LINFO1104 : Solution des codes des TP

• TP2

```
% QUESTION 1 ------------------------------------------------------------
%[a b c] a|b|c|nil a|(b|(c|nil))

declare L1 L2 L3 L4 L5 L6 L7
L1 = a|nil
L2 = a|(b|c|nil)|d|nil
L3 = proc{$}{Browse oui}end|proc{$}{Browse non}end|nil
L4 = est|une|liste|nil
L5 = (a|p|nil)|nil
L6 = ceci|L4
L7 = L2.2 %1 correspond   Head et 2   Tail

{Browse L1}
{Browse L2}
{Browse L3}
{Browse L4}
{Browse L5}
{Browse L6}
{Browse L7}

declare
fun {Head L}
   case L of H|T then H
   [] nil then nil
   end
end

fun {Tail L}
   case L of H|T then T
   [] nil then nil
   end
end

{Browse {Head a|b|c|nil}}
{Browse {Tail a|b|c|nil}}

%QUESTION 2 ------------------------------------------------------------
declare
fun {Length L}
   fun {Length2 L N}
      case L of H|T then {Length2 T N+1}
      [] nil then N
      end
   end
in
   {Length2 L 0}
end

{Browse {Length l|o|i|s|nil}}
{Browse {Length [[b o r]i s]}}

%QUESTION 3 ------------------------------------------------------------
declare
fun {Append L1 L2}
   case L1 of H|T then H|{Append T L2}
   [] nil then L2
   end
end

{Browse {Append [r a] [p h]}}
```

```
{Browse {Append [b [o r]] [i s]}}

%QUESTION 4 ------------------------------------------------------------
declare
fun {F Arg}
   case Arg of H|T then 'nonEmpty'
   [] nil then 'empty'
   else 'other'
   end
end

{Browse {F 2}}
{Browse {F a|b|c|nil}}
{Browse {F nil}}

%QUESTION 5 ------------------------------------------------------------
declare
fun {Take Xs N}
   case Xs of H|T then
      if N>0 then H|{Take T N-1}
      else nil
      end
   [] nil then nil
   end
end

fun {Drop Xs N}
   case Xs of H|T then
      if N>0 then {Drop T N-1}
      else Xs
      end
   [] nil then nil
   end
end

{Browse {Take [r a p h] 2}}
{Browse {Take [r a p h] 7}}
{Browse {Drop [r a p h] 2}}
{Browse {Drop [r a p h] 7}}

%QUESTION 6 ------------------------------------------------------------
declare
fun {MultList L}
   fun {MultL L Tot}
      case L of H|T then {MultL T Tot*H}
      [] nil then Tot
      end
   end
in
   {MultL L 1}
end

{Browse {MultList [1 2 3 4]}}

%QUESTION 8 ------------------------------------------------------------
declare
fun {Prefix L1 L2}
   case L1 of H1|T1 then
      case L2 of H2|T2 then
         if H1 == H2 then {Prefix T1 T2}
         else false
         end
      [] nil then false
      end
   [] nil then true
```

```
   end
end
fun {FindString S T}
   local IndexMax FindStringHelp
      IndexMax = {Length T}-{Length S}+1
      FindStringHelp =
      fun{$ L1 L2 I}
         if I =< IndexMax then   % !!!!!! =< et PAS <=
            if {Prefix L1 L2} then
               I|{FindStringHelp L1 L2.2 I+1}
            else {FindStringHelp L1 L2.2 I+1}
            end
         else nil
         end
      end
   in
      {FindStringHelp S T 1}
   end
end

{Browse {Prefix [1 2 1] [1 2 3 4]}}
{Browse {Prefix [1 2 3] [1 2 3 4]}}
{Browse {FindString [a b a b] [a b a b a b]}}

%QUESTION 9 ------------------------------------------------------------
declare
fun {GetPrice C N1 N2 N3}
   N1 * C.1.prix + N2 * C.2.prix + N3 * C.3.prix
end

local
   Carte = carte(menu(entree: 'salade verte aux lardons'
                      plat: 'steak frites'
                      prix: 10)
                 menu(entree: 'salade de crevettes grises'
                      plat: 'saumon fume et pommes de terre'
                      prix: 12)
                 menu(plat: 'choucroute garnie'
                      prix: 9))
in
   {Browse Carte.2}
   {Browse Carte.2.plat}
   {Browse Carte.1.entree}
   {Browse {GetPrice Carte 8 11 6}}
end

%QUESTION 10 -----------------------------------------------------------
declare
fun {Promenade Bt}   %FAUX
   if Bt.left == empty andthen Bt.right == empty then nil
   elseif Bt.left == empty then Bt.1|{Promenade Bt.right}
   elseif Bt.right == empty then Bt.1|{Promenade Bt.left}
   else Bt.1|{Promenade Bt.left}|{Promenade Bt.right}
   end
end
fun {PromenadeSoluce Bt}
   local
      fun {PromenadeHelp Bt Acc}
         case Bt of btree(V left:TL right:TR) then
            V|{PromenadeHelp TL {PromenadeHelp TR Acc}}
         [] empty then Acc
         end
      end
   in
      {PromenadeHelp Bt nil}
```

```
   end
end
fun {FoldL L F U}
   case L of nil then U
   [] H|T then {FoldL T F {F U H}}
   end
end

fun {SumTree Bt}
   local L Sum
      L = {PromenadeSoluce Bt}
      Sum =
      fun {$ Lst Tot}
         case Lst of H|T then {Sum T Tot+H}
         [] nil then Tot
         end
      end
   in
      {Sum L 0}
   end
end

Btree = btree(42
              left: btree(26
                          left: btree(54
                                      left: empty
                                      right: btree(18
                                                   left: empty
                                                   right: empty))
                          right: empty)
              right: btree(37
                           left: btree(11
                                       left: empty
                                       right: empty)
                           right: empty))
{Browse {Promenade Btree}}
{Browse {PromenadeSoluce Btree}}
{Browse {FoldL [1 2 3 4 5 6 7 8 9] fun {$ X Y} X+Y end 0}}
{Browse {SumTree Btree}}

%QUESTION 11 ------------------------------------------------------------
declare
fun {DictionaryFilter D F}
   local
      fun {DictionaryFilterHelp D F Acc}
         case D of dict(key:Key info:Info left:Left right:Right) then
            if {F D.info} then Key#Info|{DictionaryFilterHelp Left F {
               DictionaryFilterHelp Right F Acc}}
            else {DictionaryFilterHelp Left F {DictionaryFilterHelp
               Right F Acc}}
            end
         [] leaf then Acc
         end
      end
   in
      {DictionaryFilterHelp D F nil}
   end
end

Class = dict(key:10
             info:person('Christian' 19)
             left:dict(key:7
                       info:person('Denys' 25)
                       left:leaf
                       right:dict(key:9
```

```
                                    info:person('David' 7)
                                    left:leaf
                                    right:leaf))
             right:dict(key:18
                        info:person('Rose' 12)
                        left:dict(key:14
                                    info:person('Ann' 27)
                                    left:leaf
                                    right:leaf)
                        right:leaf))
fun {Old Info}
   Info.2 > 20
end
{Browse {DictionaryFilter Class Old}}
% Val --> [7#person('Denys' 25) 14#person('Ann' 27)]

%QUESTION 12 ------------------------------------------------------------

{Browse '(a b):'|{IsList '|'(a b)}|{IsTuple '|'(a b)}} %TUPLE
{Browse '|(a |(b nil)):'|{IsList '|'(a '|'(b nil))}|{IsTuple '|'(a '|'(b
    nil))}} %LISTE
{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)
{Browse 'state(1 a 2):'|{IsList state(1 a 2)}|{IsTuple state(1 a 2)}} %
   TUPLE
{Browse 'state(1 3:2 2:a):'|{IsList state(1 3:2 2:a)}|{IsTuple state(1
   3:2 2:a)}} %TUPLE
{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
{Browse 'a#b#c:'|{IsList a#b#c}|{IsTuple a#b#c}} %TUPLE
{Browse '[a b c]:'|{IsList  [a b c]}|{IsTuple  [a b c]}} %LISTE
{Browse 'm|n|o:'|{IsList m|n|o}|{IsTuple m|n|o}} %TUPLE (alors que m|n|o
   |nil est bien une liste)

%QUESTION 13 ------------------------------------------------------------

declare
fun {Applique L F}
   case L of H|T then {F H}|{Applique T F}
   [] nil then nil
   end
end
fun {Lol X}
   lol(X)
end
fun {MakeAdder X}
   fun {$ Y} X+Y
   end
end
fun {AddAll L N}
   case L of H|T then H+N|{AddAll T N}
   [] nil then nil
   end
end

{Browse {Applique [1 2 3] Lol}} % Affiche [lol(1) lol(2) lol(3)]
Add5 = {MakeAdder 5} % correspond   Add5 = fun {$ Y} Y+5
{Browse {Add5 13}} % Affiche 18
{Browse {AddAll [1 2 3 4] 2}}

%QUESTION 14 ------------------------------------------------------------

{Browse {Label a#b#c}} %#
{Browse {Width un#tres#long#tuple#tres#tres#long}} %7
```

```
{Browse {Arity 1#4#16}} % 1#2#3

fun {SameLength Xs Ys}
   case Xs#Ys
   of nil#nil then true
   [] (X|Xr)#(Y|Yr) then {SameLength Xr Yr}
   else false
   end
end
```

