'my_fun (quote (a b c d)))
    '(my_fun (lambda (x) (* x 2))  a 10  b 20  c 30  d 40)
)

> ------> (20 40 60 80)
```

```
;funciones recursivas


(eval '(fact (car (quote (4 5 6))))
      '(fact  (lambda (n) (if (eq n 0) 1
                              (* n (fact (- n 1)))))  )
)
```

```
> ------> 24
```

```
 (eval '(fact 5) '(fact
          (lambda (n)
                    (if (eq n 0) 1
                     (* n (fact (- n 1)))
                    )
                    )
          )
)
```

```
> ------> 120
```