75 - 14

Lenguajes Formales

Trabajos Prácticos

Alumno: Bello Camilletti, Nicolás
Padrón: 86676
Fecha de final: 7/5/2010

Indice

Interprete de C

```
; Problema : Interprete de C.
; Lenguajes Formales - Primer Cuatrimestre 2010
; Alumno : Bello Camilletti, Nicolás.
; Padrón : 86676

;------ Validaciones -------
(defun esVariable (var mem)
   (pertenece_ListaPares var mem)
)

(defun esAsignacion (expr memoria)
   (if (esVariable (car expr) memoria)
      t
      (and (or (equal (car expr) '++) (equal (car expr) '--))
         (esVariable (cadr expr) memoria)
      )
   )
)

(defun esFuncion (prg f)
   (equal (caar prg) f)
)

(defun isValidType (type)
   (cond
      ((eq type 'int) T)
      ((eq type 'long) T)
      ((eq type 'float) T)
      ((eq type 'double) T)
      (T nil)
   )
)

(defun isVarDef (prg)
   (if (>= (length (car prg)) 2)
      (isValidType (caar prg) )
      nil
   )
)

;----- Fin Validaciones -------

;-------- Operaciones de memoria ---------
(defun buscar (elemento listaPares)
   (if (null listaPares)
      nil
      (if (equal elemento (caar listaPares))
         (cadar listaPares)
         (buscar elemento (cdr listaPares))
      )
   )
)
```

```
(defun pertenece_ListaPares (elemento listaPares)
   (if (null listaPares)
      nil
      (if (equal elemento (caar listaPares) )
         t
         (pertenece_ListaPares elemento (cdr listaPares))
      )
   )
)

(defun modificar_Valor_Var_En_Mem (nombreElemento NuevoValor listaPares)
   (if (null listaPares)
      nil
      (if (equal nombreElemento (caar listaPares) )
         (cons (list nombreElemento NuevoValor ) (cdr listaPares) )
         (cons (car listaPares ) (modificar_Valor_Var_En_Mem nombreElemento NuevoValor (cdr listaPares)))
      )
   )
)

(defun addPairKeyValue (key value mem)
   (cons (list key value ) mem )
)

(defun agregar_Var_Con_Asignacion (var valor mem)
   (addPairKeyValue var (evaluar valor mem) mem)
)

(defun agregar_Var_A_Mem (var mem)
   (if (null var)
      mem
      (if (equal (length var) 1)
         (addPairKeyValue (car var) 0 mem )
         (if (equal (nth 1 var) '= )
            (agregar_Var_Con_Asignacion (car var) (caddr var) (agregar_Var_A_Mem (cdddr var) mem))
            (addPairKeyValue (car var) 0 (agregar_Var_A_Mem (cdr var) mem))
         )
      )
   )
)

;-------- Fin Operaciones de memoria ---------
;----- Evaluar -------
(defun esOperadorLisp (el)
   (cond
      ((equal el '+) t)
      ((equal el '*) t)
      ((equal el '-) t)
      ((equal el '/) t)
      ((equal el '<) t)
      ((equal el '>) t)
      ((equal el '<=) t)
      ((equal el '>=) t)
      ((equal el 'and) t)
      ((equal el 'or) t)
      ( t nil)
   )
)
```

```
(defun prioridad_op (op)
   (cond
      ((equal op '+) 2)
      ((equal op '*) 3)
      ((equal op '-) 2)
      ((equal op '/) 3)
      ((equal op '<) 1)
      ((equal op '>) 1)
      ((equal op '<=) 1)
      ((equal op '>=) 1)
      ( t 0)
   )
)

(defun componer_expresion (ops vars)
   (if (null ops)
      vars
      (componer_expresion (cdr ops) (cons (list (car ops) (cadr vars) (car vars)) (cddr vars) ))
   )
)

(defun inf_a_pref (expr &optional (ops nil) (vars nil))
   (if (null expr)
      (componer_expresion ops vars)
      (if (esOperadorLisp (car expr))
         (if (> (length ops) 0)
            (if (< (prioridad_op (car expr)) (prioridad_op (car ops)))
               (inf_a_pref (cdr expr) (cons (car expr)(cdr ops))
                        (cons (list (car ops) (cadr vars) (car vars)) (cddr vars) )
               )
               (inf_a_pref (cdr expr) (cons (car expr) ops) vars)
            )
            (inf_a_pref (cdr expr) (cons (car expr) ops) vars)
         )
         (if (atom (car expr))
            (inf_a_pref (cdr expr) ops (cons (car expr) vars))
            (inf_a_pref (cdr expr) ops (cons (car (inf_a_pref (car expr))) vars))
         )
      )
   )
)

(defun filtrar_nil_t (resultado)
   (cond
      ((equal resultado nil) 0)
      ((equal resultado t) 1)
      (t resultado)
   )
)

(defun operar_lista (lista)
   (if (null lista)  nil
      (if (atom (car lista))
         (eval lista)
         (operar_lista (car lista))
      )
   )
)
```

```
(defun operar (lista)
   (if (eq (length lista) 1)
      (if (atom (car lista))
         (car lista); tiene que ser variable o cte.
         (filtrar_nil_t (operar_lista (inf_a_pref lista)))
      )
      (filtrar_nil_t (operar_lista (inf_a_pref lista)))
   )
)

(defun esOperadorEnC (op)
   (cond
      ((equal op '+) t)
      ((equal op '*) t)
      ((equal op '-) t)
      ((equal op '/) t)
      ((equal op '<) t)
      ((equal op '>) t)
      ((equal op '<=) t)
      ((equal op '>=) t)
      ((equal op '&&) t)
      ((equal op '||) t)
      ( t nil)
   )
)

(defun traducirOp (op)
   (cond
      ((equal op '&&) 'and)
      ((equal op '||) 'or)
      ( t op)
   )
)

(defun getVarValue (varName mem)
   (buscar varName mem)
)

;Arma una expreción algebraica en base de la sentencia, donde figuran variables y otros, que las busca en memoria.
(defun armar_Expresion (sentencia mem)
   (if (null sentencia)
      nil
      (cond
         ;Si es parentesis, hay que evaluar lo de adentro y dejar la misma estructura
         ( (not (atom (car sentencia))) (cons (list (armar_Expresion (car sentencia) mem)) (armar_Expresion (cdr sentencia) mem))
)
         ;Si es variable, hay que buscar en memoria el valor
         ( (esVariable (car sentencia) mem) (cons (getVarValue (car sentencia) mem) (armar_Expresion (cdr sentencia) mem)) )
         ;Si es un operador traducirlo a los de lisp
         ( (esOperadorEnC (car sentencia)) (cons (traducirOp (car sentencia)) (armar_Expresion (cdr sentencia) mem)) )
         ;Caso contrario, es constante( u operacion, que para esta instancia resulta como si fuese constante).
         ( T (cons (car sentencia) (armar_Expresion (cdr sentencia) mem)) )
      )
   )
)
```

```
(defun evaluar (sentencia mem)
   ;(filtrar_nil_t (evaluar_lisp prg mem))
   (if (atom sentencia)
      sentencia
      (operar (armar_Expresion sentencia mem))
   )

)

;------- Fin Evaluar -------

;------ Funciones auxiliares para ejecutar ------
(defun asignacion (expr memoria)
   (if (esVariable (car expr) memoria)
      (cond
         ( (equal (nth 1 expr) '=) (modificar_Valor_Var_En_Mem (car expr) (evaluar (cddr expr) memoria) memoria))
         ( (equal (nth 1 expr) '++) (asignacion (list (car expr) '= (car expr) '+ 1 ) memoria))
         ( (equal (nth 1 expr) '--) (asignacion (list (car expr) '= (car expr) '- 1 ) memoria))
         ( t (asignacion (list (car l) '= (car l) (nth 1 expr) (nth 3 expr)) memoria))
      )
      (asignacion (reverse expr) memoria)
   )
)

(defun procesar_if (prg mem entrada salida)
   (if (not (equal (evaluar (cadar prg) mem) 0))
      (ejecutar (append (list (nth 2 (car prg))) (cdr prg)) mem entrada salida)
      (if (equal (length (car prg)) 5)
         (ejecutar (append (list (nth 4 (car prg))) (cdr prg)) mem entrada salida)
         (ejecutar (cdr prg) mem entrada salida)
      )
   )
)

(defun procesar_while (prg mem entrada salida)
   (if (not (equal (evaluar (nth 1 (car prg)) mem) 0))
      (ejecutar (append (cddar prg) prg) mem entrada salida)
      (ejecutar (cdr prg) mem entrada salida)
   )
)

(defun procesar_for (prg mem entrada salida)
   (ejecutar
      (append
         (list (car (nth 1 (car prg)))
            (cons 'while
               ( cons (cadr (nth 1 (car prg)))
                  (append (cddar prg) (list (caddr (nth 1 (car prg))) ) )
               )
            )
         )
         (cdr prg)
      )
      mem entrada salida
   )
)
```

```
(defun procesar_scanf (prg mem entrada salida)
   (ejecutar (cdr prg) (modificar_Valor_Var_En_Mem (cadar prg) (car entrada) mem) (cdr entrada) salida)
)

(defun procesar_printf (prg mem entrada salida)
   (ejecutar (cdr prg) mem entrada (append salida (list (evaluar (cdar prg) mem))) )
)

(defun procesar_varDef (prg mem entrada salida)
   (if (pertenece_ListaPares(cadar prg) mem)
      'Error
      (ejecutar (cdr prg) (agregar_Var_A_Mem (cdar prg) mem) entrada salida)
   )
)

;------ Fin Funciones auxiliares para ejecutar ------

(defun ejecutar (prg mem &optional (entrada nil) (salida nil))
   (if (null prg)
      salida
      (cond
         ( (esFuncion prg 'scanf) (procesar_scanf prg mem entrada salida) )
         ( (esFuncion prg 'printf) (procesar_printf prg mem entrada salida) )
         ( (isVarDef prg) (procesar_varDef prg mem entrada salida) )
         ( (esAsignacion (car prg) mem) (ejecutar (cdr prg) (asignacion (car prg) mem) entrada salida) )
         ( (esFuncion prg 'if) (procesar_if prg mem entrada salida) )
         ( (esFuncion prg 'while) (procesar_while prg mem entrada salida) )
         ( (esFuncion prg 'for) (procesar_for prg mem entrada salida) )
         ( t (list 'syntax_error (car prg)) )
      )
   )
)

(defun run (prg &optional (entrada nil) (mem nil) )
   (if (null prg)
      nil
      (cond
         ( (isVarDef prg) (run (cdr prg) entrada (agregar_Var_A_Mem (cdar prg) mem)) )
         ( (eq (caar prg) 'main) (ejecutar (cadar prg) mem entrada) )
         ( t 'Error )
      )
   )
)
```

