UNIVERSITÉ D'ORLÉANS **DEUG MIAS**
IF 3.01 - Informatique **Année 2001-2002**

Révision sur les listes

Exercice 1 Notes d'examens

Un professeur de CAML souhaite faire la gestion des notes de ses élèves avec son logiciel favori. Chaque étudiant ayant une note de partiel et une note d'examen, il faut calculer la note finale de l'étudiant, qui sera soit la note d'examen, soit la moyenne des deux notes, suivant ce qui avantage le plus l'élève.
Le professeur aimerait aussi pouvoir calculer les moyennes de ses classes, et trier les listes d'élève en fonction de leur note de partiel, d'examen, ou de leur note finale.
Seulement voilà... Ce professeur n'a pas le temps de faire ça lui-même, il faut lui donner un coup de main.

```
# type etudiant = {nom: string;
                   partiel : float;
                   examen : float;
                   final : float};;
type etudiant = {
  nom : string;
  partiel : float;
  examen : float;
  final : float;
}
```

Définissons dès maintenant une classe, c'est à dire une liste d'étudiants (les notes finales sont toutes mises à zéro, puisque leur calcul sera fait par la suite...) :

```
# let classe = [{nom="Pierre" ; partiel = 08. ; examen = 13. ; final = 0.};
                {nom="Paul" ; partiel = 13. ; examen = 11. ; final = 0.};
                {nom="Jacques" ; partiel = 20. ; examen = 18. ; final = 0.};
                {nom="Jean" ; partiel = 19.5 ; examen = 20. ; final = 0.};
                {nom="Michel" ; partiel = 08. ; examen = 09. ; final = 0.}];;
val classe : etudiant list =
  [{nom="Pierre"; partiel=8; examen=13; final=0};
   {nom="Paul"; partiel=13; examen=11; final=0};
   {nom="Jacques"; partiel=20; examen=18; final=0};
   {nom="Jean"; partiel=19.5; examen=20; final=0};
   {nom="Michel"; partiel=8; examen=9; final=0}]
```

Nous allons maintenant écrire une fonction calculant leurs notes finales.

```
# let notes_finales = let note_finale toto = {toto with final=
                max (toto.examen) ((toto.examen +. toto.partiel)/.2.)} in
                List.map note_finale;;
val notes_finales : etudiant list -> etudiant list = <fun>
# let classe = notes_finales classe;;
val classe : etudiant list =
[{nom="Pierre"; partiel=8; examen=13; final=13};
 {nom="Paul"; partiel=13; examen=11; final=12};
 {nom="Jacques"; partiel=20; examen=18; final=19};
 {nom="Jean"; partiel=19.5; examen=20; final=20};
 {nom="Michel"; partiel=8; examen=9; final=9}]
```

Écrivons maintenant une fonction qui va calculer les moyennes de classe : nous utiliserons pour cela une fonction intermédiaire qui calcule la somme des éléments d'une liste, ainsi que son nombre d'éléments.

```
# let moyenne l = let rec somme l = match l with
                  []   -> (0.,0.)
                | a::m -> let (s,n) = somme r in (s+.a, n+.1.)
        in  let (x,y) = somme l in x/.y;;
val moyenne : float list -> float = <fun>
# let partiels = let partiel toto = toto.partiel in List.map partiel;;
val partiels : etudiant list -> float list = <fun>
# let examens = let examen toto = toto.examen in List.map examen;;
val examens : etudiant list -> float list = <fun>
# let finales = let finale toto = toto.final in List.map finale;;
val finales : etudiant list -> float list = <fun>
# moyenne (partiels classe);;
- : float = 13.7
# moyenne (examens classe);;
- : float = 14.2
# moyenne (finales classe);;
- : float = 14.6
```

Ne nous arrêtons pas en si bon chemin et proposons à notre cher professeur de calculer l'écart-type des notes. L'écart-type est la racine carrée de la moyenne des carrés des différences avec la moyenne. Mathématiquement, si m est la moyenne des valeurs x_i, alors l'écart-type est