- **TP3**

```
% INTRODUCTION
    -------------------------------------------------------------
declare
fun {Fact N}
   if N==0 then 1
   else N*{Fact N-1} end
end

KernelFact in
proc {KernelFact N R}
   local Zero B1 in
      Zero = 0
      B1 = N==Zero
      if B1 then R = 1
      else
         local N1 B2 One in %Attention !!! Ne pas oublier One comme Zero
              dans if
            One = 1
            N1 = N-One
            {KernelFact N1 B2}
            R = N*B2
         end
      end
   end
end

{Browse {Fact 5}}
{Browse {KernelFact 5}}

% QUESTION 1 ----------------------------------------------------------
%Point A ---------
declare
fun {Sum N}
   if N == 1 then 1
   else N*N + {Sum N-1} end
end
KernelSum in
proc {KernelSum N ?R}
   local One B1 in
      One = 1
      B1 = N==1
      if B1 then R = 1
      else
         local N1 B2 B3 in
            N1 = N-1
            B2 = N*N
            B3 = {Sum N1}
            R = B2 + B3
         end
      end
   end
end

```

```
{Browse {Sum 9}}
{Browse {KernelSum 9}}

%Point B ---------
declare
fun {SumAux N Acc}
   if N == 1 then Acc + 1
   else {SumAux N-1 N*N+Acc} end
end
fun {Sum N}
   {SumAux N 0}
end

KernelSumAux KernelSum in %ATTENTION!!!! Bien mettre les 2 fonctions sur la m me ligne
proc {KernelSumAux N Acc ?R}
   local One B1 in
      One = 1
      B1 = N==1
      if B1 then R = Acc+1
      else
         local N1 B2 B3 in
            N1 = N-1
            B2 = N*N
            B3 = B2+Acc
            {KernelSumAux N1 B3 R}
         end
      end
   end
end %NE PAS METTRE LE 'KernelSum in' ici

proc {KernelSum N ?R}
   local Zero in
      Zero = 0
      {KernelSumAux N Zero R}
   end
end

% QUESTION 2 ------------------------------------------------------------

%[1]-> '|'(1:1 2:nil)
%[1 2 3]-> '|'(1:1 2:'|'(1:2 2:'|'(1:3 2:nil)))
% nil-> nil
%state(4 f 3)-> state(1:4 2:f 3:3)

% QUESTION 3 ------------------------------------------------------------

%Environnement contextuel
%proc {Q A} {P A+1} end ---> E_c = {}
%proc {P} {Browse A} end  ---> E_c = {Browse -> browse}

local P Q in
   proc {P A R} R=A+2 end
   local P R in
      fun {Q A}
         {P A R}
         R
      end
      proc {P A R} R=A-2 end
   end
   %% Qu'affiche {Browse {Q 4}} ? 2
   {Browse {Q 4}}
end

% QUESTION 5------------------------------------------------------------
```

```
local MakeAdd Add1 Add2 in
   proc {MakeAdd X Add}
      proc {Add Y Z}
         Z=X+Y
      end
   end
   {MakeAdd 1 Add1}
   {MakeAdd 2 Add2}
   local V in
      {Add1 42 V} {Browse V} %43
   end
   local V in
      {Add2 42 V} {Browse V} %44
   end
end
```

- # TP4

```
% QUESTION 1 ------------------------------------------------------------
local P in
   local Z in
      Z=1
      proc {P X Y} Y=X+Z end
   end
   local B A in
      A=10
      {P A B}
      {Browse B}
   end
end

%Voir feuilles   la main

% QUESTION 2 ------------------------------------------------------------
declare
fun{MakeMulFilter N}
   fun{$ I} I mod N == 0
   end
end
fun {Filter L F}
   case L of H|T then
      if {F H} then
         H|{Filter T F}
      else {Filter T F}
      end
   [] nil then nil
   end
end
fun {IsPrime Nbr}
   local IsPrimeHelp
      fun {IsPrimeHelp X N}
         if N > (X div 2) then true
         elseif X mod N \= 0 then {IsPrimeHelp X N+1}
         else false
         end
      end
   in
      {IsPrimeHelp Nbr 2}
   end
end

Mul5 = {MakeMulFilter 5}
{Browse {Mul5 15}}
{Browse {Mul5 16}}
{Browse {Filter [1 2 3 4 5 6] fun{$ A} A mod 2 == 0 end}}
```

```
{Browse {Filter [1 2 3 4 5 6] fun{$ A} A mod 3 == 0 end}}
{Browse {Filter [1 2 3 4 5 6 7 8 9] IsPrime}}

% QUESTION 3 ------------------------------------------------------------
declare
fun{MultL L}
   if L == nil then 0
   else
      local MultLHelp
         fun {MultLHelp L Tot}
            case L of H|T then {MultLHelp T Tot*H}
            [] nil then Tot
            end
         end
      in
         {MultLHelp L 1}
      end
   end
end
fun{DiffL L}
   local DiffLHelp
      fun {DiffLHelp L Tot}
         case L of H|T then {DiffLHelp T Tot-H}
         [] nil then Tot
         end
      end
   in
      {DiffLHelp L 0}
   end
end

{Browse {MultL [1 2 3 4 5 6 7 8 9]}}
{Browse {MultL nil}}
{Browse {DiffL [1 2 3 4 5 6 7 8 9]}}
{Browse {DiffL nil}}

% QUESTION 4 ------------------------------------------------------------
declare
fun {Applique L F}
   case L of H|T then {F H}|{Applique T F}
   [] nil then nil
   end
end
fun {PowSoluce Exp}
   fun {PowHelp N E Tot}
      if E>1 then {PowHelp N E-1 Tot*N}
      else Tot
      end
   end
in
   fun{$ Nbr}
      {PowHelp Nbr Exp Nbr}
   end
end

{Browse {Applique [1 2 3 4 5] {PowSoluce 3}}}
%On a donc {{PowSoluce 3} H} qui va direct aller dans le "in" pour
    r soudre  {PowHelp H 3 H}

% QUESTION 5 ------------------------------------------------------------
%Point A ---------
declare
fun {Convertir1 T}
   fun {$ V} V*T end
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 end
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 fun{MyMap L F}
146    case L of H|T then {F H}|{MyMap T F}
147    [] nil then nil end
148 end
149 fun{MyFoldL L F Acc}
150    case L of H|T then {MyFoldL T F {F Acc H}}
151    [] nil then Acc
152    end
153 end
154 fun {PipeLine N}
155    P1 P2 P3 in
156    P1 = {GenerateList N}
157    P2 = {MyFilter P1 fun {$ X} X mod 2 \= 0 end}
158    P3 = {MyMap P2 fun {$ X} X * X end}
159    {MyFoldL P3 fun {$ Acc X} X + Acc end 0}
160 end
161
162 {Browse {PipeLine 10}}
163
164 % QUESTION 7 ------------------------------------------------------------
165 local Y LB in
166    Y=10
167    proc {LB X ?Z}
168       local B in
169          B = X>=Y
170          if X>=Y then Z=X
171          else Z=Y end
172       end
173    end
174    local Y Z Five in
175       Y = 15
176       Cinq = 5
```

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

- **TP5**

LINFO1104 : TP5 Lamdba-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 progammation "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 effecuées 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

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

   - $\lambda x.xyz$ → valide : $\lambda x.((xy)z)$
   - $\lambda x.\lambda y$ → invalide
   - m → valide
   - $x\lambda wy.y$ → valide : $x(\lambda w.\lambda y.y)$
   - $x\lambda$ → invalide
   - $\lambda\lambda xz.zx$ → invalide
   - $(mnop)(qrst)vw\lambda xyz.zxy$ → valide

   (Solution)

   - valide
   - Invalide
   - valide
   - valide
   - invalide
   - invalide
   - 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.

