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

Feuille d'exercices n°4

Exercice 1 Produit cartésien

```
# type 'a equation = 'a * 'a * 'a;;
type 'a equation = 'a * 'a * 'a
# type 'a systeme = ('a equation) * ('a equation);;
type 'a systeme = 'a equation * 'a equation
# let (s:float systeme) = ((1.,1.,1.),(2.,3.,4.));;
val s : float systeme = (1, 1, 1), (2, 3, 4)

# let solve (s: float systeme) = match s with
        ((a,b,c),(d,e,f)) -> let det=a*.e-.b*.d in match det with
                 0. -> (0.,0.)
               | g  -> ((e*.c -. b*.f)/.det, (-.d*.c+.a*.f)/.det);;
val solve : float systeme -> float * float = <fun>

# solve s;;
- : float * float = -1, 2
```

Exercice 2 Équation du second degré

```
# type 'a equation = 'a * 'a * 'a;;
type 'a equation = 'a * 'a * 'a
# let (s:float equation) = (1.,0.,-.2.);;
val s : float equation = 1, 0, -2

# type 'a sol = Deux of 'a*'a | Une of 'a | Infinite | Aucune;;
type 'a sol = Deux of 'a * 'a | Une of 'a | Infinite | Aucune

# let rsolve (s: float equation) = match s with
          (0.,0.,0.) -> Infinite
        | (0.,0.,c)  -> Aucune
        | (0.,b,c)   -> Une (-.c/.b)
        | (a,b,c)    -> let discr=b*.b-.4.*.a*.c in match discr with
                          0.            -> Une (-.b/.(2.*.a))
                        | d when (d>0.) -> Deux ((-.b+.sqrt(discr))/.(2.*.a),
                                                 (-.b-.sqrt(discr))/.(2.*.a))
                        | d             -> Aucune;;


# rsolve (2., 0., 4.);;
- : float sol = Aucune
# rsolve (1., 0., -.2.);;
- : float sol = Deux (1.41421356237, -1.41421356237)
```

```
# let explique (r:float sol) = match r with
          Aucune      -> "Il n'y a pas de solution."
        | Infinite    -> "Tout reel est solution."
        | Une a       -> string_of_float(a)^" est la seule solution."
        | Deux (a,b)  -> string_of_float(a)^" et "^string_of_float(b)
                         ^ " sont les deux solutions.";;
```

Exercice 3 Les rationnels

```
# type rat = int * int;;
type rat = int * int
# let rec myreduit (f:rat) (i:int) = match i with
          1 -> f
        | i -> match f with
               (a,b) -> if ((a mod i = 0) && (b mod i = 0))
                                then myreduit (a / i, b / i) (i-1)
                                else myreduit (a,b) (i-1);;
val myreduit : rat -> int -> rat = <fun>
# let reduit (f:rat) = match f with
          (a,b) when (a>b) -> myreduit f a
        | (a,b)            -> myreduit f b;;
val reduit : rat -> rat = <fun>

# reduit (-12,45);;
- : rat = -4, 15

# let cr a b = reduit (a,b);;
val cr : int -> int -> rat = <fun>

# let somme (x:rat) (y:rat) = match x with
        (a,b) -> match y with
                 (c,d) -> reduit (cr (a*d+b*c) (d*b));;
val somme : rat -> rat -> rat = <fun>
# let prod (x:rat) (y:rat) = match x with
        (a,b) -> match y with
                 (c,d) -> reduit (cr (a*c) (b*d));;
val prod : rat -> rat -> rat = <fun>

# prod (111, 27) (9,37);;
- : rat = 1, 1
# somme (111, 27) (9,37);;
- : rat = 1450, 333
```

```
# type int_or_rat = Int of int
                  | Rat of int*int;;
type int_or_rat = Int of int | Rat of int * int
# let simplify (f:rat) = match (reduit f) with
          (a,1)  -> Int a
        | (a,-1) -> Int (-a)
        | (a,b)  -> Rat (a,b);;
val simplify : rat -> int_or_rat = <fun>
# let explain a = match simplify a with
          Int a     -> string_of_int(a)
        | Rat (a,b) -> string_of_int(a)^"/"^string_of_int(b);;

# explain (somme (111, 27) (9,37));;
- : string = "1450/333"
# explain (prod (111, 27) (9,37));;
- : string = "1"
```