Test

```
'(----- Load utils for tests -----)

(defun test (name actual expected)
   (if (equal actual expected)
      (list name 'passed)
      (list name 'fail '=> 'expected expected 'actual actual)
   )
)

'(----- Inicio tests -----)
(test 'Buscar_En_Memoria_Esta_Solo
   (buscar 'a '((a 2)))
   '2
)

(test 'Buscar_En_Memoria_Esta_Con_Otro
   (buscar 'a '( (b 3) (a 2)))
   '2
)

(test 'Buscar_En_Memoria_No_Esta
   (buscar 'c '( (b 3) (a 2)))
   nil
)

(test 'Pertenece_ListaPares_Esta_Solo
   (pertenece_ListaPares 'a '((a 2)))
   t
)

(test 'Pertenece_ListaPares_Esta_Con_Otro
   (pertenece_ListaPares 'a '( (b 3) (a 2)))
   t
)

(test 'Pertenece_ListaPares_No_Esta
   (pertenece_ListaPares 'c '( (b 3) (a 2)))
   nil
)

(test 'Modificar_Esta_Solo
   (modificar_Valor_Var_En_Mem 'a 5 '( (a 2)))
   '( (a 5))
)

(test 'Modificar_Esta_Con_Otro
   (modificar_Valor_Var_En_Mem 'a 5 '( (b 3) (a 2)))
   '( (b 3) (a 5))
)

(test 'Modificar_No_Esta
   (modificar_Valor_Var_En_Mem 'c 2 '( (b 3) (a 2)))
   '( (b 3) (a 2))
)
```

```
(test 'Agregar_No_Esta
   (agregar_Var_A_Mem '(c = 2) '( (b 3) (a 2)))
   '( (c 2) (b 3) (a 2))
)

(test 'Agregar_Esta
   (agregar_Var_A_Mem '(a = 4) '( (b 3) (a 2)))
   '( (a 4) (b 3) (a 2))
)

(test 'var_def_ok
   (isVarDef '((int i)) )
   t
)

(test 'var_def_no_name
   (isVarDef '((int)) )
   nil
)

(test 'var_def_no_type
   (isVarDef '((i)) )
   nil
)

(test 'var_def_wrong_type
   (isVarDef '((integer i)) )
   nil
)

'(----- Fin tests -----)
```