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

   (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 (α – *conversion*).** Pour chaque paire d'expression, indiquez si les expresions sont α – *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 (β).** Reduisez au maximum les expressions suivantes en utilisant la β-reduction.

- $(\lambda x.xx)y \to yy$
- $(\lambda x.axxa)y \to ayya$
- $(\lambda x.(\lambda z.zx)q)y \to (\lambda x.qx)y \to qy$
- $(\lambda x.x((\lambda z.zx)(\lambda x.bx)))y$ ($(\lambda x.bx)x \Leftrightarrow (bx)$) $\to (\lambda x.x(bx))y \to y(by)$
- $(\lambda m.m)(\lambda n.n)(\lambda c.cc)(\lambda d.d)$ $\to (\lambda d.d)(\lambda d.d) \to (\lambda d.d)$
- $\lambda z.x\lambda x.x$ → 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 (η).** Réduisez au maximum les expressions suivantes en utilisant l'η-reduction.

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

(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 β-réduction ou par η-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 $(\lambda x.\, \lambda y.\, x\ y)\ (\lambda z.\, z)\ (\lambda w.\, w)$

(Soluce)

$$\begin{aligned}(\lambda x.\, \lambda y.\, x)\ ((\lambda x.\, x)\ y) &= \lambda y.\, ((\lambda x.\, x)\ y) \\ &= \lambda y.\, y\end{aligned}$$

$$\begin{aligned}(\lambda x.\, \lambda y.\, x)\ ((\lambda x.\, x)\ y) &= \lambda x.\, ((\lambda y.\, x)\ y) \\ &= \lambda x.\, x\end{aligned}$$

$$\begin{aligned}(\lambda x.\, \lambda y.\, x\ y)\ (\lambda z.\, z)\ (\lambda w.\, w) &= (\lambda y.\, (\lambda z.\, z)\ y)\ (\lambda w.\, w)) \\ &= (\lambda z.\, z)(\lambda w.\, w) \\ &= \lambda w.\, w\end{aligned}$$

$$\begin{aligned}(\lambda x.\, \lambda y.\, x\ y)\ (\lambda z.\, z)\ (\lambda w.\, w) &= \lambda x.\, (\lambda y.\, x\ y\ (\lambda w.\, w)) \\ &= \lambda y.\, (\lambda w.\, w)\ y \\ &= \lambda y.\, y\end{aligned}$$

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

Représentation de type de donnée

7. **Arithmetiques 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 répresenter les booléens, on utilise une notation appelé 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:

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

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

(Solution)

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

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

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

Un nombre n est donc représenté par une fonction à deux arguments $f\ x$ qui execute 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-que le résultat est 3?
- $plus := \lambda m.\lambda n.\lambda f.\lambda x.mf(nfx)$

(Solution)

- $succ := \lambda n.\lambda f.\lambda x.(nfx)$
- $plus\ 1\ 2 \Rightarrow (\lambda m.\lambda n.\lambda f.\lambda x.m\ f(n\ f\ x))1\ 2 \Rightarrow \lambda f.\lambda x.1\ f\ (2\ f\ x) \Rightarrow \lambda f.\lambda x.1\ f\ ((\lambda fx.f(fx))f\ x) \Rightarrow \lambda f.\lambda x.1\ f\ (f(fx)) \Rightarrow \lambda f.\lambda x(\lambda f.\lambda x(fx))f\ (f(fx)) \Rightarrow \lambda f.\lambda x.f(f(fx))$

9. **Les listes en lambda calcul**

- $paire := \lambda a.\lambda b.\lambda f.((fa)b)$
- 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)

- $first := \lambda a.\lambda b.a$
- $second := \lambda a.\lambda b.b$
- $p := pair\ 4\ 2 \Rightarrow \lambda 4.\lambda 2.\lambda f.((fa)b) \Rightarrow \lambda f((f4)2)$
- $p\ first \Rightarrow \lambda f.((f\ 4)2)\ first \Rightarrow ((first\ 4)2) \Rightarrow ((\lambda a.\lambda b.a)4\ 2) \Rightarrow 4$
- une liste est une paire où le premier élément est une expression lambda (head) et le second élément est une liste (tail): $\lambda f.((fh)t)$

La récursivité en lambda-calcul

10. **Fonction récursive**. Contrairement à un langage de programation, 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 $f(f(...(f(c))...)) = f^n(c) = c$.

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

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

$\lambda mfx.f(mfx)$

$\lambda fx.f(f(fx))$

$m = \boxed{1} := \lambda fx.fx$

$m = 2 := \lambda fx.f(fx)$

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

$\lambda f x . f f$

Nous allons utiliser ce même principe pour créer des expression lamba 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 :

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

Si le combinateur est appliqué indéfiniment, on obtient :

$$Y\ g = g\ (Y\ g) = g\ (g\ (Y\ g)) = g\ (...g\ (Y\ g)...)$$

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 défninies dans la fonction `add` ? En d'autres termes, défninissez 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` $= \lambda y.y\ (\lambda t.\texttt{false})\ \texttt{true}$
`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 lamba calcul :
  - `is_zero 2`
  - `pred 2`

`is_zero 2` $= (\lambda f.\,f\ (\lambda t.\,\lambda xy.\,y)\ \lambda xy.\,x)\ (\lambda sz.\,s(s(z)))$
$= \lambda sz.\,s(s(z))\ (\lambda t.\,\lambda xy.\,y)\ \lambda xy.\,x$
$= (\lambda t.\,\lambda xy.\,y)((\lambda t.\,\lambda xy.\,y)(\lambda xy.\,x))$
$= (\lambda t.\,\lambda xy.\,y)(\lambda xy.\,x)$
$= (\lambda xy.\,y)$
$= \texttt{false}$

$$
\begin{aligned}
\texttt{pred}\ 2 &= (\lambda n.\,\lambda f.\,\lambda x.\,n\ (\lambda g.\,\lambda h.\,h\ (g\ f))\ (\lambda u.\,x)\ (\lambda u.\,u))\ (\lambda sz.\,s(s(z))) \\
&= \lambda f.\,\lambda x.\,(\lambda sz.\,s(s(z)))\ (\lambda g.\,\lambda h.\,h\ (g\ f))\ (\lambda u.\,x)\ (\lambda u.\,u) \\
&= \lambda f.\,\lambda x.\,(\lambda g.\,\lambda h.\,h\ (g\ f))\ ((\lambda g.\,\lambda h.\,h\ (g\ f))(\lambda u.\,x))\ (\lambda u.\,u) \\
&= \lambda f.\,\lambda x.\,(\lambda g.\,\lambda h.\,h\ (g\ f))\ (\lambda h.\,h\ (\lambda u.\,x)\ f)\ (\lambda u.\,u) \\
&= \lambda f.\,\lambda x.\,(\lambda g.\,\lambda h.\,h\ (g\ f))\ (\lambda h.\,h\ x)\ (\lambda u.\,u) \\
&= \lambda f.\,\lambda x.\,(\lambda h.\,h\ ((\lambda h.\,h\ x)\ f))\ (\lambda u.\,u) \\
&= \lambda f.\,\lambda x.\,(\lambda h.\,h\ (f\,x))\ (\lambda u.\,u) \\
&= \lambda f.\,\lambda x.\,(\lambda u.\,u\ (f\,x)) \\
&= \lambda f.\,\lambda x.\,f\,x \\
&= \lambda f.\,\lambda x.\,f(x) \\
&= 1
\end{aligned}
$$

- Continuez l'évaluation de l'expression demandée :

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

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.