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - m)^2}{n}}.$$

Cela mesure, grosso modo, la dispersion des valeurs (plus l'écart-type est petit, plus les valeurs sont « concentrées » autour de la moyenne).

```
# let ecart_type l = let m = moyenne l in
                     let f x = (x-.m)*.(x-.m) in
                     sqrt (moyenne (List.map f l));;
val ecart_type : float list -> float = <fun>
# ecart_type (partiels classe);;
- : float = 5.26877594893
# ecart_type (examens classe);;
- : float = 4.1665333312
# ecart_type (finales classe);;
- : float = 4.22374241639
```

Continuons maintenant avec le tri des étudiants suivant leurs notes. Nous utiliserons ici deux algorithmes :

– le premier, appelé *quicksort*, est très efficace en pratique, et il se défini de façon récursive comme suit : on prend la première valeur de la liste, on met au début de la liste les éléments qui sont plus petits, à la fin de la liste les éléments qui sont plus grands, et on recommence avec ces deux sous-listes.
  L'ordre utilise par ce tri sera passe en argument de la fonction, ce qui nous permettra de réutiliser cette fonction de tri dans d'autres cas.

```
# let rec separe inf a l = match l with
          []   -> ([],[])
        | b::m -> let (x,y) = separe inf a m in
                  if (inf b a) then (b::x,y) else (x,b::y);;
val separe : ('a -> 'b -> bool) -> 'b -> 'a list -> 'a list * 'a list = <fun>
# let rec quicksort inf l = match l with
          []   -> []
        | b::m -> let (x,y) = separe inf b m in
                  (quicksort inf x)@[b]@(quicksort inf y);;
val quicksort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

– Le deuxième tri, appelé *tri par sélection*, consiste à chercher dans la liste l'élément minimal, et à le mettre en tête de la liste triée, et ainsi de suite. Il nous faut donc une fonction qui cherchera l'élément maximal d'une liste, une fonction qui supprime un élément d'une liste, et la fonction de tri elle-même.

```
# let rec minimum inf l = match l with
          []   -> failwith "erreur"
        | [a]  -> a
        | a::r -> let m = minimum inf r in
                  if (inf m a) then m else a;;
val minimum : ('a -> 'a -> bool) -> 'a list -> 'a = <fun>
# let rec supprime a l = match l with
          []   -> failwith "erreur"
        | b::r -> if (a=b) then r else b::(supprime a r);;
val supprime : 'a -> 'a list -> 'a list = <fun>
# let rec tri_select inf l = match l with
          [] -> []
        | m  -> let i = minimum inf l in
                  i::(tri_select inf (supprime i l));;
val tri_select : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
```

Trions maintenant notre classe suivant la note finale. Je n'utilise que `quicksort` ici, mais c'est pareil avec `tri_select` :

```
# let inffinal toto titi = (toto.final <= titi.final);;
val inffinal : etudiant -> etudiant -> bool = <fun>
# let infpartiel toto titi = (toto.partiel <= titi.partiel);;
val infpartiel : etudiant -> etudiant -> bool = <fun>
# let infexam toto titi = (toto.examen <= titi.examen);;
val infexam : etudiant -> etudiant -> bool = <fun>
# let rec noms l = match l with
          []   -> []
        | b::m -> (b.nom)::(noms m);;
val noms : etudiant list -> string list = <fun>
# noms (quicksort inffinal classe);;
- : string list = ["Michel"; "Paul"; "Pierre"; "Jacques"; "Jean"]
# noms (quicksort infexam classe);;
- : string list = ["Michel"; "Paul"; "Pierre"; "Jacques"; "Jean"]
# noms (quicksort infpartiel classe);;
- : string list = ["Michel"; "Pierre"; "Paul"; "Jean"; "Jacques"]
```

Calculons également la note médiane... La liste étant triée, il suffit d'enlever un 'élément à chaque extrêmité jusqu'à obtenir la médiane :

