7524 - Teoría de la programación

TP Individual

Karina Alaya
Padron 75840

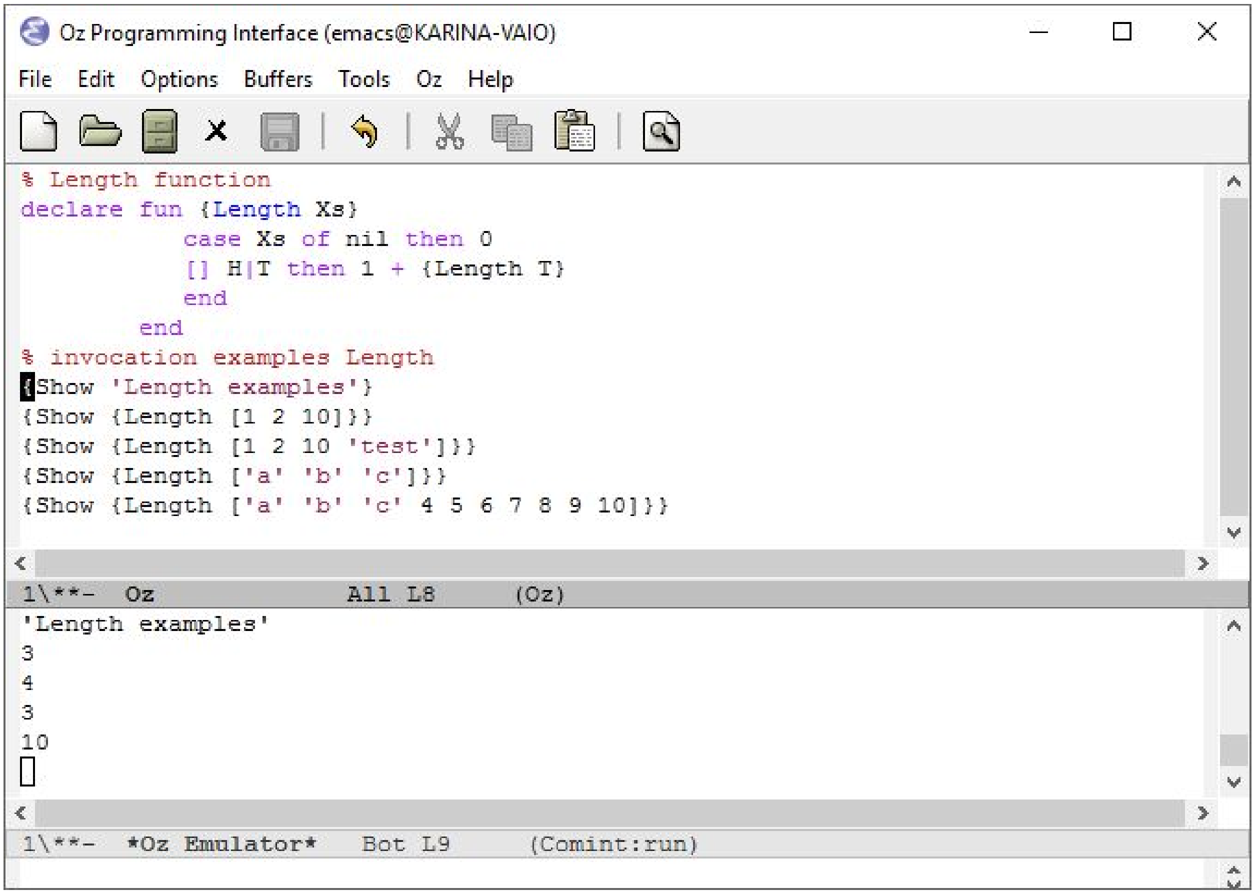
Ejercicio 1) - Procesamiento de listas

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)
File Edit Options Buffers Tools Oz Help
1**- Oz All L8 (Oz)
1**- *Oz Emulator* Bot L9 (Comint:run)

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

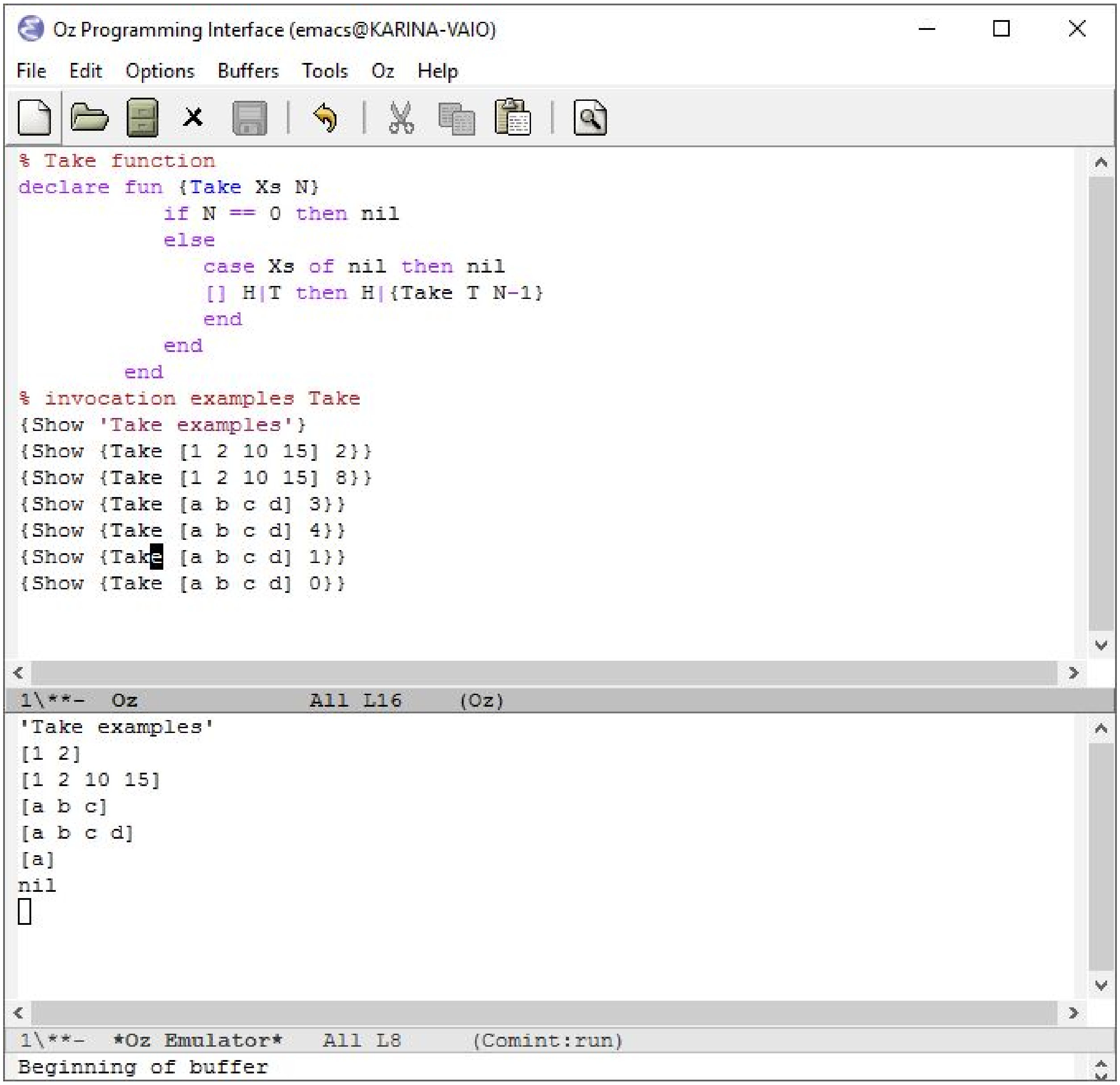
```
'Length examples'
3
4
3
10
```

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
```

b. Ejemplos de ejecución


Oz Programming Interface (emacs@KARINA-VAIO)
File Edit Options Buffers Tools Oz Help

```
% 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
% 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]
[a b c d]
[a]
nil
```

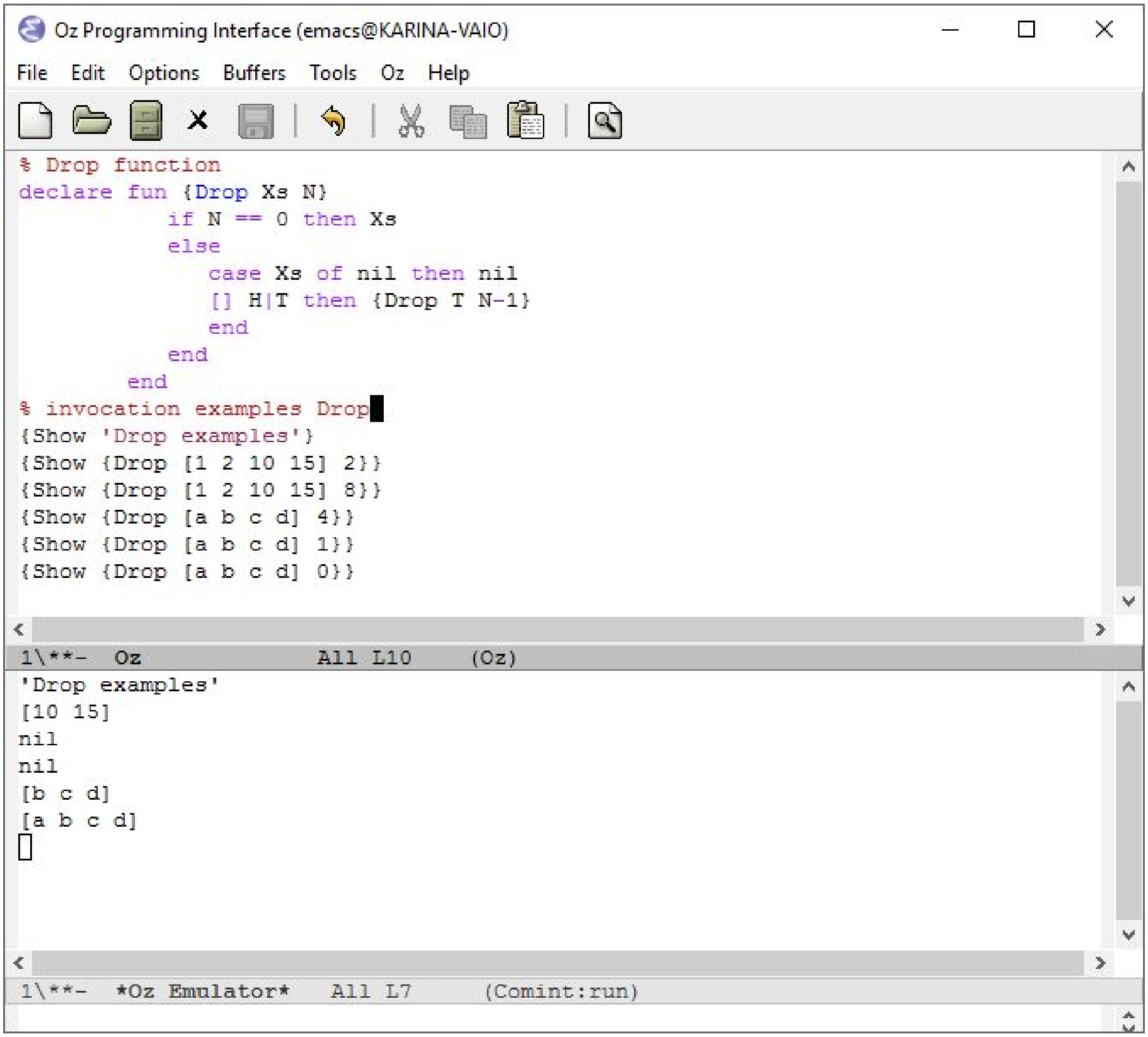
1**- *Oz Emulator*   All L8   (Comint:run)
Beginning of buffer

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
```