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

• TP6

```
% QUESTION 1 ------------------------------------------------------------
declare
fun {Reverse Xs}
   local C in
      C = {NewCell nil}
      for X in Xs do
         C := X|@C
      end
      @C
   end
end
{Browse {Reverse [1 2 3 4 5 6 7 8 9]}}

% Identique en terme d'appels rec
% L'etat n'est pas encapsule ! Il y a une Cell Y
% Xs est une liste (immutable), il ne peut pas changer !

% QUESTION 2 ------------------------------------------------------------
declare
fun {NewStack}
   {NewCell nil}
end
fun {IsEmpty S}
   @S == nil
end
proc {Push S X}            %COMME MODIFIE LA CELL IL FAUT METTRE PROC
   S := X|@S
end
fun {Pop S}
   case @S of H|T then
      S := T
      H
   [] nil then nil
   end
end
fun{Eval L}
   C in                    %ATTENTION PAS METTRE EN LOCAL
   C = {NewStack}
   for Elem in L do
      if Elem == '+' then
         {Push C {Pop C}+{Pop C}}
      elseif Elem == '*' then
         {Push C {Pop C}*{Pop C}}
      elseif Elem == '-' then
         local Pop1 in
            Pop1 = {Pop C}
            {Push C {Pop C}-Pop1}
         end
      elseif Elem == '/' then
         local Pop1 in
            Pop1 = {Pop C}
            {Push C {Pop C}/Pop1}
         end
      else
         {Push C Elem}
      end
   end
   {Pop C}
end

{Browse {Eval [13 45 '+' 89 17 '-' '*']}}

% QUESTION 3 ------------------------------------------------------------
```

```
declare
fun {NewStack}          %Plus besoin de la Cell dans les arguments des
    fonctions du coup
   C
   fun {IsEmpty}
      @C == nil
   end
   proc {Push X}
      C := X|@C
   end
   fun {Pop}
      case @C of H|T then
         C := T
         H
      [] nil then nil
      end
   end
in
   C  = {NewCell nil}   %Evidemment le C doit etre attribu  aprs le "in
       " !!!!!!
   stack(isEmpty:IsEmpty push:Push pop:Pop)
end
fun{Eval L}
   C in                     %ATTENTION PAS METTRE EN LOCAL
   C = {NewStack}
   for Elem in L do
      if Elem == '+' then
         {C.push {C.pop}+{C.pop}}
      elseif Elem == '*' then
         {C.push {C.pop}*{C.pop}}
      elseif Elem == '-' then
         local Pop1 in
            Pop1 = {C.pop}
            {C.push {C.pop}-Pop1}
         end
      elseif Elem == '/' then
         local Pop1 in
            Pop1 = {C.pop}
            {C.push {C.pop}/Pop1}
         end
      else
         {C.push Elem}
      end
   end
   {C.pop}
end

{Browse {Eval [13 45 '+' 89 17 '-' '*']}}

% QUESTION 4 ------------------------------------------------------------
declare
fun {ShuffleSoluce L} Len A C Tail Head in
   Len = {Length L}
   A = {NewArray 0 Len 0}
   C = {NewCell 0}
   for Elem in L do
      A.@C := Elem
      C := @C+1
   end
   Tail = {NewCell Head}
   for I in 0..(Len-1) do Idx Range Next in
      Range = Len - 1 - I
      Idx = {OS.rand} mod (Range+1)
      @Tail = A.Idx|Next
      Tail := Next
```

```
      A.Idx := A.Range
   end
   @Tail = nil
   Head
end
{Browse {ShuffleSoluce [a b c d e]}}

% QUESTION 5 ------------------------------------------------------------
declare
C = {NewCell nil}
L1={NewCell 0}|{NewCell 1}|{NewCell 2}|nil
L2=0|{NewCell 1|{NewCell 2|C}}
% L1 est une liste de Cell
% L2 n'est pas une liste car ne termine pas par nil

%O(1)
fun {Prepend1 X L} X|L end
fun {Prepend2 X L} X|{NewCell @L} end

% pour ajouter   la fin de L1 on utilise Append O(n)
% pour L2 O(n)
proc {Append2 L X}
   case L
   of _|T then {Append2 T X}
   else
      case @L of nil then L := X|{NewCell nil}
      [] _|C then {Append2 C X} end
   end
end
{Append2 L2 3}
{Browse @C}

declare
% both are unsafe, we need at least two elements
fun {Swap L} L.2.1|L.1|L.2.2 end
fun {Swap2 L} H T in
   H = (@(L.2)).1
   T = (@(L.2)).2
   L.2 := L.1|T
   H|L.2
end

% INTRO Classes
    ------------------------------------------------------------
declare
class Counter
   attr value
   meth init % (re)initialise le compteur
      value:=0
   end
   meth inc % incremente le compteur
      value:=@value+1
   end
   meth get(X) % renvoie la valeur courante du compteur dans X
      X=@value
   end
end
MonCompteur={New Counter init}
for X in [65 81 92 34 70] do {MonCompteur inc} end
{Browse {MonCompteur get($)}}

% QUESTION 6 ------------------------------------------------------------
declare
class Collection
   attr elements
```

```
   meth init
      elements := nil
   end
   meth put(X)
      elements := X|@elements
   end
   meth get($)
      case @elements of X|Xr then elements := Xr X end
   end
   meth isEmpty($)
      @elements == nil
   end
   meth union(C)
      if {Not {C isEmpty($)}} then
         {self put({C get($)})}
         {self union(C)}
      end
   end
   meth toList($)
      @elements
   end
end

declare
class SortedCollection from Collection
   meth put(X)
      fun {PutHelp L}
         case L of H|T then
            if H<X then H|{PutHelp T}
            else X|L
            end
         [] nil then X|L
         end
      end in
      elements :={PutHelp @elements}
   end
end

declare % tests
C1 = {New Collection init}
C2 = {New SortedCollection init}

{C1 put(0)}
{C1 put(1)}
{C2 put(2)}
{C2 put(1)}
{C2 put(3)}

{C1 union(C2)}
{Browse {C2 isEmpty($)}}
{Browse {C1 isEmpty($)}}
{Browse {C1 toList($)}}
{Browse {C1 get($)}}
{Browse {C1 toList($)}}

% if C1, C2 are Collections, union is O(|C2|)
% because put is O(1)

% if C1, C2 are SortedCollections,
% union is O(|C2|^2 + |C1|*|C2|)

% proof:
% (n1 + (n1+1) + (n1+2) + ... + (n1+n2-1))
% ~= (n1 + (n1+1) + (n1+2) + ... + (n1+n2))
% = n2*n1 + (0+1+2+...+n2)
```

```
% = n2*n1 + 0.5*n2*(n2+1)
% ~= n2*n1 + 0.5*n2*n2
% => O(n2^2 + n2*n1)

declare
Xs = [7 8 0 4 3]
C3 = {New SortedCollection init}
for X in Xs do
   {C3 put(X)}
end
{Browse {C3 toList($)}}
% insertion sort O(n^2)

% QUESTION 7 ------------------------------------------------------------
declare
class Constante
   attr value
   meth init(V) value := V end
   meth evalue($) @value end
   meth derive(V $) {New Constante init(0)} end
end
class Variable
   attr value
   meth init(V) value := V end
   meth evalue($) @value end
   meth derive(V E)
      if self == V then E = {New Constante init(1)}
      else E = {New Constante init(0)} end
   end
   meth set(V) value := V end
end
class Somme
   attr e1 e2
   meth init(Expr1 Expr2)
      e1 := Expr1
      e2 := Expr2
   end
   meth evalue($) {@e1 evalue($)} + {@e2 evalue($)} end
   meth derive(V $)
      {New Somme init({@e1 derive(V $)} {@e2 derive(V $)})}
   end
end
class Difference
   attr e1 e2
   meth init(Expr1 Expr2)
      e1 := Expr1
      e2 := Expr2
   end
   meth evalue($) {@e1 evalue($)} - {@e2 evalue($)} end
   meth derive(V $)
      {New Difference init({@e1 derive(V $)} {@e2 derive(V $)})}
   end
end
class Produit
   attr e1 e2
   meth init(Expr1 Expr2)
      e1 := Expr1
      e2 := Expr2
   end
   meth evalue($) {@e1 evalue($)} * {@e2 evalue($)} end
   meth derive(V $) D1 D2 in
      D1 = {New Produit init({@e1 derive(V $)} @e2)}
      D2 = {New Produit init({@e2 derive(V $)} @e1)}
      {New Somme init(D1 D2)}
   end
```

```
end
class Puissance
   attr e1 c
   meth init(Expr1 C)
      e1 := Expr1
      c := C
   end
   meth evalue($) {Pow {@e1 evalue($)} @c} end
   meth derive(V $) De1 P CP in
      De1 = {@e1 derive(V $)}
      P = {New Puissance init(@e1 @c-1)}
      CP = {New Produit init({New Constante init(@c)} P)}
      {New Produit init(CP De1)}
   end
end

declare
VarX={New Variable init(0)}
VarY={New Variable init(0)}
local
   ExprX2={New Puissance init(VarX 2)}
   Expr3={New Constante init(3)}
   Expr3X2={New Produit init(Expr3 ExprX2)}
   ExprXY={New Produit init(VarX VarY)}
   Expr3X2mXY={New Difference init(Expr3X2 ExprXY)}
   ExprY3={New Puissance init(VarY 3)}
in
   Formule={New Somme init(Expr3X2mXY ExprY3)}
end

{VarX set(7)}
{VarY set(23)}
{Browse {Formule evalue($)}} % affiche 12153
{VarX set(5)}
{VarY set(8)}
{Browse {Formule evalue($)}} % affiche 547

declare
Derivee={Formule derive(VarX $)} % represente 6x - y
{VarX set(7)}
{VarY set(23)}
{Browse {Derivee evalue($)}} % affiche 19

% QUESTION 8 ------------------------------------------------------------
declare
class Sequence % not very efficient ! %ON PEUT METTRE UNE LISTE DANS LE
    INIT CONTRAIREMENT AU TEMPLATE
   attr l
   meth init(L) l := L end
   meth isEmpty($) @l == nil end
   meth first($) {List.nth @l 1} end
   meth last($) {List.last @l} end
   meth insertFirst(X) l := X|@l end
   meth insertLast(X) l := {List.append @l X|nil} end
   meth removeFirst l := {List.drop @l 1} end
   meth removeLast
      if {self isEmpty($)} then skip
      else l := {List.take @l {List.length @l}-1} end
   end
end

fun {Palindrome Xs}
   S
   fun {Check}
      if {S isEmpty($)} then true
```

```
      elseif {S first($)} == {S last($)} then
         {S removeFirst}
         {S removeLast}
         {Check}
      else false end
   end
in
   S = {New Sequence init(Xs)}
   {Check}
end
{Browse {Palindrome "radar"}}
{Browse {Palindrome "abc"}}
{Browse {Palindrome "a"}}
{Browse {Palindrome "eluparcettecrapule"}}
```

