# 7524 - Teoría de la programación TP Individual

Karina Alaya

Padron 75840

<u>Ejercicio 1) - Procesamiento de listas</u>
<u>Length</u>
<u>Take</u>
<u>Drop</u>
<u>Append</u>
<u>Member</u>
<u>Position</u>
<u>Ejercicio 2) - Referencias externas</u>
Ejercicio 3) - Ejemplo de ejecución
Programa 1
Programa 2
Programa 3
Programa 4
<u>Programa 5</u>
<u>Programa 6</u>
<u>Ejercicio 4) - Case</u>
<u>Ejercicio 5) - Recursividad</u>
1. Traducción al lenguaje Kernel
2. "Tail Recursive"
3. Ejecución en la máquina abstracta
<u>Implementación básica</u>
<u>Implementación Tail Recursive</u>
Ejercicio 6) - Alto orden con listas
<u>FoldL</u>
<u>FoldR</u>
<u>Map</u>
<u>Filter</u>
<u>Ejercicio 7) - Hilos</u>
<u>WaitSome</u>
Maquina abstracta
<u>Ejercicio 8) - Evaluación perezosa</u>
8.1 Maquina Abstracta

#### 8.2 Reverse

<u>Ejercicio 9) - Mensajes</u>

Ejercicio 10) - Servidor de filtros

Ejercicio 11) - Celdas de memoria

## Ejercicio 1) - Procesamiento de listas

#### 1. Length

a. Código Oz

```
% Length function
declare fun {Length Xs}
          case Xs of nil then 0
        [] H|T then 1 + {Length T}
        end
    end
```

b. Ejemplos de ejecución

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                                  X
File Edit Options Buffers Tools Oz Help
% Length function
declare fun {Length Xs}
           case Xs of nil then 0
           [] H|T then 1 + {Length T}
        end
% invocation examples Length
{Show 'Length examples'}
{Show {Length [1 2 10]}}
{Show {Length [1 2 10 'test']}}
{Show {Length ['a' 'b' 'c']}}
{Show {Length ['a' 'b' 'c' 4 5 6 7 8 9 10]}}
1\**- Oz
                      All L8
'Length examples'
4
3
10
1\**- *Oz Emulator*
                      Bot L9
                                 (Comint:run)
```

#### 2. Take

a. Código Oz

```
% Take function
declare fun {Take Xs N}
    if N == 0 then nil
    else
        case Xs of nil then nil
        [] H|T then H|{Take T N-1}
        end
    end
end
```

```
X
Oz Programming Interface (emacs@KARINA-VAIO)
File Edit Options Buffers Tools Oz Help
                            X 1 1 1 1 1
% Take function
declare fun {Take Xs N}
           if N == 0 then nil
              case Xs of nil then nil
             [] H|T then H|{Take T N-1}
              end
           end
        end
% invocation examples Take
{Show 'Take examples'}
{Show {Take [1 2 10 15] 2}}
{Show {Take [1 2 10 15] 8}}
{Show {Take [a b c d] 3}}
{Show {Take [a b c d] 4}}
{Show {Take [a b c d] 1}}
{Show {Take [a b c d] 0}}
1\**- Oz
                      All L16
                                  (Oz)
'Take examples'
[1 2]
[1 2 10 15]
[a b c]
[abcd]
[a]
nil
1\**- *Oz Emulator*
                      All L8
                                 (Comint:run)
Beginning of buffer
```

#### 3. Drop

a. Código Oz

```
% Drop function
declare fun {Drop Xs N}
    if N == 0 then Xs
    else
        case Xs of nil then nil
     [] H|T then {Drop T N-1}
        end
    end
    end
end
```

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                            X
File Edit Options Buffers Tools Oz Help
% Drop function
declare fun {Drop Xs N}
           if N == 0 then Xs
           else
              case Xs of nil then nil
              [] H|T then {Drop T N-1}
              end
           end
        end
% invocation examples Drop
{Show 'Drop examples'}
{Show {Drop [1 2 10 15] 2}}
{Show {Drop [1 2 10 15] 8}}
{Show {Drop [a b c d] 4}}
{Show {Drop [a b c d] 1}}
{Show {Drop [a b c d] 0}}
1\**- Oz
                       All L10
                                  (Oz)
'Drop examples'
[10 15]
nil
nil
[b c d]
[abcd]
1\**- *Oz Emulator*
                                   (Comint:run)
                      All L7
```

#### 4. Append

a. Código Oz

```
% Append function
declare fun {Append Xs Ys}
    if Ys == nil then Xs
    else
        case Xs of nil then Ys
        [] H|T then H|{Append T Ys}
        end
    end
    end
end
```

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                             X
File Edit Options Buffers Tools Oz Help
% Append function
declare fun {Append Xs Ys}
           if Ys == nil then Xs
           else
              case Xs of nil then Ys
              [] H|T then H|{Append T Ys}
              end
           end
        end
% invocation examples Append
{Show 'Append examples'}
{Show {Append [1 2 10 15] [a b c d]}}
{Show {Append [1 2 10 15] nil}}
{Show {Append nil [a b]}}
{Show {Append [1 2 10 15] [a]}}
{Show {Append [1] [a b]}}
1\**- Oz
                       All L8
                                   (Oz)
'Append examples'
[1 2 10 15 a b c d]
[1 2 10 15]
[a b]
[1 2 10 15 a]
[1 a b]
1\**- *Oz Emulator*
                       All L7
                                   (Comint:run)
```

#### 5. Member

a. Código Oz

```
% Member function
declare fun {Member Xs Y}
    case Xs of nil then false
    [] H|T then
        if H == Y then true
        else
        {Member T Y}
        end
    end
end
```

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                           X
File Edit Options Buffers Tools Oz Help
                           * •
% Member function
declare fun {Member Xs Y}
           case Xs of nil then false
           [] H|T then
              if H == Y then true
              else
                 {Member T Y}
              end
           end
        end
s invocation examples Member
{Show 'Member examples'}
{Show {Member [1 2 10 15] 2}}
{Show {Member [1 2 10 15] 1}}
{Show {Member [1 2 10 15] 10}}
{Show {Member [1 2 10 a 15] 15}}
{Show {Member [1 2 10 a 15] a}}
{Show {Member [1 2 10 15] b}}
{Show {Member [1 2 10 15] 3}}
{Show {Member [1 2 10 15] 5}}
{Show {Member [1 2 10 15] nil}}
{Show {Member nil a}}
1\**- Oz
                    All L11
                                  (Oz)
'Member examples'
true
true
true
true
true
false
false
false
false
false
1\**- *Oz Emulator* All L11 (Comint:run)
```

