

# LINFO1104 : Solution des codes des TP

## . TP2

```
1  % QUESTION 1 -----
2  %[a b c] a|b|c|nil a|(b|(c|nil))
3
4  declare L1 L2 L3 L4 L5 L6 L7
5  L1 = a|nil
6  L2 = a|(b|c|nil)|d|nil
7  L3 = proc{$}{Browse oui}end|proc{$}{Browse non}end|nil
8  L4 = est|une|liste|nil
9  L5 = (a|p|nil)|nil
10 L6 = ceci|L4
11 L7 = L2.2 %1 correspond      Head et 2      Tail
12
13 {Browse L1}
14 {Browse L2}
15 {Browse L3}
16 {Browse L4}
17 {Browse L5}
18 {Browse L6}
19 {Browse L7}
20
21 declare
22 fun {Head L}
23   case L of H|T then H
24   [] nil then nil
25   end
26 end
27
28 fun {Tail L}
29   case L of H|T then T
30   [] nil then nil
31   end
32 end
33
34 {Browse {Head a|b|c|nil}}
35 {Browse {Tail a|b|c|nil}}
36
37 %QUESTION 2 -----
38 declare
39 fun {Length L}
40   fun {Length2 L N}
41     case L of H|T then {Length2 T N+1}
42     [] nil then N
43     end
44   end
45 in
46   {Length2 L 0}
47 end
48
49 {Browse {Length l|o|i|s|nil}}
50 {Browse {Length [[b o r]i s]}}
51
52 %QUESTION 3 -----
53 declare
54 fun {Append L1 L2}
55   case L1 of H|T then H|{Append T L2}
56   [] nil then L2
57   end
58 end
59
60 {Browse {Append [r a] [p h]}}
```

```

61 {Browse {Append [b [o r]] [i s]}}
62
63 %QUESTION 4 -----
64 declare
65 fun {F Arg}
66     case Arg of H|T then 'nonEmpty'
67     [] nil then 'empty'
68     else 'other'
69     end
70 end
71
72 {Browse {F 2}}
73 {Browse {F a|b|c|nil}}
74 {Browse {F nil}}
75
76 %QUESTION 5 -----
77 declare
78 fun {Take Xs N}
79     case Xs of H|T then
80         if N>0 then H|{Take T N-1}
81         else nil
82         end
83     [] nil then nil
84     end
85 end
86
87 fun {Drop Xs N}
88     case Xs of H|T then
89         if N>0 then {Drop T N-1}
90         else Xs
91         end
92     [] nil then nil
93     end
94 end
95
96 {Browse {Take [r a p h] 2}}
97 {Browse {Take [r a p h] 7}}
98 {Browse {Drop [r a p h] 2}}
99 {Browse {Drop [r a p h] 7}}
100
101 %QUESTION 6 -----
102 declare
103 fun {MultList L}
104     fun {MultL L Tot}
105         case L of H|T then {MultL T Tot*H}
106         [] nil then Tot
107         end
108     end
109 in
110     {MultL L 1}
111 end
112
113 {Browse {MultList [1 2 3 4]}}
114
115 %QUESTION 8 -----
116 declare
117 fun {Prefix L1 L2}
118     case L1 of H1|T1 then
119         case L2 of H2|T2 then
120             if H1 == H2 then {Prefix T1 T2}
121             else false
122             end
123         [] nil then false
124         end
125     [] nil then true

```

```

126     end
127 end
128 fun {FindString S T}
129     local IndexMax FindStringHelp
130     IndexMax = {Length T}-{Length S}+1
131     FindStringHelp =
132     fun{$ L1 L2 I}
133         if I <= IndexMax then % !!!!! =< et PAS <=
134             if {Prefix L1 L2} then
135                 I|{FindStringHelp L1 L2.2 I+1}
136             else {FindStringHelp L1 L2.2 I+1}
137             end
138         else nil
139         end
140     end
141     in
142     {FindStringHelp S T 1}
143     end
144 end
145
146 {Browse {Prefix [1 2 1] [1 2 3 4]}}
147 {Browse {Prefix [1 2 3] [1 2 3 4]}}
148 {Browse {FindString [a b a b] [a b a b a b]}}
149
150 %QUESTION 9 -----
151 declare
152 fun {GetPrice C N1 N2 N3}
153     N1 * C.1.prix + N2 * C.2.prix + N3 * C.3.prix
154 end
155
156 local
157     Carte = carte(menu(entree: 'salade verte aux lardons'
158                        plat: 'steak frites'
159                        prix: 10)
160                  menu(entree: 'salade de crevettes grises'
161                        plat: 'saumon fume et pommes de terre'
162                        prix: 12)
163                  menu(plat: 'choucroute garnie'
164                        prix: 9))
165 in
166     {Browse Carte.2}
167     {Browse Carte.2.plat}
168     {Browse Carte.1.entree}
169     {Browse {GetPrice Carte 8 11 6}}
170 end
171
172 %QUESTION 10 -----
173 declare
174 fun {Promenade Bt} %FAUX
175     if Bt.left == empty andthen Bt.right == empty then nil
176     elseif Bt.left == empty then Bt.1|{Promenade Bt.right}
177     elseif Bt.right == empty then Bt.1|{Promenade Bt.left}
178     else Bt.1|{Promenade Bt.left}|{Promenade Bt.right}
179     end
180 end
181 fun {PromenadeSoluze Bt}
182     local
183     fun {PromenadeHelp Bt Acc}
184         case Bt of btree(V left:TL right:TR) then
185             V|{PromenadeHelp TL {PromenadeHelp TR Acc}}
186             [] empty then Acc
187         end
188     end
189     in
190     {PromenadeHelp Bt nil}

```

```

191     end
192 end
193 fun {FoldL L F U}
194     case L of nil then U
195     [] H|T then {FoldL T F {F U H}}
196     end
197 end
198
199 fun {SumTree Bt}
200     local L Sum
201     L = {PromenadeSoluce Bt}
202     Sum =
203     fun {$ Lst Tot}
204         case Lst of H|T then {Sum T Tot+H}
205         [] nil then Tot
206         end
207     end
208     in
209     {Sum L 0}
210     end
211 end
212
213 Btree = btree(42
214     left: btree(26
215         left: btree(54
216             left: empty
217             right: btree(18
218                 left: empty
219                 right: empty))
220         right: empty)
221     right: btree(37
222         left: btree(11
223             left: empty
224             right: empty)
225         right: empty))
226 {Browse {Promenade Btree}}
227 {Browse {PromenadeSoluce Btree}}
228 {Browse {FoldL [1 2 3 4 5 6 7 8 9] fun {$ X Y} X+Y end 0}}
229 {Browse {SumTree Btree}}
230
231 %QUESTION 11 -----
232 declare
233 fun {DictionaryFilter D F}
234     local
235     fun {DictionaryFilterHelp D F Acc}
236         case D of dict(key:Key info:Info left:Left right:Right) then
237             if {F D.info} then Key#Info|{DictionaryFilterHelp Left F {
238                 DictionaryFilterHelp Right F Acc}}
239             else {DictionaryFilterHelp Left F {DictionaryFilterHelp
240                 Right F Acc}}
241             end
242         [] leaf then Acc
243         end
244     end
245     in
246     {DictionaryFilterHelp D F nil}
247     end
248 end
249
250 Class = dict(key:10
251     info:person('Christian' 19)
252     left:dict(key:7
253         info:person('Denys' 25)
254         left:leaf
255         right:dict(key:9

```

```

254         info:person('David' 7)
255         left:leaf
256         right:leaf))
257     right:dict(key:18
258         info:person('Rose' 12)
259         left:dict(key:14
260             info:person('Ann' 27)
261             left:leaf
262             right:leaf)
263         right:leaf))
264 fun {Old Info}
265     Info.2 > 20
266 end
267 {Browse {DictionaryFilter Class Old}}
268 % Val --> [7#person('Denys' 25) 14#person('Ann' 27)]
269
270 %QUESTION 12 -----
271
272 {Browse '(a b):'|{IsList '|'(a b)}|{IsTuple '|'(a b)}} %TUPLE
273 {Browse '|(a |(b nil)):'|{IsList '|'(a '|'(b nil))}|{IsTuple '|'(a '|'(b
    nil))}} %LISTE
274 {Browse '|(2:nil a):'|{IsList '|'(2:nil a)}|{IsTuple '|'(2:nil a)}} %
    LISTE (correspond a|nil car si tu print L.2 on a nil)
275 {Browse 'state(1 a 2):'|{IsList state(1 a 2)}|{IsTuple state(1 a 2)}} %
    TUPLE
276 {Browse 'state(1 3:2 2:a):'|{IsList state(1 3:2 2:a)}|{IsTuple state(1
    3:2 2:a)}} %TUPLE
277 {Browse 'tree(v:a Btree.left Btree.right:'|{IsList tree(v:a Btree.left
    Btree.right)}|{IsTuple tree(v:a Btree.left Btree.right)}} %
    ENREGISTREMENT
278 {Browse 'a#b#c:'|{IsList a#b#c}|{IsTuple a#b#c}} %TUPLE
279 {Browse '[a b c]:'|{IsList [a b c]}|{IsTuple [a b c]}} %LISTE
280 {Browse 'm|n|o:'|{IsList m|n|o}|{IsTuple m|n|o}} %TUPLE (alors que m|n|o
    |nil est bien une liste)
281
282 %QUESTION 13 -----
283
284 declare
285 fun {Applique L F}
286     case L of H|T then {F H}|{Applique T F}
287     [] nil then nil
288     end
289 end
290 fun {Lol X}
291     lol(X)
292 end
293 fun {MakeAdder X}
294     fun {$ Y} X+Y
295     end
296 end
297 fun {AddAll L N}
298     case L of H|T then H+N|{AddAll T N}
299     [] nil then nil
300     end
301 end
302
303 {Browse {Applique [1 2 3] Lol}} % Affiche [lol(1) lol(2) lol(3)]
304 Add5 = {MakeAdder 5} % correspond Add5 = fun {$ Y} Y+5
305 {Browse {Add5 13}} % Affiche 18
306 {Browse {AddAll [1 2 3 4] 2}}
307
308 %QUESTION 14 -----
309
310 {Browse {Label a#b#c}} %#
311 {Browse {Width un#tres#long#tuple#tres#tres#long}} %7

```

```

312 {Browse {Arity 1#4#16}} % 1#2#3
313
314 fun {SameLength Xs Ys}
315   case Xs#Ys
316   of nil#nil then true
317   [] (X|Xr)#(Y|Yr) then {SameLength Xr Yr}
318   else false
319   end
320 end

```

## . TP3

```

1  % INTRODUCTION
2  -----
3  declare
4  fun {Fact N}
5    if N==0 then 1
6    else N*{Fact N-1} end
7  end
8
9  KernelFact in
10 proc {KernelFact N R}
11   local Zero B1 in
12     Zero = 0
13     B1 = N==Zero
14     if B1 then R = 1
15     else
16       local N1 B2 One in %Attention !!! Ne pas oublier One comme Zero
17       dans if
18         One = 1
19         N1 = N-One
20         {KernelFact N1 B2}
21         R = N*B2
22       end
23     end
24   end
25 end
26
27 {Browse {Fact 5}}
28 {Browse {KernelFact 5}}
29
30 % QUESTION 1 -----
31 %Point A -----
32 declare
33 fun {Sum N}
34   if N == 1 then 1
35   else N*N + {Sum N-1} end
36 end
37 KernelSum in
38 proc {KernelSum N ?R}
39   local One B1 in
40     One = 1
41     B1 = N==1
42     if B1 then R = 1
43     else
44       local N1 B2 B3 in
45         N1 = N-1
46         B2 = N*N
47         B3 = {Sum N1}
48         R = B2 + B3
49       end
50     end
51   end
52 end
53 end
54 end
55