b. Ejemplos de ejecución



Oz Programming Interface (emacs@KARINA-VAIO)
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]
[a b c d]
```

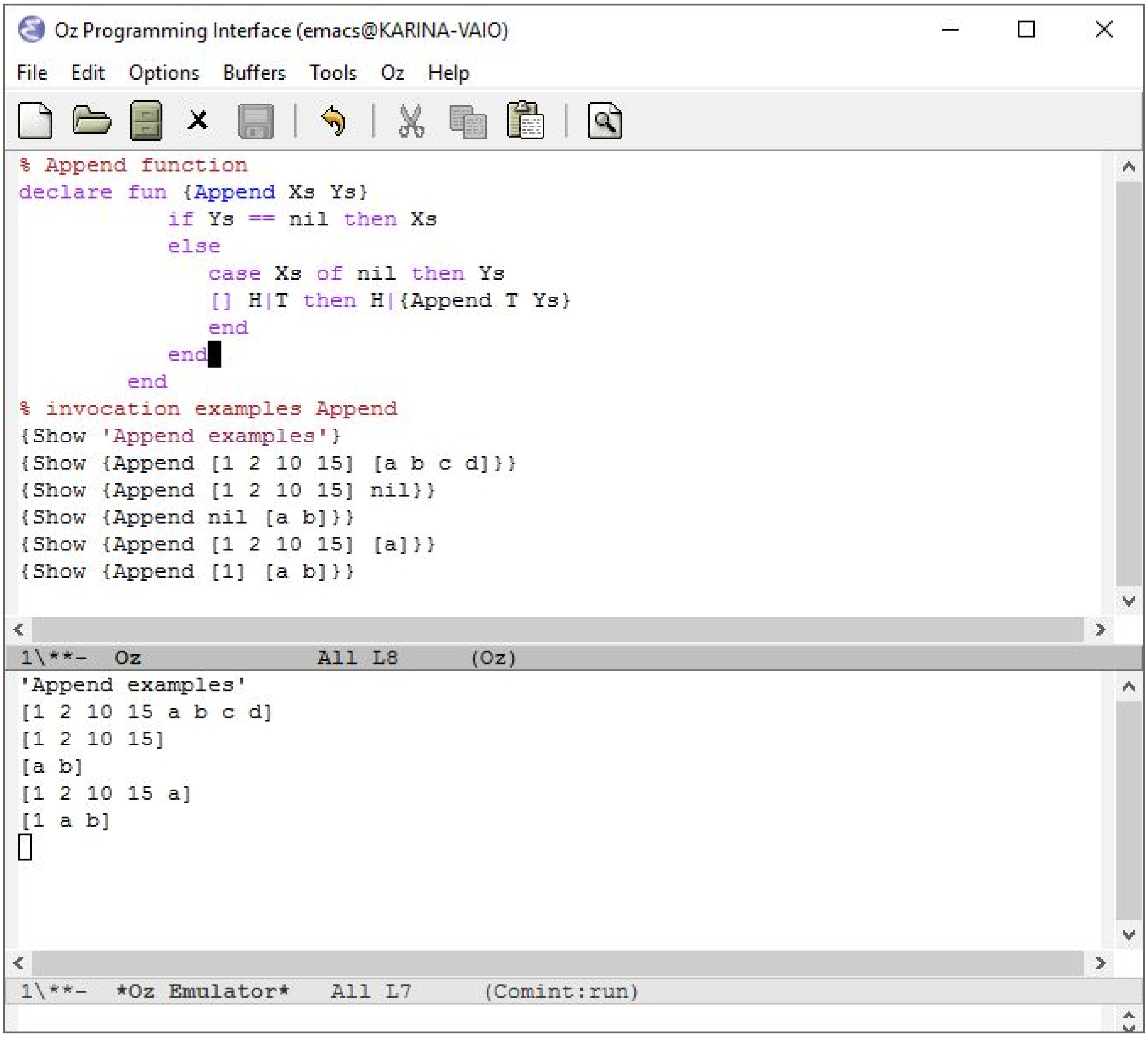
```
1\**-  *Oz Emulator*   All L7      (Comint:run)
```

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
```

b. Ejemplos de ejecución


Oz Programming Interface (emacs@KARINA-VAIO)
File Edit Options Buffers Tools Oz Help
1**- Oz All L8 (Oz)
1**- *Oz Emulator* All L7 (Comint:run)

```
% 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]}}
```

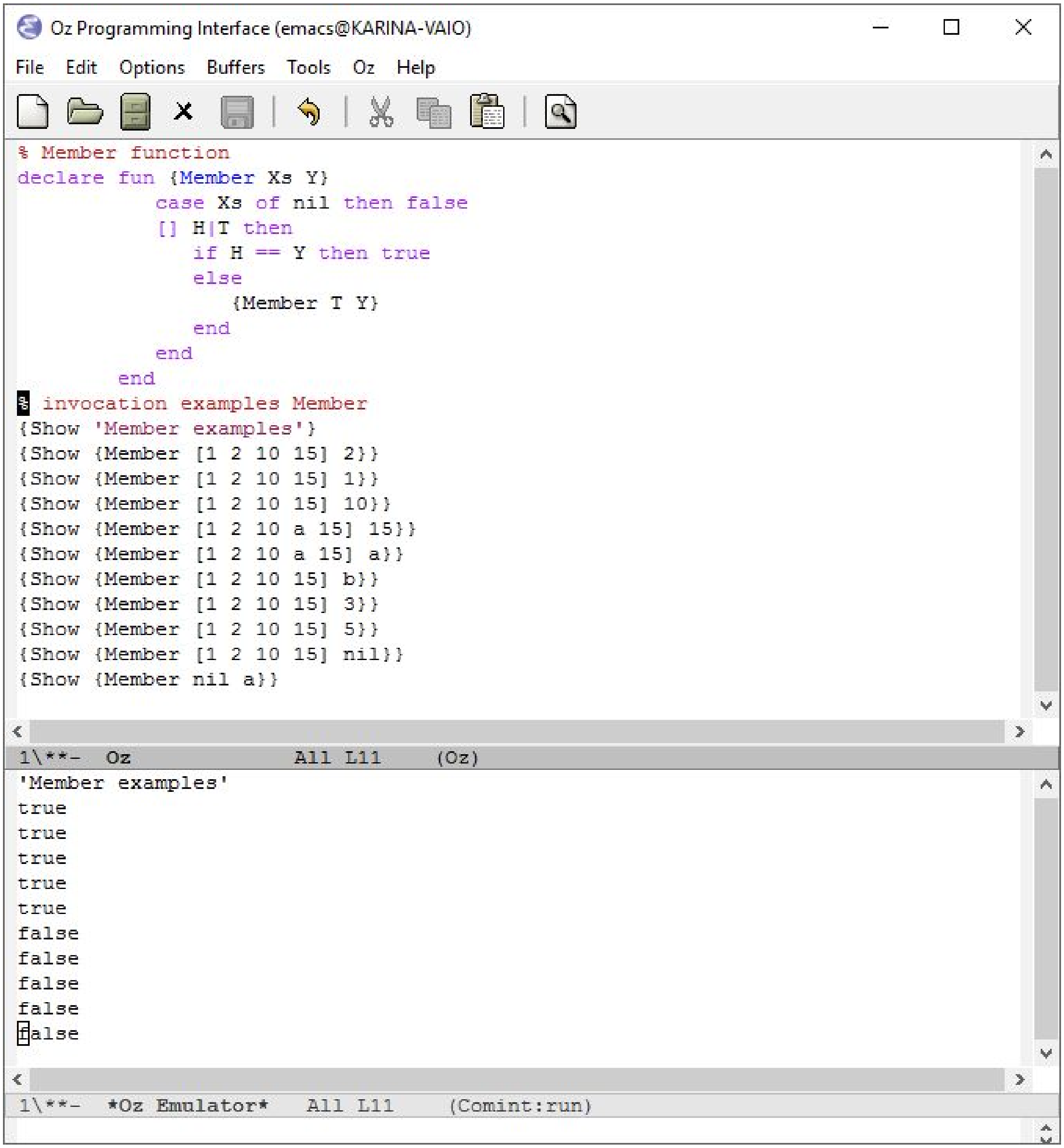
```
'Append examples'
[1 2 10 15 a b c d]
[1 2 10 15]
[a b]
[1 2 10 15 a]
[1 a b]
```

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
```

b. Ejemplos de ejecución


Oz Programming Interface (emacs@KARINA-VAIO)
File
Edit
Options
Buffers
Tools
Oz
Help
1**- Oz   All L11   (Oz)
1**- *Oz Emulator*   All L11   (Comint:run)