```
# let rec renverse l = match l with
          [] -> []
        | a::m -> (renverse m)@[a];;
val renverse : 'a list -> 'a list = <fun>
# let rec medianetriee l = match l with
          [] -> failwith("La liste est vide")
        | [a] -> a
        | [a;b] -> a
        | a::m -> match (renverse m) with
                b::n -> medianetriee n;;
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
[]
val medianetriee : 'a list -> 'a = <fun>
# let mediane inf l = let m = quicksort inf l in
                medianetriee m;;
val mediane : ('a -> 'a -> bool) -> 'a list -> 'a = <fun>
# mediane infpartiel classe;;
- : etudiant = {nom="Paul"; partiel=13; examen=11; final=12}
mediane infexam classe;;
- : etudiant = {nom="Pierre"; partiel=8; examen=13; final=13}
# mediane inffinal classe;;
- : etudiant = {nom="Pierre"; partiel=8; examen=13; final=13}
```

Exercice 2 Les anniversaires

Ceci est une simple application de ce que nous avons fait ci-dessus, ainsi que des fonctions sur les dates que nous avions écrites lors d'un précédent TD.

```
# let today={jour=14; mois="novembre"; annee=2001};;
val today : date = {jour=14; mois="novembre"; annee=2001}
# type personne = {nom: string; naissance: date};;
type personne = { nom : string; naissance : date; }
# let famille = [{nom= "Tata Germaine";
                  naissance = {jour=13 ;mois = "juin"; annee = 1956}};
                 {nom= "Tonton Dédé";
                  naissance = {jour=1 ;mois = "avril"; annee = 1946}};
                 {nom= "Cousin Hubert";
                  naissance = {jour=29 ;mois = "fevrier"; annee = 1972}};
                 {nom= "Pépé";
                  naissance = {jour=23 ;mois = "novembre"; annee = 1908}};
                 {nom= "Mémé";
                  naissance = {jour=9 ;mois = "janvier"; annee = 1913}};];;
val famille : personne list =
  [{nom="Tata Germaine"; naissance={jour=13; mois="juin"; annee=1956}};
   {nom="Tonton Dédé"; naissance={jour=1; mois="avril"; annee=1946}};
   {nom="Cousin Hubert"; naissance={jour=29; mois="fevrier"; annee=1972}};
   {nom="Pépé"; naissance={jour=23; mois="novembre"; annee=1908}};
   {nom="Mémé"; naissance={jour=9; mois="janvier"; annee=1913}}]
```

Si on veut trier ces personnes selon leur age, rien de plus simple :

```
# let plusjeune toto titi = (inferieur titi.naissance toto.naissance);;
val plusjeune : personne -> personne -> bool = <fun>
# let rec noms (l: personne list) = match l with
          []   -> []
        | b::m -> (b.nom)::(noms m);;
val noms : personne list -> string list = <fun>
# noms (quicksort plusjeune famille);;
["Cousin Hubert"; "Tata Germaine"; "Tonton Dédé"; "Mémé"; "Pépé"]
```

NB : il a fallu redéfinir la fonction `noms` ici, car elle était de type `etudiant list -> string list` alors qu'on l'applique ici à des `personne`s et pas à des `etudiant`s.
On peut aussi trier ces personnes par ordre de date d'anniversaire :

```
# let infanniv toto titi = inferieur {toto.naissance with annee = 0}
                                     {titi.naissance with annee = 0};;
val infanniv : personne -> personne -> bool = <fun>
# quicksort infanniv famille;;
[{nom="Mémé"; naissance={jour=9; mois="janvier"; annee=1913}};
 {nom="Cousin Hubert"; naissance={jour=29; mois="fevrier"; annee=1972}};
 {nom="Tonton Dédé"; naissance={jour=1; mois="avril"; annee=1946}};
 {nom="Tata Germaine"; naissance={jour=13; mois="juin"; annee=1956}};
 {nom="Pépé"; naissance={jour=23; mois="novembre"; annee=1908}}]
```