Ejemplo

```
'(----- Ejecución completa -----)
'(La salida debe ser "hola" "cond_true" 21 A B 12)

(run '(
      (int n j = 10 i k)
      (main(
         (i = 1)
         (while (i <= 10)
            (i ++)
         )
         (for ( (int w = 0) (w <= 10) ( w ++) )
            (i ++)
         )
         (-- i)
         (i --)
         (printf "hola")
         (if (1 && 2) (printf "cond_true") else (printf "cond_false"))
         (printf (i + 1))
         (scanf n)
         (printf n)
         (scanf n)
         (printf n)
         (k = ((1 + 2) * ( 3 + 1)))
         (printf k)
         )
      )
   ) ;codigo
   '(A B) ;entrada
)
```

Problema de las N reinas

Solución 1

```
; Problema : N reinas, solución planteada por la profesora, con algunas mejoras minimas.(Llega a 15 reinas. Apartir de las 14, funciona mas lento)
; Explicación : Esta solución está basada en mantener un tablero con las posibles reinas en él, y descartar posiciones según las reinas que se vayan eligiendo. Aparte se guardan un historial de las reinas para poder volver para atras.
; Lenguajes Formales - Primer Cuatrimestre 2010
; Alumno : Bello Camilletti, Nicolás.
; Padrón : 86676

(defun eliminarColumna (elemento filaTab)
   (if (null filaTab)
      nil
      (if (or (or (eq (apply '+ elemento) (apply '+ (car filaTab)))
                  (eq (apply '- elemento) (apply '- (car filaTab)))
              (= (cadar filaTab) (cadr elemento) )) )
         (eliminarColumna elemento (cdr filaTab))
         (cons (car filaTab) (eliminarColumna elemento (cdr filaTab)) )
      )
   )
)

(defun eliminaPorFila (elemento tab )
   (if (null tab)
      nil
      (cons (eliminarColumna elemento (car tab) )
            (eliminaPorFila elemento (cdr tab))
      )
   )
)

(defun eliminanulos (lista)
   (if (null lista)
      nil
      (if (null (car lista))
         (eliminanulos (cdr lista))
         (cons (car lista) (eliminanulos (cdr lista)))
      )
   )
)

(defun elimtodoslospares(listapos tab)
   (if (null listapos)
      (eliminanulos tab)
      (elimtodoslospares (cdr listapos) (eliminaPorFila (car listapos) (cdr tab)))
   )
)

(defun buscarLong2oMas (L)
   (if (null L)    nil
      ( if (>= (length (car L)) 2)
         L
         (buscarLong2oMas (cdr L))
      )
   )
)
```

Solución 2

```
; Problema : N reinas, solución propia.(Llega a 17 reinas. Apartir de las 14, funciona mas lento)
; Explicación : Esta solución está basada en validar cada reina individualmente contra el resto, esto hace que solo se mantenga en memoria la solución actual, ni siquiera se mantiene el tablero, solo la fila actual de donde se quiere sacar la nueva reina.
; Lenguajes Formales - Primer Cuatrimestre 2010
; Alumno : Bello Camilletti, Nicolás.
; Padrón : 86676

(defun validarReinaColFilDiag (reina nuevaReina)
   (if (eq (car reina) (car nuevaReina))
      nil
      (if (eq (cadr reina) (cadr nuevaReina))
         nil
         (if (or (eq (apply '+ reina) (apply '+ nuevaReina))
                 (eq (apply '- reina) (apply '- nuevaReina)))
            nil
            T
         )
      )
   )
)

;Valida la nueva reina contra las ya existentes.
(defun validarReinas (reinas nuevaReina n)
   (if (null reinas)
      T
      (if (validarReinaColFilDiag (car reinas) nuevaReina )
         (validarReinas (cdr reinas) nuevaReina n)
         nil
      )
   )
)

(defun crearFila (maxCol &optional (fila '1) (minCol '1))
   (if (> minCol maxCol)
      nil
      (cons (list fila minCol) (crearFila maxCol fila (+ minCol '1) ))
   )
)

;Busca un elemento para el cual se puede ir al siguiente, osea no sea la ultima columna
(defun buscarLong2oMas (L N)
   (if (null L)
      nil
      ( if (>= (cadar L) N )
         (buscarLong2oMas (cdr L) N)
         L
      )
   )
)
```

```
;intenta agregar una reina de la lista de posibles reinas.
(defun agregarReina (reinas nuevasReinas n)
   (if (null nuevasReinas)
      (cons nil reinas)
      (if (validarReinas reinas (car nuevasReinas) n)
         (cons (car nuevasReinas) reinas)
         (agregarReina reinas (cdr nuevasReinas) n)
      )
   )
)

;intenta agregar reinas hasta que tenga que volver para atras.
(defun ReinasAux (N posreinas )
   (If (null (car posreinas))
      posreinas
      (if (eq (length posreinas) N)
         posreinas
         (if (< (length posreinas) (caar posreinas))
            (cons nil posreinas)
            (ReinasAux N (agregarReina posreinas (crearFila N (+(length posreinas) 1) ) N ) )
         )
      )
   )
)

;crea el entorno para volver a agregar una reina cuando se descarta la elegida
(defun reinasGoingBack (n posreinas)
   (agregarReina (cdr posreinas)  (crearFila N (caar posreinas) (+ (cadar posreinas) 1) ) n )
)

;Verifica si tiene que volver atras agregando las reinas, y descartar la ultima elegida.
(defun checkIfHaveToGoBack (N posreinas )
   (If (null (car posreinas)) ;tengo que volver atras
      (checkIfHaveToGoBack N (reinasGoingBack N  (buscarLong2oMas (cdr posreinas) N)))
      (if (eq (length posreinas) N) ; Termine
         posreinas
         (if (< (length posreinas) (caar posreinas)) ; No tengo posiblidad de llegar a las N reinas.
            (checkIfHaveToGoBack N (reinasGoingBack N  (buscarLong2oMas (cdr posreinas) N)))
            posreinas
         )
      )
   )
)

;Devuelve las reinas pero en orden inverso.
(defun ReverseReinas (N &optional (posreinas '((1 1)) ) )
   (if (eq (length posreinas) N )
      posreinas
      (ReverseReinas N (checkifhaveTogoBack N (ReinasAux N posreinas)))
   )
)

;Main function
(defun Reinas (N )    (print (reverse (ReverseReinas N ))))

;Ejemplo de uso
(reinas 17)
```