Exercice 4 Rationnels avec des enregistrements

```
# type rat = {num:int; den:int};;
type rat = { num : int; den : int; }
# let rec myreduit (f:rat) (i:int) = match i with
          1 -> f
        | i -> let g = (if ((f.num mod i = 0) && (f.den mod i = 0))
                          then {num=f.num / i; den=f.den / i}
                          else {num=f.num; den=f.den}) in
                myreduit g (i-1);;
val myreduit : rat -> int -> rat = <fun>
# let min a b = if a<b then a else b;;
val min : 'a -> 'a -> 'a = <fun>
# let reduit (f:rat) = myreduit f (min (abs f.num) (abs f.den));;
val reduit : rat -> rat = <fun>

# reduit {num=6216; den=2856};;
- : rat = {num=37; den=17}

# let cr a b = reduit {num=a; den=b};;
val cr : int -> int -> rat = <fun>
# let somme (f:rat) (g:rat) = cr (f.num*g.den+f.den*g.num) (f.den*g.den);;
val somme : rat -> rat -> rat = <fun>
# let prod (f:rat) (g:rat) = cr (f.num*g.num) (f.den*g.den);;
val prod : rat -> rat -> rat = <fun>
```

```
# somme {num=111; den=27} {num=9; den=37};;
- : rat = {num=1450; den=333}
# prod {num=111; den=27} {num=9; den=37};;
- : rat = {num=1; den=1}

# type int_or_rat = Int of int
                  | Rat of rat;;
type int_or_rat = Int of int | Rat of rat
# let simplify (f:rat) = if (abs f.den = 1) then Int f.num
                                            else Rat f;;
val simplify : rat -> int_or_rat = <fun>
# let explain a = match simplify a with
          Int a     -> string_of_int(a)
        | Rat a -> string_of_int(a.num)^"/"^string_of_int(a.den);;
val explain : rat -> string = <fun>

# explain (somme {num=111; den=27} {num=9; den=37});;
- : string = "1450/333"
# explain (prod {num=111; den=27} {num=9; den=37});;
- : string = "1"
```

Exercice 5 Date

```
# type date={jour:int; mois:string; annee:int};;
type date = { jour : int; mois : string; annee : int; }
# let ecris (d:date) = string_of_int(d.jour)^" "^d.mois^" "^string_of_int(d.annee);;
val ecris : date -> string = <fun>
# let d={jour=6; mois="novembre"; annee=2001};;
val d : date = {jour=6; mois="novembre"; annee=2001}
# ecris d;;
- : string = "6 novembre 2001"
```

```
#  let aj_annee (d:date) (a:int) = {d with annee=d.annee+a};;
val aj_annee : date -> int -> date = <fun>

# let mois m = match (m mod 12) with
          0 -> "janvier"
        | 1 -> "fevrier"
        | 2 -> "mars"
        | 3 -> "avril"
        | 4 -> "mai"
        | 5 -> "juin"
        | 6 -> "juillet"
        | 7 -> "aout"
        | 8 -> "septembre"
        | 9 -> "octobre"
        | 10 -> "novembre"
        | 11 -> "decembre" ;;
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
12
val mois : int -> string = <fun>
# let siom m = match String.lowercase m with
          "janvier"  -> 0
        | "fevrier"  -> 1
        | "mars"  -> 2
        | "avril"    -> 3
        | "mai"      -> 4
        | "juin"     -> 5
        | "juillet"  -> 6
        | "aout"     -> 7
        | "septembre" -> 8
        | "octobre"   -> 9
        | "novembre"  -> 10
        | "decembre" -> 11
        | _ -> failwith "erreur" ;;
val siom : string -> int = <fun>
```

```
# let aj_mois (d:date) (m:int) = let newm = (siom d.mois)+m in
                        let newa = d.annee + (newm/12) in
                        {d with annee=newa; mois = mois newm};;
val aj_mois : date -> int -> date = <fun>

# ecris d;;
- : string = "6 novembre 2001"
# ecris (aj_mois d 5);;
- : string = "6 avril 2002"
# ecris (aj_mois d 123);;
- : string = "6 fevrier 2012"
# ecris (aj_mois {jour=31; mois="janvier"; annee=2001} 1);;
-: string = "31 fevrier 2001"
```

