Premiers pas en Oz

Exercice 1
Écrire en Oz une fonction Maxlist à un argument telle que {Maxlist L}, où L est une liste d'entiers non-négatifs, donne la valeur maximale de L (0 si la liste L est vide).

Correction :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TP1 Exo1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Version 1 : use a procedure

declare Max in
proc {Max L Result}
   case L of
      nil then Result=0
   [] H|R then
      local X in
         {Max R X}
         if H > X then Result=H else Result=X end
      end
   end
end

local Z in
   {Max [4 7 17 42 26] Z}
   {Browse Z}
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Version 2 : use a function

declare
fun {Max L}
   case L of
      nil then 0
   [] H|R then
      local MR = {Max R} in
         if H > MR then H else MR end
      end
   end
end

{Browse {Max [34 78 4 256]}}
```

Exercice 2
La fonction suivante pour la mise à jour d'une pair (clef, valeur) dans un arbre binaire recherche a été montrée en cours :

```
% insertion of a new pair (key, value) into a binary search tree
declare
fun {FunInsert Key Value TreeIn}
    case TreeIn
    of nil then tree(Key Value nil nil)
    [] tree(K1 V1 T1 T2) then
            if Key == K1 then tree(Key Value T1 T2)
            elseif Key < K1 then
                tree(K1 V1 {FunInsert Key Value T1} T2)
            else
                tree(K1 V1 T1 {FunInsert Key Value T2})
            end
    end
end
```

Vous trouverez ce fichier également sur le site web.

1. Écrire une fonction qui, quand son premier argument est une liste de paires (clef,valeur), renvoye un arbre binaire de recherche qui contient toutes ces paires.
2. Écrire une fonction qui, quand son premier argument est un arbre binaire de recherche et son deuxième argument une clef, renvoye la valeur stockée dans l'arbre pour cette clef.
3. Écrire une fonction qui, quand son premier argument est un arbre binaire de recherche et son deuxième argument une clef, renvoye l'arbre sans la paire avec la clef donnée.

Correction :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TP1 Exo2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% insertion of a new pair (key, value) into a binary search tree
declare
fun {FunInsert Key Value TreeIn}
    case TreeIn
    of nil then tree(Key Value nil nil)
    [] tree(K1 V1 T1 T2) then
            if Key == K1 then tree(Key Value T1 T2)
            elseif Key < K1 then
                tree(K1 V1 {FunInsert Key Value T1} T2)
            else
                tree(K1 V1 T1 {FunInsert Key Value T2})
            end
    end
end

% Q1: construct a tree from an association list
declare
fun {TreeOfList L}
```

```
   case L
   of nil then nil
   [] Key#Value|RestList then {FunInsert Key Value {TreeOfList RestList}}
   end
end

{Browse {TreeOfList [a#1 m#56 b#7 x#789]}}

% Q2: lookup in a binary search tree
declare
fun {LookUp SearchKey Tree}
   case Tree
   of nil then none
   [] tree(Key Value T1 T2) then
      if SearchKey == Key then Value
      elseif SearchKey < Key then {LookUp SearchKey T1}
      else {LookUp SearchKey T2}
      end
   end
end

{Browse {LookUp b {TreeOfList [a#1 m#56 b#7 x#789]}}}

% Q3: remove a key from a binary search tree
declare
fun {RemoveSmallest T}
  % if T is not empty, then returns a triple Y#V#T where Y is the smallest
  % (leftmost) key in T, V the value associated to Y, and T the tree without Y.
  % if T is empty, then returns none
   case T
   of nil then none
   [] tree(Y V T1 T2) then
      if T1==nil
      then Y#V#T2
      else
                  Yp#Vp#Tp = {RemoveSmallest T1}
           in
                  Yp#Vp#tree(Y V Tp T2)
      end
   end
end

declare
fun {Delete X T}
   case T
   of nil then nil
   [] tree(Y W T1 T2) then
      if X==Y
      then
                  if T2==nil then T1
                  else case {RemoveSmallest T2}
                           of Yp#Vp#Tp then tree(Yp Vp T1 Tp)
                           end
                  end
```

```
      elseif X<Y
      then tree(Y W {Delete X T1} T2)
      else tree(Y W T1 {Delete X T2})
      end
   end
end

{Browse {Delete b {TreeOfList [a#1 m#56 b#7 x#789]}}}
```

Exercice 3 (Optionnel)
Écrire la fonction de tri rapide d'une liste (quicksort) en Oz. La fonction prendra comme argument la liste à trier, et renvoie la liste triée.

Programmez l'algorithme de quicksort en utilisant les listes :

- une liste de longueur 0 ou 1 est déjà triée
- pour trier une liste d'au moins deux éléments on utilise le premier élément de liste comme *pivot*. La liste est partagée en deux sous-listes, une contenant les éléments strictement plus petits que le pivot, et l'autre les éléments qui sont égaux ou plus grands que le pivot. Puis on trie récursivement ces deux liste, et on recombine les résultats obtenus.

Indication : Il y a un constructeur de paires qui est notée #. Par exemple, 1#2 est la paire des valeurs 1 et 2. Vous pouvez facilement décomposer une paire p par une construction comme

```
local Left#Right = p in ... end
```

Correction :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TP1 Exo3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

declare
fun {Append L1 L2}
   case L1 of
      nil then L2
   [] H1|R1 then local Rs in {Append R1 L2 Rs} H1|Rs end
   end
end

declare
fun {Split Pivot L}
   case L of
      nil then nil#nil
   [] H|R then
      local
         Left#Right = {Split Pivot R}
      in
         if H<Pivot then (H|Left)#Right else Left#(H|Right) end
      end
   end
end

{Browse {Split 12 [3 78 5 12 888]}}

declare
fun {QuickSort L}
   case L of
      nil then nil
   [] H|Ls then
      local Left#Right = {Split H Ls}
      in {Append {QuickSort Left} H|{QuickSort Right}}
      end
   end
```

end

$\{Browse\ \{QuickSort\ [5\ \ 67\ \ 32\ \ 987\ \ 32\ \ 6\ \ 32\ \ 1000]\}\}$