Lisp en Lisp

```
; Problema : Lisp En Lisp.
; Lenguajes Formales - Primer Cuatrimestre 2010
; Alumno : Bello Camilletti, Nicolás.
; Padrón : 86676


;---------- Funciones para manejo del ambiente ----------
;obtiene el valor del elemento en el ambiente, o nil si no está.
(defun get_From_Env (elem env)
   (if (or (null env) (null elem))
      nil
      (if (eq (caar env) elem)
         (cadar env)
         (get_From_Env elem (cdr env))
      )
   )
)

;obtiene t si el elemento está en el ambiente, o nil en caso contrario.
(defun is_In_Env (elem env)
   (if (or (null env) (null elem))
      nil
      (if (eq (caar env) elem)
         T
         (is_In_Env elem (cdr env))
      )
   )
)

;agrega o reemplaza el valor de un elemento en el ambiente
(defun replace_or_add (env param new_value)
   (if (null env)
      (list (list param new_value))
      (if (eq (caar env) param)
         (cons (list param new_value) (cdr env))
         (cons (car env) (replace_or_add (cdr env) param new_value))
      )
   )
)

;expande el ambiente
(defun expand_env (env params vals)
   (if (null params)
      env
      (expand_env (replace_or_add env (car params) (car vals)) (cdr params) (cdr vals))
   )
)

;---------- Fin Funciones para manejo del ambiente ----------
```

```
;---------- Funciones auxiliares para exec ----------

;evalua una expresion quote
;(quote expresion)
(defun exec_quote (code)
   (cadr code)
)

;evalua una expresion que es un atomo
;atomo
(defun exec_atom (the_atom env)
   (if (is_In_Env the_atom env)
      (get_From_Env the_atom env)
      the_atom
   )
)

;evalua una expresion or
;(or expr1 expr2)
(defun exec_or (code env)
   (if (exec (cadr code) env)
      t
      (exec (caddr code) env)
   )
)

;evalua una expresion and
;(and expr1 expr2)
(defun exec_and (code env)
   (if (exec (cadr code) env)
      (exec (caddr code) env)
      nil
   )
)

;evalua una expresion if
;(if expresion true-code false-code)
(defun exec_if (code env)
   (if (exec (cadr code) env)
      (exec (caddr code) env)
      (exec (cadddr code) env)
   )
)

;evalua una lista de expresion del cond con el sig formato
;((expr code) (expr code) )
(defun exec_cond_list (code env)
   (if (exec (caar code) env)
      (exec (cadar code) env)
      (exec_cond_list (cdr code) env)
   )
)

;evalula una expresion cond
;(cond (expr code) (expr code))
(defun exec_cond (code env)
   (exec_cond_list (cdr code) env)
)
```

```
;aplicacion de lambda
;(caddar code): codigo a ejecutar
;(cadar code): parametros de la fcn lambda
;(cdr code): valores que toman los parametros de la funcion lambda
;env: el ambiente actual
(defun apply_lambda (code env)
   (exec (caddar code) (expand_env env (cadar code) (cdr code)))
)

(defun esFuncion (code f)
   (equal (car code) f)
)

(defun apply_Code (code)
   (apply (car code) (cdr code))
)

;implementacion del mapcar
(defun exec_Mapcar (f l env)
   (if (null l)
      nil
      (cons (exec (list f (list 'quote (car l))) env) (exec_Mapcar  f (cdr l) env))
   )
)

;evalue la lista de argumentos de una funcion
;(param1 param2 ... paramn)
(defun eval_args_list (code env)
   (if (null code)
      nil
      (cons (exec (car code) env) (eval_args_list (cdr code) env))
   )
)

;evalue la lista de argumentos de una funcion
;(fun param1 param2 ... paramn)
(defun eval_args (code env)
   (cons (car code) (eval_args_list (cdr code) env))
)

(defun exec_fun (code env)
   (if (atom (car code))
      (cond
         ((esFuncion code 'nth) (nth (cadr code) (caddr code)))
         ((esFuncion code 'cons) (cons (cadr code) (caddr code)))
         ((esFuncion code 'append) (append (cadr code) (caddr code)))
         ((esFuncion code 'apply) (apply (cadr code) (caddr code)))
         ((esFuncion code 'mapcar) (exec_Mapcar (cadr code) (caddr code) env))
         ;buscar en el ambiente por si hay una funcion con el nombre 'car code'
         ((is_In_Env (car code) env) (exec (cons (get_From_Env (car code) env) (cdr code)) env))
         (t (apply_Code code)); caso contrario ejecuta con la función que venga de nombre, y los parametros siguiente.
      )
      (cond
         ((esFuncion (car code) 'lambda) (apply_lambda code env))
         (t nil)
      )
   )
)
```

```
;evalua una expresion lisp
(defun exec (code &optional (env nil))
  (if (null code) nil
    (cond
      ((atom code) (exec_atom code env))
      ((esFuncion code 'quote) (exec_quote code))
      ((esFuncion code 'or) (exec_or code env))
      ((esFuncion code 'and) (exec_and code env))
      ((esFuncion code 'if) (exec_if code env))
      ((esFuncion code 'cond) (exec_cond code env))
      ((esFuncion code 'lambda) code )
      (t (exec_fun (eval_args code env) env))
    )
  )
)
```