#### 6. Position

a. Código Oz

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                             X
File Edit Options Buffers Tools Oz Help
% Position function
declare fun {Position Xs Y}
           case Xs of nil then 0
            [] H|T then
              if H == Y then 0
              else 1 + {Position T Y}
              end
           end
        end
% invocation examples Position
{Show 'Position examples'}
{Show {Position [1 2 10 15] 10}}
{Show {Position [1 2 10 a 15] 15}}
{Show {Position [1 2 10 a 15] a}}
{Show {Position [1 2 10 15] 1}}
{Show {Position [1 2 10 15] 3}}
1\**- Oz
                       All L5
                                   (Oz)
'Position examples'
3
0
4
<
1\**- *Oz Emulator*
                        All L7
                                    (Comint:run)
```

# Ejercicio 2) - Referencias externas

1.  $proc \{P X Y\} local Z in \{Q Z U\} end end$ 

Referencias externas: Q, U

2.  $proc \{P X Y\} local Z in \{Q Z Y\} end end$ 

Referencias externas: Q

3.  $proc \{P X Y\} local Z in \{P Z Y\} end end$ 

Referencias externas: ninguna

# Ejercicio 3) - Ejemplo de ejecución

## Programa 1

#### **Estado inicial**

Stack	Store	E
local B in if B then Skip else skip end end	-	

#### => Ejecución de declaración de variable

Stack	Store	Е
if B then skip else	b1	B-> b1
skip end		

=> Ejecución de condicional. Como E(B) no está determinado, el programa se suspende, a la espera que b1 tome algun valor.

### Programa 2

#### **Estado inicial**

Stack	Store	Е
local B in  B = false  if B then  skip  else  skip  end	-	-
end		

#### => Ejecución de declaración de variable y composición

Stack	Store	Е
B = false if B then skip else skip end	b1	B-> b1

#### => Ejecución de binding de variables

Stack	Store	E
if B then skip else skip end	b1 = <b>false</b>	B-> b1

# => Ejecución de condicional, como B = false entonces se agrega en el stack la sentencia dentro del else

Stack	Store	Е
skip	b1 = false	B-> b1

=> Ejecución del skip (St6), no hay cambios en el store y se quita la sentencia.

Stack	Store	Е
-	b1 = false	B-> b1

No hay nada mas en el stack, entonces el programa finaliza.

## Programa 3

#### **Estado inicial**

Stack	Store	E
local X Z A B P in proc {P X Y} Y = X+Z	-	-
end Z=7		
X=4 {P X A}		
{P A B}		
end		

#### => Ejecución de declaración de variables

Stack	Store	E
proc {P X Y} Y = X+Z end Z=7 X=4 {P X A} {P A B}	x1 z1 a1 b1 p1	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

#### => Ejecución de procedure value

Stack	Store	E
Z=7	x1	X -> x1
X=4	z1	Z-> z1
{P X A}	a1	A -> a1
{P A B}	b1	B -> b1
	p1 = <b>proc</b> { <b>P X Y</b> }	P-> p1
	Y = X+Z	
	end, CE = {Z -> z1}	

## => Ejecución de binding de variable

Stack	Store	Е
X=4 {P X A} {P A B}	x1 z1 = <b>7</b> a1 b1 p1 = proc {P X Y} Y = X+Z end, CE = {Z -> z1}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

#### => Ejecución de binding de variable

Stack	Store	E
{P X A} {P A B}	x1 = <b>4</b> z1 = 7 a1 b1 p1 = proc {P X Y} Y = X+Z end, CE = {Z -> z1}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

## => Ejecución de procedure value

Stack	Store	E
<b>A</b> = <b>X</b> + <b>Z</b> {P A B}	x1 = 4 z1 = 7 a1 b1 p1 = proc {\$ X Y}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

#### => Asignacion de variable mas suma en base al store

Stack	Store	Е
{P A B}	x1 = 4 z1 = 7 a1 = <b>11</b> b1 p1 = proc {\$ X Y} Y = X+Z end, CE = {Z -> z1}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

## => Ejecución de procedure value

Stack	Store	E
B = A + Z	x1 = 4 z1 = 7 a1 = 11 b1 p1 = proc {\$ X Y}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

#### => Asignacion de variable mas suma en base al store

Stack	Store	Е
-	x1 = 4 z1 = 7 a1 = 11 b1 = <b>18</b> p1 = proc {\$ X Y}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

# Programa 4

#### **Estado inicial**

Stack	Store	E
local X Z A B P in proc {P X Y} Y = X+Z	-	-
end Z=10 local Z in Z = 2		
X=4 {P X A} {P A B} end end		

## => Ejecución de declaración de variables

Stack	Store	E
proc {P X Y}	x1 z1 a1 b1 p1	X - > x1 Z -> z1 A -> a1 B -> b1 P -> p1
{P X A} {P A B} end		

## => Ejecución de declaración de procedure value

Stack	Store	Е
Z=10	x1	X - > x1
local Z in	z1	Z -> z1
Z = 2	a1	A -> a1
X=4	b1	B -> b1
{P X A}	p1 = <b>proc</b> { <b>\$ X Y</b> }	P -> p1
{P A B}	Y = X+Z	
end	end, CE = {Z -> z1}	

## => Ejecución de asignación de variable

Stack	Store	Е
local Z in	x1	X - > x1
Z = 2	z1 = <b>10</b>	Z -> z1
X=4	a1	A -> a1
{P X A}	b1	B -> b1
{P A B}	p1 = proc {\$ X Y}	P -> p1
end	Y = X+Z	·
	end, $CE = \{Z -> z1\}$	

## => Ejecución de declaración de variable Z

Stack	Store	Е
Z = 2	x1	X - > x1
X=4	z1 = 10	Z -> <b>z2</b>
{P X A}	<b>z2</b>	A -> a1
{P A B}	a1	B -> b1
	b1	P -> p1
	p1 = proc {\$ X Y}	

Y = X+Z end, CE = {Z -> z1}	
(2 * 21)	

#### => Ejecución de asignación de variable Z (en el nuevo entorno)

Stack	Store	Е
X=4 {P X A} {P A B}	x1 z1 = 10 z2 = <b>2</b> a1 b1 p1 = proc {\$ X Y}	X - > x1 Z -> z2 A -> a1 B -> b1 P -> p1

#### => Ejecución de asignación de variable X (en el nuevo entorno)

Stack	Store	Е
{P X A} {P A B}	x1 = <b>4</b> z1 = 10 z2 = 2 a1 b1 p1 = proc {\$ X Y}	X - > x1 Z -> z2 A -> a1 B -> b1 P -> p1

#### => Ejecución de procedure value (en el nuevo entorno usando su entorno contextual)

Stack	Store	E
A = X + Z {P A B}	x1 = 4 z1 = 10 z2 = 2 a1 b1 p1 = proc {\$ X Y}	X - > x1 Z -> z2 A -> a1 B -> b1 P -> p1

# => Ejecución de la única sentencia del procedure (en el nuevo entorno usando su entorno contextual)

Stack	Store	E
-------	-------	---

{P A B}	x1 = 4	X - > x1
	z1 = 10	Z -> z2
	z2 = 2	A -> a1
	a1 = <b>14 (Z -&gt; z1 = 10)</b>	B -> b1
	b1	P -> p1
	p1 = proc {\$ X Y}	
	Y = X+Z	
	end, CE = {Z -> z1}	

#### => Ejecución del procedure value(en el nuevo entorno usando su entorno contextual)

Stack	Store	Е
B = A + Z	x1 = 4 z1 = 10 z2 = 2 a1 = 14 b1 p1 = proc {\$ X Y}	X - > x1 Z -> z2 A -> a1 B -> b1 P -> p1

# => Ejecución de la única sentencia del procedure (en el nuevo entorno usando su entorno contextual)

Stack	Store	Е
-	x1 = 4 z1 = 10 z2 = 2 a1 = 14 b1 = <b>24 (a1:14 +z1:10)</b> p1 = proc {\$ X Y} Y = X+Z end, CE = {Z -> z1}	X - > x1 Z -> z2 A -> a1 B -> b1 P -> p1

#### Fin del programa

# Programa 5

#### **Estado inicial**

Stack	Store	E
local X Y Z P Q in	-	•
X=6		
Y=4		
proc {P A B}		
proc {B U V}		
local F in		
F=A+1		
V=U+F		
end		
end		
end		
{P X Q}		
{Q Y Z}		
end		

#### => Declaracion de variables

Stack	Store	E
X=6	x1	X -> x1
Y=4	y1	Y -> y1
proc {P A B}	z1	Z -> z1
proc {B U V}	p1	P -> p1
local F in	q1	Q -> q1
F=A+1		
V=U+F		
end		
end		
end		
{P X Q}		
{Q Y Z}		

#### => Asignación de variable

Stack	Store	Е
Y=4 proc {P A B} proc {B U V} local F in	x1 = <b>6</b> y1 z1 p1	X -> x1 Y -> y1 Z -> z1 P -> p1

	F=A+1 V=U+F	q1	Q -> q1
end			
end			
end			
{P X Q}			
{P X Q} {Q Y Z}			

## => Asignación de variable

Stack	Store	Е
proc {P A B} proc {B U V}	x1 = 6 y1 = <b>4</b>	X -> x1 Y -> y1