- # TP8

```
% QUESTION 1 ------------------------------------------------------------
declare A B C D
thread D = C+1 end
thread C = B+1 end
thread A = 1 end
thread B = A+1 end
{Browse D}

%DCABBrowse
%ABCD

% QUESTION 2 ------------------------------------------------------------
%Partie A--------
local X Y Z in
   thread if X==1 then Y=2 else Z=2 end end
   thread if Y==1 then X=1 else Z=2 end end
   X=1
   {Browse X|Y|Z|nil} %1|2|2
end

%Partie B--------
local X Y Z in
   thread if X==1 then Y=2 else Z=2 end end
   thread if Y==1 then X=1 else Z=2 end end
   X=2
   {Browse X|Y|Z|nil} %2|_|2
end

% QUESTION 3 ------------------------------------------------------------
declare
fun {ProduceInts N}
   fun {ProduceIntsHelp I}
      {Delay 1000}
      if I==N then N|nil
      else I|{ProduceIntsHelp I+1} end
   end
in
   {ProduceIntsHelp 1}
end
fun {Sum Xs}
   fun {SumHelp L Acc}
      {Delay 1000}
      case L of nil then Acc
      [] H|T then {SumHelp T Acc+H}
      end
   end
in
   {SumHelp Xs 0}
```

```
end
declare Xs S
thread Xs = {ProduceInts 666} end
thread S = {Sum Xs} end
{Browse S}

declare Xs S
Xs = {ProduceInts 666}
S = {Sum Xs}
{Browse S}

% QUESTION 4 ------------------------------------------------------------
declare
fun {Producer X}
   local
      fun {ProducerHelp N Lst}
         if N > ~1 then
            {ProducerHelp N-1 N|Lst}
         else
            Lst
         end
      end
   in
      {ProducerHelp X-1 nil}
   end
end

fun {FilterOddNumber Lst}
   local
      fun {FilterOddNumberHelp Lst OddNumbers}
         case Lst
         of nil then {Reverse OddNumbers} %pour  iter  de faire trop d'
             append
         [] H|T then
            if H mod 2 \= 0 then {FilterOddNumberHelp T H|OddNumbers}
            else  {FilterOddNumberHelp T OddNumbers}
            end
         end
      end
   in
      {FilterOddNumberHelp Lst nil}
   end
end

thread Xs = {Producer 10000} end
thread S = {FilterOddNumber Xs} end
{Browse S}

declare
fun {ServeBeer}
   local
      Beer = {OS.rand} mod 2
   in
      if Beer == 0 then 'trappist'
      else 'pils'
      end
   end
end

fun {SmellTrappist Beer}
   case Beer
   of 'trappist' then true
   else false
   end
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
```

```
   end
end

local InS in
   {Browse {Counter InS}}
   InS=a|b|a|c|_
end

%Solution

declare
fun {CounterSol S} %ne renvoit rien non plus.....
   fun {Update C L}
      case L of (H1#H2)|T then
         if H1 == C then H1#(H2+1)|T
         else (H1#H2)|{Update C T} end
      else nil end
   end
   fun {CAlt S A}
      case S of H|T then
         A2 = {Update H A}
      in
         A2|{CAlt T A2}
      else nil
      end
   end
in
   thread {CAlt S nil} end
end

local InS in
   {Browse {Counter InS}}
   InS=a|b|a|c|_
end
```

- **TP9**

```
% QUESTION 3 ------------------------------------------------------------
declare
fun {MakeBinaryGate F} % Complexit de {MakeBinaryGate F} est O(1) et
    ses fonctions O(n)
   fun {$ S1 S2} % $ reprsente {MakeBinaryGate F}
      fun {Gate SA SB}
         case SA#SB
         of (HA|TA)#(HB|TB) then
            {F HA HB}|{Gate TA TB}
         else
            nil
         end
      end
   in
      thread {Gate S1 S2} end %bien mettre thread car stream
   end
end

S1 = 0|0|1|1|_
S2 = 0|1|0|1|_
And = {MakeBinaryGate fun {$ A B} A*B end}
Or = {MakeBinaryGate fun {$ A B} A+B-A*B end}
Xor = {MakeBinaryGate fun {$ A B} A+B-2*A*B end}
Nor = {MakeBinaryGate fun {$ A B} 1-(A+B-A*B) end}
{Browse 'and'#{And S1 S2}}
{Browse 'or'#{Or S1 S2}}
{Browse 'xor'#{Xor S1 S2}}
{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 pendent  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
```

```
   of H|T then {Browse H} {PrintMsg T}
   else skip
   end
end

{PrintMsg S}

% QUESTION 2 ------------------------------------------------------------
declare
fun {Pow N E}
   local
      fun {PowHelp E Acc}
         if E > 0 then {PowHelp E-1 Acc*N}
         else Acc
         end
      end
   in
      {PowHelp E 1}
   end
end

fun {LaunchServer}
   local S P
      proc {ParseStream S}
         case S
         of H|T then
            case H
            of add(X Y R) then R = X+Y {ParseStream T}
            [] pow(X Y R) then R = {Pow X Y} {ParseStream T}
            [] 'div'(X Y R) then if Y \= 0 then R = X div Y else R = '
                division par 0 impossible' end {ParseStream T}
            else
               {Browse 'je ne comprends pas ton message'} {ParseStream T
                   }
            end
         else skip
         end
      end
   in
      P = {NewPort S}
      thread {ParseStream S} end
      P
   end
end

A B N S Res1 Res2 Res3 Res4 Res5 Res6
S = {LaunchServer}
{Send S add(321 345 Res1)}
{Browse Res1}
{Send S pow(2 N Res2)}
N = 8
{Browse Res2}
{Send S add(A B Res3)}
{Send S add(10 20 Res4)}
{Send S foo}
{Browse Res4}
A = 3
B = 0-A
{Send S 'div'(90 Res3 Res5)}
{Send S 'div'(90 Res4 Res6)}
{Browse Res3}
{Browse Res5}
{Browse Res6}

% QUESTION 3 ------------------------------------------------------------
```