Test

```
'(--------- Tests ----------)
;testing function(cons (exec (list f (list 'quote (car l))) env) (exec_Mapcar  f (cdr l) env))
;============================
(defun test (name actual expected)
  (if (equal actual expected)
    (list name 'passed)
    (list name 'fail '=> 'expected expected 'actual actual)
  )
)
;============================

;numeros
(test 'numero (exec '2) '2)

;letras
(test 'letra (exec 'A) 'A)

;true false
(test 'tf1 (exec nil) nil)
(test 'tf2 (exec 'nil) nil)
(test 'tf3 (exec 't) t)

;variables de ambiente
(test 'amb1 (exec 'A '((A 2)) ) '2)
(test 'amb2 (exec 'B '((A 2) (B 10)) ) '10)

;quote
(test 'quote1 (exec '(quote A) ) 'A)
(test 'quote2 (exec '(quote 1) ) '1)
(test 'quote3 (exec '(quote (car a)) ) '(car a))
(test 'quote4 (exec '(quote ((2 3) (4 5))) ) '((2 3) (4 5)) )

;or
(test 'or1 (exec '(or t t) ) 't)
(test 'or2 (exec '(or t nil) ) 't)
(test 'or3 (exec '(or nil nil) ) 'nil)
(test 'or4 (exec '(or nil t) ) 't)

;and
(test 'and1 (exec '(and nil nil) ) 'nil)
```

```
(test 'and2 (exec '(and t nil) ) 'nil)
(test 'and3 (exec '(and nil t) ) 'nil)
(test 'and4 (exec '(and t t) ) 't)

;and y or
(test 'andor1 (exec '(and (or t nil) t) ) 't)
(test 'andor2 (exec '(and (or t nil) (or nil nil)) ) 'nil)
(test 'andor3 (exec '(or (or t nil) (or nil nil )) ) 't)

;if
(test 'if1 (exec '(if t 1 2)) '1)
(test 'if2 (exec '(if t 1 2)) '1)
(test 'if3 (exec '(if (or t nil) 1 2)) '1)

;cond
(test 'cond1 (exec '(cond (t 2) )) '2)
(test 'cond2 (exec '(cond (nil 5) (t 2) )) '2)
(test 'cond3 (exec '(cond ((and nil nil) 5) (t 2) )) '2)
(test 'cond4 (exec '(cond ((and nil nil) 5) (nil 99) (t 2) )) '2)
(test 'cond5 (exec '(cond (nil 99) ((and t t) 5) (t 2) )) '5)
(test 'cond6 (exec '(cond ((and t t) 5) (nil 99)  (t 2) )) '5)

;list
(test 'list1 (exec '(list 2 3 4)) '(2 3 4))
(test 'list2 (exec '(list 2 3 4 5)) '(2 3 4 5))
(test 'list3 (exec '(list t)) '(t))
(test 'list4 (exec '(list 1 a (quote (1 2))) '((a 10))) '(1 10 (1 2)))

;list con proceso
(test 'listproc1 (exec '(list (or t t))) '(t))
(test 'listproc2 (exec '(list (or t t) (and t nil))) '(t nil))
(test 'listproc3 (exec '(list (quote (2 3 4)))) '((2 3 4)))

;car
(test 'car1 (exec '(car (quote (2 3)))) '2)
(test 'car2 (exec '(car (quote (4 2 3)))) '4)
(test 'car3 (exec '(car (quote ( (2 3) (4 5))))) '(2 3))

;cdr
(test 'cdr1 (exec '(cdr (quote (4 2 3)))) '(2 3))

;caar
(test 'caar1 (exec '(caar (quote ((4 2 3))))) '4)

;cdar
(test 'cdar1 (exec '(cdar (quote ((4 2 3))))) '(2 3))

;car + ambiente
(test 'car-amb1 (exec '(car (list a 2 3)) '((a 100)) ) '100)

;cdr + ambiente
(test 'cdr-amb1 (exec '(cdr (list a b c)) '((a 100) (b 99) (c 98)) ) '(99 98))

;not
(test 'not1 (exec '(not t)) nil)
(test 'not2 (exec '(not nil)) t)
(test 'not3 (exec '(not a) '((a nil))) t)
```

```
;lambda
(test 'lambda1 (exec '((lambda (x) (* x 2)) 2)) '4)
(test 'lambda2 (exec '((lambda (x y) (+ (* x 2) y)) 2 4)) '8)
(test 'lambda3 (exec '(lambda (x) (* x 2))) '(lambda (x) (* x 2)))
(test 'lambda4 (exec '(mapcar (lambda (x) (cons x (cdr '(3 4 5)))) '(1 2 3))) '((1 4 5) (2 4 5) (3 4 5)))

;expandir ambiente
(test 'exp_amb1 (replace_or_add nil 'a '1) '((a 1)))
(test 'exp_amb2 (replace_or_add '((a 2)) 'a '1) '((a 1)))
(test 'exp_amb3 (replace_or_add '((b 10) (a 2)) 'a '1) '((b 10) (a 1)))
(test 'exp_amb4 (expand_env nil '(x y) '(10 20)) '((x 10) (y 20)))
(test 'exp_amb5 (expand_env '((x 50)) '(x y) '(10 20)) '((x 10) (y 20)))

;recursion
(test 'rec1 (exec '(car (car (quote((2 3 4))))) ) '2)

;aritmeticas
(test 'aritm1 (exec '(+ 2 3) ) '5)
(test 'aritm2 (exec '(+ 2 3 4) ) '9)
(test 'aritm3 (exec '(- 3 4 5) ) '-6)
(test 'aritm4 (exec '(* 3 4) ) '12)
(test 'aritm5 (exec '(/ 12 4) ) '3)
(test 'aritm6 (exec '(* (/ 12 4) 10 )) '30)
(test 'aritm7 (exec '(+ (* (/ 12 4) 10 ) 1000)) '1030)

;atom
(test 'atom1 (exec '(atom 2)) t)
(test 'atom2 (exec '(atom nil)) t)
(test 'atom3 (exec '(atom (quote (2 3 4)))) nil)

;listp
(test 'listp1 (exec '(listp 2)) nil)
(test 'listp2 (exec '(listp nil)) t)
(test 'listp3 (exec '(listp (quote (2 3)))) t)

;numberp
(test 'numberp1 (exec '(numberp (quote (2 3)))) nil)
(test 'numberp2 (exec '(numberp 2)) t)
(test 'numberp3 (exec '(numberp nil)) nil)

;null
(test 'null1 (exec '(null nil)) t)
(test 'null2 (exec '(null 3)) nil)

;nth
(test 'nth1 (exec '(nth 0 (quote (0 1)))) '0)
(test 'nth2 (exec '(nth 1 (quote (0 1)))) '1)
(test 'nth3 (exec '(nth 4 (quote (0 1 2 3 4)))) '4)

;cons
(test 'cons1 (exec '(cons 4 (quote (0 1 2 3 4)))) '(4 0 1 2 3 4))
(test 'cons2 (exec '(cons 4 (quote (4)))) '(4 4))
(test 'conscdr1 (exec '(cons x (cdr y)) '((x a)(y (b c)))) '(a c))
(test 'conscdr2 (exec '(cons 'a (quote (1 2)))) '(a 1 2))

;append
(test 'append1 (exec '(append (quote (4)) (quote (4)))) '(4 4))
(test 'append2 (exec '(append (quote (4 5 6)) (quote (4 5 6)))) '(4 5 6 4 5 6))
```

```
;length
(test 'length1 (exec '(length (quote (4 5 6)))) '3)

;apply
(test 'apply1 (exec '(apply '+ (quote (4 5 6)))) '15)
(test 'apply2 (exec '(apply '* (quote (4 5)))) '20)

;mapcar
(test 'mapcar1 (exec '(mapcar 'numberp (quote (4)))) '(t))
(test 'mapcar2 (exec '(mapcar 'numberp (quote (4 5 6 nil)))) '(t t t nil))
(test 'mapcar3 (exec '(mapcar 'car (quote ( (2 3) (4 5 ))) )) '(2 4))

;reverse
(test 'reverse (exec '(reverse (quote (4 5 6 7)))) '(7 6 5 4))

;===========
;funcionales
;===========

(test 'fun1 (exec
   '(my_fun 1)
   '( (my_fun (lambda (x) (* x 2))) )
   )
   '2
)

(test 'fun2 (exec
   '(mapcar 'numberp (quote (1 2 3 4)))
   )
   '(t t t t)
)

(test 'fun3 (exec
   '(mapcar 'my_fun (quote (1 2 3 4)))
   '((my_fun (lambda (x) (* x 2))))
   )
   '(2 4 6 8)
)

(test 'fun4 (exec
   '(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)
)
```