```

```

52 {Browse {Sum 9}}
53 {Browse {KernelSum 9}}
54
55 %Point B -----
56 declare
57 fun {SumAux N Acc}
58   if N == 1 then Acc + 1
59   else {SumAux N-1 N*N+Acc} end
60 end
61 fun {Sum N}
62   {SumAux N 0}
63 end
64
65 KernelSumAux KernelSum in %ATTENTION!!!! Bien mettre les 2 fonctions sur
   la mme ligne
66 proc {KernelSumAux N Acc ?R}
67   local One B1 in
68     One = 1
69     B1 = N==1
70     if B1 then R = Acc+1
71     else
72       local N1 B2 B3 in
73         N1 = N-1
74         B2 = N*N
75         B3 = B2+Acc
76         {KernelSumAux N1 B3 R}
77       end
78     end
79   end
80 end %NE PAS METTRE LE 'KernelSum in' ici
81
82 proc {KernelSum N ?R}
83   local Zero in
84     Zero = 0
85     {KernelSumAux N Zero R}
86   end
87 end
88
89 % QUESTION 2 -----
90
91 %[1]-> '|' (1:1 2:nil)
92 %[1 2 3]-> '|' (1:1 2:'|' (1:2 2:'|' (1:3 2:nil)))
93 % nil-> nil
94 %state(4 f 3)-> state(1:4 2:f 3:3)
95
96 % QUESTION 3 -----
97
98 %Environnement contextuel
99 %proc {Q A} {P A+1} end ---> E_c = {}
100 %proc {P} {Browse A} end ---> E_c = {Browse -> browse}
101
102 local P Q in
103   proc {P A R} R=A+2 end
104   local P R in
105     fun {Q A}
106       {P A R}
107     R
108   end
109   proc {P A R} R=A-2 end
110 end
111 %% Qu'affiche {Browse {Q 4}} ? 2
112 {Browse {Q 4}}
113 end
114
115 % QUESTION 5-----

```

```

116 local MakeAdd Add1 Add2 in
117   proc {MakeAdd X Add}
118     proc {Add Y Z}
119       Z=X+Y
120     end
121   end
122   {MakeAdd 1 Add1}
123   {MakeAdd 2 Add2}
124   local V in
125     {Add1 42 V} {Browse V} %43
126   end
127   local V in
128     {Add2 42 V} {Browse V} %44
129   end
130 end

```

## • TP4

```

1  % QUESTION 1 -----
2  local P in
3    local Z in
4      Z=1
5      proc {P X Y} Y=X+Z end
6    end
7    local B A in
8      A=10
9      {P A B}
10     {Browse B}
11   end
12 end
13
14 %Voir feuilles    la main
15
16 % QUESTION 2 -----
17 declare
18 fun{MakeMulFilter N}
19   fun{$ I} I mod N == 0
20   end
21 end
22 fun {Filter L F}
23   case L of H|T then
24     if {F H} then
25       H|{Filter T F}
26     else {Filter T F}
27     end
28   [] nil then nil
29   end
30 end
31 fun {IsPrime Nbr}
32   local IsPrimeHelp
33   fun {IsPrimeHelp X N}
34     if N > (X div 2) then true
35     elseif X mod N \= 0 then {IsPrimeHelp X N+1}
36     else false
37     end
38   end
39   in
40     {IsPrimeHelp Nbr 2}
41   end
42 end
43
44 Mul5 = {MakeMulFilter 5}
45 {Browse {Mul5 15}}
46 {Browse {Mul5 16}}
47 {Browse {Filter [1 2 3 4 5 6] fun{$ A} A mod 2 == 0 end}}

```



```

48 {Browse {Filter [1 2 3 4 5 6] fun{$ A} A mod 3 == 0 end}}
49 {Browse {Filter [1 2 3 4 5 6 7 8 9] IsPrime}}
50
51 % QUESTION 3 -----
52 declare
53 fun{MultL L}
54   if L == nil then 0
55   else
56     local MultLHelp
57     fun {MultLHelp L Tot}
58       case L of H|T then {MultLHelp T Tot*H}
59       [] nil then Tot
60       end
61     end
62     in
63       {MultLHelp L 1}
64     end
65   end
66 end
67 fun{DiffL L}
68   local DiffLHelp
69   fun {DiffLHelp L Tot}
70     case L of H|T then {DiffLHelp T Tot-H}
71     [] nil then Tot
72     end
73   end
74   in
75     {DiffLHelp L 0}
76   end
77 end
78
79 {Browse {MultL [1 2 3 4 5 6 7 8 9]}}
80 {Browse {MultL nil}}
81 {Browse {DiffL [1 2 3 4 5 6 7 8 9]}}
82 {Browse {DiffL nil}}
83
84 % QUESTION 4 -----
85 declare
86 fun {Applique L F}
87   case L of H|T then {F H}|{Applique T F}
88   [] nil then nil
89   end
90 end
91 fun {PowSolve Exp}
92   fun {PowHelp N E Tot}
93     if E>1 then {PowHelp N E-1 Tot*N}
94     else Tot
95     end
96   end
97   in
98     fun{$ Nbr}
99       {PowHelp Nbr Exp Nbr}
100     end
101 end
102
103 {Browse {Applique [1 2 3 4 5] {PowSolve 3}}}
104 %On a donc {{PowSolve 3} H} qui va direct aller dans le "in" pour
    r soudre {PowHelp H 3 H}
105
106 % QUESTION 5 -----
107 %Point A -----
108 declare
109 fun {Convertir1 T}
110   fun {$ V} V*T end
111 end

```

```

112
113 %Point B -----
114 declare
115 fun {Convertir2 T Extra}
116     fun {$ V} V*T + Extra end
117 end
118
119 PiedsEnMetres = {Convertir1 0.3048}
120 {Browse {PiedsEnMetres 10.0}}
121 FahrenheitEnDegres = {Convertir2 0.56 ~17.78}
122 {Browse {FahrenheitEnDegres 10.0}}
123
124 % QUESTION 6 -----
125 declare
126 fun{GenerateList N}
127     local GenerateListHelp
128     fun {GenerateListHelp Begin End}
129         if Begin > End then nil
130         else
131             Begin|{GenerateListHelp Begin+1 End}
132         end
133     end
134 in
135     {GenerateListHelp 0 N}
136 end
137
138 fun{MyFilter L F}
139     case L of H|T then
140         if {F H} then H|{MyFilter T F}
141         else {MyFilter T F} end
142     [] nil then nil
143     end
144 end
145
146 fun{MyMap L F}
147     case L of H|T then {F H}|{MyMap T F}
148     [] nil then nil end
149 end
150
151 fun{MyFoldL L F Acc}
152     case L of H|T then {MyFoldL T F {F Acc H}}
153     [] nil then Acc
154 end
155
156 fun {PipeLine N}
157     P1 P2 P3 in
158     P1 = {GenerateList N}
159     P2 = {MyFilter P1 fun {$ X} X mod 2 \= 0 end}
160     P3 = {MyMap P2 fun {$ X} X * X end}
161     {MyFoldL P3 fun {$ Acc X} X + Acc end 0}
162 end
163
164 {Browse {PipeLine 10}}
165
166 % QUESTION 7 -----
167 local Y LB in
168     Y=10
169     proc {LB X ?Z}
170         local B in
171             B = X>=Y
172             if X>=Y then Z=X
173             else Z=Y end
174         end
175     end
176 local Y Z Five in
177     Y = 15
178     Cinq = 5

```

```
177         {LB Cinq Z}  
178         {Browse Z}  
179     end  
180 end
```

## • TP5

## LINFO1104 : TP5 Lambda-Calcul

Le lambda calcul est un système formel sur lequel se base la programmation fonctionnelle. C'est un langage qui est Turing complet, ce qui veut dire qu'il peut faire tous les calculs qu'un langage de programmation "normal" peut faire. Il a l'avantage d'être petit et d'avoir une syntaxe simple. Le lambda-calcul définit un ensemble d'expressions comme étant valide. Ces expressions sont appelées des lambda-termes et se divisent en trois catégories:

**variables** : une variable  $x$  est lambda-terme.

**abstraction** : Si  $m$  est un lambda-terme et  $x$  est une variable, alors  $(\lambda x.m)$  est un lambda-terme. Dans ce cas-ci,  $m$  est appelé le corps de l'abstraction et  $x$  en est l'argument.

**applications** : Si  $m$  et  $n$  sont des lambda-termes, alors  $(mn)$  est un lambda-terme.

Seules ces règles sont utilisées pour définir ce qu'est une expression valide en lambda-calcul. Pour clarifier les expressions, il est courant en lambda-calcul d'omettre certaines parenthèses et de compresser les séquences d'abstractions.

- Les parenthèses les plus à l'extérieur sont retirées:  $(mn)$  devient  $mn$ .
- Les applications sont effectuées en faisant l'association par la gauche:  $((mn)p)$  devient  $mnp$ .
- Le corps d'une expression s'étend autant que possible vers la droite.  $\lambda x.mn$  signifie  $\lambda x.(mn)$  et non  $(\lambda x.m)n$ .
- Une séquence d'abstraction peut être abrégée.  $\lambda x.\lambda y.\lambda z.n$  est abrégé en  $\lambda xyz.n$ .

### Définition formelle du lambda-calcul

$x(\lambda w.\lambda y.y)$

1. **Validité d'une expression.** Pour les expressions suivantes, identifiez celles qui sont valides.

- |  |   |
|--|---|
| • $\lambda x.xyz \rightarrow \text{valide} : \lambda x.((xy)z)$        | • $x\lambda \rightarrow \text{invalid}$                     |
| • $\lambda x.\lambda y \rightarrow \text{invalid}$                     | • $\lambda\lambda xz.zx \rightarrow \text{invalid}$         |
| • $m \rightarrow \text{valide}$  | • $(mnop)(qrst)vw\lambda xyz.zxy \rightarrow \text{valide}$ |
| • $x\lambda wy.y \rightarrow \text{valide} : x(\lambda w.\lambda y.y)$ |   |

(Solution)

- |            |            |
|------------|------------|
| • valide   | • invalide |
| • Invalide | • invalide |
| • valide   | • valide   |
| • valide   |            |

2. **Variable libre et variable liée.** En lambda-calcul, le concept de variable libre et variable liée existe aussi. Quand une variable fait partie des arguments d'une abstraction, on dit qu'elle est liée à cette abstraction. Pour les expressions suivantes, identifiez pour chaque abstraction leur corps et indiquez pour chaque variable  $x$  à quelle abstraction est-elle liée.

- |  |  |
|--|--|
| • $\lambda x(\lambda y.x) \rightarrow \lambda x.(\lambda y.x)$     | • $\lambda x(x\lambda x.x) \rightarrow \lambda x.(x(\lambda x.x))$ |
| • $\lambda x.\lambda x.x \rightarrow \lambda x.(\lambda x.x)$      | • $\lambda z.x\lambda y.x \rightarrow \lambda z.(x(\lambda y.x))$  |
| • $\lambda x(x\lambda y.x) \rightarrow \lambda x.(x(\lambda y.x))$ | • $\lambda z.x\lambda x.x \rightarrow \lambda z.(x(\lambda x.x))$  |

(Solution)

- $(\lambda x.(\lambda y.x))$
- $(\lambda x.(\lambda x.x))$
- $(\lambda x.x(\lambda y.x))$
- $(\lambda x.x(\lambda x.x))$
- $(\lambda z.x(\lambda y.x))$
- $(\lambda z.x(\lambda x.x))$

3. **Renommage de variable ( $\alpha$  - conversion).** Pour chaque paire d'expression, indiquez si les expressions sont  $\alpha$  - *equivalent*.