local F in F=A+1 V=U+F	z1 p1 q1	Z -> z1 P -> p1 Q -> q1
end end end		
{P X Q} {Q Y Z}		

## => Declaración procedure value

Stack	Store	Е
{P X Q}	x1 = 6	X -> x1
{Q Y Z}	y1 = 4	Y -> y1
	z1	Z -> z1
	p1 = <b>proc</b> { <b>\$ A B</b> }	P -> p1
	proc {B U V}	Q -> q1
	local F in	
	F=A+1	
	V=U+F	
	end	
	end	
	end, CE = {}	
	q1	

## => Ejecución procedure value

Stack	Store	Е
proc {Q U V}	x1 = 6	X -> x1
local F in	y1 = 4	Y -> y1
F = X + 1	z1	Z -> z1
V = U + F	p1 = proc {\$ A B}	P -> p1
end	proc {B U V}	Q -> q1
end	local F in	
{Q Y Z}	F=A+1	
	V=U+F	
	end	
	end	
	end, CE = {}	
	q1	

#### => Declaración procedure value

Stack	Store	Е
{Q Y Z}	x1 = 6 y1 = 4 z1 p1 = proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end, CE = {} q1 = proc {\$ U V} local F in F = X + 1 V = U + F end End, CE = {X-> x1}	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

# => Ejecución procedure value

Stack	Store	Е
local F in	x1 = 6 y1 = 4 z1 p1 = proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end, CE = {} q1 = proc {\$ U V} local F in F = X + 1 V = U + F end End, CE = {X-> x1}	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

#### => declaración de variable

Store	Е
<pre>x1 = 6 y1 = 4 z1 f1 p1 = proc {\$ A B}     proc {B U V}</pre>	E  X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1
F = X + 1 V = U + F end End, $CE = \{X-> x1, F-> f1\}$	
	x1 = 6 y1 = 4 z1 f1 p1 = proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end, CE = {} q1 = proc {\$ U V} local F in F = X + 1 V = U + F end

## => Asignación y suma de variables

Stack	Store	Е
Z=Y+F	x1 = 6 y1 = 4 z1 f1 = <b>7</b> p1 = proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end end, CE = {} q1 = proc {\$ U V} local F in F = X + 1 V = U + F end End, CE = {X-> x1, F -> f1}	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

## => Asignación y suma de variables

Stack	Store	Е
-	x1 = 6	X -> x1
	y1 = 4	Y -> y1
	z1 = <b>7(f1)</b> + <b>4(y1)</b> = <b>11</b>	Z -> z1
	f1 = 7	P -> p1
	p1 = proc {\$ A B}	Q -> q1
	proc {B U V}	·
	local F in	
	F=A+1	
	V=U+F	
	end	
	end	
	end, CE = {}	
	q1 = proc {\$ U V}	
	local F in	
	F = X + 1	
	V = U + F	
	end	
	End, CE = $\{X-> x1, F-> f1\}$	

# Programa 6

#### **Estado inicial**

Stack	Store	E
local X Y Z in     X = Y     try     X = 1Y = 2 Z = 3     catch Exception then     skip     end     {Browse X#Y#Z} end	-	-

#### => Declaración de variables

Stack	Store	E
X=Y	x1	X -> x1
try	y1	Y -> y1
X = 1 Y = 2 Z = 3	z1	Z -> z1
catch Exception then		
skip		
end		
{Browse X#Y#Z}		

#### => Asignación de variables

Stack	Store	E
try	x1	X -> y1
X = 1	y1	Y -> y1
Y = 2	z1	Z -> z1
Z = 3		
catch Exception then		
skip		
end		
{Browse X#Y#Z}		

## => Ejecución de try

Stack	Store	E
X = 1	x1	X -> y1
Y = 2	y1	Y -> y1

Z = 3	z1	Z -> z1
catch Exception then		
skip		
end		
{Browse X#Y#Z}		

#### => Ejecución de asignación de variable (dentro del catch)

Stack	Store	Е
Y = 2 Z = 3 catch Exception then skip end {Browse X#Y#Z}	x1 y1 = <b>1</b> z1	X -> y1 Y -> y1 Z -> z1

#### => Ejecución de asignación de variable (dentro del catch)

Stack	Store	Е
Y = 2 Z = 3 catch Exception then skip end {Browse X#Y#Z}	x1 y1 = 1 z1	X -> y1 Y -> y1 Z -> z1

Acá se intenta asignar a y1 el valor 2, pero esta variable ya tiene valor porque X = Y, entonces dispara la Exception y se quitan todas las operaciones hasta la del catch

#### => Ejecución Exception

Stack	Store	E
skip {Browse X#Y#Z}	x1 y1 = 1 z1	X -> y1 Y -> y1 Z -> z1

#### => Ejecución Skip

Stack	Store	Е
{Browse X#Y#Z}	x1 y1 = 1 z1	X -> y1 Y -> y1 Z -> z1

#### => Ejecución Browse

Stack	Store	Е
	x1 y1 = 1 z1	X -> y1 Y -> y1 Z -> z1

Como el valor de Z no esta determinado el browse mostrará un '\_' indicando que aun no se definió, en otras operaciones podría suspenderse la ejecución, pero no ocurre con Browse. La ejecución mostrará:

1#1#\_

# Ejercicio 4) - Case

#### {Test [b c a]}

Predicción: 'case'(4)

Ejecución: 'case'(4)

La lista no empieza con a como primer elemento, ni es un record, es una lista pero el primer y elemento no son iguales, luego es una lista, entonces case 4.

#### {Test f(b(3))}

Predicción: 'case'(5)

Ejecución: 'case'(5)

No es lista que empiece con a, si bien es un tupla llamada f, su valor no es a sino b(3), tampoco es una lista, con lo cual saltamos al caso 5 que cumple, dado que es un tupla llamada f y Y tomará el valor b(3)

#### {Test f(a)}

Predicción: 'case'(2)

Ejecución: 'case'(2)

No es lista, entonces analiza el case 2 donde coincide el nombre de la tupla y el elemento que contiene.

## {Test f(a(3))}

Predicción: 'case'(5)

Ejecución: 'case'(5)

No es lista, luego es una tupla llamada f pero el elemento no es a, luego caso 3 y 4 no cumple por no ser una lista, en el caso 5 satisface la condición porque se llama f y el valor de la variable Y será a(3)

#### {Test f(d)}

Predicción: 'case'(5)

Ejecución: 'case'(5)

No es lista, luego es una tupla llamada f pero el elemento no es a, luego caso 3 y 4 no cumple por no ser una lista, en el caso 5 satisface la condición porque se llama f y el valor de la variable Y será d

#### {Test [a b c]}

Predición: 'case'(1)

Ejecución: 'case'(1)

Es una lista que empieza con el valor a, cumple la primer condición.

#### {Test [c a b]}

Prediccion: 'case'(4)

Ejecución: case (4)

No es una lista que comience con el valor a, luego no es una tupla, luego no es una lista con los primeros dos elementos iguales. Finalmente es una lista por lo que entra en el caso 4.

#### {Test ala}

Predicción: 'case'(1)

Ejecución: 'case'(1)

Es una lista que comienza con el valor a.

#### {Test '|'(v b)}

Predicción: 'case'(6)

Ejecución: 'case'(4)

No es una lista que comience con a, tampoco es tupla, no es una lista con dos elementos iguales al inicio, pero si es una lista donde el valor de Zes b. El motivo por el cual mi predicción fue incorrecta es que pensé que el formato aceptado era una cabeza con una cola, o sea que debía ser 'l'(v [b]), pero es incorrecto.

#### {Test '|'(a a)}

Predicción: 'case'(6)

Ejecución: 'case'(1)

Es una lista que comienza con a, por el mismo motivo que el punto anterior me equivoqué en la predicción.

#### {Test 'l'(b b)}

Predicción 'case'(6)

Ejecución: 'case'(3)

No es una lista que comienza con a, luego tampoco es tupla, luego si es una lista con ambos elementos primero y segundo iguales.

#### {Test '|'(a b c)}

Predicción 'case'(6)

Ejecución: 'case'(6)

Al tener 3 elementos no coincide con una lista iniciando con a, luego no es tupla ni lista con ambos elementos iguales, tampoco es lista en la 4ta opcion, tampoco tupla llamada f, y queda como única opción el caso 6.

#### {Test 'l'(a [b c]}

Predicción: 'case'(1)

Ejecución: 'case'(1)

En este caso si crea con la cabeza y la cola la lista con 3 elementos que empiezan con a, por lo tanto coincide la primer condición.

# Ejercicio 5) - Recursividad

## 1. Traducción al lenguaje Kernel

