7524 - Teoría de la programación

TP Individual

Karina Alaya

Padron 75840

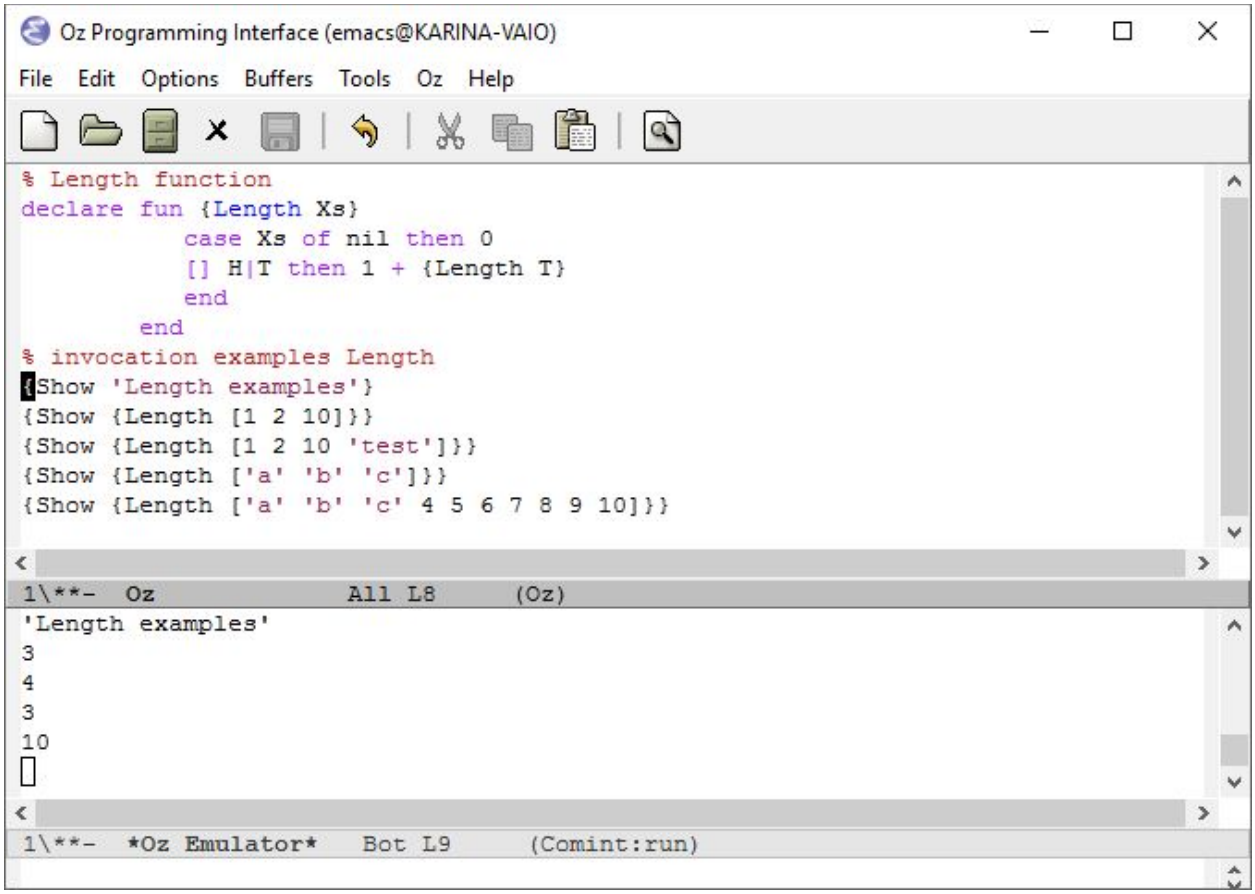
Ejercicio 1) - Procesamiento de listas

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

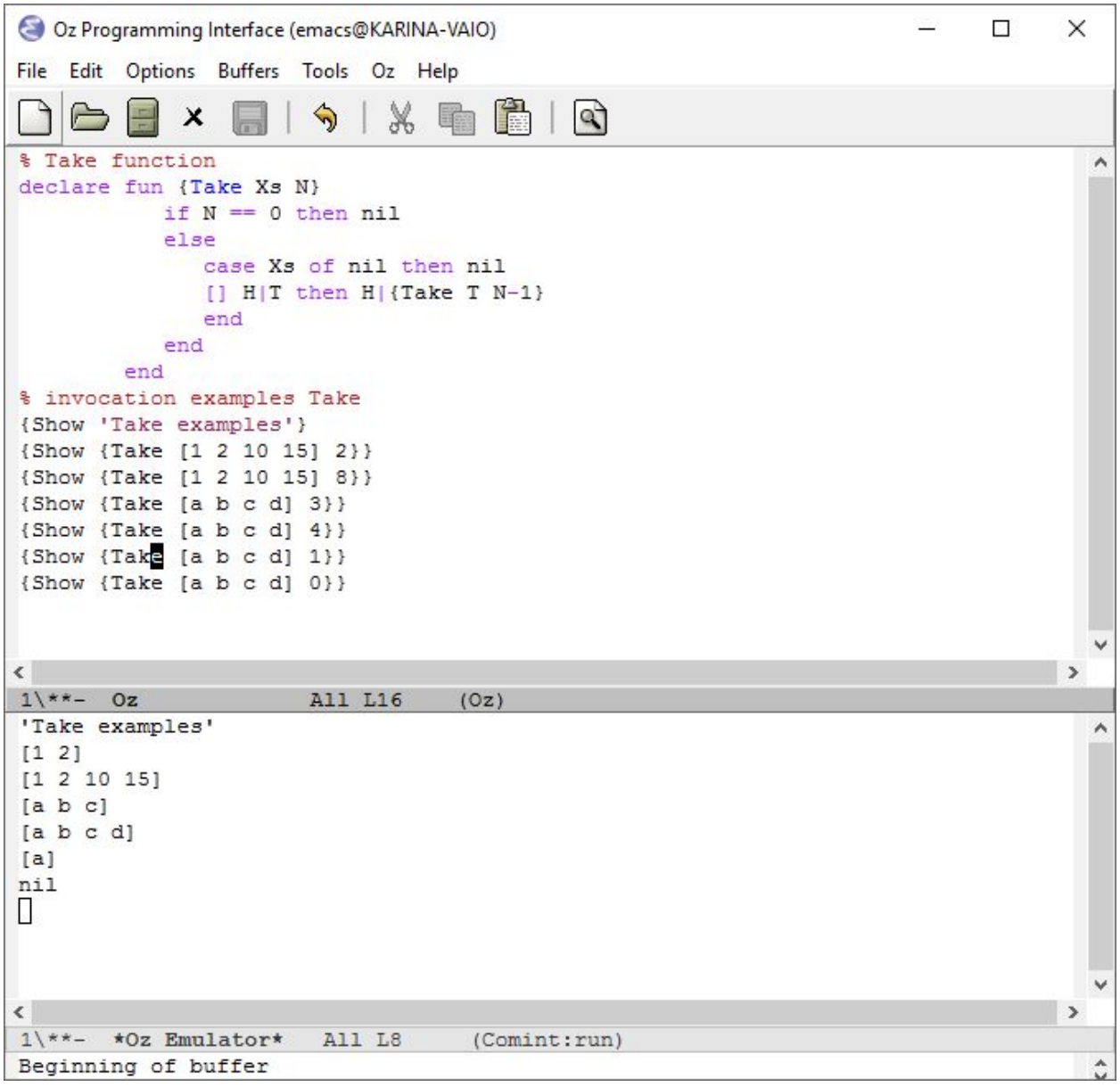


2. Take

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

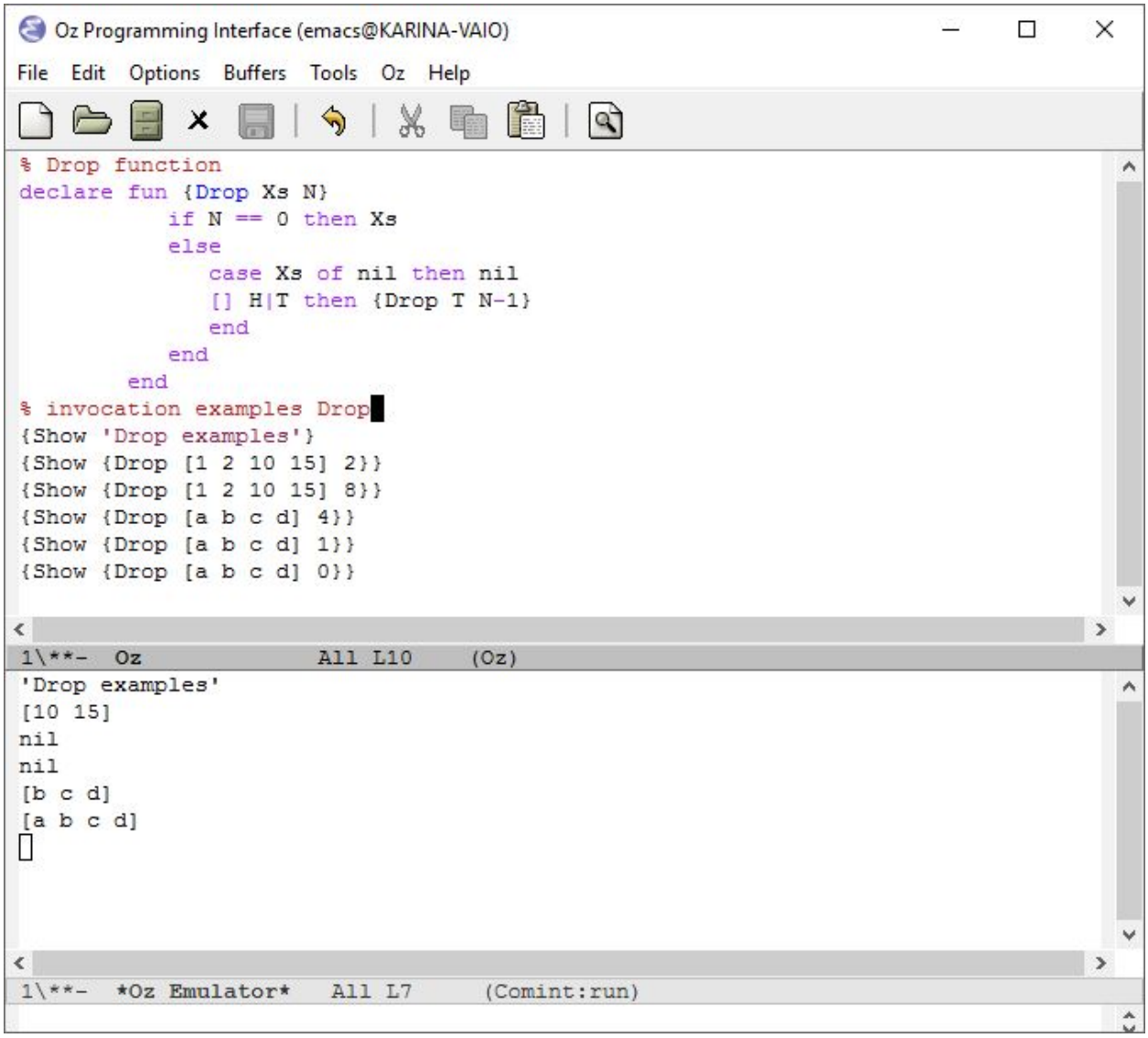


3. Drop

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

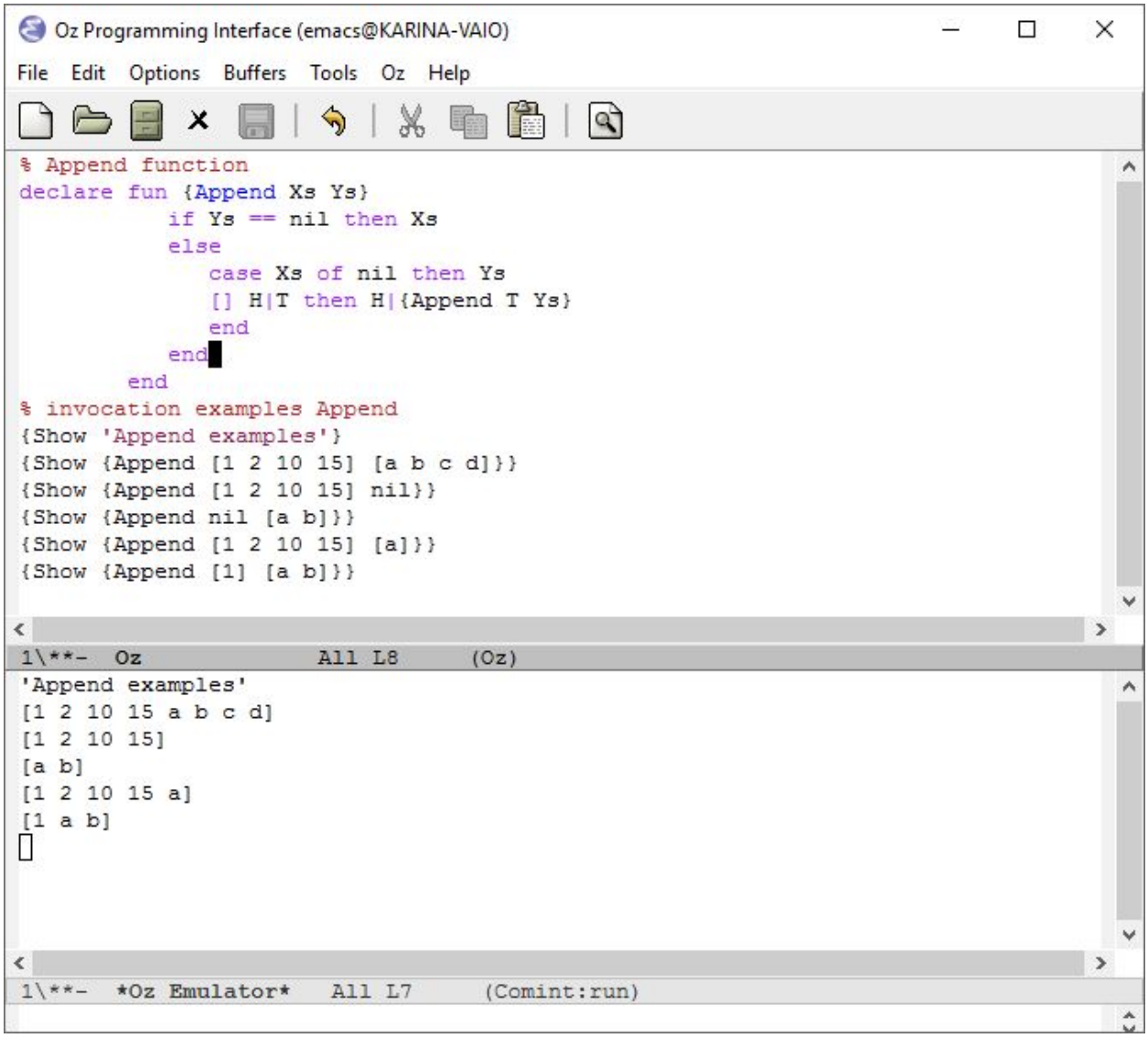