$\lambda a.(\lambda b.(abb))$	$\lambda b.(\lambda a.(baa))$	OUI
$\lambda a.\lambda b.\lambda a.bb$	$\lambda i.\lambda j.jji$	NON
$\lambda x.x\lambda y.x$	$\lambda e.e\lambda f.f$	NON
$\lambda x.x\lambda y.x$	$\lambda e.e\lambda f.e$	OUI

(Solution)

$\lambda a.\lambda b.abb$	$\lambda b.\lambda a.baa$	Oui
$\lambda a.\lambda b.\lambda a.bb$	$\lambda i.\lambda j.jji$	Non
$\lambda x.x\lambda y.x$	$\lambda e.e\lambda f.f$	Non
$\lambda x.x\lambda y.x$	$\lambda e.e\lambda f.e$	Oui

4. **Réduction d'expression ( $\beta$ ).** Réduisez au maximum les expressions suivantes en utilisant la  $\beta$ -reduction.

- $(\lambda x.xx)y \rightarrow yy$
- $(\lambda x.axxa)y \rightarrow ayya$
- $(\lambda x.(\lambda z.zx)q)y \rightarrow (\lambda x.qx)y \rightarrow qy$
- $(\lambda x.x((\lambda z.zx)(\lambda x.bx)))y \rightarrow (\lambda x.x(bx))y \rightarrow y(bx)$
- $(\lambda m.m)(\lambda n.n)(\lambda c.cc)(\lambda d.d) \rightarrow (\lambda d.d)(\lambda d.d) \rightarrow (\lambda d.d)$
- $\lambda z.x\lambda x.x \rightarrow \text{Non reduc.}$

(Solution)

- $(\lambda x.xx)y \Rightarrow yy$
- $(\lambda x.axxa)y \Rightarrow ayya$
- $(\lambda x.(\lambda z.zx)q)y \Rightarrow qy$
- $(\lambda x.x((\lambda z.zx)(\lambda x.bx)))y \Rightarrow y(by)$
- $(\lambda m.m)(\lambda n.n)(\lambda c.cc)(\lambda d.d) \Rightarrow (\lambda d.d)$
- $\lambda z.x\lambda x.x$

5. **Reduction d'expression ( $\eta$ ).** Réduisez au maximum les expressions suivantes en utilisant l' $\eta$ -reduction.

- $\lambda x.(\lambda y.y)x \rightarrow \lambda y.y$
- $\lambda x.(\lambda y.(\lambda z.p)y)x \rightarrow \lambda z.p$
- $\lambda x.(\lambda y.(\lambda z.z))x \rightarrow \lambda y.(\lambda z.z)$
- $\lambda x.(\lambda y.yx)p \rightarrow \lambda x.(\lambda y.yx)p$
- $(\lambda f.fx)(\lambda y.gy) \rightarrow$

(Solution)

- $\lambda x.(\lambda y.y)x \Rightarrow \lambda y.y$
- $\lambda x.(\lambda y.(\lambda z.p)y)x \Rightarrow \lambda z.p$
- $\lambda x.(\lambda y.(\lambda z.z))x \Rightarrow \lambda y.(\lambda z.z)$
- $\lambda x.(\lambda y.yx)p$
- $(\lambda f.fx)(\lambda y.gy) \Rightarrow (\lambda f.fx)g$

On dit d'une expression qui n'est pas réductible par  $\beta$ -réduction ou par  $\eta$ -réduction est en *forme normale  $\beta$  - eta*.

## Propriété de Church-Rosser

6. Le théorème de Church-Rosser déclare que lorsqu'on applique des règles de réduction à des termes du lambda-calcul, l'ordre dans lequel les réductions sont choisies ne fait pas de différence au résultat final.

S'il y a deux réductions ou séquences de réductions distinctes qui peuvent être appliquées au même terme, alors il existe un terme qui peut être atteint à partir des deux résultats, en appliquant des séquences (éventuellement vides) de réductions supplémentaires.

L'expression suivante peut être réduite de deux façons :  $(\lambda x. \lambda y. x) ((\lambda x. x) y)$ .

- Simplifiez cette expression en réduisant d'abord les lambda-termes les plus à gauche possible. Refaites ensuite la même opération en réduisant d'abord les lambda-termes les plus à droite possible. Que remarquez-vous ?
- Refaites la même procédure pour  $(\lambda x. \lambda y. x y) (\lambda z. z) (\lambda w. w)$

(Solution)

$$\begin{array}{c} \beta \\ \curvearrowright \\ (\lambda x. \lambda y. x) ((\lambda x. x) y) = \lambda y. ((\lambda x. x) y) \\ = \lambda y. y \end{array}$$

$$\begin{array}{c} (\lambda x. (\lambda y. x)) (y) \\ (\lambda x. \lambda y. x) ((\lambda x. x) y) = \lambda x. ((\lambda y. x) y) \\ = \lambda x. x \end{array} \quad ?$$

$$\begin{array}{c} (\lambda x. \lambda y. x y) (\lambda z. z) (\lambda w. w) = (\lambda y. (\lambda z. z) y) (\lambda w. w) \\ = (\lambda z. z) (\lambda w. w) \\ = \lambda w. w \end{array}$$

$$\begin{array}{c} (\lambda x. \lambda y. x y) (\lambda z. z) (\lambda w. w) = \lambda x. (\lambda y. x y (\lambda w. w)) \\ = \lambda y. (\lambda w. w) y \\ = \lambda w. w \end{array}$$

Le théorème de Church-Rosser stipule que la  $\beta$ -réduction est confluente. Si une expression conduit à deux formes normales irréductibles, elles sont  $\alpha$ -équivalentes (équivalentes au renommage près).

## Représentation de type de donnée

7. **Arithmétiques des booléens.** Il n'y a pas de booléen ou de nombre en lambda-calcul mais il est possible de les modéliser juste avec des fonctions. Pour représenter les booléens, on utilise une notation appelée Church-booléen:

- $true := \lambda x. \lambda y. x$
- $false := \lambda x. \lambda y. y$

Avec ces lambda-termes, les opérateurs logiques peuvent être aussi définis par des lambda-termes, par exemple l'opérateur *and* peut être défini de la façon suivante:

$$and := \lambda p. \lambda q. p q p$$

- Définissez les opérateurs *not* et *or*.
- Évaluez l'expression *or true false* en utilisant les lambda-termes définis au dessus.

(Solution)

- $not := \lambda p. p \text{ false } true$
- $or := \lambda p. \lambda q. p \text{ } q$
- $or \text{ true } false \Rightarrow (\lambda p. \lambda q. p \text{ } q) \text{ true } false \Rightarrow true \text{ true } false \Rightarrow (\lambda x. \lambda y. x) \text{ true } false \Rightarrow true$

8. **Arithmétiques des nombres.** Pour représenter les nombres naturels, on utilise une notation appelé Church-numéral:

- $0 := \lambda f. \lambda x. x$  ou  $\lambda f x. x$
- $1 := \lambda f. \lambda x. f x$  ou  $\lambda f x. (f x)$
- $2 := \lambda f. \lambda x. f(f x)$  ou  $\lambda f x. f(f x)$
- $3 := \lambda f. \lambda x. f(f(f x))$  ou  $\lambda f x. f(f(f x))$

$$\lambda m. f(f^m x)$$

$$f(\lambda f. \lambda x. f x)$$

Un nombre  $n$  est donc représenté par une fonction à deux arguments  $f x$  qui exécute  $f$  sur  $x$   $n$  fois.  $f$  est une fonction et  $x$  est une variable.

- Définissez une fonction *succ* qui renvoie le successeur d'un nombre 1. La fonction a 3 arguments.
- La fonction *plus* est définie comme ci-dessous, en calculant *plus 1 2*, est-ce que le résultat est 3?
- $plus := \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x)$

(Solution)

- $succ := \lambda n. \lambda f. \lambda x. f (n f x)$
- $plus \text{ 1 } 2 \Rightarrow (\lambda m. \lambda n. \lambda f. \lambda x. m f (n f x)) \text{ 1 } 2 \Rightarrow \lambda f. \lambda x. 1 f (2 f x) \Rightarrow \lambda f. \lambda x. 1 f ((\lambda f x. f(f x)) f x) \Rightarrow \lambda f. \lambda x. 1 f (f(f x)) \Rightarrow \lambda f. \lambda x. (\lambda f. \lambda x. f(f x)) f (f(f x)) \Rightarrow \lambda f. \lambda x. f(f(f x))$

9. **Les listes en lambda calcul**

- $paire := \lambda a. \lambda b. \lambda f. ((f a) b)$
- Définissez une fonction *first* et *second* qui prennent en argument une paire et renvoie respectivement le premier et le second élément de la paire.
- Créer une *paire* 4 2 et appliquer vos fonction pour récupérer le premier et second élément
- En utilisant les paires, définissez une fonction *list* qui crée une liste.

(Solution)

- $first := \lambda a. \lambda b. a$
- $second := \lambda a. \lambda b. b$
- $p := \text{pair } 4 \text{ 2} \Rightarrow \lambda 4. \lambda 2. \lambda f. ((f a) b) \Rightarrow \lambda f ((f 4) 2)$
- $p \text{ first} \Rightarrow \lambda f. ((f 4) 2) \text{ first} \Rightarrow ((first 4) 2) \Rightarrow ((\lambda a. \lambda b. a) 4 \text{ 2}) \Rightarrow 4$
- une liste est une paire où le premier élément est une expression lambda (head) et le second élément est une liste (tail):  $\lambda f. ((f h) t)$

## La récursivité en lambda-calcul

10. **Fonction récursive.** Contrairement à un langage de programmation, le lambda-calcul ne supporte pas directement la récursion en appelant une fonction  $f$  à l'intérieur de la fonction  $f$ . Le lambda-calcul utilise alors une fonction "point-fixe".

Le point fixe d'une fonction  $f$  est un élément du domaine de  $f$ , mis en correspondance avec lui-même dans  $f$ . Par exemple, " $c$ " est le point fixe de  $f$  si  $f(c) = c$ . De même  $f(f(\dots(f(c))\dots)) = f^n(c) = c$ .

Pour la fonction  $f(x) = x^2$ , 0 et 1 sont les seuls points fixes de  $f$  puisque  $f(0) = 0$  et  $f(1) = 1$ .

- $plus := \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x)$

↑  
f      x

$$\lambda m f x. \underline{f (m f x)}$$

$$\lambda f x. f (f (f x))$$

$$m = 1 := \lambda f x. \overbrace{f x}^{\sim}$$

$$m = 2 := \lambda f x. \overbrace{f (f x)}^{\sim}$$



$$\lambda m f x. (m f x)$$

$$\lambda f x. f(f x)$$

$$\lambda f x. f f$$

Nous allons utiliser ce même principe pour créer des expression lambda récursives. Il existe un combinateur qui permet de créer une version récursive d'une fonction. Ce combinateur est lui même une fonction qui retournera un point fixe pour n'importe quelle fonction que l'on lui passe :

$$Y = \lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))$$

Lorsque  $Y$  est utilisé comme "constructeur", elle crée un point fixe sur l'argument qui lui est passé.

Regardons ce qu'il se passe lorsque  $g$  est appliqué à  $Y$  :

$$\begin{aligned} Y g &= (\lambda f. (\lambda x. f(x x)) (\lambda x. f(x x))) g \\ &= (\lambda x. g(x x)) (\lambda x. g(x x)) \\ &= g((\lambda x. g(x x)) (\lambda x. g(x x))) \\ &= g(Y g) \end{aligned}$$

Si le combinateur est appliqué indéfiniment, on obtient :

$$Y g = g(Y g) = g(g(Y g)) = g(\dots g(Y g) \dots)$$

C'est ainsi que la récursion est créée en s'assurant que  $g$  et  $Y$  continuent à s'étendre à chaque application.