```
% 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
% 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}}
```

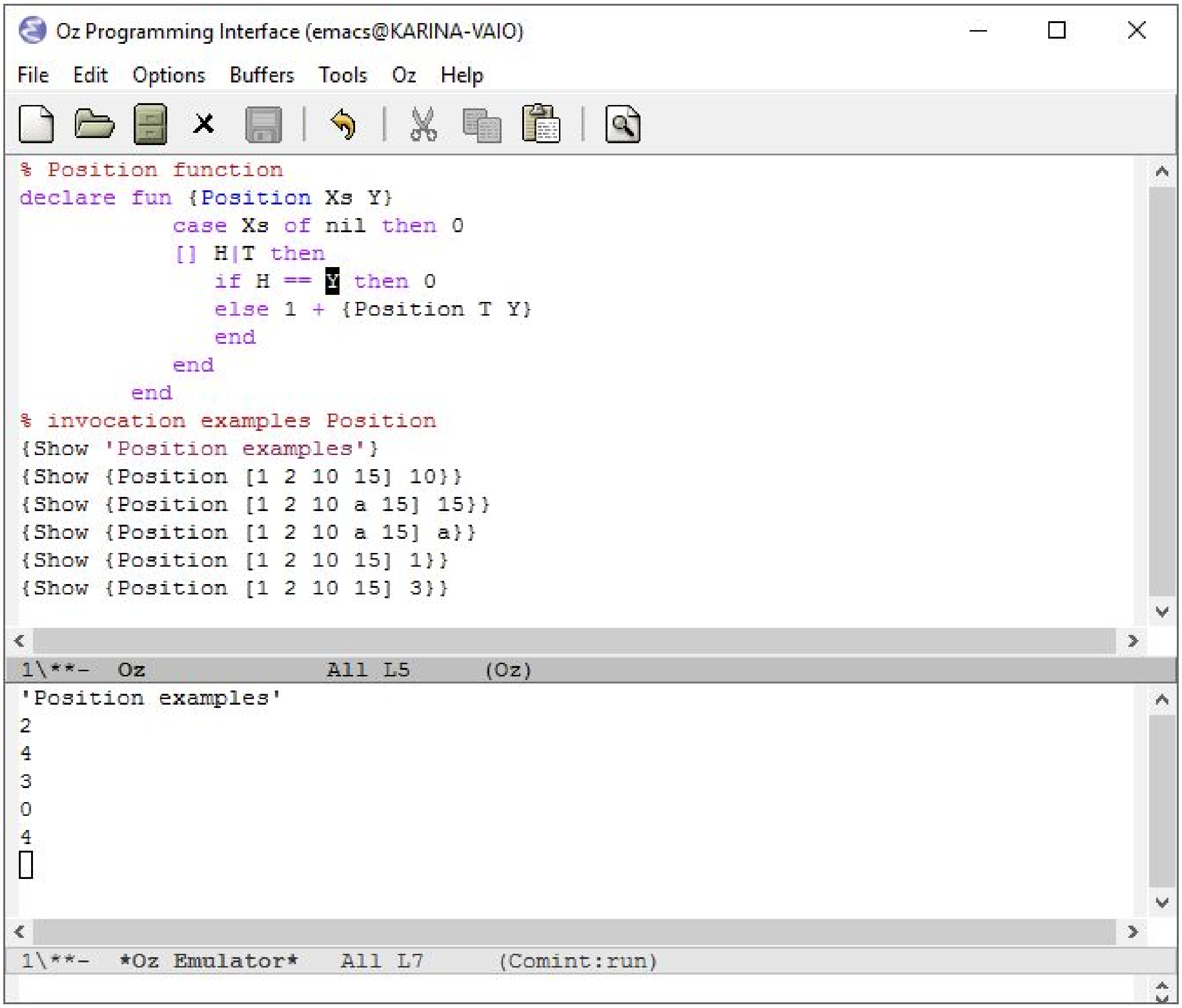
```
'Member examples'
true
true
true
true
true
false
false
false
false
false
```

6. Position

a. Código Oz

```
% 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
```

b. Ejemplos de ejecución


Oz Programming Interface (emacs@KARINA-VAIO)
File
Edit
Options
Buffers
Tools
Oz
Help
1**- Oz   All L5   (Oz)
1**- *Oz Emulator*   All L7   (Comint:run)

```
% 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}}
```

```
'Position examples'
2
4
3
0
4
```

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	E
	b1	**B-> b1**

```
if B then
          skip
else
          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**	**E**

local B in
        B = false
        if B then
                skip
        else
                skip
        end
end
	-	**-**

=> Ejecución de declaración de variable y composición

Stack	**Store**	**E**

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 = **false**	B-> b1

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

Stack	**Store**	**E**
skip	b1 = false	B-> b1

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

Stack	**Store**	**E**
-	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**
	x1 **z1** **a1** **b1** **p1**	**X -> x1** **Z-> z1** **A -> a1** **B -> b1** **P-> p1**

```
proc {P X Y}
          Y = X+Z
end
Z=7
X=4
{P X A}
{P A B}
```

=> Ejecución de procedure value

Stack	**Store**	**E**

Z=7
X=4
{P X A}
{P A B}
	x1
	z1
	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**

X=4
{P X A}
{P A B}
	x1
	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 binding de variable**

Stack	**Store**	**E**

{P X A}
{P A B}
	x1 = **4**
	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**

A = X + Z
{P A B}
	x1 = 4
	z1 = 7
	a1
	b1
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		X -> x1
		Z-> z1
		A -> a1
		B -> b1
		P-> p1

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

Stack	**Store**	**E**

{P A B}	x1 = 4
	z1 = 7
	a1 = **11**
	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}
	Y = X+Z
	end, CE = {Z -> z1}
		X -> x1
		Z-> z1
		A -> a1
		B -> b1
		P-> p1

=> Asignacion de variable mas suma en base al store

Stack	**Store**	**E**

-	x1 = 4
	z1 = 7
	a1 = 11
	b1 **= 18**
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		X -> x1
		Z-> z1
		A -> a1
		B -> b1
		P-> p1

Programa 4

Estado inicial

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

Stack:

```
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}
Y = X+Z
end
Z=10
local Z in
Z = 2
X=4
{P X A}
{P A B}
end
	x1
	z1
	a1
	b1
	p1
		X - > x1
		Z -> z1
		A -> a1
		B -> b1
		P -> p1

=> Ejecución de declaración de procedure value

Stack	Store	E

Z=10
local Z in
Z = 2
X=4
{P X A}
{P A B}
end
	x1
	z1
	a1
	b1
	p1 = **proc {$ X Y}**
	Y = X+Z
	end, CE = {Z -> z1}
		X - > x1
		Z -> z1
		A -> a1
		B -> b1
		P -> p1

=> Ejecución de asignación de variable

Stack	Store	E

local Z in
Z = 2
X=4
{P X A}
{P A B}
end
	x1
	z1 = **10**
	a1
	b1
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		X - > x1
		Z -> z1
		A -> a1
		B -> b1
		P -> p1

=> Ejecución de declaración de variable Z

Stack	Store	E

Z = 2
X=4
{P X A}
{P A B}
	x1
	z1 = 10
	z2
	a1
	b1
	p1 = proc {$ X Y}
		X - > x1
		Z -> **z2**
		A -> a1
		B -> b1
		P -> p1

	Y = X+Z
	end, CE = {Z -> z1}	

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

Stack	**Store**	**E**

X=4
{P X A}
{P A B}
	x1
	z1 = 10
	z2 = **2**
	a1
	b1
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		X - > x1
		Z -> z2
		A -> a1
		B -> b1
		P -> p1

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

Stack	**Store**	**E**

{P X A}
{P A B}
	x1 = **4**
	z1 = 10
	z2 = 2
	a1
	b1
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		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}
	Y = X+Z
	end, CE = {Z -> z1}
		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
	z1 = 10
	z2 = 2
	a1 = **14 (Z -> z1 = 10)**
	b1
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		X - > x1
		Z -> z2
		A -> a1
		B -> b1
		P -> p1

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

Stack	**Store**	**E**

B = A + Z	x1 = 4
	z1 = 10
	z2 = 2
	a1 = 14
	b1
	p1 = proc {$ X Y}
	Y = X+Z
	end, CE = {Z -> z1}
		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**

-	x1 = 4
	z1 = 10
	z2 = 2
	a1 = 14
	b1 = **24 (a1:14 +z1:10)**
	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

```
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
```

Store	**E**
-	**-**

=> **Declaracion de variables**

Stack

```
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}
```

Store	**E**
x1 **y1** **z1** **p1** **q1**	**X -> x1** **Y -> y1** **Z -> z1** **P -> p1** **Q -> q1**

=> **Asignación de variable**

Stack

```
Y=4
proc {P A B}
        proc {B U V}
                local F in
```

Store	E
x1 = **6** y1 z1 p1	X -> x1 Y -> y1 Z -> z1 P -> p1

```
            F=A+1
            V=U+F
        end
    end
end
{P X Q}
{Q Y Z}
```

q1	Q -> q1

=> **Asignación de variable**

Stack

```
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}
```

Store	E
x1 = 6 y1 = **4** z1 p1 q1	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

=> **Declaración procedure value**

Stack

```
{P X Q}
{Q Y Z}
```

Store

```
x1 = 6
y1 = 4
z1
p1 = proc {$ A B}
    proc {B U V}
        local F in
            F=A+1
            V=U+F
        end
    end
end, CE = {}
q1
```

E
X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

=> **Ejecución procedure value**

Stack

```
proc {Q U V}
        local F in
                F = X + 1
                V = U + F
        end
end
{Q Y Z}
```

Store

```
x1 = 6
y1 = 4
z1
p1 = proc {$ A B}
        proc {B U V}
                local F in
                        F=A+1
                        V=U+F
                end
        end
end, CE = {}
q1
```

E

```
X -> x1
Y -> y1
Z -> z1
P -> p1
Q -> q1
```

=> **Declaración procedure value**

Stack

```
{Q Y Z}
```

Store

```
x1 = 6
y1 = 4
z1
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}
```

E

```
X -> x1
Y -> y1
Z -> z1
P -> p1
Q -> q1
```

=> **Ejecución procedure value**

Stack

```
local F in
        F = X + 1
        Z = Y + F
end
```

Store

```
x1 = 6
y1 = 4
z1
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}
```

E

```
X -> x1
Y -> y1
Z -> z1
P -> p1
Q -> q1
```

=> **declaración de variable**

Stack

```
F = X + 1
Z = Y + F
```

Store

```
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
end, CE = {}
q1 = proc {$ U V}
        local F in
                F = X + 1
                V = U + F
        end
End, CE = {X-> x1, F -> f1}
```

E

```
X -> x1
Y -> y1
Z -> z1
P -> p1
Q -> q1
```

=> **Asignación y suma de variables**

Stack	Store	E

Z = Y + F	x1 = 6
	y1 = 4
	z1
	f1 = **7**
	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	E

-	x1 = 6
	y1 = 4
	z1 = **7(f1) + 4(y1) = 11**
	f1 = 7
	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

Programa 6

Estado inicial

Stack	**Store**	**E**

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

=> **Declaración de variables**

Stack	**Store**	**E**

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

=> **Asignación de variables**

Stack	**Store**	**E**

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

=> **Ejecución de try**

Stack	**Store**	**E**

X = 1
Y = 2
	x1
	y1
		X -> y1
		Y -> y1

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

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

Stack	**Store**	**E**

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

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

Stack	**Store**	**E**

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**	**E**
{Browse X#Y#Z}	x1 y1 = 1 z1	X -> y1 Y -> y1 Z -> z1

=> **Ejecución Browse**

Stack	**Store**	**E**
	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 'l'(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 'l'(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 'l'(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”

```
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
```

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**
(ver código abajo)	-	-

```
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		

=> **Declaración de variable**

Stack

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

Store

length

E

length -> Length

=> **Asignación de procedure value**

Stack

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

Store

```
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, CE = {})
```

E

length -> Length

=> **Declaración variable K**

Stack

```
{Length [1 2 3 4] K}
{Show K}
```

Store

```
length = (proc {$ Xs N}
        case Xs of nil then
           N = 0
```

E

k-> K

```
        else
           case Xs of _|T then
              local U in
                        {Length T U}
                        N = U + 1
              end
           else
              skip
           end
        end
end, CE = {})
k
```