4. Append

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

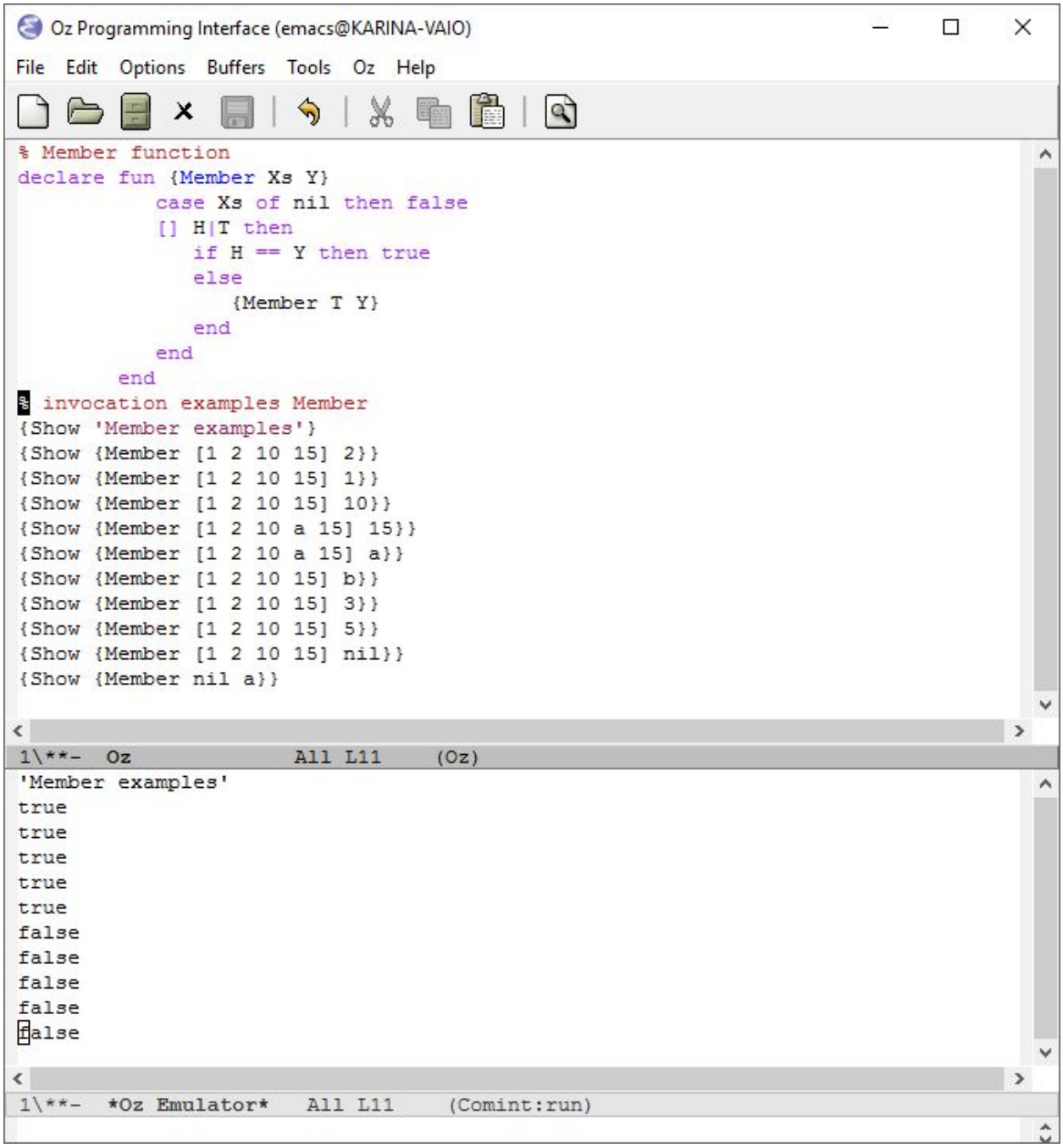


5. Member

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

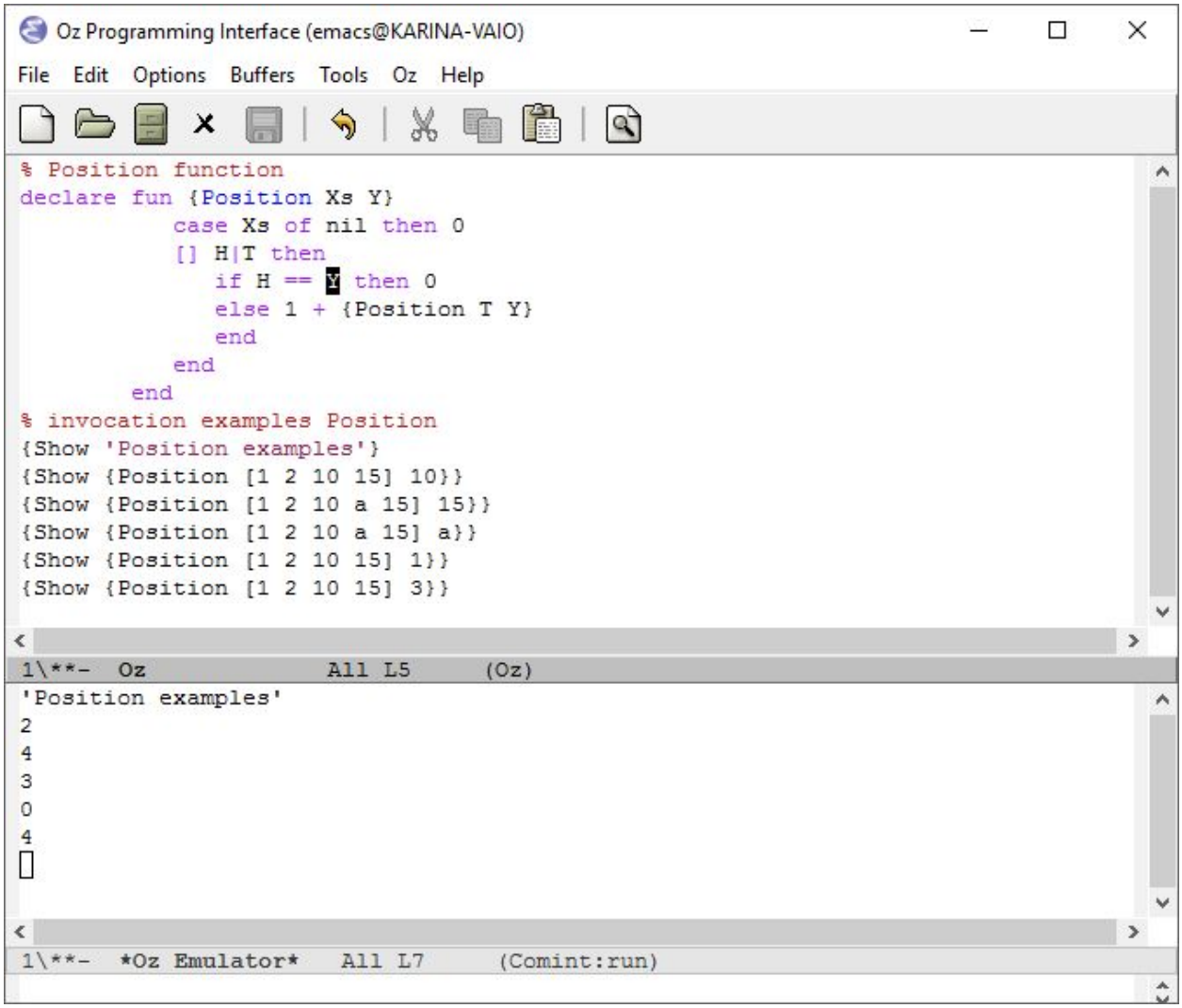

Oz Programming Interface (emacs@KARINA-VAIO)
File Edit Options Buffers Tools Oz Help
% Member function
declare fun {Member Xs Y}
case Xs of nil then false
[] H|T then
if H == Y then true
else
{Member T Y}
end
end
end
% invocation examples Member
{Show 'Member examples'}
{Show {Member [1 2 10 15] 2}}
{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}}
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

```
% 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



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**

if B then
skip
else
skip
end
	b1	**B-> b1**

=> 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
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 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
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	**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 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}	**x1** **y1** **z1** **p1** **q1**	**X -> x1** **Y -> y1** **Z -> z1** **P -> p1** **Q -> q1**

=> Asignación de variable

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

=> Declaración procedure value

Stack	Store	E
{P X Q} {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** **end, CE** = {} q1	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

=> Ejecución procedure value

Stack	Store	E

proc {Q U V}
  local F in
  F = X + 1
  V = U + F
  end
end
{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
	end, CE = {}
	q1
		X -> x1
		Y -> y1
		Z -> z1
		P -> p1
		Q -> q1

=> Declaración procedure value

Stack	Store	E

{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
	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	E
local F in **F = X + 1** **Z = Y + F** **end**	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}	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1

=> declaración de variable

Stack	Store	E
F = X + 1 Z = Y + F	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**}	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 a|a}

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