GPS

```
; Problema : GPS.
; Lenguajes Formales - Primer Cuatrimestre 2010
; Alumno : Bello Camilletti, Nicolás.
; Padrón : 86676

(defun pertenece (A L)
   (if (null L)
      nil
      ( if (eq A (car L))
         T
         (pertenece A (cdr L))
      )
   )
)

(defun diferencia (a b)
   (if (null a)
      nil
      (if (null b)
         a
         (if (pertenece (car a) b)
            (diferencia (cdr a) b)
            (cons (car a) (diferencia (cdr a) b))
         )
      )
   )
)

(defun vecinos (actual L)
   (if (null L)
      nil
      ( if (eq actual (caar L))
         (cadar L)
         (vecinos actual (cdr L))
      )
   )
)

(defun distribuir (cam vec)
   (if (null vec)
      nil
      (cons (cons (car vec) cam) (distribuir cam (cdr vec)))
   )
)

; obtiene todos los caminos posibles de i a f en el grafo
(defun GPS (i f grafo &optional(caminos (list( list i))) )
   (if (null caminos)
      nil
      (if (eq (caar caminos) f)
         (cons (car caminos) (GPS i f grafo (cdr caminos) ) )
         (GPS i f grafo (append (distribuir(car caminos)
                                 (diferencia (vecinos (caar caminos) grafo) (car caminos))
                                )
                                (cdr caminos)
                        )
```

```
        )
      )
   )
)

(defun obtenerMinimo (L)
   (if (eq (length L) 1)
      (car L)
      (if (null (car L) )
         (obtenerMinimo (cdr L) )
         (if (< (length (car L)) (length (cadr L)) )
            (obtenerMinimo (cons (car L) (cddr L)) )
            (obtenerMinimo (cdr L) )
         )
      )
   )
)

(defun obtenerMaximo (L)
   (if (eq (length L) 1)
      (car L)
      (if (null (car L) )
         (obtenerMaximo (cdr L) )
         (if (< (length (car L)) (length (cadr L)) )
            (obtenerMaximo (cdr L) )
            (obtenerMaximo (cons (car L) (cddr L)) )
         )
      )
   )
)

(defun caminoMinimo (i f grafo)
   ( obtenerMinimo (GPS i f grafo) )
)

(defun caminoMaximo (i f grafo)
   ( obtenerMaximo (GPS i f grafo) )
)

'(----- Codificador / Decodificador -----)

(defun traductor ( c diccionario)
   (if (null diccionario)
      nil
      ( if (eq c (caar diccionario))
         (cadar diccionario)
         (traductor c (cdr diccionario))
      )
   )
)

(defun traductorList ( L diccionario)
   (if (null L)
      nil
      ( cons (traductor (car L) diccionario ) (traductorList (cdr L) diccionario ) )
   )
)
```

```
(defun compararPorCalle (calle otroTerm)
   (or (equal calle (car otroTerm) )
      (equal calle (cadr otroTerm) )
   )
)

(defun comparar (term otroTerm)
   (and (compararPorCalle (car term) otroTerm) (compararPorCalle (cadr term) otroTerm) )
)

(defun codificador ( term diccionario)
   (if (null diccionario)
      nil
      ( if (comparar term (cadar diccionario))
         (caar diccionario)
         (codificador term (cdr diccionario))
      )
   )
)

; obtiene cual es el comun entre 2 esquinas, para saber por cual calle va.
(defun obtenerComun (term otroTerm)
   (if (null otroTerm)
      (car term)
      (if (compararPorCalle (car term) otroTerm)
         (car term)
         (if (compararPorCalle (cadr term) otroTerm)
            (cadr term)
            nil
         )
      )
   )
)

; obtiene el comun de los 2 primeros del recorrido
(defun obtenerComunPrimeros (recorrido )
   (if (null recorrido)
      nil
      (if (= (length recorrido) 1)
         (caar recorrido)
         (obtenerComun (car recorrido) (cadr recorrido))
      )
   )
)


(defun crearNuevoActual (recorrido)
   (list (obtenerComunPrimeros recorrido) 1)
)

(defun aumentarContadorCuadras (actual)
   (list (car actual) (+ (cadr actual) 1))
)

(defun agregarAlFinal (lista item)
   (if (null lista)
      (list item)
      (append lista (list item))
```

```
   )
)

; comprime el recorrido generando pares calle por la que va, cantidad de cuadras.
(defun comprimirRecorrido (recorrido &optional (result nil)
                                ( actual (list (obtenerComunPrimeros recorrido) 0)) )
   (if (null recorrido)
      (agregarAlFinal result actual)
      (if (equal (car actual) (obtenerComunPrimeros recorrido) ) ; en la proxima sigue en la misma calle
         (comprimirRecorrido (cdr recorrido) result (aumentarContadorCuadras actual) )
         (comprimirRecorrido (cdr recorrido) (agregarAlFinal result actual) (crearNuevoActual recorrido ))
      )
   )
)

(defun armarDescripcion (actual)
   (list 'luego 'gire 'en (car actual) 'y 'avance (cadr actual) 'cuadras.)
)

; Genera la descripción del circuito basado en el recorrido comprimido
(defun describir (rec &optional (desc nil))
   (if (null rec)
      (append desc '(hasta llegar a destino.))
      (if (null desc)
         (describir (cdr rec) (list 'Tome (caar rec) 'y 'avance (cadar rec) 'cuadras.) )
         (if (eq (cadar rec) 1)
            (describir (cdr rec) (append desc (list 'Doble 'en (caar rec) 'y 'avance 'una 'cuadra.) ) )
            (describir (cdr rec) (append desc (armarDescripcion (car rec)) ) )
         )
      )
   )
)

(defun traducirRecorrido (rec diccionario)
   (describir (comprimirRecorrido (traductorList rec diccionario)) )
)
```