=> **Ejecución de procedure**

Stack

```
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
{Show K}
```

Store

```
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, *)
k
```

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

E

k-> K

=> **Ejecución de case**

Stack

```
case Xs of _|T then
      local U in
            {Length T U}
            N = U + 1
      end
else
      skip
end
{Show K}
```

Store

```
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, *)
```

E

k-> K

<table>
<tr><td></td><td>k
<table><tr><td>Store*</td><td>CE*</td></tr><tr><td>xs = [1 2 3 4]
n =</td><td>xs -> Xs
n -> K</td></tr></table></td><td></td></tr>
</table>

=> **Ejecución de case**

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td><pre>
local U in
      {Length T U}
      N = U + 1
end
{Show K}
</pre></td><td><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
              skip
           end
        end
end, *)
k
</pre><table><tr><td>Store*</td><td>CE*</td></tr><tr><td>xs = [1 2 3 4]
n =
t = [2 3 4]</td><td>xs -> Xs
n -> K
t -> T</td></tr></table></td><td>k-> K</td></tr>
</table>

=> **Declaración de U**

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td><pre>
{Length T U}
N = U + 1
{Show K}
</pre></td><td><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
              skip
           end
        end
end, *)
k
</pre></td><td>k-> K</td></tr>
</table>

Store*	CE*

xs = [1 2 3 4]
n =
t = [2 3 4]
u =
	n -> K
	t -> T
	u->U

=> **Ejecución de Length**

Stack

```
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
N = U + 1
{Show K}
```

Store

```
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, *)
k
```

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

E

k-> K

=> **Ejecución de case**

Stack

```
case Xs of _|T then
      local U in
            {Length T U}
            N = U + 1
      end
else
      skip
end
```

Store

```
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
```

E

k-> K

```
N = U + 1
{Show K}
```

```
            end
         end
end, *)
```

k

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

```
local U in
      {Length T U}
      N = U + 1
end
N = U + 1
{Show K}
```

Store

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, *)
```

k

Store*	CE**
xs = [1 2 3 4] n = t = [2 3 4] u = xs’=[2 3 4] n’= **t’=[3 4]**	xs’ -> Xs n’ -> N **t’ -> T** u->U

E

k-> K

=> **Declaración de U**

Stack

```
{Length T U}
N = U + 1
```

Store

length = (proc {$ Xs N}

```
        case Xs of nil then
           N = 0
```

E

k-> K

```
N = U + 1
{Show K}
```

```
         else
            case Xs of _|T then
               local U in
                         {Length T U}
                         N = U + 1
               end
            else
               skip
            end
         end
end, *)
```

k

Store*	CE**
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
see below	see below	k-> K

Stack

```
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
N = U + 1
N = U + 1
{Show K}
```

Store

```
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, *)
```

k

Store*	CE***
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]	

E

k-> K

u'= **xs'' = [3 4]** **n'' =**	

=> Ejecución de case

Stack

```
case Xs of _|T then
      local U in
            {Length T U}
            N = U + 1
      end
else
      skip
end
N = U + 1
N = U + 1
{Show K}
```

Store

```
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, *)
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

E

k-> K

=> Ejecución de case

Stack

```
local U in
      {Length T U}
      N = U + 1
end
N = U + 1
N = U + 1
{Show K}
```

Store

```
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
```

E

k-> K

<table>
<tr>
<td></td>
<td>

```
            skip
         end
      end
end, *)
```

k

<table>
<tr><th>Store*</th><th>CE***</th></tr>
<tr>
<td>xs = [1 2 3 4]
n =
t = [2 3 4]
u =
xs’=[2 3 4]
n’=
t’=[3 4]
u’=
xs’’ = [3 4]
n’’ =
<b>t’’ = [4]</b></td>
<td>xs’’ -> Xs
n’’ -> N
<b>t’’ -> T</b>
u’->U</td>
</tr>
</table>

</td>
<td></td>
</tr>
</table>

=> **Declaración de variable U**

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr>
<td>

`{Length T U}`
`N = U + 1`
`N = U + 1`
`N = U + 1`
`{Show K}`

</td>
<td>

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, *)
```

k

<table>
<tr><th>Store*</th><th>CE***</th></tr>
<tr>
<td>xs = [1 2 3 4]
n =
t = [2 3 4]
u =
xs’=[2 3 4]
n’=
t’=[3 4]
u’=
xs’’ = [3 4]</td>
<td>xs’’ -> Xs
n’’ -> N
t’’ -> T
<b>u’’->U</b></td>
</tr>
</table>

</td>
<td>k-> K</td>
</tr>
</table>

	(see nested table below)	

n’’ = t’’ = [4] **u’’ =**	

=> **Ejecución de Length**

Stack

```
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
N = U + 1
N = U + 1
N = U + 1
{Show K}
```

Store

```
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, *)
k
```

Store*	CE****
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’=	
xs’’ = [3 4]	
n’’ =	
t’’ = [4]	
u’’ =	
xs’’’= [4]	
n’’’=	

E

k-> K

=> **Ejecución de case**

Stack

```
case Xs of _|T then
      local U in
            {Length T U}
```

Store

```
length = (proc {$ Xs N}
        case Xs of nil then
           N = 0
        else
```

E

k-> K

```
            N = U + 1
      end
else
      skip
end
N = U + 1
N = U + 1
N = U + 1
{Show K}
```

```
           case Xs of _|T then
              local U in
                        {Length T U}
                        N = U + 1
              end
           else
              skip
           end
        end
end, *)
```

k

Store*	CE****
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'=	
xs'' = [3 4]	
n'' =	
t'' = [4]	
u'' =	
xs'''= [4]	
n'''=	

=> Ejecución de case

Stack

```
local U in
      {Length T U}
      N = U + 1
end
N = U + 1
N = U + 1
N = U + 1
{Show K}
```

Store

```
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, *)
```

k

Store*	CE****
xs = [1 2 3 4]	xs'''-> Xs

E

k-> K

<table>
<tr>
<td></td>
<td>
<table>
<tr>
<td>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</td>
<td>n’’’ -> N
t’’’ -> T
u’’->U</td>
</tr>
</table>
</td>
<td></td>
</tr>
</table>

=> **Declaración variable U**

<table>
<tr>
<th>Stack</th>
<th>Store</th>
<th>E</th>
</tr>
<tr>
<td><pre>{Length T U}
N = U + 1
N = U + 1
N = U + 1
N = U + 1
{Show K}</pre></td>
<td><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
              skip
           end
        end
end, *)
k</pre>
<table>
<tr>
<th>Store*</th>
<th>CE****</th>
</tr>
<tr>
<td>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]</td>
<td>xs’’’-> Xs
n’’’ -> N
t’’’ -> T
u’’’->U</td>
</tr>
</table>
</td>
<td>k-> K</td>
</tr>
</table>

<table>
<tr><td></td><td><table><tr><td>u’’ =
xs’’’= [4]
n’’’=
t’’’ = nil
u’’’ =</td><td></td></tr></table></td><td></td></tr>
</table>

=> **Ejecución Length**

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td><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
N = U + 1
N = U + 1
N = U + 1
N = U + 1
{Show K}
</pre></td><td>length = (<pre>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, *)</pre>k<table><tr><th>Store*</th><th>CE****</th></tr><tr><td>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’’’ =
<b>xs’’’’= nil</b>
<b>n’’’’=</b></td><td><b>xs’’’’-> Xs</b>
<b>n’’’’ -> N</b>
t’’’ -> T
u’’’->U</td></tr></table></td><td>k-> K</td></tr>
</table>

=> **Ejecución case**

Stack	Store	E

N = 0
N = U + 1
N = U + 1
N = U + 1
N = U + 1
{Show K}
	length = (proc {$ Xs N}
	k
		k-> K

```
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, *)
k
```

Store*	CE****
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'=	
xs'' = [3 4]	
n'' =	
t'' = [4]	
u'' =	
xs'''= [4]	
n'''=	
t''' = nil	
u''' =	
xs''''= nil	
n''''=	

=> Ejecución asignación valor

Stack	Store	E

N = U + 1
N = U + 1
N = U + 1
N = U + 1
{Show K}
	length = (proc {$ Xs N}	k-> K

```
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, *)
```

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””= 0**	xs””-> Xs n”” -> N t’” -> T u’”->U

=> **Ejecución asignación valor**

Stack	**Store**	**E**

`N = U + 1`
`N = U + 1`
`N = U + 1`
`{Show K}`
	length = (`proc {$ Xs N}`	k-> K

```
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, *)
```

k

Store*	CE***
xs = [1 2 3 4]	n’” -> N

<table>
<tr>
<td></td>
<td>
<table>
<tr>
<td>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”’= 1
t”’ = nil
u”’ = 0</td>
<td>u”’->U</td>
</tr>
</table>
</td>
<td></td>
</tr>
</table>

=> **Ejecución asignación valor**

<table>
<tr>
<th>Stack</th>
<th>Store</th>
<th>E</th>
</tr>
<tr>
<td><code>N = U + 1</code>
<code>N = U + 1</code>
<code>{Show K}</code></td>
<td>
<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
              skip
           end
        end
end, *)
</pre>
k
<table>
<tr>
<th>Store*</th>
<th>CE**</th>
</tr>
<tr>
<td>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</td>
<td>n” -> N
u”->U</td>
</tr>
</table>
</td>
<td>k-> K</td>
</tr>
</table>

	t'' = [4] u'' = 1	

=> Ejecución asignación valor

Stack	**Store**	**E**

`N = U + 1`
`{Show K}`
		k-> K

```
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, *)
```

k

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

=> Ejecución asignación valor

Stack	**Store**	**E**
`{Show K}`		k-> K

```
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, *)
```

	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
		k-> K

```
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, *)
k = 4
```

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**
	length =	**Length ->length**

```
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

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

=> **Asignación procedure value**

Stack	**Store**	**E**
		Length ->length

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

```
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, CE={}}
```

=> Declaración variable K

Stack

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

Store

```
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, CE={}}
```

K=

E

Length ->length
K -> k

=> Ejecución de Length

Stack

```
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}
```

Store

```
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, *}
K=
```

Store*	CE

xs=[1 2 3 4]
a =0
n=
k =
	Xs -> xs
	A -> a
	N->n
	K->k

E

Length ->length
K -> k

=> Ejecución de case

Stack

```
case Xs of _|T then
        local X in
                X = A + 1
                {Length T X N}
```

Store

```
length ={proc {$ Xs A N}
 case Xs of nil then
        N = A
 else
```

E

Length ->length
K -> k

```
     end
else
        skip
end
{Show K}
```

```
         case Xs of _|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]
a =0
n=
k =
	Xs -> xs
	A -> a
	N->n
	K->k

=> **Ejecución de case**

Stack

```
local X in
        X = A + 1
        {Length T X N}
end
{Show K}
```

Store

```
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, *}
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**

E

Length ->length
K -> k

=> **Declaración de variable local X**

Stack	**Store**	**E**

`X = A + 1`
`{Length T X N}`
`{Show K}`
		Length ->length
		K -> k

```
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, *}
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=	**X->x**

=> Asignación de valor

Stack	**Store**	**E**

`{Length T X N}`
`{Show K}`
		Length ->length
		K -> k