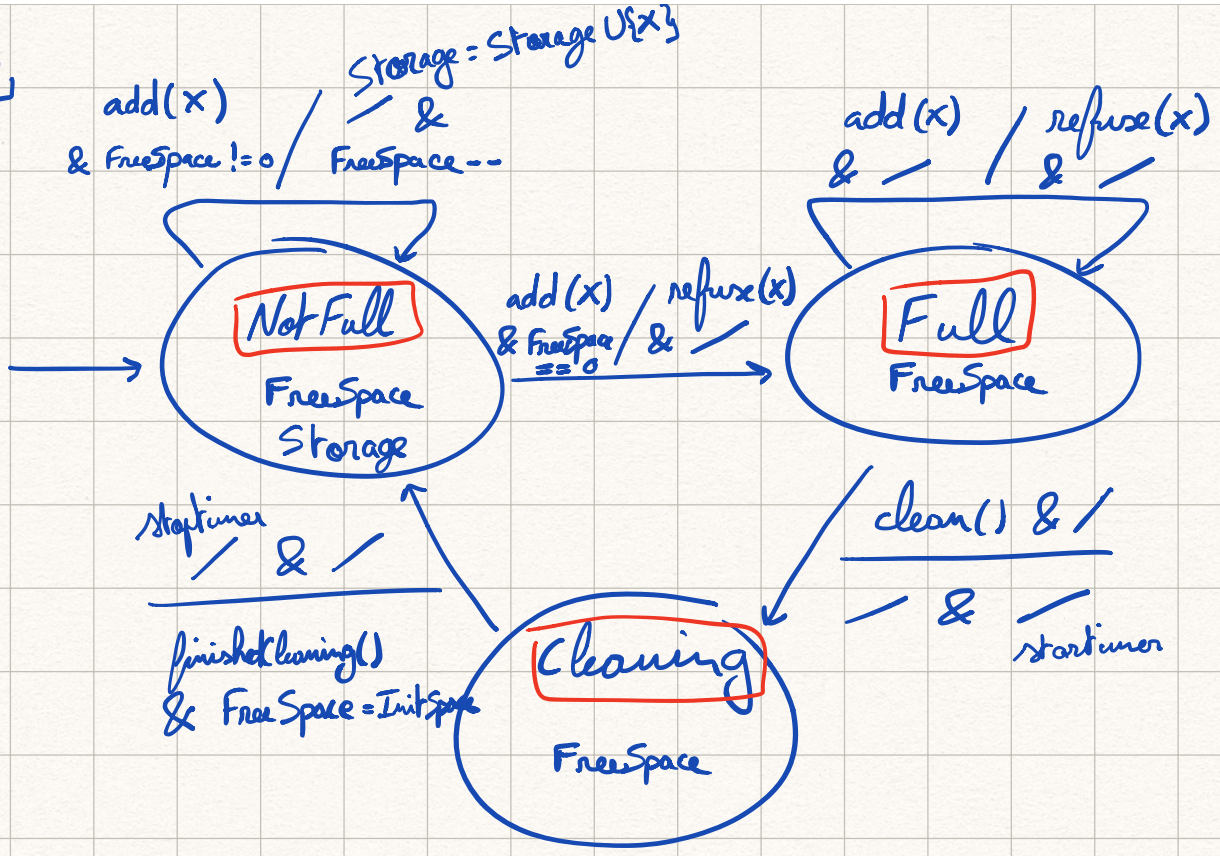
```
declare
fun {StudentRMI}
   S in
   thread
      for ask(howmany:Beers) in S do
         Beers={OS.rand} mod 24
      end
   end
   {NewPort S}
end

fun {StudentCallBack}
   S in
   thread
      for ask(howmany:P) in S do
         {Send P {OS.rand} mod 24}
      end
   end
   {NewPort S}
end

fun {Student}
   local N in
      N = {OS.rand} mod 2
      if N == 0 then 'StudentRMI'
      else 'StudentCallBack'
      end
   end
end

fun {CreateUniversity Size}
   fun {CreateLoop I}
      if I =< Size then
         % pour {Student} choisissez soit StudentRMI ou StudentCallBack,
         %  d fini  plus haut, selon l'humeur de Charlotte
         {Student}|{CreateLoop I+1}
      else nil end
   end
in
   {CreateLoop 1}
end

proc {Charlotte}
   local Students TotalBeers
      proc {P L}
         {Browse L}
      end
   in
      Students = {CreateUniversity 10}
      {P Students}
   end
end

{Charlotte}

%pas compris et pas  r solu  correctement...

% QUESTION 4 ------------------------------------------------------------
declare
fun {MakePortier}
   local P S
      proc {PortierHelp S Tot}
         case S
         of H|T then
            case H
```

```
            of getIn(N) then {PortierHelp T Tot+N}
            [] getOut(N) then {PortierHelp T Tot-N}
            [] getCount(N) then N = Tot {PortierHelp T Tot}
            else {Browse 'commande non valide'}
            end
         else skip
         end
      end
   in
      P = {NewPort S}
      thread {PortierHelp S 0} end
      P
   end
end

Portier = {MakePortier}
{Send Portier getIn(10)}
{Browse {Send Portier getCount($)}}
{Send Portier getOut(5)}
{Browse {Send Portier getCount($)}}

% QUESTION 5 ------------------------------------------------------------
declare
fun {NewStack} %/!\ comme rien n'est en local il faut avoir des noms de
    variables diffrents
   Stream
   proc {ParseStack S Stack}
      case S  %liste d'instructions stream alors que stack est une liste
      of push(X)|T then {ParseStack T X|Stack}
      [] pop(X)|T then X = Stack.1 {ParseStack T Stack.2}
      [] isEmpty(X)|T then X = (Stack==nil) {ParseStack T Stack}
      []_|T then {ParseStack T Stack}
      else {Browse 'can do that'}
      end
   end
in
   thread {ParseStack Stream nil} end
   {NewPort Stream}
end
proc {Push Port X}
   {Send Port push(X)}
end
fun {Pop Port}
   {Send Port pop($)}
end
fun {IsEmpty Port}
   {Send Port isEmpty($)}
end

MyStack = {NewStack}
{Push MyStack 'banane'}
{Browse {Pop MyStack}}
{Browse {IsEmpty MyStack}}
{Push MyStack 'kiwi'}
{Push MyStack 'fraise'}
{Browse {IsEmpty MyStack}}


% QUESTION 7 ------------------------------------------------------------
declare
fun {Counter Output}
   S
   fun {UpdateStream Elem State}
      case State
      of nil then {Append State Elem#1|nil}
```

```
      [] H|T then
         if Elem == H.1 then
            H.1#H.2+1|T
         else
            H|{UpdateStream Elem T}
         end
      end
   end
   proc {TreatStream S State Output}
      case S
      of H|T then
         local NewState NewOutput in
            NewState = {UpdateStream H State}
            Output = NewState|NewOutput
            {TreatStream T NewState NewOutput}
         end
      [] _|T then {TreatStream T State Output}
      else skip
      end
   end
in
   thread {TreatStream S nil Output} end
   {NewPort S}
end
Out
C = {Counter Out}
{Browse out#Out}
{Delay 512}
{Send C a}
{Delay 512}
{Send C a}
{Delay 512}
{Send C b}
{Delay 512}
{Send C c}
{Delay 512}
{Send C a}
{Delay 512}
{Send C b}
```

- **TP11**

1)



2) 1 seul controller pour tts les ascenseurs donc 1 seul ascenceur en mvt à la fois.