Test

```
'(----- Load utils for tests -----)
(setq grafoTest '((a(b c)) (b(a e d)) (c(a d e)) (d(b c e)) (e(e b d)) ) )
(setq diccionarioTest '(
                  (a (PaseoColon Independencia))
                  (b (PaseoColon Chile))
                  (f (Independencia Balcarse))
                  (g (Independencia Defensa))
                  (h (Defensa Chile))
                  (k (Defensa Balcarse ))
                  (l (Belgrano Balcarse) )
               )
)

(defun test (name a e)
   (if (equal a e)
      (list name 'passed)
      (list name 'fail '=> 'expected e 'actual a)
   )
)
```

```
'(----- Inicio tests -----)

(test 'diferenciaNull
   (diferencia '(a b) nil)
   '(a b)
)

(test 'diferenciaOneElement
   (diferencia '(a b c) '(b))
   '(a c)
)

(test 'diferenciaMoreElements
   (diferencia '(a b c) '(a c))
   '(b)
)

'(---------)

(test 'obtenerMinimoEqualsAndNil
   (obtenerMinimo '((a) (b) nil (c)) )
   '(c)
)

(test 'obtenerMinimoLast
   (obtenerMinimo '((a b c) (a b)) )
   '(a b)
)

(test 'obtenerMinimoFirst
   (obtenerMinimo '((a b) (a b c)) )
   '(a b)
)

(test 'obtenerMinimoMiddle
   (obtenerMinimo '((a b c) (a b) (a b c d)) )
   '(a b)
)

'(---------)

(test 'obtenerMaximoEqualsAndNil
   (obtenerMaximo '((a) (b) nil (c)) )
   '(a)
)

(test 'obtenerMaximoFirst
   (obtenerMaximo '((a b c) (a b)) )
   '(a b c)
)

(test 'obtenerMaximoLast
   (obtenerMaximo '((a b) (a b c)) )
   '(a b c)
)
```

```
(test 'obtenerMaximoMiddle
   (obtenerMaximo '((a b) (a b c) (a)) )
   '(a b c)
)

'(---------)
(test 'vecinosNull
   (vecinos 'a nil)
   'nil
)

(test 'vecinosFirst
   (vecinos 'a '((a(b c)) (b(a e d)) ))
   '(b c)
)

(test 'vecinosMiddle
   (vecinos 'c '((a(b c)) (c(a d e)) (b(e b d)) ))
   '(a d e)
)

(test 'vecinosLast
   (vecinos 'b '((a(b c)) (c(a d e)) (b(e b d)) ))
   '(e b d)
)

'(---------)

(test 'GPS
   (GPS 'a 'e grafoTest)
   '((E B A) (E C D B A) (E D B A) (E B D C A) (E D C A) (E C A))
)

(test 'GPSFinalNotExist
   (GPS 'a 'w grafoTest)
   'nil
)

(test 'GPSInitialNotExist
   (GPS 'w 'a grafoTest)
   'nil
)

'(---------)

(test 'caminoMaximo
   (caminoMaximo 'a 'e grafoTest)
   '(E C D B A)
)

(test 'caminoMinimo
   (caminoMinimo 'a 'e grafoTest)
   '(E C A)
)

'(---------)
```

```
(test 'compararEsta
   (comparar '(PaseoColon Chile) '(PaseoColon Chile) )
   'T
)

(test 'compararEstaInvertido
   (comparar '(Chile PaseoColon) '(PaseoColon Chile) )
   'T
)

(test 'compararNoEsta
   (comparar '(No esta) '(PaseoColon Chile) )
   'nil
)

'(---------)

(test 'codificadorEsta
   (codificador '(PaseoColon Chile) diccionarioTest )
   'b
)

(test 'codificadorEstaInvertido
   (codificador '(Chile PaseoColon) diccionarioTest )
   'b
)

(test 'codificadorNoEsta
   (codificador '(No esta) diccionarioTest )
   'nil
)

'(---------)

(test 'traductorEsta
   (traductor 'b diccionarioTest )
   '(PaseoColon Chile)
)

(test 'traductorNoEsta
   (traductor 'w diccionarioTest )
   'nil
)

'(----- Fin tests -----)
```