```
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, *}
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	

=> Ejecución de Length

Stack	Store	E

Stack

```
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}
```

Store

```
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, *}
```

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'=	

E

Length ->length
K -> k

=> Ejecución de case

Stack	Store	E

Stack

```
case Xs of _|T then
        local X in
                X = A + 1
                {Length T X N}
     end
else
        skip
end
Show K}
```

Store

```
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, *}
```

E

Length ->length
K -> k

<table>
<tr>
<td></td>
<td>K=
<table>
<tr><th>Store*</th><th>CE*</th></tr>
<tr><td>xs=[1 2 3 4]
a =0
n=
k =
t=[2 3 4]
x=1
<b>xs’=[2 3 4]</b>
<b>a’=1</b>
<b>n’=</b></td><td><b>Xs -> xs’</b>
<b>A -> a’</b>
<b>N->n’</b>
K->k
T->t
X->x</td></tr>
</table>
</td>
<td></td>
</tr>
</table>

=> **Ejecución de case**

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr>
<td><pre>
local X in
        X = A + 1
        {Length T X N}
end
{Show K}
</pre></td>
<td>length =<pre>
{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, *}
</pre>K=
<table>
<tr><th>Store*</th><th>CE*</th></tr>
<tr><td>xs=[1 2 3 4]
a =0
n=
k =
t=[2 3 4]
x=1
xs’=[2 3 4]
a’=1
n’=
<b>t’=[3 4]</b></td><td>Xs -> xs’
A -> a’
N->n’
K->k
<b>T->t’</b>
X->x</td></tr>
</table>
</td>
<td>Length ->length
K -> k</td>
</tr>
</table>

=> Declaración de X

Stack

```
X = A + 1
{Length T X N}
{Show K}
```

Store

```
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, *}
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’=	

E

Length ->length
K -> k

=> Asignación de valor

Stack

```
{Length T X N}
{Show K}
```

Store

```
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, *}
K=
```

E

Length ->length
K -> k

<table>
<tr>
<td></td>
<td>
<table>
<tr><th>Store*</th><th>CE*</th></tr>
<tr>
<td>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]
<b>x’= 2</b></td>
<td>Xs -> xs’
A -> a’
N->n’
K->k
T->t’
X->x’</td>
</tr>
</table>
</td>
<td></td>
</tr>
</table>

=> Ejecución de Length

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr>
<td>
<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>
</td>
<td>
length ={proc {$ Xs A N}
<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, *}
</pre>
K=
<table>
<tr><th>Store*</th><th>CE**</th></tr>
<tr>
<td>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</td>
<td><b>Xs -> xs”</b>
<b>A -> a”</b>
<b>N->n”</b>
K->k
T->t’
X->x’</td>
</tr>
</table>
</td>
<td>Length ->length
K -> k</td>
</tr>
</table>

xs”=[3 4] **a”=2** **n”=**	

=> **Ejecución de case**

Stack	Store	E

Stack

```
case Xs of _|T then
        local X in
                X = A + 1
                {Length T X N}
     end
else
        skip
end
{Show K}
```

Store

```
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, *}
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	
xs”=[3 4]	
a”=2	
n”=	

E

Length ->length
K -> k

=> **Ejecución de case**

Stack	Store	E

```
local X in
        X = A + 1
        {Length T X N}
end
{Show K}
```

```
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, *}
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	
xs’’=[3 4]	
a’’=2	
n’’=	
t’’=[4]	

Length ->length
K -> k

=> **Declaración de variable X**

Stack

```
X = A + 1
{Length T X N}
{Show K}
```

Store

```
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, *}
K=
```

E

Length ->length
K -> k

<table>
<tr><td></td><td>
<table>
<tr><th>Store*</th><th>CE**</th></tr>
<tr><td>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’’=
t’’=[4]
x’’=</td><td>Xs -> xs’’
A -> a’’
N->n’’
K->k
T->t’’
X->x’’</td></tr>
</table>
</td><td></td></tr>
</table>

=> **Asignación de valor**

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td><pre>{Length T X N}
{Show K}</pre></td><td>
<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}
             end
        else
                skip
        end
 end, *}
K=</pre>
<table>
<tr><th>Store*</th><th>CE**</th></tr>
<tr><td>xs=[1 2 3 4]
a =0
n=
k =
t=[2 3 4]
x=1</td><td>Xs -> xs’’
A -> a’’
N->n’’
K->k
T->t’’
X->x’’</td></tr>
</table>
</td><td>Length ->length
K -> k</td></tr>
</table>

	xs’=[2 3 4] a’=1 n’= t’=[3 4] x’= 2 xs’’=[3 4] a’’=2 n’’= t’’=[4] **x’’= 3**		

=> **Ejecución de Length**

Stack

```
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}
```

Store

```
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, *}
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] x’= 2 xs’’=[3 4] a’’=2 n’’= t’’=[4]	**Xs -> xs’’’** **A -> a’’’** **N->n’’’** K->k T->t’’ X->x’’

E

Length ->length
K -> k

x’’= 3 **xs’’’=[4]** **a’’’=3** **n’’’=**		

=> **Ejecución de Case**

Stack

```
case Xs of _|T then
        local X in
                X = A + 1
                {Length T X N}
     end
else
        skip
end
{Show K}
```

Store

```
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, *}
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] x’= 2 xs’’=[3 4] a’’=2 n’’= t’’=[4] x’’= 3 xs’’’=[4] a’’’=3 n’’’=	Xs -> xs’’’ A -> a’’’ N->n’’’ K->k T->t’’ X->x’’

E

Length ->length
K -> k

=> **Ejecución de Case**

Stack

```
local X in
        X = A + 1
        {Length T X N}
end
{Show K}
```

Store

```
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, *}
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	
xs’’=[3 4]	
a’’=2	
n’’=	
t’’=[4]	
x’’= 3	
xs’’’=[4]	
a’’’=3	
n’’’=	
t’’’=nil	

E

Length ->length
K -> k

=> **Declaración de variable X**

Stack

```
X = A + 1
{Length T X N}
{Show K}
```

Store

```
length ={proc {$ Xs A N}
 case Xs of nil then
        N = A
```

E

Length ->length
K -> k

```
 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=[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	
xs''=[3 4]	
a''=2	
n''=	
t''=[4]	
x''= 3	
xs'''=[4]	
a'''=3	
n'''=	
t'''=nil	
x'''=	

=> **Asignación de valor**

Stack	Store	E

`{Length T X N}`
`{Show K}`
	length =`{proc {$ Xs A N}`	Length ->length
		K -> k

```
 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, *}
```
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	
xs''=[3 4]	
a''=2	
n''=	
t''=[4]	
x''= 3	
xs'''=[4]	
a'''=3	
n'''=	
t'''=nil	
x'''=4	

=> **Ejecución de Length**

Stack

```
 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}
```

Store

```
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, *}
K=
```

E

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] x’= 2 xs”=[3 4] a”=2 n”= t”=[4] x”= 3 xs”’=[4] a”’=3 n”’= t”’=nil x”’=4 **xs””=nil** **a””=4** **n””=**	**Xs -> xs””** **A -> a””** **N->n””** K->k T->t”’ X->x”’

=> **Ejecución de case**

Stack	Store	E

N = A
{Show K}
	length =	Length ->length
		K -> k

```
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, *}
```

K=

Store*	CE****

<table>
<tr>
<td></td>
<td>
<table>
<tr>
<td>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’’=
t’’=[4]
x’’= 3
xs’’’=[4]
a’’’=3
n’’’=
t’’’=nil
x’’’=4
xs’’’’=nil
a’’’’=4
n’’’’=</td>
<td>Xs -> xs’’’’
A -> a’’’’
N->n’’’’
K->k
T->t’’’
X->x’’’</td>
</tr>
</table>
</td>
<td></td>
</tr>
</table>

=> **Asignación de valor**

<table>
<tr>
<th>Stack</th>
<th>Store</th>
<th>E</th>
</tr>
<tr>
<td>{Show K}</td>
<td>

```
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, *}
```

K=4

<table>
<tr>
<th>Store*</th>
<th>CE****</th>
</tr>
<tr>
<td>xs=[1 2 3 4]</td>
<td>Xs -> xs’’’’</td>
</tr>
</table>
</td>
<td>Length ->length
K -> k</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>
<table>
<tr>
<td>a =0
n=
<b>k =4</b>
t=[2 3 4]
x=1
xs’=[2 3 4]
a’=1
<b>n’=4</b>
t’=[3 4]
x’= 2
xs’’=[3 4]
a’’=2
<b>n’’=4</b>
t’’=[4]
x’’= 3
xs’’’=[4]
a’’’=3
<b>n’’’=4</b>
t’’’=nil
x’’’=4
xs’’’’=nil
a’’’’=4
<b>n’’’’=4</b></td>
<td>A -> a’’’’
N->n’’’’
K->k
T->t’’’
X->x’’’</td>
</tr>
</table>
</td>
<td></td>
</tr>
</table>

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

Stack	**Store**	**E**
--		Length ->length K -> k

```
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, *}
```

K=4

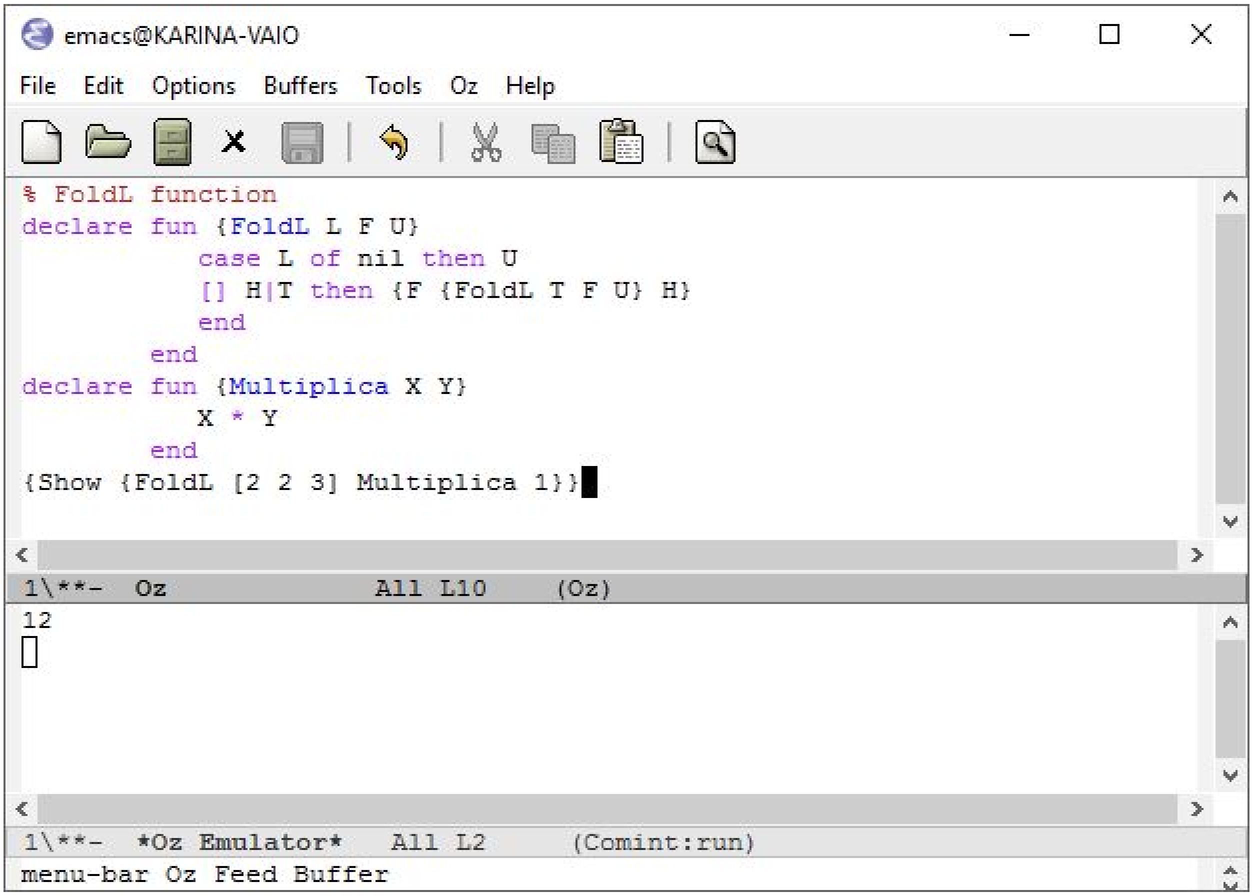
Ejercicio 6) - Alto orden con listas

FoldL

a. Código Oz