**Exercice:** utilisez le combinateur  $Y$  pour appliquer une fonction *add* qui additionne 1 avec 1. Pour vous aider, traduisez ce pseudo-code en lambda-calcul :

```
def add f x y =
  if is_zero y then x
  else f (succ x) (pred x)
```

- Comment traduisez-vous les fonctions définies dans la fonction **add** ? En d'autres termes, définissez respectivement :

- if c then x else y
- is\_zero y
- succ n
- pred n

(Solution)

```
if c then x else y =  $\lambda x. \lambda y. \lambda c. ((c\ x)\ y)$ 
is_zero y =  $\lambda y. y\ (\lambda t. \text{false})\ \text{true}$ 
succ n =  $\lambda n. \lambda f. \lambda x. f\ (n\ f\ x)$ 
pred n =  $\lambda n. \lambda f. \lambda x. n\ (\lambda g. \lambda h. h\ (g\ f))\ (\lambda u. x)\ (\lambda u. u)$ 
```

- Utilisez les résultats de vos calculs précédents pour calculer en lambda calcul :

- is\_zero 2
- pred 2

$\lambda m f x . m\ x$      $\lambda f x . f\ x$

$\lambda$

$$\begin{aligned} \text{is\_zero } 2 &= (\lambda f. f\ (\lambda t. \lambda xy. y)\ \lambda xy. x)\ (\lambda sz. s(s(z))) \\ &= \lambda sz. s(s(z))\ (\lambda t. \lambda xy. y)\ \lambda xy. x \\ &= (\lambda t. \lambda xy. y)\ ((\lambda t. \lambda xy. y)\ (\lambda xy. x)) \\ &= (\lambda t. \lambda xy. y)\ (\lambda xy. x) \\ &= (\lambda xy. y) \\ &= \text{false} \end{aligned}$$

$$\begin{aligned}
\text{pred } 2 &= (\lambda n. \lambda f. \lambda x. n (\lambda g. \lambda h. h (g f)) (\lambda u. x) (\lambda u. u)) (\lambda sz. s(s(z))) \quad \textcolor{blue}{1} \\
&= \lambda f. \lambda x. (\lambda sz. s(s(z))) (\lambda g. \lambda h. h (g f)) (\lambda u. x) (\lambda u. u) \quad \textcolor{blue}{2} \\
&= \lambda f. \lambda x. (\lambda g. \lambda h. h (g f)) ((\lambda g. \lambda h. h (g f)) (\lambda u. x)) (\lambda u. u) \quad \textcolor{blue}{3} \\
&= \lambda f. \lambda x. (\lambda g. \lambda h. h (g f)) (\lambda h. h (\lambda u. x) f) (\lambda u. u) \quad \textcolor{blue}{4} \\
&= \lambda f. \lambda x. (\lambda g. \lambda h. h (g f)) (\lambda h. h x) (\lambda u. u) \quad \textcolor{blue}{5} \\
&= \lambda f. \lambda x. (\lambda h. h ((\lambda h. h x) f)) (\lambda u. u) \quad \textcolor{blue}{6} \\
&= \lambda f. \lambda x. (\lambda h. h (f x)) (\lambda u. u) \quad \textcolor{blue}{7} \\
&= \lambda f. \lambda x. (\lambda u. u (f x)) \quad \textcolor{blue}{8} \\
&= \lambda f. \lambda x. f x \quad \textcolor{blue}{9} \\
&= \lambda f. \lambda x. f(x) \quad \textcolor{blue}{10} \\
&= 1
\end{aligned}$$

- Continuez l'évaluation de l'expression demandée :

$$Y \text{ add } 1 \ 1 = \dots(\text{your job})$$

Traduire l'entièreté des fonctions en lambda calcul rendrait le calcul beaucoup trop long. Utilisez les fonction définies à l'exercice précédent pour calculer le résultat.

$$\begin{aligned}
Y \text{ add } 1 \ 1 &= \text{add } Y \text{ add } 1 \ 1 \\
&= \lambda f. \lambda x. \lambda y. (\text{if is\_zero } y \text{ then } x \text{ else } f (\text{succ } x) (\text{pred } y)) (Y \text{ add}) 1 \ 1 \\
&= \lambda x. \lambda y. (\text{if is\_zero } y \text{ then } x \text{ else } Y \text{ add } (\text{succ } x) (\text{pred } y)) 1 \ 1 \\
&= \text{if is\_zero } 1 \text{ then } 1 \text{ else } Y \text{ add } (\text{succ } 1) (\text{pred } 1) \\
&= \text{if is\_zero } 1 \text{ then } 1 \text{ else } Y \text{ add } 2 \ 0 \\
&= Y \text{ add } 2 \ 0 \\
&= \lambda f. \lambda x. \lambda y. (\text{if is\_zero } y \text{ then } x \text{ else } f (\text{succ } x) (\text{pred } y)) (Y \text{ add}) 2 \ 0 \\
&= \lambda x. \lambda y. (\text{if is\_zero } y \text{ then } x \text{ else } Y \text{ add } (\text{succ } x) (\text{pred } y)) 2 \ 0 \\
&= \text{if is\_zero } 0 \text{ then } 2 \text{ else } Y \text{ add } (\text{succ } 2) (\text{pred } 0) \\
&= \text{if is\_zero } 1 \text{ then } 1 \text{ else } Y \text{ add } 3 \ 1 \\
&= 2
\end{aligned}$$

## . TP6

```
1  % QUESTION 1 -----
2  declare
3  fun {Reverse Xs}
4      local C in
5          C = {NewCell nil}
6          for X in Xs do
7              C := X|@C
8          end
9          @C
10     end
11 end
12 {Browse {Reverse [1 2 3 4 5 6 7 8 9]}}
13
14 % Identique en terme d'appels rec
15 % L'etat n'est pas encapsule ! Il y a une Cell Y
16 % Xs est une liste (immutable), il ne peut pas changer !
17
18 % QUESTION 2 -----
19 declare
20 fun {NewStack}
21     {NewCell nil}
22 end
23 fun {IsEmpty S}
24     @S == nil
25 end
26 proc {Push S X}                %COMME MODIFIE LA CELL IL FAUT METTRE PROC
27     S := X|@S
28 end
29 fun {Pop S}
30     case @S of H|T then
31         S := T
32         H
33     [] nil then nil
34     end
35 end
36 fun{Eval L}
37     C in                        %ATTENTION PAS METTRE EN LOCAL
38     C = {NewStack}
39     for Elem in L do
40         if Elem == '+' then
41             {Push C {Pop C}+{Pop C}}
42         elseif Elem == '*' then
43             {Push C {Pop C}*{Pop C}}
44         elseif Elem == '-' then
45             local Pop1 in
46                 Pop1 = {Pop C}
47                 {Push C {Pop C}-Pop1}
48             end
49         elseif Elem == '/' then
50             local Pop1 in
51                 Pop1 = {Pop C}
52                 {Push C {Pop C}/Pop1}
53             end
54         else
55             {Push C Elem}
56         end
57     end
58     {Pop C}
59 end
60
61 {Browse {Eval [13 45 '+' 89 17 '-' '*']}}
62
63 % QUESTION 3 -----
```

```

64 declare
65 fun {NewStack}           %Plus besoin de la Cell dans les arguments des
    fonctions du coup
66   C
67   fun {IsEmpty}
68     @C == nil
69   end
70   proc {Push X}
71     C := X|@C
72   end
73   fun {Pop}
74     case @C of H|T then
75       C := T
76       H
77     [] nil then nil
78     end
79   end
80 in
81   C = {NewCell nil}      %Evidemment le C doit etre attribue aprs le "in
    " !!!!!!!
82   stack(isEmpty:IsEmpty push:Push pop:Pop)
83 end
84 fun{Eval L}
85   C in                    %ATTENTION PAS METTRE EN LOCAL
86   C = {NewStack}
87   for Elem in L do
88     if Elem == '+' then
89       {C.push {C.pop}+{C.pop}}
90     elseif Elem == '*' then
91       {C.push {C.pop}*{C.pop}}
92     elseif Elem == '-' then
93       local Pop1 in
94         Pop1 = {C.pop}
95         {C.push {C.pop}-Pop1}
96       end
97     elseif Elem == '/' then
98       local Pop1 in
99         Pop1 = {C.pop}
100        {C.push {C.pop}/Pop1}
101      end
102    else
103      {C.push Elem}
104    end
105  end
106  {C.pop}
107 end
108
109 {Browse {Eval [13 45 '+' 89 17 '-' '*']}}
110
111 % QUESTION 4 -----
112 declare
113 fun {ShuffleSolu L} Len A C Tail Head in
114   Len = {Length L}
115   A = {NewArray 0 Len 0}
116   C = {NewCell 0}
117   for Elem in L do
118     A.@C := Elem
119     C := @C+1
120   end
121   Tail = {NewCell Head}
122   for I in 0..(Len-1) do Idx Range Next in
123     Range = Len - 1 - I
124     Idx = {OS.rand} mod (Range+1)
125     @Tail = A.Idx|Next
126     Tail := Next

```

```

127         A.Idx := A.Range
128     end
129     @Tail = nil
130     Head
131 end
132 {Browse {ShuffleSoluce [a b c d e]}}
133
134 % QUESTION 5 -----
135 declare
136 C = {NewCell nil}
137 L1={NewCell 0}|{NewCell 1}|{NewCell 2}|nil
138 L2=0|{NewCell 1}|{NewCell 2}|C}
139 % L1 est une liste de Cell
140 % L2 n'est pas une liste car ne termine pas par nil
141
142 %O(1)
143 fun {Prepend1 X L} X|L end
144 fun {Prepend2 X L} X|{NewCell @L} end
145
146 % pour ajouter la fin de L1 on utilise Append O(n)
147 % pour L2 O(n)
148 proc {Append2 L X}
149     case L
150     of _|T then {Append2 T X}
151     else
152         case @L of nil then L := X|{NewCell nil}
153         [] _|C then {Append2 C X} end
154     end
155 end
156 {Append2 L2 3}
157 {Browse @C}
158
159 declare
160 % both are unsafe, we need at least two elements
161 fun {Swap L} L.2.1|L.1|L.2.2 end
162 fun {Swap2 L} H T in
163     H = (@(L.2)).1
164     T = (@(L.2)).2
165     L.2 := L.1|T
166     H|L.2
167 end
168
169 % INTRO Classes
170 -----
171 declare
172 class Counter
173     attr value
174     meth init % (re)initialise le compteur
175         value:=0
176     end
177     meth inc % incremente le compteur
178         value:=@value+1
179     end
180     meth get(X) % renvoie la valeur courante du compteur dans X
181         X=@value
182     end
183 end
184 MonCompteur={New Counter init}
185 for X in [65 81 92 34 70] do {MonCompteur inc} end
186 {Browse {MonCompteur get($)}}
187
188 % QUESTION 6 -----
189 declare
190 class Collection
191     attr elements

```

```

191     meth init
192         elements := nil
193     end
194     meth put(X)
195         elements := X|@elements
196     end
197     meth get($)
198         case @elements of X|Xr then elements := Xr X end
199     end
200     meth isEmpty($)
201         @elements == nil
202     end
203     meth union(C)
204         if {Not {C isEmpty($)}} then
205             {self put({C get($)}}}
206             {self union(C)}
207         end
208     end
209     meth toList($)
210         @elements
211     end
212 end
213
214 declare
215 class SortedCollection from Collection
216     meth put(X)
217         fun {PutHelp L}
218             case L of H|T then
219                 if H<X then H|{PutHelp T}
220                 else X|L
221                 end
222             [] nil then X|L
223             end
224         end in
225         elements :={PutHelp @elements}
226     end
227 end
228
229 declare % tests
230 C1 = {New Collection init}
231 C2 = {New SortedCollection init}
232
233 {C1 put(0)}
234 {C1 put(1)}
235 {C2 put(2)}
236 {C2 put(1)}
237 {C2 put(3)}
238
239 {C1 union(C2)}
240 {Browse {C2 isEmpty($)}}
241 {Browse {C1 isEmpty($)}}
242 {Browse {C1 toList($)}}
243 {Browse {C1 get($)}}
244 {Browse {C1 toList($)}}
245
246 % if C1, C2 are Collections, union is O(|C2|)
247 % because put is O(1)
248
249 % if C1, C2 are SortedCollections,
250 % union is O(|C2|^2 + |C1|*|C2|)
251
252 % proof:
253 % (n1 + (n1+1) + (n1+2) + ... + (n1+n2-1))
254 % ~= (n1 + (n1+1) + (n1+2) + ... + (n1+n2))
255 % = n2*n1 + (0+1+2+...+n2)