```
local Length in
   Length = proc {$ Xs N}
            case Xs of nil then
               N = 0
            else
               case Xs of _|T then
                  local U in
                        {Length T U}
                        N = U + 1
                  end
               else
                  skip
               end
            end
           end
end
local K in
   {Length [1 2 3 4] K}
   {Show K}
end
```

#### 2. "Tail Recursive"

```
end
end
end
end
local K in
{Length [1 2 3 4] 0 K}
{Show K}
end
```

En el primer caso no es la llamada recursiva lo último que se ejecuta, entonces voy a tener las operaciones que siguen acumuladas en el stack y hasta que no se termine la invocación de la última llamada recursiva no voy a poder liberar el stack. Al hacerlo tail recursive no tengo en el stack operaciones pendientes, el cual me queda claramente más pequeño. Este es el motivo por el cual siempre debemos tratar de hacer la invocación a la recursividad al final. Esto se verá claramente en el punto 3.

#### 3. Ejecución en la máquina abstracta

Implementación básica

#### **Estado inicial**

Stack	Store	E
<pre>local Length in   Length = proc {\$ Xs N}         case Xs of nil then         N = 0         else         case Xs of _ T then             local U in             {Length T U}             N = U + 1             end         else             skip         end         end</pre>	-	-
end		
<pre>end local K in   {Length [1 2 3 4] K}   {Show K}</pre>		

end

#### => Declaración de variable

Stack	Store	E
<pre>Length = proc {\$ Xs N}     case Xs of nil then        N = 0     else        case Xs of _ T then        local U in        {Length T U}</pre>	length	length -> Length
N = U + 1 end		
else		
skip end		
end		
end local K in		
{Length [1 2 3 4] K} {Show K} end		

#### => Asignación de procedure value

Stack	Store	E
local K in {Length [1 2 3 4] K} {Show K} end	<pre>length = (proc {\$ Xs N}</pre>	length -> Length

#### => Declaración variable K

Stack	Store	E
{Length [1 2 3 4] K} {Show K}	<pre>length = (proc {\$ Xs N}      case Xs of nil then      N = 0</pre>	k-> K

## => Ejecución de procedure

Stack	Store	E
case Xs of nil then N = 0	<pre>length = (proc {\$ Xs N}     case Xs of nil then     N = 0</pre>	k-> K
else  case Xs of _ T then  local U in  {Length T U}  N = U + 1  end  else  skip  end	else	
end {Show K}	Store* CE*  xs = [1 2 3 4] xs -> Xs n = n -> K	

## => Ejecución de case

Stack	Store	E
<pre>case Xs of _ T then</pre>	<pre>length = (proc {\$ Xs N}</pre>	k-> K

k		
Store*	CE*	
xs = [1 2 3 4] n =	xs -> Xs n -> K	
n =	n -> K	

## => Ejecución de case

{Length T U} N = U + 1 end {Show K}  {Show K}     Case Xs of nil then   N = 0	Stack	E
Store* CE* xs = [1 2 3 4] xs -> Xs	{Length T U} N = U + 1 end	k-> K
n =		

#### => Declaración de U

Stack	Store	E
{Length T U} N = U + 1 {Show K}	<pre>length = (proc {\$ Xs N}</pre>	k-> K

Store*	CE*	
xs = [1 2 3 4] n = t = [2 3 4] u =	n -> K t -> T <b>u-&gt;U</b>	

## => Ejecución de Length

Stack	Store		E
<pre>case Xs of nil then     N = 0     else     case Xs of _ T then         local U in         {Length T U}         N = U + 1         end     else         skip     end</pre>	<pre>length = (proc {\$ Xs</pre>	il then f _ T then	k-> K
end N = U + 1 {Show K}	Store*  xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'=	CE**  xs' -> Xs n' -> N  t -> T u->U	

## => Ejecución de case

Stack	Store	E
<pre>case Xs of _ T then   local U in   {Length T U}   N = U + 1</pre>	<pre>length = (proc {\$ Xs N}     case Xs of nil then        N = 0     else        case Xs of _ T then</pre>	k-> K
end else skip	local U in {Length T U} N = U + 1 end	
end	else skip	

N = U + 1 {Show K}	end end end, *) k	end end, *)	
	Store*	CE**	
	xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'=	xs' -> Xs n' -> N t -> T u->U	

## => Ejecución de case

Stack	Store		E
local U in {Length T U} N = U + 1 end N = U + 1 {Show K}	length = (proc {\$ xs case Xs of n N = 0 else case Xs o local  end else skip end end end, *) k  Store*  xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4]	il then f _ T then	k-> K

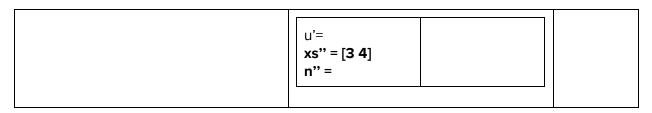
#### => Declaración de U

Stack	Store	E
{Length T U} N = U + 1	<pre>length = (proc {\$ Xs N}      case Xs of nil then      N = 0</pre>	k-> K

```
N = U + 1
                                                      else
                                                          case Xs of _|T then local U in
{Show K}
                                                                       {Length T U}
                                                                       N = U + 1
                                                             end
                                                          else
                                                             skip
                                                          end
                                                      end
                                               end, *)
                                                                     CE**
                                                Store*
                                               xs = [1 2 3 4]
                                                                     xs' -> Xs
                                                                     n' -> N
                                                n =
                                                t = [2 \ 3 \ 4]
                                                                     t' -> T
                                                                     u'->U
                                                u =
                                                xs'=[2 3 4]
                                                n'=
                                                t'=[3 4]
                                                u'=
```

## => Ejecución de Length

Stack	Store	E
<pre>case Xs of nil then      N = 0 else     case Xs of _ T then      local U in         {Length T U}      N = U + 1      end     else      skip     end end</pre>	<pre>length = (proc {\$ Xs N}</pre>	k-> K
N = U + 1 N = U + 1 {Show K}	Store*   CE***   xs = [1 2 3 4]  n =  t = [2 3 4]  u =  xs'' -> Xs  n" -> N  t' -> T  u'-> U  xs'=[2 3 4]  n'=  t'=[3 4]	



## => Ejecución de case

Stack	Store		E
<pre>case Xs of _ T then</pre>	<pre>length = (proc {\$ Xs</pre>	il then f _ T then	k-> K
	Store*	CE***	
	xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs" = [3 4] n" =	xs" -> Xs n" -> N t' -> T u'->U	

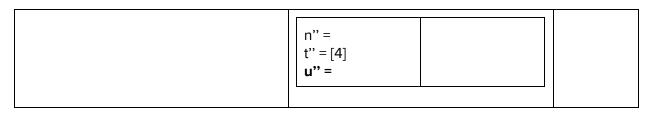
## => Ejecución de case

Stack	Store	E
<pre>local U in      {Length T U}      N = U + 1 end N = U + 1 N = U + 1 {Show K}</pre>	<pre>length = (proc {\$ Xs N}     case Xs of nil then     N = 0     else     case Xs of _ T then         local U in         {Length T U}         N = U + 1         end     else</pre>	k-> K

skip end end end, *) k		
Store*	CE***	
xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs'' = [3 4] n'' = t'' = [4]	xs" -> Xs n" -> N <b>t" -&gt; T</b> u'->U	

### => Declaración de variable U

Stack	Store		E
{Length T U} N = U + 1 N = U + 1 N = U + 1 {Show K}	<pre>length = (proc {\$ Xs N}</pre>		k-> K
	Store*	CE***	
	xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs'' = [3 4]	xs" -> Xs n" -> N t" -> T <b>u"-&gt;U</b>	



## => Ejecución de Length

Stack	Store		E
<pre>case Xs of nil then</pre>	length = (proc {\$ Xs case Xs of no N = 0 else	il then  f _ T then	k-> K
(SHOW K)	t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs'' = [3 4] n'' = t'' = [4] u'' = xs'''= [4] n'''=	t" -> T u"->U	

## => Ejecución de case

Stack	Store	E
<pre>case Xs of _ T then   local U in   {Length T U}</pre>	<pre>length = (proc {\$ Xs N}     case Xs of nil then     N = 0     else</pre>	k-> K