```
% 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}}
```

b. Ejemplo de ejecución



```
% 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
```

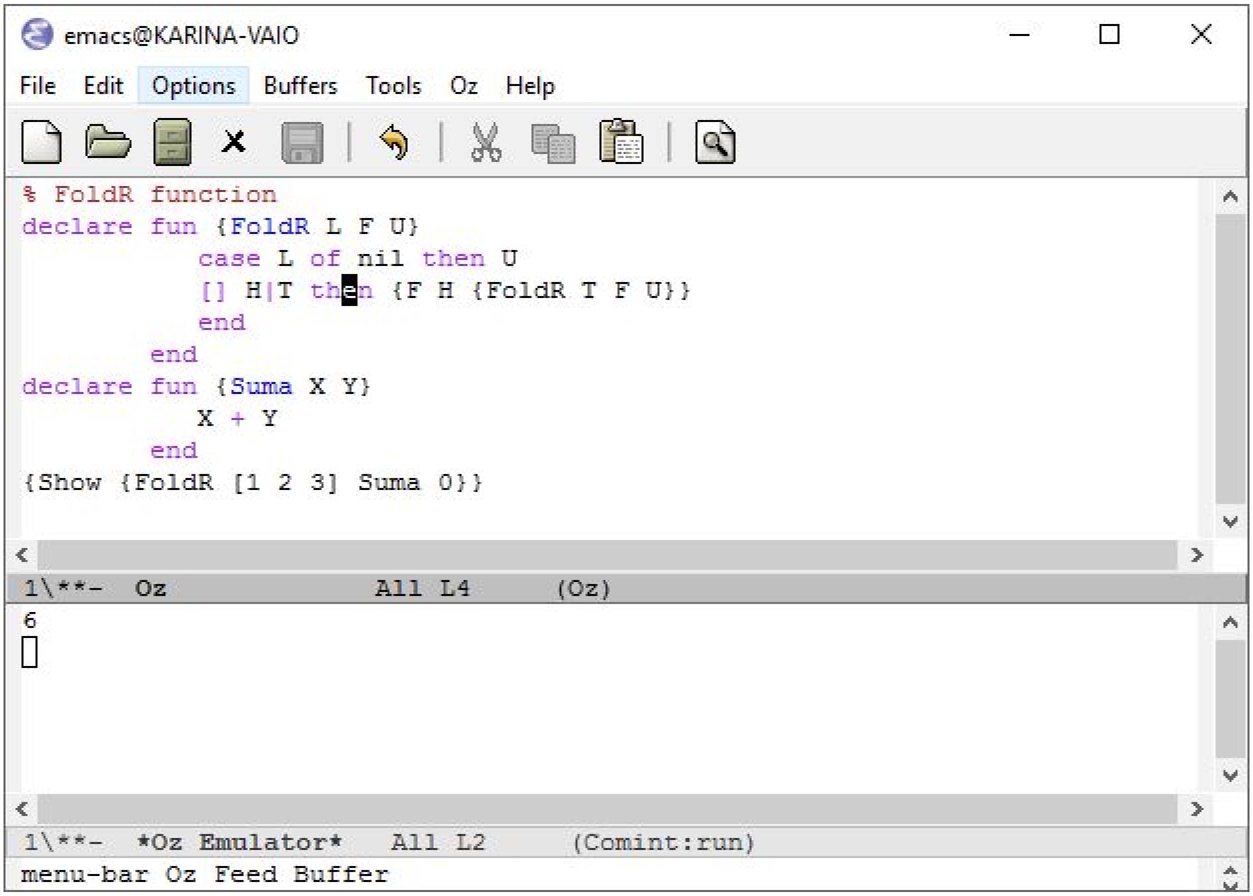
```
1\**-  *Oz Emulator*   All L2      (Comint:run)
menu-bar Oz Feed Buffer
```

FoldR

a. Código Oz

```
% FoldR function
declare fun {FoldR L F U}
         case L of nil then U
         [] H|T then {F H {FoldR T F U}}
         end
      end
declare fun {Suma X Y}
         X + Y
      end
{Show {FoldR [1 2 3] Suma 0}}
```

b. Ejemplo de ejecución


emacs@KARINA-VAIO
File
Edit
Options
Buffers
Tools
Oz
Help
1**- Oz   All L4   (Oz)
1**- *Oz Emulator*   All L2   (Comint:run)

```
% FoldR function
declare fun {FoldR L F U}
           case L of nil then U
           [] H|T then {F H {FoldR T F U}}
           end
        end
declare fun {Suma X Y}
           X + Y
        end
{Show {FoldR [1 2 3] Suma 0}}
```

```
6
```

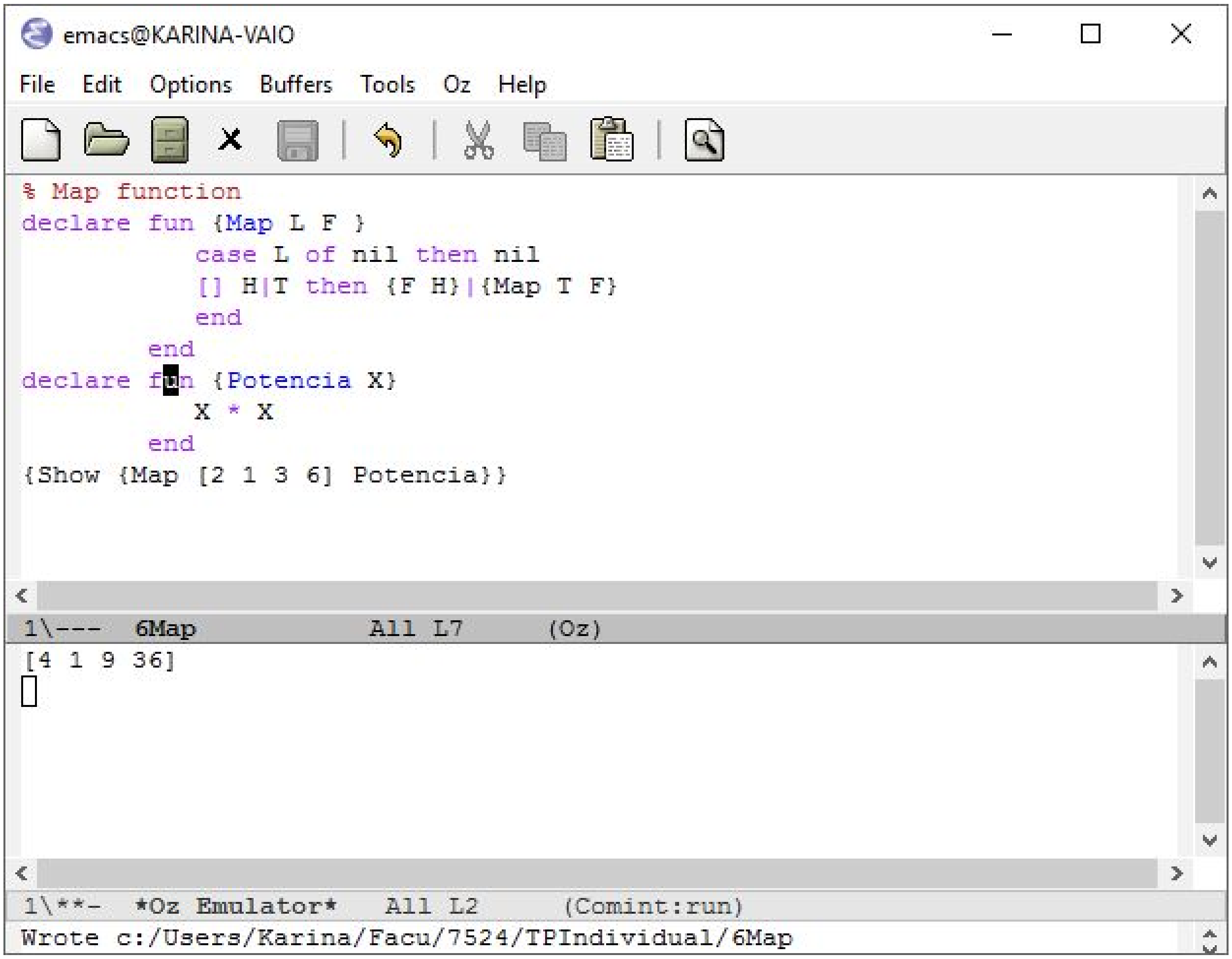
menu-bar Oz Feed Buffer

Map

a. Código Oz

```
% Map function
declare fun {Map L F }
         case L of nil then nil
         [] H|T then {F H}|{Map T F}
         end
      end
declare fun {Potencia X}
         X * X
      end
{Show {Map [2 1 3 6] Potencia}}
```

b. Ejemplo de ejecución



```
% Map function
declare fun {Map L F }
           case L of nil then nil
           [] H|T then {F H}|{Map T F}
           end
        end
declare fun {Potencia X}
           X * X
        end
{Show {Map [2 1 3 6] Potencia}}
```