```

```

256 % = n2*n1 + 0.5*n2*(n2+1)
257 % ~= n2*n1 + 0.5*n2*n2
258 % => 0(n2^2 + n2*n1)
259
260 declare
261 Xs = [7 8 0 4 3]
262 C3 = {New SortedCollection init}
263 for X in Xs do
264     {C3 put(X)}
265 end
266 {Browse {C3 toList($)}}
267 % insertion sort O(n^2)
268
269 % QUESTION 7 -----
270 declare
271 class Constante
272     attr value
273     meth init(V) value := V end
274     meth evaluate($) @value end
275     meth derive(V $) {New Constante init(0)} end
276 end
277 class Variable
278     attr value
279     meth init(V) value := V end
280     meth evaluate($) @value end
281     meth derive(V E)
282         if self == V then E = {New Constante init(1)}
283         else E = {New Constante init(0)} end
284     end
285     meth set(V) value := V end
286 end
287 class Somme
288     attr e1 e2
289     meth init(Expr1 Expr2)
290         e1 := Expr1
291         e2 := Expr2
292     end
293     meth evaluate($) {@e1 evaluate($)} + {@e2 evaluate($)} end
294     meth derive(V $)
295         {New Somme init({@e1 derive(V $)} {@e2 derive(V $)})}
296     end
297 end
298 class Difference
299     attr e1 e2
300     meth init(Expr1 Expr2)
301         e1 := Expr1
302         e2 := Expr2
303     end
304     meth evaluate($) {@e1 evaluate($)} - {@e2 evaluate($)} end
305     meth derive(V $)
306         {New Difference init({@e1 derive(V $)} {@e2 derive(V $)})}
307     end
308 end
309 class Produit
310     attr e1 e2
311     meth init(Expr1 Expr2)
312         e1 := Expr1
313         e2 := Expr2
314     end
315     meth evaluate($) {@e1 evaluate($)} * {@e2 evaluate($)} end
316     meth derive(V $) D1 D2 in
317         D1 = {New Produit init({@e1 derive(V $)} @e2)}
318         D2 = {New Produit init({@e2 derive(V $)} @e1)}
319         {New Somme init(D1 D2)}
320     end

```



```

321 end
322 class Puissance
323   attr e1 c
324   meth init(Expr1 C)
325     e1 := Expr1
326     c := C
327   end
328   meth evaluate($) {Pow {@e1 evaluate($)} @c} end
329   meth derive(V $) De1 P CP in
330     De1 = {@e1 derive(V $)}
331     P = {New Puissance init(@e1 @c-1)}
332     CP = {New Produit init({New Constante init(@c)} P)}
333     {New Produit init(CP De1)}
334   end
335 end
336
337 declare
338 VarX={New Variable init(0)}
339 VarY={New Variable init(0)}
340 local
341   ExprX2={New Puissance init(VarX 2)}
342   Expr3={New Constante init(3)}
343   Expr3X2={New Produit init(Expr3 ExprX2)}
344   ExprXY={New Produit init(VarX VarY)}
345   Expr3X2mXY={New Difference init(Expr3X2 ExprXY)}
346   ExprY3={New Puissance init(VarY 3)}
347 in
348   Formule={New Somme init(Expr3X2mXY ExprY3)}
349 end
350
351 {VarX set(7)}
352 {VarY set(23)}
353 {Browse {Formule evaluate($)}} % affiche 12153
354 {VarX set(5)}
355 {VarY set(8)}
356 {Browse {Formule evaluate($)}} % affiche 547
357
358 declare
359 Derivee={Formule derive(VarX $)} % represente 6x - y
360 {VarX set(7)}
361 {VarY set(23)}
362 {Browse {Derivee evaluate($)}} % affiche 19
363
364 % QUESTION 8 -----
365 declare
366 class Sequence % not very efficient ! %ON PEUT METTRE UNE LISTE DANS LE
  INIT CONTRAIREMENT AU TEMPLATE
367   attr l
368   meth init(L) l := L end
369   meth isEmpty($) @l == nil end
370   meth first($) {List.nth @l 1} end
371   meth last($) {List.last @l} end
372   meth insertFirst(X) l := X|@l end
373   meth insertLast(X) l := {List.append @l X|nil} end
374   meth removeFirst l := {List.drop @l 1} end
375   meth removeLast
376     if {self isEmpty($)} then skip
377     else l := {List.take @l {List.length @l}-1} end
378   end
379 end
380
381 fun {Palindrome Xs}
382   S
383   fun {Check}
384     if {S isEmpty($)} then true

```

```

385         elseif {S first($)} == {S last($)} then
386             {S removeFirst}
387             {S removeLast}
388             {Check}
389         else false end
390     end
391 in
392     S = {New Sequence init(Xs)}
393     {Check}
394 end
395 {Browse {Palindrome "radar"}}
396 {Browse {Palindrome "abc"}}
397 {Browse {Palindrome "a"}}
398 {Browse {Palindrome "eluparcettecrapule"}}

```

## • TP8

```

1  % QUESTION 1 -----
2  declare A B C D
3  thread D = C+1 end
4  thread C = B+1 end
5  thread A = 1 end
6  thread B = A+1 end
7  {Browse D}
8
9  %DCABBrowse
10 %ABCD
11
12 % QUESTION 2 -----
13 %Partie A-----
14 local X Y Z in
15     thread if X==1 then Y=2 else Z=2 end end
16     thread if Y==1 then X=1 else Z=2 end end
17     X=1
18     {Browse X|Y|Z|nil} %1|2|2
19 end
20
21 %Partie B-----
22 local X Y Z in
23     thread if X==1 then Y=2 else Z=2 end end
24     thread if Y==1 then X=1 else Z=2 end end
25     X=2
26     {Browse X|Y|Z|nil} %2|_|2
27 end
28
29 % QUESTION 3 -----
30 declare
31 fun {ProduceInts N}
32     fun {ProduceIntsHelp I}
33         {Delay 1000}
34         if I==N then N|nil
35         else I|{ProduceIntsHelp I+1} end
36     end
37 in
38     {ProduceIntsHelp 1}
39 end
40 fun {Sum Xs}
41     fun {SumHelp L Acc}
42         {Delay 1000}
43         case L of nil then Acc
44         [] H|T then {SumHelp T Acc+H}
45         end
46     end
47 in
48     {SumHelp Xs 0}

```

```

49 end
50 declare Xs S
51 thread Xs = {ProduceInts 666} end
52 thread S = {Sum Xs} end
53 {Browse S}
54
55 declare Xs S
56 Xs = {ProduceInts 666}
57 S = {Sum Xs}
58 {Browse S}
59
60 % QUESTION 4 -----
61 declare
62 fun {Producer X}
63   local
64     fun {ProducerHelp N Lst}
65       if N > ~1 then
66         {ProducerHelp N-1 N|Lst}
67       else
68         Lst
69       end
70     end
71   in
72     {ProducerHelp X-1 nil}
73   end
74 end
75
76 fun {FilterOddNumber Lst}
77   local
78     fun {FilterOddNumberHelp Lst OddNumbers}
79       case Lst
80       of nil then {Reverse OddNumbers} %pour iter de faire trop d'
81         append
82         [] H|T then
83           if H mod 2 \= 0 then {FilterOddNumberHelp T H|OddNumbers}
84           else {FilterOddNumberHelp T OddNumbers}
85           end
86         end
87       in
88         {FilterOddNumberHelp Lst nil}
89       end
90   end
91
92 thread Xs = {Producer 10000} end
93 thread S = {FilterOddNumber Xs} end
94 {Browse S}
95
96 declare
97 fun {ServeBeer}
98   local
99     Beer = {OS.rand} mod 2
100   in
101     if Beer == 0 then 'trappist'
102     else 'pils'
103     end
104   end
105 end
106
107 fun {SmellTrappist Beer}
108   case Beer
109   of 'trappist' then true
110   else false
111   end
112 end

```

```

113
114 fun {Barman N}
115   if N == 0 then nil
116   else
117     {Delay 3000}
118     {ServeBeer}|{Barman N-1}
119   end
120 end
121
122 fun {Charlotte Lst}
123   local
124     fun {CharlotteHelp Lst A}
125       case Lst
126       of nil then A
127       [] H|T then
128         if {SmellTrappist H} then
129           {CharlotteHelp T A+1}
130         else
131           {CharlotteHelp T A}
132         end
133       end
134     end
135   in
136     {CharlotteHelp Lst 0}
137   end
138 end
139
140 Xs = {Barman 10}
141 {Browse Xs}
142 {Browse 'Charlotte'#{Charlotte Xs}}
143 {Browse 'Ami'#{Length Xs}-{Charlotte Xs}}
144
145 % QUESTION 5 -----
146 declare
147 fun {AddOrIncrementElement Xs Elem}
148   case Xs
149   of nil then
150     Elem#1|nil
151   [] H|T then
152     if H.1 == Elem then
153       H.1#H.2+1|T
154     else
155       H|{AddOrIncrementElement T Elem}
156     end
157   end
158 end
159
160 fun {Counter Xs} %fonctionne seulement pour les listes finies a cause du
  reverse
161   local LstReverse
162   fun {CounterHelp Xs Lst}
163     case Xs
164     of H|T then
165       if Lst == nil then
166         {CounterHelp T [H#1]|nil}
167       else
168         {CounterHelp T {AddOrIncrementElement Lst.1 H}|Lst}
169       end
170     else
171       LstReverse = {Reverse Lst}
172       LstReverse
173     end
174   end
175   in
176     thread {CounterHelp Xs nil} end

```

```

177     end
178 end
179
180 local InS in
181     {Browse {Counter InS}}
182     InS=a|b|a|c|_
183 end
184
185 %Solution
186
187 declare
188 fun {CounterSol S} %ne renvoie rien non plus.....
189     fun {Update C L}
190         case L of (H1#H2)|T then
191             if H1 == C then H1#(H2+1)|T
192             else (H1#H2)|{Update C T} end
193         else nil end
194     end
195     fun {CAlt S A}
196         case S of H|T then
197             A2 = {Update H A}
198             in
199                 A2|{CAlt T A2}
200             else nil
201             end
202         end
203     in
204         thread {CAlt S nil} end
205     end
206
207 local InS in
208     {Browse {Counter InS}}
209     InS=a|b|a|c|_
210 end

```

## • TP9

```

1  % QUESTION 3 -----
2  declare
3  fun {MakeBinaryGate F} % Complexit de {MakeBinaryGate F} est O(1) et
   ses fonctions O(n)
4      fun {$ S1 S2} % $ represente {MakeBinaryGate F}
5          fun {Gate SA SB}
6              case SA#SB
7              of (HA|TA)#(HB|TB) then
8                  {F HA HB}|{Gate TA TB}
9              else
10                 nil
11             end
12         end
13     in
14         thread {Gate S1 S2} end %bien mettre thread car stream
15     end
16 end
17
18 S1 = 0|0|1|1|1|_
19 S2 = 0|1|0|1|1|_
20 And = {MakeBinaryGate fun {$ A B} A*B end}
21 Or = {MakeBinaryGate fun {$ A B} A+B-A*B end}
22 Xor = {MakeBinaryGate fun {$ A B} A+B-2*A*B end}
23 Nor = {MakeBinaryGate fun {$ A B} 1-(A+B-A*B) end}
24 {Browse 'and'#{And S1 S2}}
25 {Browse 'or'#{Or S1 S2}}
26 {Browse 'xor'#{Xor S1 S2}}
27 {Browse 'nor'#{Nor S1 S2}}

