Informe de Taller # 2

```
Wilson Andres Martinez Rivera    2266319
Juan David Camacho Cataño 2266292
Juan Gabriel Paredes Chaves 2266183
```

```
;;Expresiones condicionales
(expresion ("empty") list-empty-exp)
(expresion ("cons" "(" expresion expresion ")") list-cons-exp)
(expresion ("length" "(" expresion ")") list-length-exp)
(expresion ("first" "(" expresion ")") list-first-exp)
(expresion ("rest" "(" expresion ")") list-rest-exp)
(expresion ("nth" "(" expresion "," expresion ")") list-nth-exp)

;;Expresion Cond
(expresion ("cond" (arbno expresion "==>" expresion) "else" "==>" expresion "end") cond-exp)
```

Nueva implementación de la gramática de las listas con la variaciones de (empty), (cons), (length), (first), (rest), (nth), ademas se pone la expresion de Cond con el resplado de else.

```
; Datatype para listas personalizadas
(define-datatype lista lista?
  (lista-vacia)
  (lista-cons
   (primer-elemento number?)
   (resto-lista lista?)))

; Funciones auxiliares de lista
(define es-lista-vacia?
  (lambda (lst)
    (cases lista lst
      (lista-vacia () #t)
      (lista-cons (p r) #f))))

(define obtener-primer-elemento
  (lambda (lst)
    (cases lista lst
      (lista-vacia () (eopl:error "No se puede obtener el primer elemento de una lista vacía"))
      (lista-cons (p r) p))))

(define obtener-resto
  (lambda (lst)
    (cases lista lst
      (lista-vacia () (eopl:error "No se puede obtener el resto de una lista vacía"))
      (lista-cons (p r) r))))

(define obtener-longitud
  (lambda (lst)
    (cases lista lst
      (lista-vacia () 0)
      (lista-cons (p r) (+ 1 (obtener-longitud r))))))
```

```
(define obtener-nth
  (lambda (lst n)
    (cond
      [(es-lista-vacia? lst) (eopl:error "Índice fuera de rango")]
      [(= n 0) (obtener-primer-elemento lst)]
      [else (obtener-nth (obtener-resto lst) (- n 1))])))
```

Aquí está la implementación de la definición de las expresiones de las listas funciona de una manera individual.

```
;;Evaluar expresion
(define evaluar-expresion
 (lambda (exp amb)
   (cases expresion exp
     ;;Casos base
     (lit-exp (dato) dato)
     (var-exp (id) (apply-env amb id))
     (true-exp () #t)
     (false-exp () #f)
     ;;Nuevos casos de lista
     (list-empty-exp () (lista-vacia))

     (list-cons-exp (elem lista)
       (let ([elem-evaluando (evaluar-expresion elem amb)]
             [lista-evaluada (evaluar-expresion lista amb)])
         (if (lista? lista-evaluada)
             (lista-cons elem-evaluando lista-evaluada)
             (eopl:error "El segundo argumento de cons no es una lista" lista-evaluada ))))

     (list-length-exp (lst)
       (let ([lista-evaluada (evaluar-expresion lst amb)])
         (obtener-longitud lista-evaluada)))

     (list-first-exp (lst)
       (let ([lista-evaluada (evaluar-expresion lst amb)])
         (obtener-primer-elemento lista-evaluada)))

     (list-rest-exp (lst)
       (let ([lista-evaluada (evaluar-expresion lst amb)])
         (obtener-resto lista-evaluada)))
```

```
(list-nth-exp (lst n)
  (let ([lista-evaluada (evaluar-expresion lst amb)]
        [n-evaluado (evaluar-expresion n amb)])
    (obtener-nth lista-evaluada n-evaluado)))

;;Caso cond
(cond-exp (condiciones-expresiones else-palabra else-expresion)
  (let loop ([pairs condiciones-expresiones])
    (cond
      [(null? pairs) (evaluar-expresion else-expresion amb)]
      [else
      (let* ([condicion (car pairs)]
             [expresion (cadr pairs)]
             [resultado-condicion (evaluar-expresion condicion amb)])
        (if (not (= resultado-condicion 0))
            (evaluar-expresion expresion amb)
            (loop (cddr pairs))))])))
```

Aquí se planteó la expresión en la cual se evalúa cada acción gramatical, nueva implementada por medio de las listas.

