# 7524 - Teoría de la programación TP Individual

Karina Alaya

Padron 75840

#### Ejercicio 1) - Procesamiento de listas

**Length** 

<u>Take</u>

Drop

<u>Append</u>

<u>Member</u>

**Position** 

#### Ejercicio 2) - Referencias externas

#### Ejercicio 3) - Ejemplo de ejecución

Programa 1

Programa 2

Programa 3

Programa 4

Programa 5

Programa 6

Ejercicio 4) - Case

<u>Ejercicio 5) - Recursividad</u>

# Ejercicio 1) - Procesamiento de listas

#### 1. Length

a. Código Oz

```
% Length function

declare fun {Length Xs}

    case Xs of nil then 0

[] HIT 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

[] HIT then HI{Take T N-1}

end

end

end
```

```
X
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
              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

[] HIT then {Drop T N-1}

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*
                      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

[] HIT then HI{Append T Ys}

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

[] HIT then

    if H == Y then true

    else

    {Member T Y}

    end

    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

```
% Position function

declare fun {Position Xs Y}

    case Xs of nil then 0

[] HIT then

    if H == Y then 0

    else 1 + {Position T Y}

    end

    end

    end

    end
```

```
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	Е
local B in if B then Skip else	-	-
skip end end		

#### => Ejecución de declaración de variable

Stack	Store	Е
if B then skip	b1	B-> b1
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	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	Е
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	E
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	-	-
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

#### => Ejecución de procedure value

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

# => Ejecución de binding de variable

Stack	Store	E
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	Е
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

# => 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}	
eria, CL - (Z > Zij	

#### => 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	Е
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

0.00	F=A+1 V=U+F	q1	Q -> q1
end	a e		
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	E
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	<pre>x1 = 6 y1 = 4 z1 p1 = proc {\$ A B}     proc {B U V}</pre>	E  X -> x1  Y -> y1  Z -> z1  P -> p1  Q -> q1
	, ()	

#### => declaración de variable

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

# => 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) + 4(y1) = 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
<pre>X = Y try      X = 1Y = 2 Z = 3 catch Exception then      skip end {Browse X#Y#Z}</pre>	x1 y1 z1	X -> x1 Y -> y1 Z -> z1

# => 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	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 '|'(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