```

```

28
29 % QUESTION 4 -----
30 local R=1|1|1|0|_ S=0|1|0|0|_ Q NotQ
31   fun {DelayG S}
32     0|S
33   end
34   NorG = {MakeBinaryGate fun {$ A B} 1-(A+B-A*B) end}
35   proc {Bascule Rs Ss Qs NotQs}
36     DelayedQs = {DelayG Qs}
37     DelayedNotQs = {DelayG NotQs}
38   in
39     Qs = {NorG Rs DelayedNotQs} %Si on n'utilise pas de Delay les 2
      fonctions ne se lancent pas pcq elles d dependent l'une de l'
      autre et que Qs et NotQs sont vides
40     NotQs = {NorG Ss DelayedQs}
41   end
42 in
43   {Bascule R S Q NotQ}
44   {Browse Q#NotQ}
45 end
46
47 local A B C D in
48   thread D = C+1 end
49   thread C = B+1 end
50   thread {Delay 5000} A = 1 end
51   thread B = A+1 end
52   {Browse D}
53 end
54
55 % QUESTION 5 -----
56 declare
57 proc {ForCollect Xs P Ys}
58   N = {NewCell 1}
59   Acc={NewCell Ys}
60   proc{C X} R2 in
61     @Acc=X|R2
62     {Browse @N#R2}
63     Acc:=R2
64     {Browse r22#R2}
65     {Cell.assign N @N+1}
66   end
67 in
68   for X in Xs do
69     {P C X}
70   end
71   @Acc = nil
72 end
73 {Browse {ForCollect [0 2 4 6 8]
74   proc{$ Collect X} {Collect X div 2} end}} %Affiche [0 1 2 3 4]
75
76 %proc {ForCollectDecl Xs Proc Ys}

```

## • TP10

```

1 % QUESTION 1 -----
2 declare
3 P S
4 {NewPort S P}
5 {Send P foo}
6 {Send P bar}
7
8 {Browse S}
9
10 proc {PrintMsg S}
11   case S

```

```

12     of H|T then {Browse H} {PrintMsg T}
13     else skip
14     end
15 end
16
17 {PrintMsg S}
18
19 % QUESTION 2 -----
20 declare
21 fun {Pow N E}
22     local
23         fun {PowHelp E Acc}
24             if E > 0 then {PowHelp E-1 Acc*N}
25             else Acc
26             end
27         end
28     in
29         {PowHelp E 1}
30     end
31 end
32
33 fun {LaunchServer}
34     local S P
35     proc {ParseStream S}
36         case S
37         of H|T then
38             case H
39             of add(X Y R) then R = X+Y {ParseStream T}
40             [] pow(X Y R) then R = {Pow X Y} {ParseStream T}
41             [] 'div'(X Y R) then if Y \= 0 then R = X div Y else R = '
42                 division par 0 impossible' end {ParseStream T}
43             else
44                 {Browse 'je ne comprends pas ton message'} {ParseStream T}
45             }
46         end
47     else skip
48     end
49 in
50     P = {NewPort S}
51     thread {ParseStream S} end
52     P
53 end
54
55 A B N S Res1 Res2 Res3 Res4 Res5 Res6
56 S = {LaunchServer}
57 {Send S add(321 345 Res1)}
58 {Browse Res1}
59 {Send S pow(2 N Res2)}
60 N = 8
61 {Browse Res2}
62 {Send S add(A B Res3)}
63 {Send S add(10 20 Res4)}
64 {Send S foo}
65 {Browse Res4}
66 A = 3
67 B = 0-A
68 {Send S 'div'(90 Res3 Res5)}
69 {Send S 'div'(90 Res4 Res6)}
70 {Browse Res3}
71 {Browse Res5}
72 {Browse Res6}
73
74 % QUESTION 3 -----

```

```

75 declare
76 fun {StudentRMI}
77   S in
78     thread
79       for ask(howmany:Beers) in S do
80         Beers={OS.rand} mod 24
81       end
82     end
83     {NewPort S}
84 end
85
86 fun {StudentCallBack}
87   S in
88     thread
89       for ask(howmany:P) in S do
90         {Send P {OS.rand} mod 24}
91       end
92     end
93     {NewPort S}
94 end
95
96 fun {Student}
97   local N in
98     N = {OS.rand} mod 2
99     if N == 0 then 'StudentRMI'
100     else 'StudentCallBack'
101     end
102   end
103 end
104
105 fun {CreateUniversity Size}
106   fun {CreateLoop I}
107     if I <= Size then
108       % pour {Student} choisissez soit StudentRMI ou StudentCallBack,
109       % dfini plus haut, selon l'humeur de Charlotte
110       {Student}|{CreateLoop I+1}
111     else nil end
112   end
113 in
114   {CreateLoop 1}
115 end
116
117 proc {Charlotte}
118   local Students TotalBeers
119   proc {P L}
120     {Browse L}
121   end
122 in
123   Students = {CreateUniversity 10}
124   {P Students}
125 end
126
127
128 {Charlotte}
129
130 %pas compris et pas r solu correctement...
131
132 % QUESTION 4 -----
133 declare
134 fun {MakePortier}
135   local P S
136   proc {PortierHelp S Tot}
137     case S
138     of H|T then
139       case H

```



```

140         of getIn(N) then {PortierHelp T Tot+N}
141         [] getOut(N) then {PortierHelp T Tot-N}
142         [] getCount(N) then N = Tot {PortierHelp T Tot}
143         else {Browse 'commande non valide'}
144         end
145     else skip
146     end
147 end
148 in
149     P = {NewPort S}
150     thread {PortierHelp S 0} end
151     P
152 end
153 end
154
155 Portier = {MakePortier}
156 {Send Portier getIn(10)}
157 {Browse {Send Portier getCount($)}}
158 {Send Portier getOut(5)}
159 {Browse {Send Portier getCount($)}}
160
161 % QUESTION 5 -----
162 declare
163 fun {NewStack} %/!\ comme rien n'est en local il faut avoir des noms de
    variables diffrents
164     Stream
165     proc {ParseStack S Stack}
166         case S %liste d'instructions stream alors que stack est une liste
167         of push(X)|T then {ParseStack T X|Stack}
168         [] pop(X)|T then X = Stack.1 {ParseStack T Stack.2}
169         [] isEmpty(X)|T then X = (Stack==nil) {ParseStack T Stack}
170         []_|T then {ParseStack T Stack}
171         else {Browse 'can do that'}
172         end
173     end
174 in
175     thread {ParseStack Stream nil} end
176     {NewPort Stream}
177 end
178 proc {Push Port X}
179     {Send Port push(X)}
180 end
181 fun {Pop Port}
182     {Send Port pop($)}
183 end
184 fun {IsEmpty Port}
185     {Send Port isEmpty($)}
186 end
187
188 MyStack = {NewStack}
189 {Push MyStack 'banane'}
190 {Browse {Pop MyStack}}
191 {Browse {IsEmpty MyStack}}
192 {Push MyStack 'kiwi'}
193 {Push MyStack 'fraise'}
194 {Browse {IsEmpty MyStack}}
195
196
197 % QUESTION 7 -----
198 declare
199 fun {Counter Output}
200     S
201     fun {UpdateStream Elem State}
202         case State
203         of nil then {Append State Elem#1|nil}

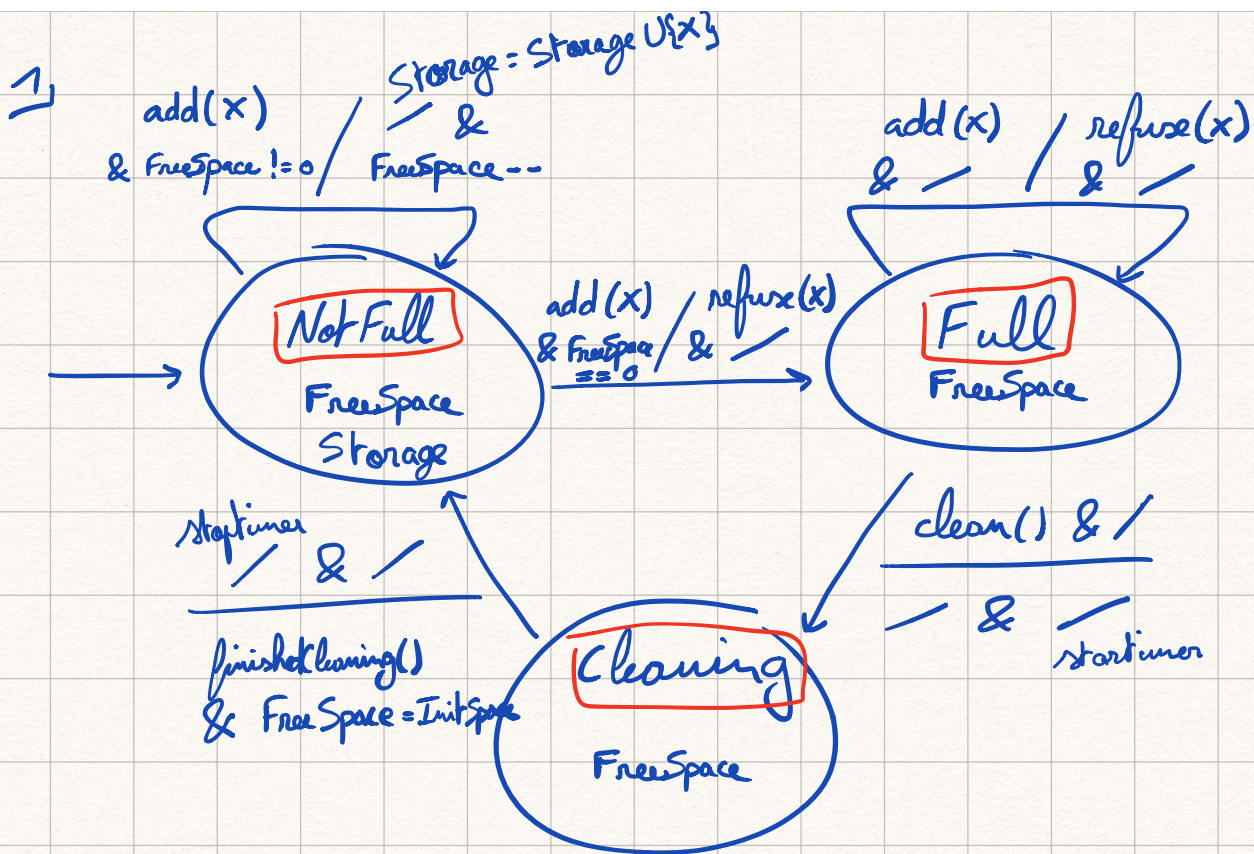
```

```

204     [] H|T then
205         if Elem == H.1 then
206             H.1#H.2+1|T
207         else
208             H|{UpdateStream Elem T}
209         end
210     end
211 end
212 proc {TreatStream S State Output}
213     case S
214     of H|T then
215         local NewState NewOutput in
216             NewState = {UpdateStream H State}
217             Output = NewState|NewOutput
218             {TreatStream T NewState NewOutput}
219         end
220     [] _|T then {TreatStream T State Output}
221     else skip
222     end
223 end
224 in
225     thread {TreatStream S nil Output} end
226     {NewPort S}
227 end
228 Out
229 C = {Counter Out}
230 {Browse out#Out}
231 {Delay 512}
232 {Send C a}
233 {Delay 512}
234 {Send C a}
235 {Delay 512}
236 {Send C b}
237 {Delay 512}
238 {Send C c}
239 {Delay 512}
240 {Send C a}
241 {Delay 512}
242 {Send C b}

```

## . TP11



2) 1 seul controller pour tous les ascenseurs donc 1 seul ascenseur en mut à la fois.