Estos fueron los únicos cambios que se hizo en el interpretador de asignación y se paso el interpretador de clase.

```
-->cons(1 cons(2 cons(3 empty)))
#(struct:lista-cons 1 #(struct:lista-cons 2 #(struct:lista-cons 3 #(struct:lista-vacia))))
-->empty
#(struct:lista-vacia)
-->cons(cons(1 empty) cons(2 empty))
lista-cons: bad value for primer-elemento field: #(struct:lista-cons 1 #(struct:lista-vacia))
```

Ejercicio

```
-->cons(let x = 3 in x cons(3 empty))
#(struct:lista-cons 3 #(struct:lista-cons 3 #(struct:lista-vacia)))
-->cons(1 2)
El segundo argumento de cons no es una lista 2
```

```
-->cond -(x, 1) ==> 1 -(x, 2) ==> 2 -(x, 3) ==> 4 else ==> 9 end
2
-->cond else ==> 9 end
9
-->
```

Ejemplo de lista anidada

```
#lang eopl.rkt - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
#lang eopl.rkt (define ...)  Check Syntax  Debug  Macro Stepper  Run  Stop
1  #lang eopl
136                                                (buscar-variable (cdr lid) vec (+ pos 1)  )]
137                                                )
138                                              )
139                                            )
140                                  )
141                            (buscar-variable lid vec 0)
142                            )
143
144                          )
145
146        )
147      )
148    )
149
150  (define ambiente-inicial
151    (ambiente-extendido '(x y z) '(4 2 5)
152                        (ambiente-extendido '(a b c) '(4 5 6)
153                                            (ambiente-vacio))))

Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->cons (2 cons(7 cons (9 cons (12 empty))))
(2 7 9 12)
-->
eof
```

Ejemplo de listas first

```
#lang eopl.rkt - DrRacket*
File Edit View Language Racket Insert Scripts Tabs Help
#lang eopl.rkt (define ...)  Check Syntax  Debug  Macro Stepper  Run  Stop
1  #lang eopl
358      (primitiva-setref! ref val)))
359
360  (define primitiva-setref! imported from eopl
361    (lambda (ref val)
362      (cases referencia ref
363        (a-ref (pos vec)
364               (vector-set! vec pos val)))))
365
366
367  ;;Interpretador
368  (define interpretador
369    (sllgen:make-rep-loop "-->" evaluar-programa
370                          (sllgen:make-stream-parser
371                           especificacion-lexica especificacion-gramatical)))
372
373
374  (interpretador)

Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->first (cons (10 cons (8 cons (12 empty))))
10
-->
eof
```

Ejemplo de listas anidadas

#lang eopl.rkt - DrRacket*

File Edit View Language Racket Insert Scripts Tabs Help

#lang eopl.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang eopl
    (primitiva-setref! ref val)))

(define primitiva-setref!
  (lambda (ref val)
    (cases referencia ref
      (a-ref (pos vec)
             (vector-set! vec pos val)))))


;;Interpretador
(define interpretador
  (sllgen:make-rep-loop "-->" evaluar-programa
                        (sllgen:make-stream-parser
                         especificacion-lexica especificacion-gramatical)))


(interpretador)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->cons (100 cons (2 cons(9 empty)))
(100 2 9)
-->
```

eof

Ejemplo de lista con elementos vacíos

#lang eopl.rkt - DrRacket*

File Edit View Language Racket Insert Scripts Tabs Help

#lang eopl.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang eopl
    (primitiva-setref! ref val)))

(define primitiva-setref!
  (lambda (ref val)
    (cases referencia ref
      (a-ref (pos vec)
             (vector-set! vec pos val)))))


;;Interpretador
(define interpretador
  (sllgen:make-rep-loop "-->" evaluar-programa
                        (sllgen:make-stream-parser
                         especificacion-lexica especificacion-gramatical)))


(interpretador)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->cons (3 cons (empty cons(8 empty)))
(3 () 8)
-->
```

eof

Ejemplo de lista simple

File Edit View Language Racket Insert Scripts Tabs Help

#lang eopl.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang eopl
                        (cons head-val tail-val)
                        (eopl:error "El segundo argumento de cons debe ser una lista" tail-val))))
        ;; Expresión para lista vacia
        (list-empty-exp () '())
        (lit-exp (dato) dato)
        (var-exp (id) (apply-env amb id))
        ;;Booleanos
        (true-exp () #true)
        (false-exp () #false)
        ;;Fin booleanos
        (prim-exp (prim args)
                  (let
                    (
                     (lista-numeros (map (lambda (x) (evaluar-expresion x amb)) args))
                     )
                   (evaluar-primitiva prim lista-numeros)
                   )
                  )
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->cons (7 cons(9 empty))
(7 9)
-->
```

eof

Ejemplo de numero en la lista

#lang eopl.rkt - DrRacket*

File Edit View Language Racket Insert Scripts Tabs Help

#lang eopl.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang eopl
    (primitiva-setref! ref val)))

(define primitiva-setref!
  (lambda (ref val)
    (cases referencia ref
      (a-ref (pos vec)
             (vector-set! vec pos val)))))


;;Interpretador
(define interpretador
  (sllgen:make-rep-loop "-->" evaluar-programa
                        (sllgen:make-stream-parser
                         especificacion-lexica especificacion-gramatical)))


(interpretador)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->length (cons (1 cons(2 empty)))
2
-->
```

eof

Ejemplo de lista rest

#lang eopl.rkt - DrRacket*

File Edit View Language Racket Insert Scripts Tabs Help

#lang eopl.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang eopl
    (primitiva-setref! ref val)))