```
case Xs of _|T then
  local U in
               N = U + 1
        end
                                                                    {Length T U}
else
                                                                    N = U + 1
                                                          end
       skip
                                                       else
end
                                                          skip
                                                       end
N = U + 1
                                                    end
N = U + 1
                                            end, *)
N = U + 1
                                             k
{Show K}
                                                                  CE****
                                              Store*
                                                                  xs'"-> Xs
                                              xs = [1234]
                                              n =
                                                                  n''' -> N
                                                                  t'' -> T
                                              t = [2 \ 3 \ 4]
                                                                  u"->U
                                              u =
                                              xs'=[2 3 4]
                                              n'=
                                              t'=[3 4]
                                              u'=
                                              xs'' = [3 4]
                                              n'' =
                                              t'' = [4]
                                              u" =
                                              xs''' = [4]
                                              n""=
```

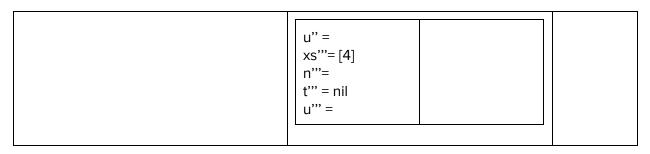
#### => Ejecución de case

Stack	Store		E
local U in {Length T U} N = U + 1 end N = U + 1 N = U + 1 N = U + 1 Show K}	<pre>length = (proc {\$ xs</pre>	il then f _ T then	k-> K

		1
n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs" = [3 4] n" = t" = [4] u" = xs""= [4] n""= t"" = nil	n''' -> N t''' -> T u''->U	

## => Declaración variable U

Stack	Store		E
{Length T U} N = U + 1 N = U + 1 N = U + 1 N = U + 1 N = U + 1 {Show K}	<pre>length = (proc {\$ Xs</pre>	il then f _ T then	k-> K
	Store*  xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs'' = [3 4] n'' = t'' = [4]	CE****  xs'"-> Xs n'"-> N t'"-> T u"'-> U	



## => Ejecución Length

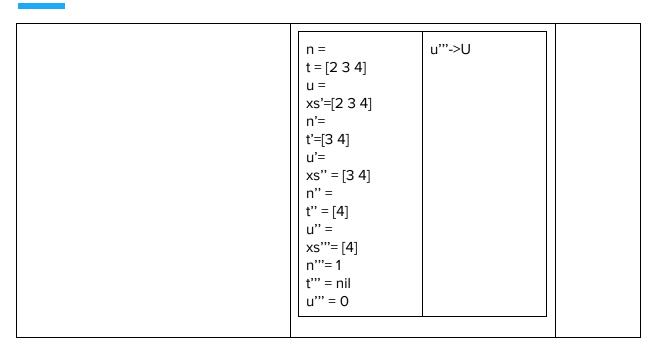
Stack	Store		E
<pre>case Xs of nil then     N = 0 else     case Xs of _ T then         local U in         {Length T U}         N = U + 1         end     else         skip     end end</pre>	end else skip end end end end k	il then  f _ T then  J in {Length T U}  N = U + 1	k-> K
N = U + 1 N = U + 1 N = U + 1 {Show K}	Store*  xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs" = [3 4] n" = t' = [4] u" = xs"'= [4] n""= t'" = nil u" = xs"'= nil n""=	xs'''-> Xs n''' -> N t''' -> T u'''->U	

## => Ejecución case

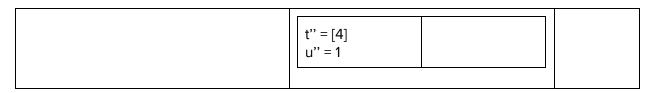
Stack	Store		E
N = 0 N = U + 1 N = U + 1 N = U + 1 N = U + 1 {Show K}	<pre>length = (proc {\$ Xs N}</pre>		k-> K
	Store*	CE****	
	xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs'' = [3 4] n'' = t'' = [4] u'' = xs'''= [4] n'''= t''' = nil u''' = xs''''= nil n''''=	xs''"-> Xs n''" -> N t'" -> T u'"->U	

Stack	Store	E
N = U + 1 N = U + 1 N = U + 1 N = U + 1 {Show K}	<pre>length = (proc {\$ Xs N}</pre>	k-> K

Stack	Store		E
N = U + 1 N = U + 1 N = U + 1 {Show K}	<pre>length = (proc {\$ Xs</pre>	il then f _ T then	k-> K
	Store*	CE***	
	xs = [1 2 3 4]	n''' -> N	



Stack	Store		E
N = U + 1 N = U + 1 {Show K}	<pre>length = (proc {\$ Xs</pre>	il then f _ T then	k-> K
	Store*	CE**	
	xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= t'=[3 4] u'= xs'' = [3 4] n'' = 2	n" -> N u"->U	



Store		E
case Xs of N = 0 else case Xs of local end else skip end end end, *)	<pre>case Xs of nil then   N = 0 else   case Xs of _ T then       local U in</pre>	
Store*	CE*	
xs = [1 2 3 4] n = t = [2 3 4] u = xs'=[2 3 4] n'= 3 t'=[3 4] u'= 2	n' -> N u'->U	
	length = (proc {\$ X:	<pre>length = (proc {\$ Xs N}</pre>

Stack	Store	E
{Show K}	<pre>length = (proc {\$ Xs N}</pre>	k-> K

k = 4		
Store*	CE	
xs = [1 2 3 4] n = 4 t = [2 3 4] u = 3	n -> N u->U	

## => Ejecución de show, muestra un 4

Stack	Store	E
	<pre>length = (proc {\$ Xs N}</pre>	k-> K

## Implementación Tail Recursive

## Estado inicial

Stack	Store	E
local Length in	-	-
Length = proc {\$ Xs A N}		
case Xs of nil then		
N = A		
else		
case Xs of _ T then		
local X in		
X = A + 1		
{Length T X N}		
end		
else		
skip		
end		
end		

```
end
end
local K in
{Length [1 2 3 4] 0 K}
{Show K}
end
```

### => Declaración de variable

Stack	Store	E
<pre>Length = proc {\$ Xs A N}   case Xs of nil then       N = A   else       case Xs of _ T then       local X in       X = A + 1       {Length T X N}</pre>	length =	Length ->length
end else skip end end end local K in {Length [1 2 3 4] 0 K}		
{Show K} end		

## => Asignación procedure value

Stack	Store	E
local K in {Length [1 2 3 4] 0 K} {Show K} end	<pre>length ={proc {\$ Xs A N}     case Xs of nil then         N = A     else         case Xs of _ T then</pre>	Length ->length

#### => Declaración variable K

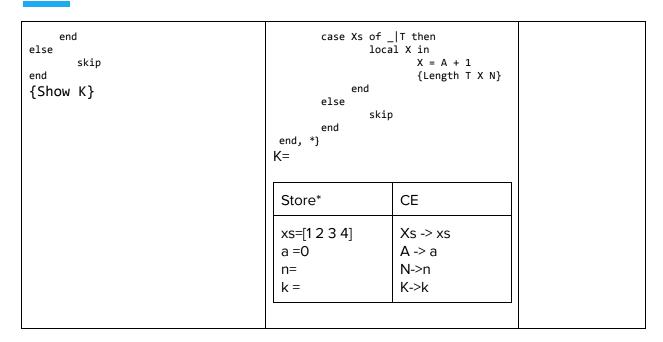
Stack	Store	E
{Length [1 2 3 4] 0 K} {Show K}	<pre>length ={proc {\$ Xs A N}   case Xs of nil then</pre>	Length ->length <b>K -&gt; k</b>

## => Ejecución de Length

Stack	Store	E
<pre>case Xs of nil then     N = A else      case Xs of _ T then</pre>	<pre>length ={proc {\$ Xs A N}     case Xs of nil then</pre>	Length ->length K -> k
	Store* CE  xs=[1 2 3 4]	

## => Ejecución de case

Stack	Store	E
<pre>case Xs of _ T then     local X in         X = A + 1         {Length T X N}</pre>	<pre>length ={proc {\$ Xs A N}   case Xs of nil then       N = A   else</pre>	Length ->length K -> k



### => Ejecución de case

Stack	Store		Е
<pre>local X in      X = A + 1      {Length T X N} end {Show K}</pre>	length ={proc {\$ Xs} case Xs of nil the	n _ T then al X in	Length ->length K -> k

#### => Declaración de variable local X

Stack	Store	E
-------	-------	---