```
1\---   6Map            All L7      (Oz)
```

```
[4 1 9 36]
```

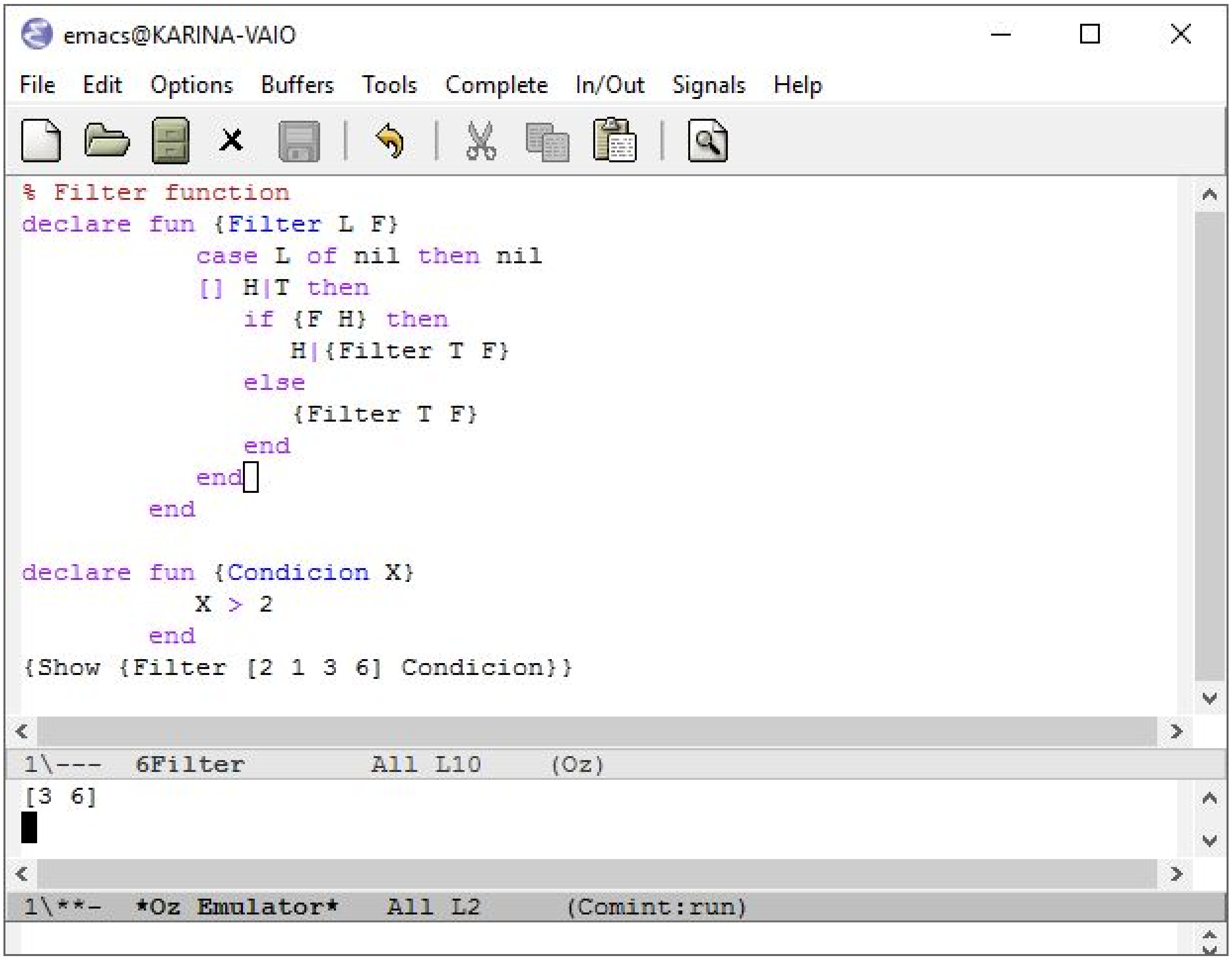
```
1\**-   *Oz Emulator*    All L2      (Comint:run)
Wrote c:/Users/Karina/Facu/7524/TPIndividual/6Map
```

Filter

a. Código Oz

```
% 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
      end
{Show {Filter [2 1 3 6] Condicion}}
```

b. Ejemplo de ejecución


emacs@KARINA-VAIO
File Edit Options Buffers Tools Complete In/Out Signals Help

```
% 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
        end
{Show {Filter [2 1 3 6] Condicion}}
```

```
1\---  6Filter        All L10     (Oz)
```

```
[3 6]
```

```
1\**-  *Oz Emulator*   All L2      (Comint:run)
```

Ejercicio 7) - Hilos

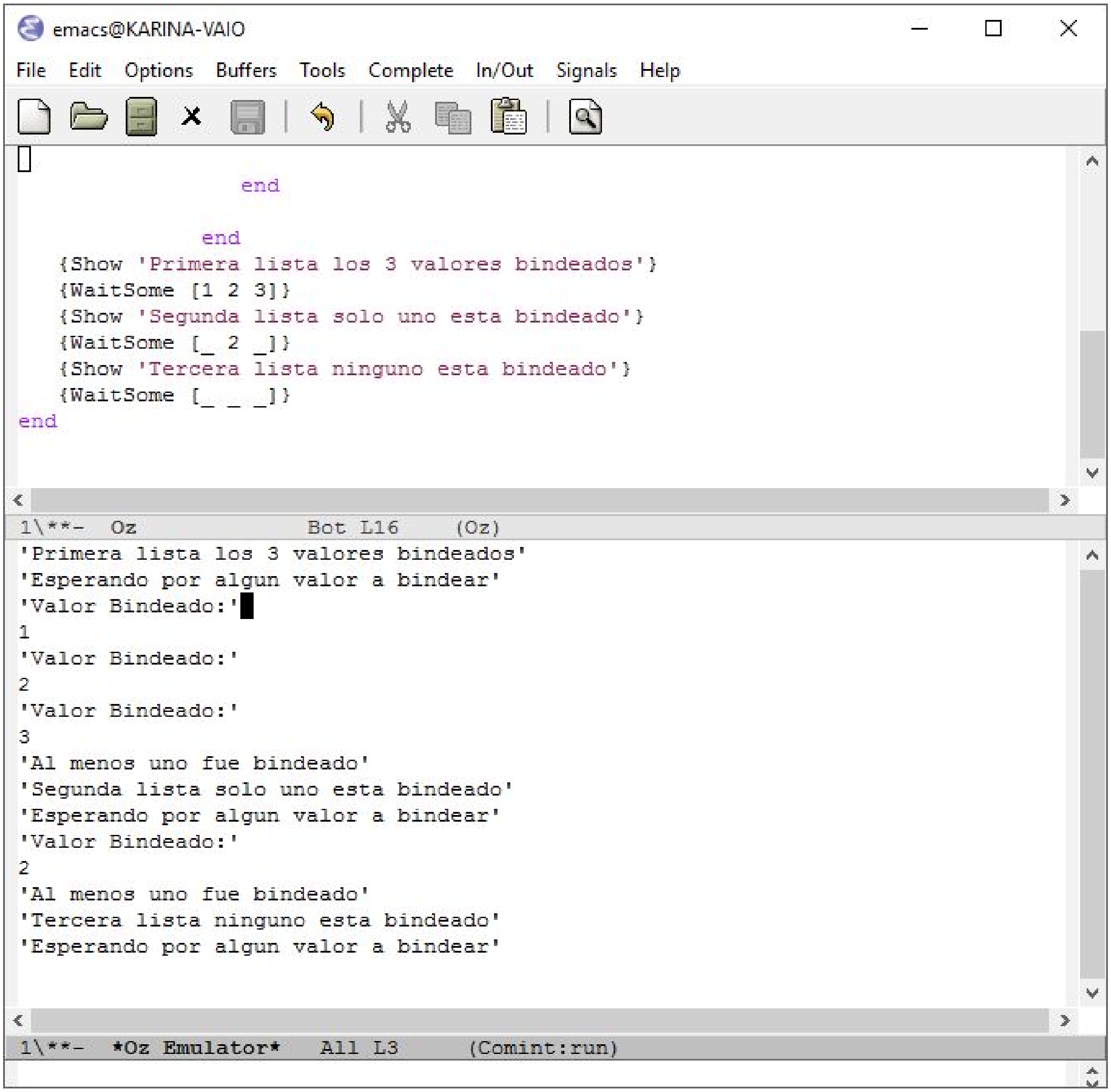
WaitSome

a. Código Oz

```
local WaitSome in
   WaitSome = proc {$ Xs}
                local Y in
                   {ForAll Xs
                    proc {$ X}
                     thread {Wait X}
                        Y = true
                        {Show 'Valor Bindeado:'}
                        {Show X}
                     end
```

```
                 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 [_ _ _]}
end
```

b. Ejemplo de ejecución


emacs@KARINA-VAIO
File Edit Options Buffers Tools Complete In/Out Signals Help

```
                  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 [_ _ _]}
end
```

1**- Oz   Bot L16   (Oz)

```
'Primera lista los 3 valores bindeados'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
1
'Valor Bindeado:'
2
'Valor Bindeado:'
3
'Al menos uno fue bindeado'
'Segunda lista solo uno esta bindeado'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
2
'Al menos uno fue bindeado'
'Tercera lista ninguno esta bindeado'
'Esperando por algun valor a bindear'
```

Maquina abstracta

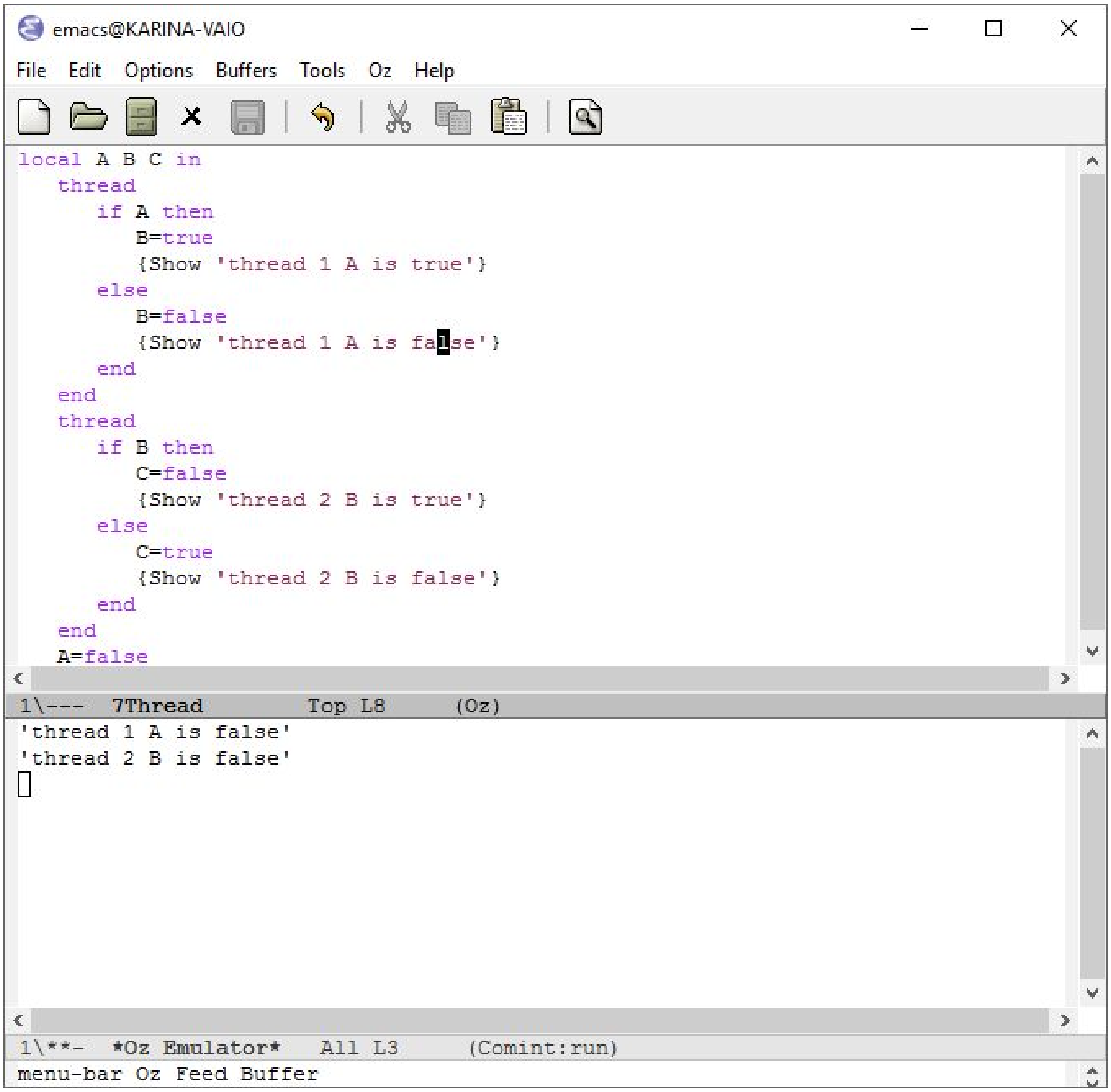
a. Código Oz