3) → Cfr. code

Impact? vitesse de traitement des requêtes par ascenseurs car ils "bloqueraient" → sinon RIEN d'autre

Diagramme? plus besoin du controller!

```

1  % Code Lift -----
2  declare
3  proc {NewPortObject Init Fun ?P}
4      proc {MsgLoop S1 State}
5
6          case S1 of Msg|S2 then
7              {MsgLoop S2 {Fun Msg State}}
8              [] nil then skip end
9          end
10         Sin
11     in
12         thread {MsgLoop Sin Init} end
13         {NewPort Sin P}
14     end
15 proc {NewPortObject2 Proc ?P}
16     Sin in
17         thread for Msg in Sin do {Proc Msg} end end
18         {NewPort Sin P}
19     end
20 proc {Controller ?Cid}
21     {NewPortObject2
22         proc {$ Msg}
23             case Msg
24             of step(Lid Pos Dest) then
25                 if Pos<Dest then
26                     {Delay 1000} {Send Lid 'at'(Pos+1)}
27                 elseif Pos>Dest then
28                     {Delay 1000} {Send Lid 'at'(Pos-1)}
29                 end
30             end
31         end Cid}
32     end
33 proc {Floor Num Init Lifts ?Fid}
34     {NewPortObject Init
35         fun {$ Msg state(Called)}
36             case Msg
37             of call then
38                 {Browse 'Floor '#Num#' calls a lift!'}
39                 if {Not Called} then Lran in
40                     Lran=Lifts.(1+{OS.rand} mod {Width Lifts})
41                     {Send Lran call(Num)}
42                 end
43                 state(true)
44             [] arrive(Ack) then
45                 {Browse 'Lift at floor '#Num#': open doors'}
46                 {Delay 2000}
47                 {Browse 'Lift at floor '#Num#': close doors'}
48                 Ack=unit
49                 state(false)
50             end
51         end Fid}
52     end
53 proc {Lift Num Init Cid Floors ?Lid}
54     {NewPortObject Init
55         fun {$ Msg state(Pos Sched Moving)}
56             case Msg
57             of call(N) then
58                 {Browse 'Lift '#Num#' needed at floor '#N'}
59                 if N==Pos andthen {Not Moving} then
60                     {Wait {Send Floors.Pos arrive($)}}
61                     state(Pos Sched false)
62                 else Sched2 in
63                     Sched2={Append Sched [N]}
64                     if {Not Moving} then

```

```

65         {Send Cid step(Lid Pos Sched2.1)} end
66         state(Pos Sched2 true)
67     end
68     [] 'at'(NewPos) then
69         {Browse 'Lift '#Num# at floor '#NewPos}
70         case Sched
71         of nil then
72             state(NewPos Sched Moving)
73         [] S|Sched2 then
74             if NewPos==S then
75                 {Wait {Send Floors.S arrive($)}}
76                 if Sched2==nil then
77                     state(NewPos nil false)
78                 else
79                     {Send Cid step(Lid NewPos Sched2.1)}
80                     state(NewPos Sched2 true)
81                 end
82             else
83                 {Send Cid step(Lid NewPos S)}
84                 state(NewPos Sched true)
85             end
86         end
87     end
88     end Lid}
89 end
90 proc {Building FN LN ?Floors ?Lifts}
91     Lifts={MakeTuple lifts LN}
92     for I in 1..LN do C in
93         {Controller C}
94         Lifts.I={Lift I state(1 nil false) C Floors}
95     end
96     Floors={MakeTuple floors FN}
97     for I in 1..FN do
98         Floors.I={Floor I state(false) Lifts}
99     end
100 end
101
102 % QUESTION 4 -----
103 declare
104 proc {NewPortObject Init Fun ?P}
105     proc {MsgLoop S1 State}
106         case S1 of Msg|S2 then
107             {MsgLoop S2 {Fun Msg State}}
108         [] nil then skip end
109         end
110     Sin
111 in
112     thread {MsgLoop Sin Init} end
113     {NewPort Sin P}
114 end
115
116 proc {NewPortObject2 Proc ?P}
117     Sin in
118     thread
119         for Msg in Sin do
120             {Proc Msg}
121         end
122     end
123     {NewPort Sin P}
124 end
125
126 proc {Floor Num Init Lifts ?Fid}
127     {NewPortObject Init
128     fun {$ Msg state(Called)}
129         case Msg

```

```

130         of call then
131             {Browse 'Floor '#Num#' calls a lift!'}
132             if {Not Called} then
133                 Lran in
134                 Lran=Lifts.(1+{OS.rand} mod {Width Lifts})
135                 {Send Lran call(Num)}
136             end
137             state(true)
138         [] arrive(Ack) then
139             {Browse 'Lift at floor '#Num#': open doors'}
140             {Delay 2000}
141             {Browse 'Lift at floor '#Num#': close doors'}
142             Ack=unit
143             state(false)
144         end
145     end
146     Fid}
147 end
148 proc {Lift Num Init Cid Floors ?Lid}
149     fun {Controller Dest state(Pos Sched)}
150         NewPos in
151         if Pos<Dest then
152             {Delay 1000} NewPos = Pos+1
153         elseif Pos>Dest then
154             {Delay 1000} NewPos = Pos-1
155         end
156         state(NewPos Sched)
157     end
158     fun {Scheduler state(Pos Sched)}
159         {Browse 'Lift '#Num#' at floor '#Pos'}
160         case Sched
161         of S|Sched2 then
162             if Pos==S then
163                 {Wait {Send Floors.S arrive($)}}
164                 if Sched2==nil then
165                     state(Pos nil)
166                 else
167                     {Scheduler {Controller Sched2.1
168                                 state(Pos Sched2)}}
169                 end
170             else
171                 {Scheduler {Controller S state(Pos Sched
172                             )}}
173             end
174         else
175             state(Pos Sched)
176         end
177     end
178 in
179     {NewPortObject Init
180     fun {$ Msg state(Pos Sched)}
181         case Msg
182         of call(N) then
183             {Browse 'Lift '#Num#' needed at floor '#N'}
184             if N==Pos andthen {Not Moving} then
185                 {Wait {Send Floors.Pos arrive($)}}
186                 state(Pos Sched)
187             else
188                 NewSched NewState in
189                 NewSched = {Append Sched [N]}
190                 {Scheduler state(Pos NewSched)}
191             end
192         end
193     end
194     Lid}

```

```

193 end
194 proc {Building FN LN ?Floors ?Lifts}
195     Lifts={MakeTuple lifts LN}
196     for I in 1..LN do C in
197         {Controller C}
198         Lifts.I={Lift I state(1 nil false) C Floors}
199     end
200     Floors={MakeTuple floors FN}
201     for I in 1..FN do
202         Floors.I={Floor I state(false) Lifts}
203     end
204 end
205
206 % QUESTION 5 -----
207 declare
208 proc {NewPortObject Init Fun ?P}
209     proc {MsgLoop S1 State}
210         case S1 of Msg|S2 then
211             {MsgLoop S2 {Fun Msg State}}
212         [] nil then skip end
213         end
214         Sin
215 in
216     thread {MsgLoop Sin Init} end
217     {NewPort Sin P}
218 end
219
220 proc {NewPortObject2 Proc ?P}
221     Sin in
222     thread
223         for Msg in Sin do
224             {Proc Msg}
225         end
226     end
227     {NewPort Sin P}
228 end
229
230 proc {Controller ?Cid}
231     {NewPortObject2
232     proc {$ Msg}
233         case Msg
234         of step(Lid Pos Dest) then
235             if Pos<Dest then
236                 {Delay 1000} {Send Lid 'at'(Pos+1)}
237             elseif Pos>Dest then
238                 {Delay 1000} {Send Lid 'at'(Pos-1)}
239             end
240         end
241     end
242     Cid}
243 end
244
245 proc {Floor Num Init Lifts ?Fid}
246     {NewPortObject Init
247     fun {$ Msg state(Called)}
248         case Msg
249         of call then
250             {Browse 'Floor '#Num#' calls a lift!'}
251             if {Not Called} then
252                 Lran in
253                 Lran=Lifts.(1+{OS.rand} mod {Width Lifts})
254                 {Send Lran call(Num)}
255             end
256             state(true)
257         [] arrive(Ack) then

```

```

258         {Browse 'Lift at floor '#Num#': open doors'}
259         {Delay 2000}
260         {Browse 'Lift at floor '#Num#': close doors'}
261         Ack=unit
262         state(false)
263     end
264 end
265     Fid}
266 end
267 proc {Lift Num Init Cid Floors ?Lid}
268     fun {Scheduler state(NewPos Scheduled Moving)}
269         fun {ChooseBestDest Scheduled Best}
270             case Scheduled
271             of H|T then
272                 if {Abs NewPos-H} < Best then {
273                     ChooseBestDest T H}
274                 else {ChooseBestDest T Best} end
275             end
276         in
277             case Scheduled
278             of Dest|Tail then
279                 if Dest==NewPos then
280                     {Wait {Send Floors.Dest arrive($
281                         )}}
282                     if Tail==nil then
283                         state(NewPos nil false)
284                     else
285                         {Send Cid step(Lid
286                             NewPos Tail.1)}
287                         state(NewPos Tail true)
288                     end
289                 else
290                     BestDest = {ChooseBestDest
291                         Scheduled Dest} in
292                     {Send Cid step(Lid NewPos
293                         BestDest)}
294                     state(NewPos Scheduled true)
295                 end
296             else state(Pos Scheduled Moving) end
297         end
298     in
299         {NewPortObject Init
300         fun {$ Msg state(Pos Sched Moving)}
301             case Msg
302             of call(N) then
303                 {Browse 'Lift '#Num#' needed at floor '#N}
304                 if N==Pos andthen {Not Moving} then
305                     {Wait {Send Floors.Pos arrive($)}}
306                     state(Pos Sched false)
307                 else
308                     Sched2 in
309                     Sched2={Append Sched [N]}
310                     if {Not Moving} then
311                         {Send Cid step(Lid Pos Sched2.1)
312                         }
313                     end
314                     state(Pos Sched2 true)
315                 end
316                 [] 'at'(NewPos) then
317                     {Browse 'Lift '#Num#' at floor '#NewPos}
318                     case Sched
319                     of nil then
320                         state(NewPos Sched Moving)
321                     [] S|Sched2 then

```



```

317         if NewPos==S then
318             {Wait {Send Floors.S arrive($)}}
319             if Sched2==nil then
320                 state(NewPos nil false)
321             else
322                 {Send Cid step(Lid
323                     NewPos Sched2.1)}
324                 state(NewPos Sched2 true
325                     )
326             end
327         else
328             {Send Cid step(Lid NewPos S)}
329             state(NewPos Sched true)
330         end
331     end
332     Lid}
333 end
334 proc {Building FN LN ?Floors ?Lifts}
335     Lifts={MakeTuple lifts LN}
336     for I in 1..LN do C in
337         {Controller C}
338         Lifts.I={Lift I state(1 nil false) C Floors}
339     end
340     Floors={MakeTuple floors FN}
341     for I in 1..FN do
342         Floors.I={Floor I state(false) Lifts}
343     end
344 end

```

## • TP12

```

1  % Question 1 -----
2  -module(wrapper).
3  -export([hello_world/0, my_add/2, my_pow/2, my_fib/1, start/1, start/0,
4      single/1, with_args/2]).
5
6  hello_world() ->
7      io:fwrite("Hello, World!\n").
8
9  my_add(A, B) ->  A+B.
10
11 my_pow(Base, Exp) when Exp >= 0 -> my_pow(Base, Exp, 1).
12 my_pow(_, 0, Res) -> Res;
13 my_pow(Base, Exp, Res) -> my_pow(Base, Exp-1, Res*Base).
14
15 my_fib(Nb) -> my_fib(0, 1, Nb).
16 my_fib(Res,_,0) -> Res;
17 my_fib(Res,Fut,Nb) -> my_fib(Fut, Res+Fut, Nb-1).
18
19 single(hello_world) -> hello_world();
20 single(_) ->'unk'.
21
22 with_args(Fun, _Args) ->
23     Valid_Members = [my_add, my_pow, my_fib],
24     case lists:member(Fun, Valid_Members) of
25         true -> apply(hello, Fun, _Args);
26         false ->'unk'
27     end.
28
29 eval([]) ->'unk';
30 eval([ _Head ]) -> single(_Head);
31 eval([ _Head | _Tail ]) ->