```
X = A + 1
                                                                                      Length ->length
                                          length ={proc {$ xs A N}
{Length T X N}
                                           case Xs of nil then
                                                                                      K \rightarrow k
{Show K}
                                                  N = A
                                           else
                                                  case Xs of _|T then local X in
                                                                   X = A + 1
                                                                   {Length T X N}
                                                       end
                                                  else
                                                           skip
                                                  end
                                           end, *}
                                          K=
                                           Store*
                                                                CE
                                                                Xs \rightarrow xs
                                           xs=[1 2 3 4]
                                           a =0
                                                                A -> a
                                           n=
                                                                N->n
                                           k =
                                                                K->k
                                           t=[2 3 4]
                                                                T->t
                                           x=
                                                                X->x
```

## => Asignación de valor

Stack	Store		Е
{Show K}			Length ->length K -> k
	Store*	CE	
	xs=[1 2 3 4] a =0 n= k = t=[2 3 4]	Xs -> xs A -> a N->n K->k T->t	

x=1	X->x	
-	<u>.</u>	

## => Ejecución de Length

Stack		Store		E
	<pre>of nil then N = A  case Xs of _ T then</pre>	length ={proc {\$ Xs case Xs of nil ther N = A else case Xs of _	T then  I X in  X = A + 1  {Length T X N}	Length ->length K -> k
		a'=1 n'=		

## => Ejecución de case

Stack	Store	E
<pre>case Xs of _ T then</pre>	<pre>length ={proc {\$ Xs A N}     case Xs of nil then</pre>	Length ->length K -> k

Store*       CE*         xs=[1 2 3 4]       Xs -> xs'         a = 0       A -> a'         n=       N->n'         k =       K->k         t=[2 3 4]       T->t         y=1       Y->y	K=		
a =0 n= k = t=[2 3 4] A -> a' N->n' K->k T->t	Store*	CE*	
xs'=[2 3 4] a'=1 n'=	a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1	A -> a' N->n' K->k	

## => Ejecución de case

Stack	Store		E
<pre>local X in</pre>	end else		Length ->length K -> k
	Store*	CE*	
	xs=[1 2 3 4] a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1 n'= t'=[3 4]	Xs -> xs' A -> a' N->n' K->k T->t' X->x	

#### => Declaración de X

Stack	Store		E
<pre>X = A + 1 {Length T X N} {Show K}</pre>	l end else		Length ->length K -> k
	Store*  xs=[1 2 3 4] a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1 n'= t'=[3 4] x'=	CE*  Xs -> xs' A -> a' N->n' K->k T->t' X->x'	

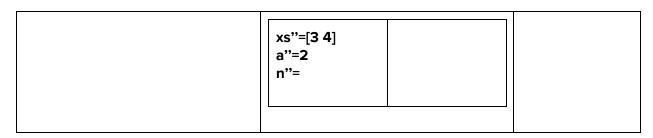
## => Asignación de valor

Stack	Store	E
{Length T X N} {Show K}	<pre>length ={proc {\$ Xs A N}     case Xs of nil then</pre>	Length ->length K -> k

Store*         CE*           xs=[1 2 3 4]         Xs -> xs'           a = 0         A -> a'           n=         N->n'           k =         K->k           t=[2 3 4]         T->t'           x=1         X->x'           xs'=[2 3 4]         a'=1           n'=         t'=[3 4]           x'= 2         x'= 2			
xs=[1 2 3 4]			
a = 0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1 n'= t'=[3 4]	Store*	CE*	
	a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1 n'= t'=[3 4]	A -> a' N->n' K->k T->t'	

## => Ejecución de Length

Stack	Store		E
<pre>case Xs of nil then     N = A else      case Xs of _ T then</pre>	<pre>length ={proc {\$ Xs   case Xs of nil then      N = A   else</pre>	T then I X in X = A + 1 {Length T X N}	Length ->length K -> k
	Store*  xs=[1 2 3 4] a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1	CE**  Xs -> xs" A -> a" N->n" K->k T->t' X->x'	
	n'= t'=[3 4] x'= 2		



## => Ejecución de case

Stack	Store		Е
<pre>case Xs of _ T then</pre>	<pre>length ={proc {\$ Xs}   case Xs of nil ther     N = A   else         case Xs of _         loca          end         else         skip     end   end, *} K=</pre>	T then    X in   X = A + 1   {Length T X N}	Length ->length K -> k
	Store*  xs=[1 2 3 4] a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1 n'= t'=[3 4] x'= 2 xs''=[3 4] a''=2 n''=	Xs -> xs" A -> a" N->n" K->k T->t' X->x'	

## => Ejecución de case

Stack	Store	E	

```
local X in
                                                                                  Length ->length
                                        length ={proc {$ Xs A N}
       X = A + 1
                                         case Xs of nil then
                                                                                  K \rightarrow k
       {Length T X N}
                                                N = A
end
                                         else
                                                case Xs of _|T then local X in
{Show K}
                                                                X = A + 1
                                                                {Length T X N}
                                                     end
                                                else
                                                        skip
                                                end
                                         end, *}
                                        K=
                                         Store*
                                                             CE**
                                                             Xs -> xs"
                                         xs=[1 2 3 4]
                                                             A -> a"
                                         a =0
                                                             N->n"
                                         n=
                                                             K->k
                                         k =
                                                             T->t"
                                         t=[2 3 4]
                                         x=1
                                                             X->x'
                                         xs'=[2 3 4]
                                         a'=1
                                         n'=
                                         t'=[3 4]
                                         x'= 2
                                         xs"=[3 4]
                                         a"=2
                                         n"=
                                         t"=[4]
```

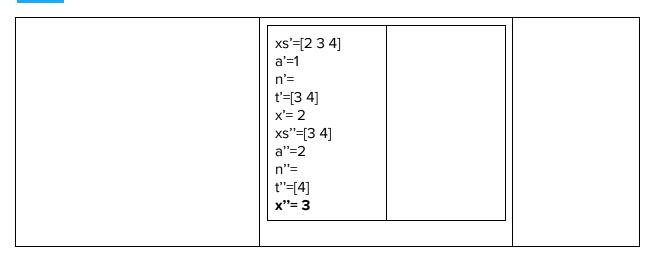
#### => Declaración de variable X

Stack	Store	E
<pre>X = A + 1 {Length T X N} {Show K}</pre>	<pre>length ={proc {\$ Xs A N}     case Xs of nil then</pre>	Length ->length K -> k

n=

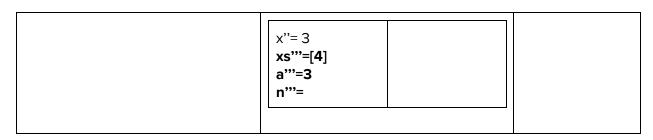
## => Asignación de valor

Stack	Store		E
{Length T X N} {Show K}	end else		Length ->length K -> k
	Store*	CE**	
	xs=[1 2 3 4] a =0 n= k = t=[2 3 4] x=1	Xs -> xs" A -> a" N->n" K->k T->t" X->x"	



## => Ejecución de Length

Stack	Store		Е
case Xs of nil then     N = A else  case Xs of _ T then     local X in     X = A + 1     {Length T X N}     end else     skip end end {Show K}	<pre>length ={proc {\$ Xs   case Xs of nil then      N = A   else      case Xs of _</pre>	T then   X in   X = A + 1   {Length T X N}	Length ->length K -> k
	n"= t"=[4]		



## => Ejecución de Case

Stack	Store		E
<pre>case Xs of _ T then</pre>	length ={proc {\$ xs case Xs of nil the N = A else case Xs of local end else skillend end, *} K=  Store*  Store*  xs=[1 2 3 4] a = 0 n= k = t=[2 3 4] x=1 xs'=[2 3 4] a'=1	_ T then al X in X = A + 1 {Length T X N}	E Length ->length K -> k
	n'= t'=[3 4] x'= 2 xs''=[3 4] a''=2 n''= t''=[4] x''= 3 xs'''=[4] a'''=3 n'''=		

## => Ejecución de Case

Stack	Store		E
<pre>local X in</pre>	end else s end end, *} K=	hen  f _ T then ocal X in	Length ->length K -> k
	Store*  xs=[1 2 3 4] a =0 n= k = t=[2 3 4] x=1 xs'=[2 3 4]	CE***  Xs -> xs''' A -> a''' N->n''' K->k T->t''' X->x'''	
	a'=1 n'= t'=[3 4] x'= 2 xs"=[3 4] a"=2 n"=		
	t"=[4] x"= 3 xs""=[4] a""=3 n""= t""=nil		

### => Declaración de variable X

Stack	Store	E
<pre>X = A + 1 {Length T X N} {Show K}</pre>	<pre>length ={proc {\$ Xs A N}   case Xs of nil then     N = A</pre>	Length ->length K -> k