3) → Cfr. code

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

Diagramme? plus besoin du controller!

```
% Code Lift ------------------------------------------------------------
declare
proc {NewPortObject Init Fun ?P}
   proc {MsgLoop S1 State}

      case S1 of Msg|S2 then
         {MsgLoop S2 {Fun Msg State}}
      [] nil then skip end
   end
   Sin
in
   thread {MsgLoop Sin Init} end
   {NewPort Sin P}
end
proc {NewPortObject2 Proc ?P}
   Sin in
   thread for Msg in Sin do {Proc Msg} end end
   {NewPort Sin P}
end
proc {Controller ?Cid}
   {NewPortObject2
    proc {$ Msg}
       case Msg
       of step(Lid Pos Dest) then
          if Pos<Dest then
             {Delay 1000} {Send Lid 'at'(Pos+1)}
          elseif Pos>Dest then
             {Delay 1000} {Send Lid 'at'(Pos-1)}
          end
       end
    end Cid}
end
proc {Floor Num Init Lifts ?Fid}
   {NewPortObject Init
    fun {$ Msg state(Called)}
       case Msg
       of call then
          {Browse 'Floor '#Num#' calls a lift!'}
          if {Not Called} then Lran in
             Lran=Lifts.(1+{OS.rand} mod {Width Lifts})
             {Send Lran call(Num)}
          end
          state(true)
       [] arrive(Ack) then
          {Browse 'Lift at floor '#Num#': open doors'}
          {Delay 2000}
          {Browse 'Lift at floor '#Num#': close doors'}
          Ack=unit
          state(false)
       end
    end Fid}
end
proc {Lift Num Init Cid Floors ?Lid}
   {NewPortObject Init
    fun {$ Msg state(Pos Sched Moving)}
       case Msg
       of call(N) then
          {Browse 'Lift '#Num#' needed at floor '#N}
          if N==Pos andthen {Not Moving} then
             {Wait {Send Floors.Pos arrive($)}}
             state(Pos Sched false)
          else Sched2 in
             Sched2={Append Sched [N]}
             if {Not Moving} then
```

```
                {Send Cid step(Lid Pos Sched2.1)} end
             state(Pos Sched2 true)
          end
       [] 'at'(NewPos) then
          {Browse 'Lift '#Num#' at floor '#NewPos}
          case Sched
          of nil then
             state(NewPos Sched Moving)
          [] S|Sched2 then
             if NewPos==S then
                {Wait {Send Floors.S arrive($)}}
                if Sched2==nil then
                   state(NewPos nil false)
                else
                   {Send Cid step(Lid NewPos Sched2.1)}
                   state(NewPos Sched2 true)
                end
             else
                {Send Cid step(Lid NewPos S)}
                state(NewPos Sched true)
             end
          end
       end
    end Lid}
end
proc {Building FN LN ?Floors ?Lifts}
   Lifts={MakeTuple lifts LN}
   for I in 1..LN do C in
      {Controller C}
      Lifts.I={Lift I state(1 nil false) C Floors}
   end
   Floors={MakeTuple floors FN}
   for I in 1..FN do
      Floors.I={Floor I state(false) Lifts}
   end
end

% QUESTION 4 ------------------------------------------------------------
declare
proc {NewPortObject Init Fun ?P}
        proc {MsgLoop S1 State}
        case S1 of Msg|S2 then
                {MsgLoop S2 {Fun Msg State}}
        [] nil then skip end
        end
        Sin
in
        thread {MsgLoop Sin Init} end
        {NewPort Sin P}
end

proc {NewPortObject2 Proc ?P}
        Sin in
        thread
                for Msg in Sin do
                   {Proc Msg}
                end
        end
        {NewPort Sin P}
end

proc {Floor Num Init Lifts ?Fid}
        {NewPortObject Init
    fun {$ Msg state(Called)}
        case Msg
```

```
        of call then
                        {Browse 'Floor '#Num#' calls a lift!'}
                        if {Not Called} then
                                Lran in
                        Lran=Lifts.(1+{OS.rand} mod {Width Lifts})
                        {Send Lran call(Num)}
                        end
                        state(true)
        [] arrive(Ack) then
                        {Browse 'Lift at floor '#Num#': open doors'}
                        {Delay 2000}
                        {Browse 'Lift at floor '#Num#': close doors'}
                        Ack=unit
                        state(false)
        end
    end
        Fid}
end
proc {Lift Num Init Cid Floors ?Lid}
        fun {Controller Dest state(Pos Sched)}
                NewPos in
                if Pos<Dest then
                        {Delay 1000} NewPos = Pos+1
                elseif Pos>Dest then
                        {Delay 1000} NewPos = Pos-1
                end
                state(NewPos Sched)
        end
        fun {Scheduler state(Pos Sched)}
                {Browse 'Lift '#Num#' at floor '#Pos}
                case Sched
                of S|Sched2 then
                        if Pos==S then
                                {Wait {Send Floors.S arrive($)}}
                                if Sched2==nil then
                                        state(Pos nil)
                                else
                                        {Scheduler {Controller Sched2.1
                                            state(Pos Sched2)}}
                                end
                        else
                                {Scheduler {Controller S state(Pos Sched
                                    )}}
                        end
                else
                        state(Pos Sched)
                end
        end
in
        {NewPortObject Init
    fun {$ Msg state(Pos Sched)}
        case Msg
        of call(N) then
                        {Browse 'Lift '#Num#' needed at floor '#N}
                        if N==Pos andthen {Not Moving} then
                        {Wait {Send Floors.Pos arrive($)}}
                        state(Pos Sched)
                        else
                                NewSched NewState in
                        NewSched = {Append Sched [N]}
                                {Scheduler state(Pos NewSched)}
                        end
        end
    end
        Lid}
```

```
end
proc {Building FN LN ?Floors ?Lifts}
   Lifts={MakeTuple lifts LN}
   for I in 1..LN do C in
      {Controller C}
      Lifts.I={Lift I state(1 nil false) C Floors}
   end
   Floors={MakeTuple floors FN}
   for I in 1..FN do
      Floors.I={Floor I state(false) Lifts}
   end
end

% QUESTION 5 ------------------------------------------------------------
declare
proc {NewPortObject Init Fun ?P}
        proc {MsgLoop S1 State}
        case S1 of Msg|S2 then
                {MsgLoop S2 {Fun Msg State}}
        [] nil then skip end
        end
        Sin
in
        thread {MsgLoop Sin Init} end
        {NewPort Sin P}
end

proc {NewPortObject2 Proc ?P}
        Sin in
        thread
                for Msg in Sin do
                   {Proc Msg}
                end
        end
        {NewPort Sin P}
end

proc {Controller ?Cid}
        {NewPortObject2
    proc {$ Msg}
        case Msg
        of step(Lid Pos Dest) then
                        if Pos<Dest then
                        {Delay 1000} {Send Lid 'at'(Pos+1)}
                        elseif Pos>Dest then
                        {Delay 1000} {Send Lid 'at'(Pos-1)}
                        end
        end
    end
        Cid}
end

proc {Floor Num Init Lifts ?Fid}
        {NewPortObject Init
    fun {$ Msg state(Called)}
        case Msg
        of call then
                        {Browse 'Floor '#Num#' calls a lift!'}
                        if {Not Called} then
                                Lran in
                        Lran=Lifts.(1+{OS.rand} mod {Width Lifts})
                        {Send Lran call(Num)}
                        end
                        state(true)
        [] arrive(Ack) then
```

```
                        {Browse 'Lift at floor '#Num#': open doors'}
                        {Delay 2000}
                        {Browse 'Lift at floor '#Num#': close doors'}
                        Ack=unit
                        state(false)
        end
    end
        Fid}
end
proc {Lift Num Init Cid Floors ?Lid}
        fun {Scheduler state(NewPos Scheduled Moving)}
                        fun {ChooseBestDest Scheduled Best}
                                case Scheduled
                                of H|T then
                                        if {Abs NewPos-H} < Best then {
                                            ChooseBestDest T H}
                                        else {ChooseBestDest T Best} end
                                else Best end
                        end
                in
                        case Scheduled
                        of Dest|Tail then
                                if Dest==NewPos then
                                        {Wait {Send Floors.Dest arrive($
                                            )}}
                                        if Tail==nil then
                                                state(NewPos nil false)
                                        else
                                                {Send Cid step(Lid
                                                    NewPos Tail.1)}
                                                state(NewPos Tail true)
                                        end
                                else
                                        BestDest = {ChooseBestDest
                                            Scheduled Dest} in
                                        {Send Cid step(Lid NewPos
                                            BestDest)}
                                        state(NewPos Scheduled true)
                                end
                        else state(Pos Scheduled Moving) end
        end
in
        {NewPortObject Init
    fun {$ Msg state(Pos Sched Moving)}
        case Msg
        of call(N) then
                        {Browse 'Lift '#Num#' needed at floor '#N}
                        if N==Pos andthen {Not Moving} then
                        {Wait {Send Floors.Pos arrive($)}}
                        state(Pos Sched false)
                        else
                                Sched2 in
                        Sched2={Append Sched [N]}
                        if {Not Moving} then
                                        {Send Cid step(Lid Pos Sched2.1)
                                            }
                                end
                        state(Pos Sched2 true)
                        end
        [] 'at'(NewPos) then
                        {Browse 'Lift '#Num#' at floor '#NewPos}
                        case Sched
                        of nil then
                        state(NewPos Sched Moving)
                        [] S|Sched2 then
```

```
                        if NewPos==S then
                                {Wait {Send Floors.S arrive($)}}
                                if Sched2==nil then
                                        state(NewPos nil false)
                                else
                                        {Send Cid step(Lid
                                            NewPos Sched2.1)}
                                        state(NewPos Sched2 true
                                            )
                                end
                            else
                                {Send Cid step(Lid NewPos S)}
                                state(NewPos Sched true)
                            end
                        end
        end
    end
        Lid}
end
proc {Building FN LN ?Floors ?Lifts}
   Lifts={MakeTuple lifts LN}
   for I in 1..LN do C in
      {Controller C}
      Lifts.I={Lift I state(1 nil false) C Floors}
   end
   Floors={MakeTuple floors FN}
   for I in 1..FN do
      Floors.I={Floor I state(false) Lifts}
   end
end
```