```
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.



```
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
```

```
1\---  7Thread        Top L8      (Oz)
```

```
'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

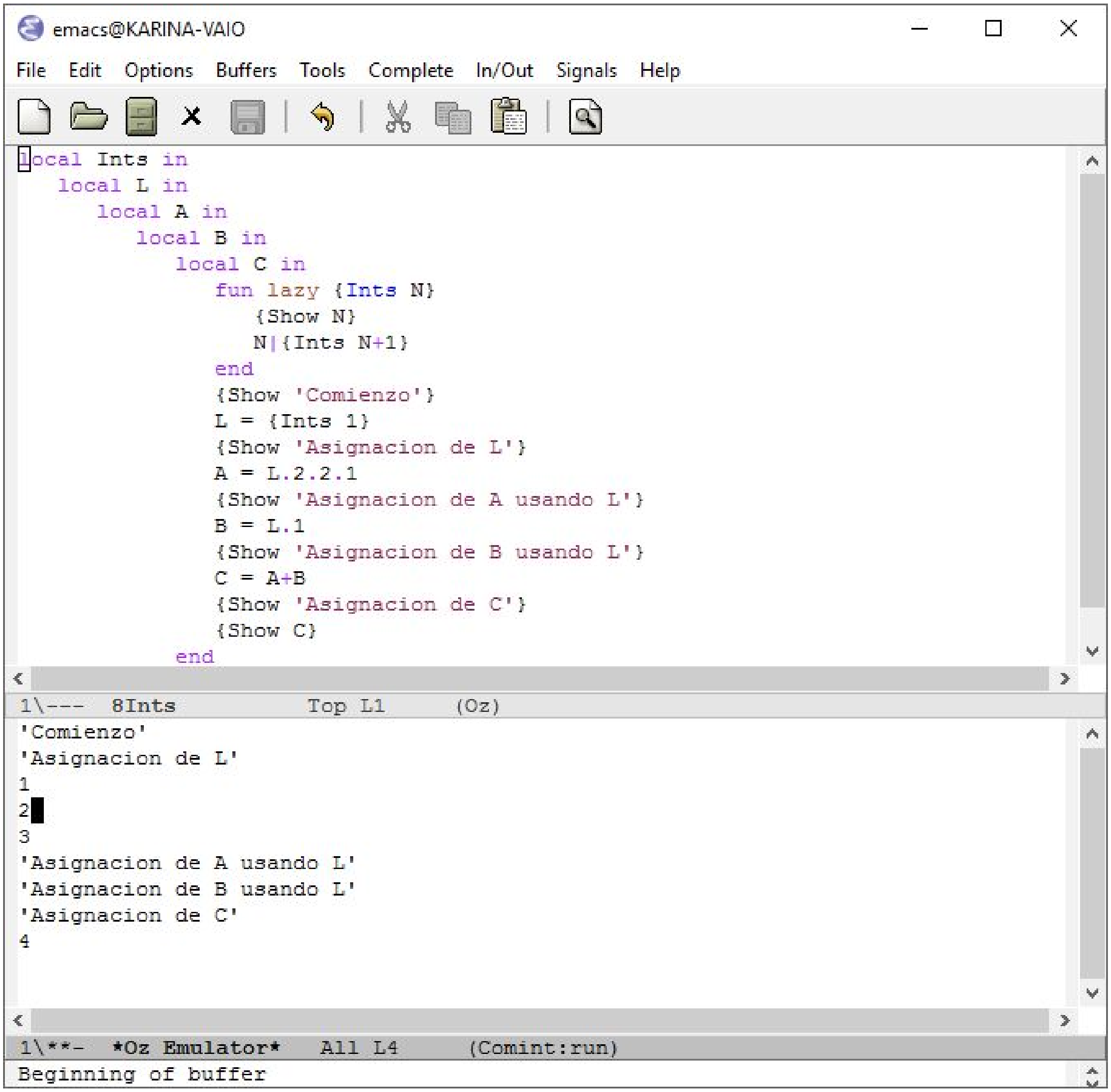
8.1 Maquina Abstracta

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
```

b. Ejemplo de ejecución


emacs@KARINA-VAIO
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'
1
2
3
'Asignacion de A usando L'
'Asignacion de B usando L'
'Asignacion de C'
4
```

1**- *Oz Emulator*   All L4   (Comint:run)
Beginning of buffer

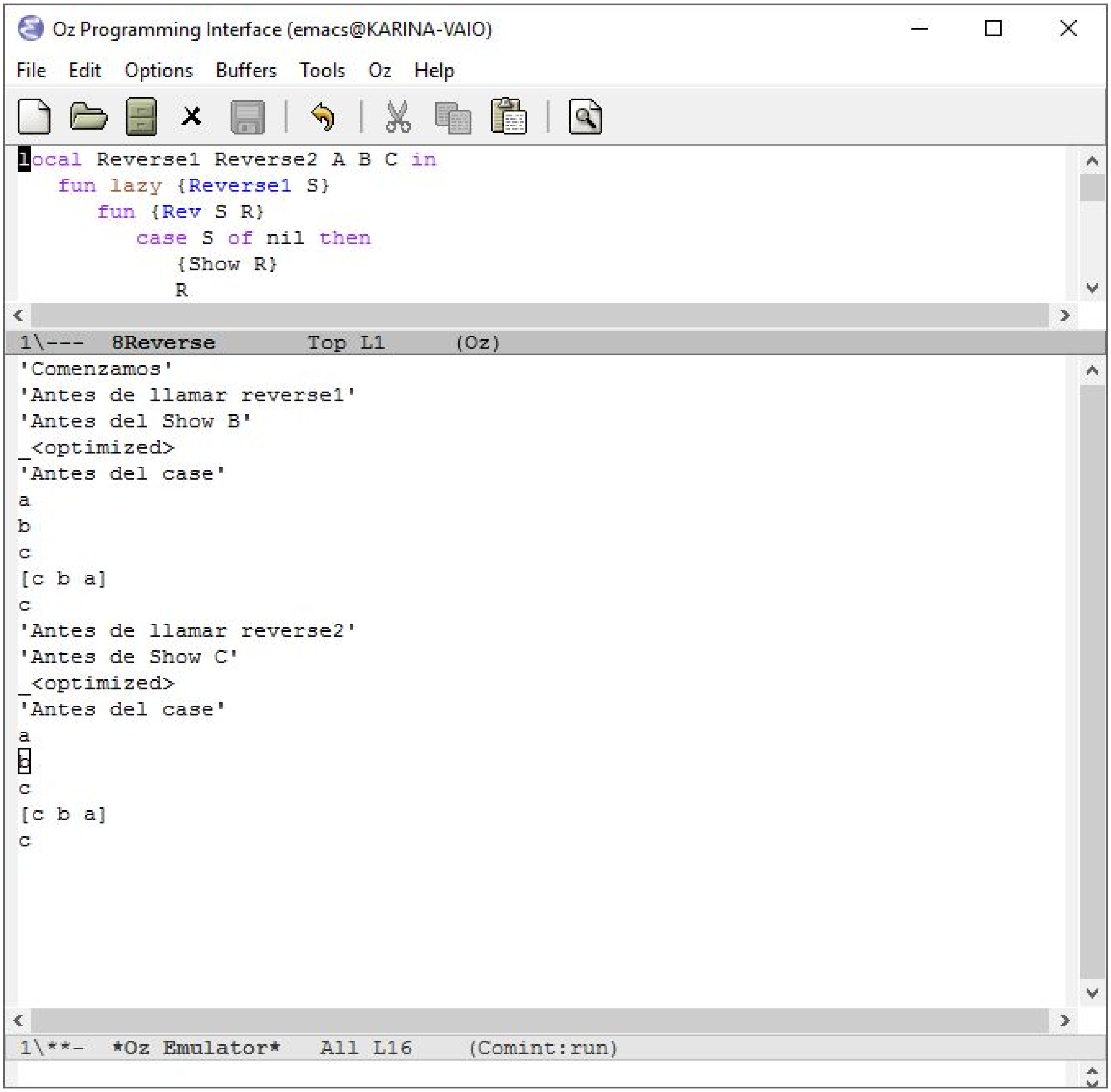
8.2 Reverse

- 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}
          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
```


Oz Programming Interface (emacs@KARINA-VAIO)
File
Edit
Options
Buffers
Tools
Oz
Help
1\--- 8Reverse   Top L1   (Oz)
1**- *Oz Emulator*   All L16   (Comint:run)

```
local Reverse1 Reverse2 A B C in
   fun lazy {Reverse1 S}
      fun {Rev S R}
         case S of nil then
            {Show R}
            R
```

```
'Comenzamos'
'Antes de llamar reverse1'
'Antes del Show B'
_<optimized>
'Antes del case'
a
b
c
[c b a]
c
'Antes de llamar reverse2'
'Antes de Show C'
_<optimized>
'Antes del case'
a
b
c
[c b a]
c
```

Ejercicio 9) - Mensajes

Ejercicio 10) - Servidor de filtros

Ejercicio 11) - Celdas de memoria