```
else
       case Xs of \_|\mathsf{T} then
               local X in
                       X = A + 1
                       {Length T X N}
            end
       else
                skip
       end
end, *}
K=
 Store*
                     CE***
 xs=[1 2 3 4]
                    Xs -> xs'"
                     A -> a'''
 a =0
 n=
                    N->n'''
                     K->k
 k =
                    T->t'''
 t=[2 3 4]
                    X->x'''
 x=1
xs'=[2 3 4]
 a'=1
 n'=
 t'=[3 4]
 x'= 2
xs"=[3 4]
 a"=2
 n"=
t"=[4]
 x"= 3
xs'"=[4]
 a'''=3
 n'''=
 t""=nil
 χ";=
```

## => Asignación de valor

Stack	Store	E
{Length T X N} {Show K}	<pre>length ={proc {\$ Xs A N}   case Xs of nil then       N = A   else       case Xs of _ T then</pre>	Length ->length K -> k

skip end end, *} K=	
Store*       CE***         xs=[1 2 3 4]       Xs -> xs"'         a = 0       A -> a"'         n=       N->n"         k =       K->k         t=[2 3 4]       T->t"'         xs1       X->x"         xs2=[2 3 4]       X->x"         a'=1       a'=1         n'=       t'=[3 4]         a"=2       n"=         t"=[4]       a"=3         n"=       t"=nil         x"=4       x"=4	

## => Ejecución de Length

Stack	Store	E
<pre>case Xs of nil then     N = A else  case Xs of _ T then     local X in     X = A + 1     {Length T X N}     end     else         skip     end end {Show K}</pre>	<pre>length ={proc {\$ Xs A N}     case Xs of nil then</pre>	Length ->length K -> k

## => Ejecución de case

Stack	Store		E
N = A {Show K}	<pre>length ={proc {\$ Xs   case Xs of nil ther       N = A   else       case Xs of _       loca        end       else       skip   end   end, *} K=</pre> Store*	T then  I X in  X = A + 1  {Length T X N}	Length ->length K -> k

## => Asignación de valor

Stack	Store		E
{Show K}	<pre>length ={proc {\$ Xs   case Xs of nil ther      N = A   else</pre>	T then I X in X = A + 1 {Length T X N}	Length ->length K -> k
	Store*	CE****	
	xs=[1 2 3 4]	Xs -> xs''''	

## => Ejecución de show, muestra un K=4 y finaliza la ejecución

Stack	Store	E
	<pre>length ={proc {\$ Xs A N}     case Xs of nil then</pre>	Length ->length K -> k

## Ejercicio 6) - Alto orden con listas

### FoldL

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                   X
File Edit Options Buffers Tools Oz Help
% FoldL function
declare fun {FoldL L F U}
           case L of nil then U
           [] H|T then {F {FoldL T F U} H}
           end
       end
declare fun {Multiplica X Y}
          X * Y
        end
{Show {FoldL [2 2 3] Multiplica 1}}
1\**- Oz
                      All L10
                               (Oz)
12
1\**- *Oz Emulator* All L2 (Comint:run)
menu-bar Oz Feed Buffer
```

### FoldR

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                    X
File Edit Options Buffers Tools Oz Help
% FoldR function
declare fun {FoldR L F U}
           case L of nil then U
           [] H|T then {F H {FoldR T F U}}
        end
declare fun {Suma X Y}
           X + Y
        end
{Show {FoldR [1 2 3] Suma 0}}
1\**- Oz
                      All L4
                                  (Oz)
1\**- *Oz Emulator* All L2 (Comint:run)
menu-bar Oz Feed Buffer
```

## Map

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                    X
File Edit Options Buffers Tools Oz Help
                                          a
% Map function
declare fun {Map L F }
           case L of nil then nil
           [] H|T then {F H}| {Map T F}
           end
        end
declare fon {Potencia X}
           X * X
{Show {Map [2 1 3 6] Potencia}}
1\--- 6Map
                      All L7
                                  (Oz)
[4 1 9 36]
1\**- *Oz Emulator*
                      All L2 (Comint:run)
Wrote c:/Users/Karina/Facu/7524/TPIndividual/6Map
```

## Filter

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                    X
File Edit Options Buffers Tools Complete In/Out Signals Help
            X
% Filter function
declare fun {Filter L F}
           case L of nil then nil
           [] H|T then
              if {F H} then
                 H | {Filter T F}
              else
                 {Filter T F}
              end
           end
        end
declare fun {Condicion X}
           X > 2
{Show {Filter [2 1 3 6] Condicion}}
1\--- 6Filter
                     A11 L10
                                 (Oz)
[3 6]
1\**- *Oz Emulator* All L2 (Comint:run)
```

## Ejercicio 7) - Hilos

#### WaitSome

a. Código Oz

```
end
}
{Show 'Esperando por algun valor a bindear'}
{Wait Y}
{Show 'Al menos uno fue bindeado'}

end

end
{Show 'Primera lista los 3 valores bindeados'}
{WaitSome [1 2 3]}
{Show 'Segunda lista solo uno esta bindeado'}
{WaitSome [_ 2 _]}
{Show 'Tercera lista ninguno esta bindeado'}
{WaitSome [_ 2 _]}
end
```

```
@ emacs@KARINA-VAIO
                                                                             X
File Edit Options Buffers Tools Complete In/Out Signals Help
                  end
   {Show 'Primera lista los 3 valores bindeados'}
   {WaitSome [1 2 3]}
   {Show 'Segunda lista solo uno esta bindeado'}
   {WaitSome [_ 2 _]} {Show 'Tercera lista ninguno esta bindeado'}
   {WaitSome [_ _ _]}
end
1\**- Oz
                      Bot L16
'Primera lista los 3 valores bindeados'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
'Valor Bindeado:'
'Valor Bindeado:'
'Al menos uno fue bindeado'
'Segunda lista solo uno esta bindeado'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
'Al menos uno fue bindeado'
'Tercera lista ninguno esta bindeado'
'Esperando por algun valor a bindear'
1\**- *Oz Emulator* All L3
                                   (Comint:run)
```

### Maquina abstracta

a. Código Oz

```
local A B C in
   thread
   if A then
   B=true
   {Show 'thread 1 A is true'}
   else
```

```
B=false
   {Show 'thread 1 A is false'}
   end
end
thread
   if B then
    C=false
   {Show 'thread 2 B is true'}
   else
    C=true
   {Show 'thread 2 B is false'}
   end
end
A=false
end
```

#### b. Ejemplo de ejecución

Le agregué impresiones por pantalla para poder ver claramente el seguimiento del código. Vemos que se lanzan dos threads de ejecución, ambos comienzan con una sentencia if, la cual se suspende porque ni A ni B están definidos, entonces se suspende. En el momento de ejecutar A=false, el thread 1 se libera puesto que tiene asignado el valor de A y ejecuta la sentencia "B = false". En consecuencia, B está definido por lo que la ejecución del thread 2 puede continuar y evaluar el if, para finalmente asignar el valor C=true.