- **TP12**

```
% Question 1 ------------------------------------------------
-module(wrapper).
-export([hello_world/0, my_add/2, my_pow/2, my_fib/1, start/1, start/0,
    single/1, with_args/2]).

hello_world() ->
    io:fwrite("Hello, World!\n").

my_add(A, B) ->  A+B.

my_pow(Base, Exp) when Exp >= 0 -> my_pow(Base, Exp, 1).
my_pow(_, 0, Res) -> Res;
my_pow(Base, Exp, Res) -> my_pow(Base, Exp-1, Res*Base).

my_fib(Nb) -> my_fib(0, 1, Nb).
my_fib(Res,_,0) -> Res;
my_fib(Res,Fut,Nb) -> my_fib(Fut, Res+Fut, Nb-1).

single(hello_world) -> hello_world();
single(_) ->'unk'.

with_args(Fun, _Args) ->
    Valid_Members = [my_add, my_pow, my_fib],
    case lists:member(Fun, Valid_Members) of
        true -> apply(hello, Fun, _Args);
        false ->'unk'
    end.

eval([]) ->'unk';
eval([ _Head ]) -> single(_Head);
eval([ _Head | _Tail ]) ->
```

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

```
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     end,
111     if Max < NbBeer -> NbBeer;
112         true -> Max
113     end,
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 combient il a
            bu de bières
145     receive
146         {Stud,-1} -> charlotte(StudQueue,BeerManager);
147         {Stud, Nb} when Nb >= 0 ->
148             BeerManager ! {self(), Nb},
149             receive
150                 {BeerManager,ok} -> charlotte(StudQueue,BeerManager); %
                        attends que le BeerManager ait mis les infos à jour
151                 _-> "error"
152             end;
153         _-> "error"
154     end.
155
156 student() ->%TODOrand:uniform(N) génère un entier aléatoire dans l'
```

```
    intervalle [1; N].
    receive
        {From, how_much_beers} -> %récupère la requete envoyée par
            charlotte()
            From ! {self(), rand:uniform(30)}; %répond à la requête
        {From, _} ->
            From ! {self(), -1};
        _-> no
    end.

build_students(Tot) ->
    [spawn(?MODULE, student, []) || _ <- lists:seq(1, Tot)].

start() ->
    BM = spawn(?MODULE, beer_manager, [{0,-1,0,0}]), %crée le
        beerManager
    Students = build_students(98956), %génère le flot d'étudiants
    Result = charlotte(Students, BM), %lance l'analyse de charlotte
    BM ! shutdown,
    case Result of
        {Nb, Avg, Min,Max, NbStuds} ->
            io:format("Total Beers: ~p~nAverage: ~p~nMin: "
            "~p~nMax: ~p~nStudents: ~p~n",
            [Nb, Avg,Min,Max,NbStuds]),
            Result;
        _ -> "error"
    end.

% Question 5 ------------------------------------------
-module(processus).
-export([start/0, divisor_checker/1]).

busy_wait(Atom) ->
    case whereis(Atom) of
        undefined ->
            timer:sleep(500),
            busy_wait(Atom);
        _ -> ok
    end.

divisor_request(Nb, Divisor) ->
    process_flag(trap_exit, true),% little hack
    Ref = make_ref(),

    ok=busy_wait(div_check),% little hack

    div_check ! {self(), Ref, Nb, Divisor},
    receive
        {'EXIT', Pid, Reason} -> divisor_request(Nb, Divisor);
        {Ref, Response} -> Response
    after2000 ->
        timeout
    end.

random_kill(P) ->
    caserand:uniform() < P of
        true -> exit(killed);
        _ -> ok
    end.

divisor_checker(KillProb) when KillProb < 1.0 ->
    receive
        {Pid, Ref, Nb, Divisor} ->
            link(Pid),% little hack
            Res = case Nb rem Divisor of
```

```
                0 -> true;
                _ -> false
            end,
        Pid ! {Ref, Res}
    end,
    ok = random_kill(KillProb),
    divisor_checker(KillProb).

start_divisor_checker(KillProb) ->
    spawn(?MODULE, restarter, [KillProb]).

restarter(P) ->
    process_flag(trap_exit, true),
    Pid = spawn_link(?MODULE, divisor_checker, [P]),
    register(div_check, Pid),
    receive
        {'EXIT', Pid, normal} -> % divisor_checker has termined
            correctly
            ok;
        {'EXIT', Pid, shutdown} -> % not a crash, manual temrination
            ok;
        {'EXIT', Pid, _} -> % this is a crash
            restarter(P)
    end.

start() ->
    Overlay_Pid = start_divisor_checker(0.35),
    % Checks the divisor of two from 1 to 10
    Lst = [ divisor_request(X, 2) || X <- lists:seq(1,10)],
    io:format("~p~n", [Lst]),
    exit(Overlay_Pid,kill).

% Question 6 ---------------------------------------------
-module(prime_number).
-export([start/1, worker_response/3]).

random_kill(P) ->
    case
        rand:uniform() < P of
        true -> exit(killed);
        _ -> ok
    end.

is_prime(P) when P =< 1 -> false;
is_prime(P) -> is_prime(P, P-1).
is_prime(_, Curr_Divisor) when Curr_Divisor =< 1 -> true;
is_prime(P, Curr_Divisor) ->
    case P rem Curr_Divisor of
        0 -> false;
        _ -> is_prime(P, Curr_Divisor-1)
    end.

prime_worker(PKill, From, To, PidMain, Ref) ->
    Lst = prime_range(PKill, From, To),
    PidMain ! {Ref, Lst}.

prime_range(PKill, From, To) ->  prime_range(PKill, From, To, []).
prime_range(_, To, To, Lst_Res) -> lists:reverse(Lst_Res);
prime_range(PKill, From, To, Lst_Res) ->
    A = is_prime(From),
    ok = random_kill(PKill),
    case A of
        true ->
            prime_range(PKill, From+1, To, [From | Lst_Res]);
        _ ->
```

```
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,
                Still_Alive - 1);
327         _ -> worker_response__(Ref, Lst, Still_Alive)
328     end.
329
330 start(Workers) ->
331     Ref = make_ref(),
332     Pid = spawn(?MODULE, worker_response, [Workers, self(), Ref]),
333     get_prime(0.0001, Workers, 100000, Pid, Ref),
334     receive
335         {Pid, Ref, Lst} ->  io:format("~p~n", [Lst])
336     end.
```