```

```

31     Args = lists:map(fun(Y) -> list_to_integer(atom_to_list(Y)) end,
32         _Tail),
33     with_args(_Head, Args).
34
35 start() -> io:fwrite("unk~n").
36 start(Lst) ->
37     io:fwrite("~p~n", [eval(Lst)]).
38
39 % Question 2 -----
40 -module(auto).
41 -export([voiture/0, voiture/1, start/0, run_voiture/0]).
42
43 voiture() -> voiture(idle).
44 voiture(State) ->
45     receive
46     start when State == idle ->
47         io:format("Engine is started~n"),
48         voiture(runing);
49     move when State == runing ->
50         io:format("I'm moving~n"),
51         voiture(runing);
52     stop when State == runing ->
53         io:format("Engine is down~n"),
54         voiture(idle);
55     _ ->
56         io:format("Aaah noooo! Kernel Panic~n"),
57         voiture(State)
58     end.
59
60 start() ->
61     V = spawn(?MODULE, voiture, []),
62     V ! start.
63
64 run_voiture() ->
65     Voiture = spawn(?MODULE, voiture, []),% ou spawn(?MODULE, voiture, [
66         idle])
67     Voiture ! move,% illégal, pas démarrée
68     Voiture ! start,% ok
69     Voiture ! move,% ok
70     Voiture ! start,% illégal, déjà démarrée
71     Voiture ! move,% ok
72     Voiture ! stop,% ok
73     Voiture ! stop.% illégal, déjà stoppée
74
75 % Question 3 -----
76 -module(msg_between_agents).
77 -export([pow_main/0, sum_pow/4, sum_pow/3, my_pow/2, my_pow/3, my_pow/0]).
78
79 my_pow(Base, Exp) -> my_pow(Base, Exp, 1).
80 my_pow(_, 0, Res) -> Res;
81 my_pow(Base, Exp, Res) -> my_pow(Base, Exp-1, Res*Base).
82 my_pow() ->
83     receive
84     {Pid, Base, Exp} -> Pid ! {self(), my_pow(Base, Exp)}
85     end,
86     my_pow().
87
88 sum_pow(_, From, To, Res) when From > To -> Res;
89 sum_pow(Pid, From, To, Res) ->
90     Pid ! {self(), From, 2},
91     receive
92     {Pid, Comp} -> sum_pow(Pid, From+1, To, Res+Comp)
93     end.

```

```

94 sum_pow(PowPid, From, To) -> sum_pow(PowPid, From, To, 0).
95
96 pow_main() ->
97     PowPid = spawn(?MODULE, my_pow, []),
98     sum_pow(PowPid, 1, 100).
99
100 % Question 4 -----
101 -module(charlotte).
102 -export([student/0, beer_manager/1,start/0]).
103
104 % Mise à jour de l'état interne du manager
105 update_stats({Nb, Min, Max, NbStud}, NbBeer) ->
106     { Nb + NbBeer,
107       if Min == -1 -> NbBeer;
108         Min > NbBeer -> NbBeer;
109         true -> Min
110     },
111     if Max < NbBeer -> NbBeer;
112       true -> Max
113     },
114     NbStud + 1}.
115
116 % Génère les statistiques pour un message "stats"
117 do_stats({Nb, Min, Max, NbStud}) ->
118     {Nb,
119      (Nb * 1.0) / NbStud,
120      Min, Max, NbStud}.
121
122 beer_manager({Nb, Min, Max, NbStud}) ->
123     %TODO.
124     receive
125         {From, stats} ->
126             From ! {self(), do_stats({Nb,Min,Max,NbStud})},
127             beer_manager({Nb, Min, Max, NbStud});
128         {From, BeerNb} ->
129             From ! {self(), ok},
130             beer_manager(update_stats({Nb,Min,Max,NbStud}, BeerNb));
131         shutdown -> ok
132     end.
133
134 charlotte([], BeerManager) ->
135     %TODO;
136     BeerManager ! {self(), stats},
137     receive
138         {BeerManager, Stats} -> Stats;
139         _-> "error"
140     end;
141
142 charlotte([Stud | StudQueue], BeerManager) ->
143     %TODO.
144     Stud ! {self(),how_much_beers}, %demande à l'étudiant combien il a
145     bu de bières
146     receive
147         {Stud,-1} -> charlotte(StudQueue,BeerManager);
148         {Stud, Nb} when Nb >= 0 ->
149             BeerManager ! {self(), Nb},
150             receive
151                 {BeerManager,ok} -> charlotte(StudQueue,BeerManager); %
152                 attends que le BeerManager ait mis les infos à jour
153                 _-> "error"
154             end;
155         _-> "error"
156     end.
157
158 student() ->%TODOrand:uniform(N) génère un entier aléatoire dans l'

```

```

157     intervalle [1; N].
158     receive
159         {From, how_much_beers} -> %récupère la requête envoyée par
160             charlotte()
161             From ! {self(), rand:uniform(30)}; %répond à la requête
162         {From, _} ->
163             From ! {self(), -1};
164         _-> no
165     end.
166
167 build_students(Tot) ->
168     [spawn(?MODULE, student, []) || _ <- lists:seq(1, Tot)].
169
170 start() ->
171     BM = spawn(?MODULE, beer_manager, [{0,-1,0,0}]), %crée le
172     beerManager
173     Students = build_students(98956), %génère le flot d'étudiants
174     Result = charlotte(Students, BM), %lance l'analyse de charlotte
175     BM ! shutdown,
176     case Result of
177         {Nb, Avg, Min,Max, NbStuds} ->
178             io:format("Total Beers: ~p~nAverage: ~p~nMin: "
179                 "~p~nMax: ~p~nStudents: ~p~n",
180                 [Nb, Avg,Min,Max,NbStuds]),
181             Result;
182         _ -> "error"
183     end.
184
185 % Question 5 -----
186 -module(processus).
187 -export([start/0, divisor_checker/1]).
188
189 busy_wait(Atom) ->
190     case whereis(Atom) of
191         undefined ->
192             timer:sleep(500),
193             busy_wait(Atom);
194         _ -> ok
195     end.
196
197 divisor_request(Nb, Divisor) ->
198     process_flag(trap_exit, true),% little hack
199     Ref = make_ref(),
200
201     ok=busy_wait(div_check),% little hack
202
203     div_check ! {self(), Ref, Nb, Divisor},
204     receive
205         {'EXIT', Pid, Reason} -> divisor_request(Nb, Divisor);
206         {Ref, Response} -> Response
207     after2000 ->
208         timeout
209     end.
210
211 random_kill(P) ->
212     caserand:uniform() < P of
213         true -> exit(killed);
214         _ -> ok
215     end.
216
217 divisor_checker(KillProb) when KillProb < 1.0 ->
218     receive
219         {Pid, Ref, Nb, Divisor} ->
220             link(Pid),% little hack
221             Res = case Nb rem Divisor of

```

```

219         0 -> true;
220         _ -> false
221     end,
222     Pid ! {Ref, Res}
223 end,
224 ok = random_kill(KillProb),
225 divisor_checker(KillProb).
226
227 start_divisor_checker(KillProb) ->
228     spawn(?MODULE, restarter, [KillProb]).
229
230 restarter(P) ->
231     process_flag(trap_exit, true),
232     Pid = spawn_link(?MODULE, divisor_checker, [P]),
233     register(div_check, Pid),
234     receive
235         {'EXIT', Pid, normal} -> % divisor_checker has terminated
236             correctly
237             ok;
238         {'EXIT', Pid, shutdown} -> % not a crash, manual temrination
239             ok;
240         {'EXIT', Pid, _} -> % this is a crash
241             restarter(P)
242     end.
243
244 start() ->
245     Overlay_Pid = start_divisor_checker(0.35),
246     % Checks the divisor of two from 1 to 10
247     Lst = [ divisor_request(X, 2) || X <- lists:seq(1,10)],
248     io:format("~p~n", [Lst]),
249     exit(Overlay_Pid,kill).
250
251 % Question 6 -----
252 -module(prime_number).
253 -export([start/1, worker_response/3]).
254
255 random_kill(P) ->
256     case
257         rand:uniform() < P of
258             true -> exit(killed);
259             _ -> ok
260         end.
261
262 is_prime(P) when P =< 1 -> false;
263 is_prime(P) -> is_prime(P, P-1).
264 is_prime(_, Curr_Divisor) when Curr_Divisor =< 1 -> true;
265 is_prime(P, Curr_Divisor) ->
266     case P rem Curr_Divisor of
267         0 -> false;
268         _ -> is_prime(P, Curr_Divisor-1)
269     end.
270
271 prime_worker(PKill, From, To, PidMain, Ref) ->
272     Lst = prime_range(PKill, From, To),
273     PidMain ! {Ref, Lst}.
274
275 prime_range(PKill, From, To) -> prime_range(PKill, From, To, []).
276 prime_range(_, To, To, Lst_Res) -> lists:reverse(Lst_Res);
277 prime_range(PKill, From, To, Lst_Res) ->
278     A = is_prime(From),
279     ok = random_kill(PKill),
280     case A of
281         true ->
282             prime_range(PKill, From+1, To, [From | Lst_Res]);
283         _ ->

```

```

283         prime_range(PKill, From+1, To, Lst_Res)
284     end.
285
286 on_exit(Pid, Fun) ->
287     spawn(fun() ->
288         Ref = monitor(process, Pid),
289         receive
290             {'DOWN', Ref, process, Pid, normal} -> ok;
291             {'DOWN', Ref, process, Pid, Why} ->
292                 Fun(Why)
293         end
294     end).
295
296 keep_alive(Fun) ->
297     process_flag(trap_exit, true),
298     Pid = spawn_link(Fun),
299     on_exit(Pid, fun(Why) ->
300         io:format("Process Died: ~p, restarting~n", [Why]),
301         keep_alive(Fun)end),
302     Pid.
303
304 splice(Len, Chunk) -> splice(Len, Chunk, 0, [], 1).
305 splice(_, M, M, TupList, _) -> lists:reverse(TupList);
306 splice(N, M, K, TupList, B) ->
307     Q = (N - K) div M,
308     R = (N - K) rem M,
309     A = B,
310     BB = B + Q + case R of
311         0 -> 0;
312         _ -> 1
313     end,
314     splice(N, M, K+1, [ {A, BB} | TupList ], BB).
315
316 get_prime(PKill, NBProcess, N, Pid, Ref) ->
317     [keep_alive(fun() -> prime_worker(PKill, X, Y, Pid, Ref)end) ||
318     {X, Y} <- splice(N+1, NBProcess)].
319
320 worker_response(Still_Alive, PidFwd, Ref) ->
321     Lst = worker_response__(Ref, [], Still_Alive),
322     PidFwd ! {self(), Ref, Lst}.
323 worker_response__(_, Lst, 0) -> Lst;
324 worker_response__(Ref, Lst, Still_Alive) ->
325     receive
326         {Ref, A} when is_list(A) -> worker_response__(Ref, Lst++A,
327             Still_Alive - 1);
328         _ -> worker_response__(Ref, Lst, Still_Alive)
329     end.
330
331 start(Workers) ->
332     Ref = make_ref(),
333     Pid = spawn(?MODULE, worker_response, [Workers, self(), Ref]),
334     get_prime(0.0001, Workers, 100000, Pid, Ref),
335     receive
336         {Pid, Ref, Lst} -> io:format("~p~n", [Lst])
337     end.

```