(define primitiva-setref!
  (lambda (ref val)
    (cases referencia ref
      (a-ref (pos vec)
             (vector-set! vec pos val)))))


;;Interpretador
(define interpretador
  (sllgen:make-rep-loop "-->" evaluar-programa
                        (sllgen:make-stream-parser
                         especificacion-lexica especificacion-gramatical)))


(interpretador)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->zest (cons (7 cons (3 empty)))
(3)
-->
```

eof

Ejemplo de lista en vacío.

#lang eopl.rkt - DrRacket*

File Edit View Language Racket Insert Scripts Tabs Help

#lang eopl.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang eopl
    (primitiva-setref! ref val)))

(define primitiva-setref!
  (lambda (ref val)
    (cases referencia ref
      (a-ref (pos vec)
             (vector-set! vec pos val)))))


;;Interpretador
(define interpretador
  (sllgen:make-rep-loop "-->" evaluar-programa
                        (sllgen:make-stream-parser
                         especificacion-lexica especificacion-gramatical)))


(interpretador)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: eopl, with debugging; memory limit: 128 MB.
-->empty
()
-->
```

eof

Ejemplo de lista con cond else

```
  )

;;Definiciones para los procedimientos
(define-datatype procval procval?
  (closure (lid (list-of symbol?))
           (body expresion?)
           (amb-creation ambiente?)))

;;Referencias

(define-datatype referencia referencia?
  (a-ref (pos number?)
         (vec vector?)))

;;Extractor de referencias
(define deref
  (lambda (ref)
    (primitiva-deref ref)))

(define primitiva-deref
  (lambda (ref)
    (cases referencia ref
      (a-ref (pos vec)
             (vector-ref vec pos)))))

;;Asignación/cambio referencias
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Racket Output

```
PS C:\Users\usuario> racket  'c:/Users/usuario/Desktop/Taller #2/Taller #2/interpretadorClase.rkt'
-->cond else ==> 20 end
20
-->
```

Ejemplo de lista de cond con condicionales

```
C: > Users > usuario > Desktop > Taller #2 > Taller #2 > interpretadorClase.rkt
363 (define operacion-prim
364   (lambda (ival op term)
374   )
375
376 ;;Definiciones para los procedimientos
377 (define-datatype procval procval?
378   (closure (lid (list-of symbol?))
379            (body expresion?)
380            (amb-creation ambiente?)))
381
382 ;;Referencias
383
384 (define-datatype referencia referencia?
385   (a-ref (pos number?)
386          (vec vector?)))
387
388 ;;Extractor de referencias
389 (define deref
390   (lambda (ref)
391     (primitiva-deref ref)))
392
393 (define primitiva-deref
394   (lambda (ref)
395     (cases referencia ref
396       (a-ref (pos vec)
397              (vector-ref vec pos)))))
398
399 ;;Asignación/cambio referencias
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS        Racket Output

PS C:\Users\usuario> racket  'c:/Users/usuario/Desktop/Taller #2/Taller #2/interpretadorClase.rkt'
-->cond +(x, 5) ==> 5 +(x, 10) ==> 10 else ==> 15 end
14
-->
```

Ejemplo de listas con múltiples cond

```
363 (define operacion-prim
364   (lambda (ival op term)
374   )
375
376 ;;Definiciones para los procedimientos
377 (define-datatype procval procval?
378   (closure (lid (list-of symbol?))
379            (body expresion?)
380            (amb-creation ambiente?)))
381
382 ;;Referencias
383
384 (define-datatype referencia referencia?
385   (a-ref (pos number?)
386          (vec vector?)))
387
388 ;;Extractor de referencias
389 (define deref
390   (lambda (ref)
391     (primitiva-deref ref)))
392
393 (define primitiva-deref
394   (lambda (ref)
395     (cases referencia ref
396       (a-ref (pos vec)
397              (vector-ref vec pos)))))
398
399 ;;Asignación/cambio referencias
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS        Racket Output

PS C:\Users\usuario> racket  'c:/Users/usuario/Desktop/Taller #2/Taller #2/interpretadorClase.rkt'
-->cond +(x, 4) ==> 4 +(x, 5) ==> 5 +(x, 6) ==> 6 else ==> 20 end
9
-->
```