Ejemplo de uso

```
(setq grafoMain '(
      (a(b f)) (b(a c )) (c(b d)) (d(c e)) (e(d l))
       (f(g))                      (l(k))
       (g(h))  (h(i))    (i(c j)) (j(d k)) (k())
          )
)

(setq diccionario '(
                    (a (PaseoColon Independencia))
                    (b (PaseoColon Chile))
                    (c (PaseoColon Mexico ))
                    (d (PaseoColon Venezuela))
                    (e (PaseoColon Belgrano))
                    (f (Independencia Balcarse))
                    (g (Independencia Defensa))
                    (h (Defensa Chile))
                    (i (Defensa Mexico))
                    (j (Defensa Venezuela))
                    (k (Defensa Balcarse ))
                    (l (Belgrano Balcarse) )
                )
)

(GPS (codificador '(PaseoColon Independencia) diccionario)
   (codificador '(PaseoColon Belgrano) diccionario)
   grafoMain
)

(traductorList (caminoMinimo (codificador '(PaseoColon Independencia) diccionario)
                    (codificador '(PaseoColon Belgrano) diccionario)
                    grafoMain
            )
            diccionario
)

(traductorList (caminoMaximo (codificador '(PaseoColon Independencia) diccionario)
                    (codificador '(PaseoColon Belgrano) diccionario)
                    grafoMain
            )
            diccionario
)

(traducirRecorrido (caminoMaximo (codificador '(PaseoColon Independencia) diccionario)
                    (codificador '(PaseoColon Belgrano) diccionario)
                    grafoMain
            )
            diccionario
)
```

Pattern matching

```
; Problema : Pattern matching.
; Lenguajes Formales - Primer Cuatrimestre 2010
; Alumno : Bello Camilletti, Nicolás.
; Padrón : 86676

(defun verif (P asoc E L)
   (if (eq asoc 'NohayAsoc)
      (cons (list P E) L)
      (if (equal asoc E)
         L
         'NoMatchea
      )
   )
)

(defun buscar (V L)
   (if (null L)
      'NohayAsoc
      (if (eq V (caar L))
         (cadar L)
         (buscar v (cdr L))
      )
   )
)

(defun pertenece (A L)
   (if (null L)
      nil
      ( if (eq A (car L))
         T
         (pertenece A (cdr L))
      )
   )
)

(defun esvar (P Vars)
   (if (atom P)
      (pertenece P Vars)
      nil
   )
)

(defun PatternMat (P E Vars &optional(L nil))
   (if (eq L 'NoMatchea)
      'NoMatchea
      (if (esvar P Vars)
         (verif P (buscar P L) E L)
         (if (atom P)
            (if (eq P E)
               L
               'NoMatchea
            )
            (PatternMat (cdr P) (cdr E) Vars (PatternMat (car P) (car E) Vars L ))
         )
      )
   )
)
```

Tests

```
'(----- Load utils for tests -----)

(defun test (name actual expected)
   (if (equal actual expected)
      (list name 'passed)
      (list name 'fail '=> 'expected expected 'actual actual)
   )
)

'(----- Inicio tests -----)
(test 'No_Vars_Pero_Igual
   (PatternMat '(Hola mundo) '(Hola mundo) nil)
   nil
)

(test 'No_Vars_No_matchea
   (PatternMat '(Pepe mundo) '(Hola mundo) nil)
   'NoMatchea
)

(test 'Var_Simple
   (PatternMat '(A) '( Hola) '(A) )
   '((A HOLA))
)

(test 'Var_Simple_Mezclado_Palabras
   (PatternMat '(A mundo) '( Hola mundo) '(A) )
   '((A HOLA))
)

(test 'Var_Doble_Mezclado_Palabras_Repetido
   (PatternMat '(A mundo A) '( Hola mundo Hola) '(A))
   '((A HOLA))
)

(test 'Var_Doble_Mezclado_Palabras_Repetido_No_matchea
   (PatternMat '(A mundo A) '( Hola mundo Chau) '(A) )
   'NoMatchea
)

(test 'Var_Doble_Distinto_Mezclado_Palabras
   (PatternMat '(A mundo B) '( Hola mundo Chau) '(A B) )
   '((B CHAU) (A HOLA))
)

(test 'Var_Triple_Con_Parentesis_Distinto_Mezclado_Palabras
   (PatternMat '((B y C) A ) '((hola y chau) mundo) '(A B C))
   '((A MUNDO) (C CHAU) (B HOLA))
)

'(----- Fin tests -----)

;Ejemplo de uso
 (PatternMat '((B y C) A ) '((hola y chau) mundo) '(A B C))
```