```
@ emacs@KARINA-VAIO
                                                                         X
File Edit Options Buffers Tools Oz Help
                         | * • • • •
local A B C in
   thread
     if A then
        {Show 'thread 1 A is true'}
      else
        B=false
        {Show 'thread 1 A is false'}
     end
   end
   thread
     if B then
        C=false
        {Show 'thread 2 B is true'}
      else
        C=true
        {Show 'thread 2 B is false'}
     end
   end
   A=false
                                 (Oz)
1\--- 7Thread
                     Top L8
'thread 1 A is false'
'thread 2 B is false'
1\**- *Oz Emulator* All L3
                                (Comint:run)
menu-bar Oz Feed Buffer
```

## Ejercicio 8) - Evaluación perezosa

## 8.1 Maquina Abstracta

a. Código Oz

```
local Ints in
local L in
local A in
```

```
local B in
          local C in
             fun lazy {Ints N}
             {Show N}
             N|{Ints N+1}
             end
             {Show 'Comienzo'}
             L = \{Ints 1\}
             {Show 'Asignacion de L'}
             A = L.2.2.1
             {Show 'Asignacion de A usando L'}
             B = L.1
             {Show 'Asignacion de B usando L'}
             C = A+B
             {Show 'Asignacion de C'}
             {Show C}
          end
      end
      end
   end
end
```

```
emacs@KARINA-VAIO
                                                                                   X
                                                                             File Edit Options Buffers Tools Complete In/Out Signals Help
local Ints in
   local L in
      local A in
         local B in
            local C in
                fun lazy {Ints N}
                   {Show N}
                  N | {Ints N+1}
                end
                {Show 'Comienzo'}
               L = {Ints 1}
                {Show 'Asignacion de L'}
               A = L.2.2.1
                {Show 'Asignacion de A usando L'}
               B = L.1
                {Show 'Asignacion de B usando L'}
               C = A+B
                {Show 'Asignacion de C'}
                {Show C}
            end
1\--- 8Ints
                       Top L1
                                   (Oz)
'Comienzo'
'Asignacion de L'
2
'Asignacion de A usando L'
'Asignacion de B usando L'
'Asignacion de C'
1\**- *Oz Emulator*
                        All L4
                                    (Comint:run)
Beginning of buffer
```

#### 8.2 Reverse

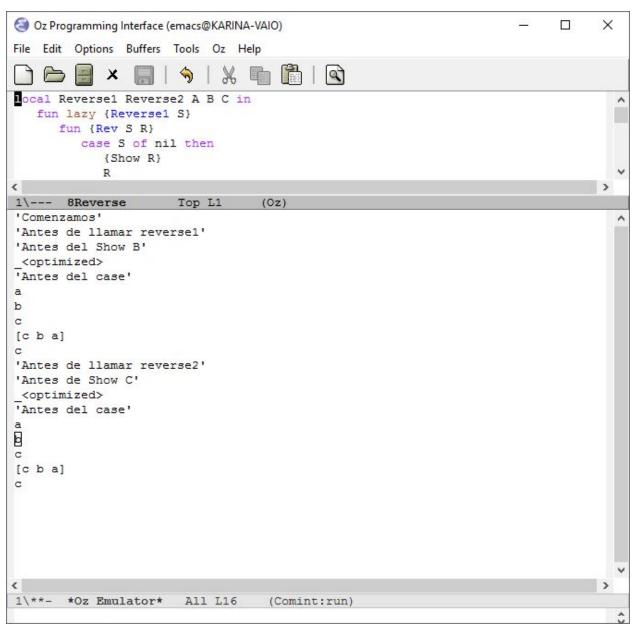
La diferencia en comportamiento es que la función Rev al ser lazy en la segunda implementación, solo va a ser ejecutada, cuando realmente sea necesario y no antes.
 Lo que ocurre es que como a su vez se encuentra dentro de otra funcion lazy, tambien va a ser ejecutada cuando necesitemos los valores de la lista y eso va a ocurrir al mismo tiempo, la necesidad de ejecución de una, en este caso fuerza a la ejecución de la otra.

- Ambas funciones retornan el mismo resultado, recién se ejecutan la función recursiva al hacer uso de los elementos de la lista. A continuación un ejemplo de ejecución:

#### Codigo oz

```
local Reverse1 Reverse2 A B C in
   fun lazy {Reverse1 S}
     fun {Rev S R}
      case S of nil then
          {Show R}
          R
       [] X|S2 then
          {Show X}
          {Rev S2 X|R}
       end
      end
   in {Rev S nil} end
  fun lazy {Reverse2 S}
      fun lazy {Rev S R}
      case S of nil then
          {Show R}
       [] X|S2 then
          {Show X}
          {Rev S2 X|R}
       end
      end
   in {Rev S nil} end
   {Show 'Comenzamos'}
   A = [a b c]
   {Show 'Antes de llamar reverse1'}
   B = {Reverse1 A}
   {Show 'Antes del Show B'}
 % {Show {Reverse1 A}}
 % {Show A}
   {Show B}
  {Show 'Antes del case'}
   case B of K|_ then
      {Show K}
   end
   {Show 'Antes de llamar reverse2'}
```

```
C = {Reverse2 A}
  {Show 'Antes de Show C'}
  %{Show {Reverse2 A}}
  {Show C}
  {Show 'Antes del case'}
  case C of M|_ then
     {Show M}
  end
end
```



## Ejercicio 9) - Mensajes

a. Código Oz

```
declare P in
local S in
   {NewPort S P}
   thread {ForAll S proc {$ M} {Browse M} end} end
   {Send P a}
   {Send P b}
   {Send P c}
   {Send P 1}
end
```

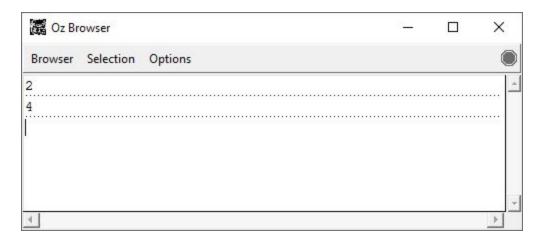
```
Browser Selection Options

a
b
c
1
```

## Ejercicio 10) - Servidor de filtros

a. Código Oz

```
%funcion unaria
declare fun {EsPar A}
         if (A \mod 2 == 0) then
            true
         else
            false
         end
      end
%P1 puerto donde envio todos los elementos
%P2 puerto donde envio solo los elementos que pasan el filtro
declare P1 P2 in
local S1 S2 in
   {NewPort S1 P1}
   {NewPort S2 P2}
   thread {ForAll S1 proc {$ M}
                  if {EsPar M} then
                     {Send P2 M}
                  end
                 end} end
   thread {ForAll S2 proc {$ M} {Browse M} end} end
   {Send P1 1}
   {Send P1 2}
   {Send P1 3}
   {Send P1 4}
end
```



# Ejercicio 11) - Celdas de memoria

## Explicación linea a linea

<pre>declare X={NewCell 0}</pre>	Declara la celda X y le asigna el valor inicial 0.
{Assign X 5}	A la celda X le asigna el nuevo valor 5
Y=X	A la celda Y le asigna X, ambas variables son la misma y tienen el mismo valor.
{Assign Y 10}	A la celda Y e asigna el valor 10, también cambiará el valor de X porque es la misma celda que puedo acceder con dos identificadores.
{Browse {Access X}==10}	Pregunta si el valor de X es 10, lo cual es verdadero. Imprime true
{Browse X==Y}	Pregunta si la celda X es igual a la celda Y, dado que son la misma celda, el valor es verdadero.  Imprime true
Z={NewCell 10}	Declara una nueva celda Z, le asigna el valor 10
{Browse Z==Y}	Pregunta si la celda Z es igual a la celda Y, como son dos celdas diferentes la comparación es falsa.  Imprime false
{Browse @X==@Z}	Pregunta si el valor de la celda X, que es 10 es igua al valor de la celda Z que también es 10 son iguales, la condicion es verdadera.  Imprime true