Bon, là, il y a un petit problème... pas trop compliqué à résoudre :

```
# let nb_j_dans_m d = match siom d.mois with
          0 -> 31
        | 1 -> if ((d.annee mod 4 = 0 && not (d.annee mod 100 =0)) ||
                   (d.annee mod 400 = 0)) then 29 else 28
        | 2 -> 31
        | 3 -> 30
        | 4 -> 31
        | 5 -> 30
        | 6 -> 31
        | 7 -> 31
        | 8 -> 30
        | 9 -> 31
        | 10 -> 30
        | 11 -> 31
        | _ -> 0 ;;
val nb_j_dans_m : date -> int = <fun>
# let aj_mois (d:date) (m:int) = let newm = (siom d.mois)+m in
                        let newa = d.annee + (newm/12) in
                        let newd={d with annee=newa; mois = mois newm} in
                        let j=min d.jour (nb_j_dans_m newd) in
                        {newd with jour=j};;
val aj_mois : date -> int -> date = <fun>

# ecris (aj_mois {jour=31; mois="janvier"; annee=2001} 1);;
- : string = "28 fevrier 2001"
# ecris (aj_mois {jour=31; mois="janvier"; annee=2000} 1);;
- : string = "29 fevrier 2000"
# ecris (aj_mois {jour=31; mois="janvier"; annee=1900} 1);;
- : string = "28 fevrier 1900"
```

On peut bien entendu, de la même façon, ajouter un certain nombre de jours à une date :

```
# let aj_un_jour d = let j=d.jour + 1 in
                        if (j > nb_j_dans_m d) then {(aj_mois d 1) with jour=1}
                                                   else {d with jour=j};;
val aj_un_jour : date -> date = <fun>
```

```
# let rec aj_n_jour d i = match i with
          n when (n<=0)  -> d;
        | n -> aj_n_jour (aj_un_jour d) (n-1);;
val aj_n_jour : date -> int -> date = <fun>
# aj_n_jour d 1000;;
- : date = {jour=2; mois="aout"; annee=2004}
# aj_n_jour d 123456;;
- : date = {jour=11; mois="novembre"; annee=2339}

# let inferieur d1 d2 = ((d1.annee < d2.annee) ||
                        ((d1.annee = d2.annee) && (siom d1.mois < siom d2.mois)) ||
                        ((d1.annee = d2.annee) && (siom d1.mois =  siom d2.mois) &&
                                        (d1.jour < d2.jour)));;
val inferieur : date -> date -> bool = <fun>
```

Profitons-en pour calculer la différence entre deux dates :

```
# let rec mydifference d1 d2 i =  if (inferieur d1 d2) then (i-1)
                else mydifference d1 (aj_un_jour d2) (i+1);;
val mydifference : date -> date -> int -> int = <fun>
# let difference d1 d2 = if inferieur d1 d2 then mydifference d2 d1 0
                                            else mydifference d1 d2 0;;
val difference : date -> date -> int = <fun>
```

Calculons aussi (nous en aurons besoin dans l'exercice suivant) la différence, en mois, entre deux dates :

```
# let mydifference_mois d1 d2 = 12*(d2.annee - d1.annee) +
        (siom d2.mois - siom d1.mois);;
val mydifference_mois : date -> date -> int = <fun>
# let difference_mois d1 d2 = if inferieur d1 d2 then mydifference_mois d2 d1
                                            else mydifference_mois d1 d2;;
val difference_mois : date -> date -> int = <fun>
```

Exercice 6 Les chameaux

Il sera utile de conserver tout ce qu'on a fait dans l'exercice précédent... Je supposerai donc ici que l'on travaille à la suite de l'exercice précédent.

```
# type chameau = {nom: string; naissance : date; kpm : int; dernier_controle : date; };;
type chameau = {
  nom : string;
  naissance : date;
  kpm : int;
  dernier_controle : date;
}
# let today = {jour=6; mois="novembre"; annee=2001};;
val today : date = {jour=6; mois="novembre"; annee=2001}
# let vieux_chameau c d = (inferieur (aj_annee c.naissance 9) d);;
val vieux_chameau : chameau -> date -> bool = <fun>
# let dist_depuis_controle c d  = c.kpm * (difference_mois d c.dernier_controle);;
val dist_depuis_controle : chameau -> date -> int = <fun>
```

```
# let rec vieux_prochain_controle c d =
                        if ((difference_mois (aj_mois d 1) c.dernier_controle >= 24) ||
                            (dist_depuis_controle c (aj_mois d 1) >= 4000))
                                        then d
                                        else vieux_prochain_controle c (aj_mois d 1);;
val vieux_prochain_controle : chameau -> date -> date = <fun>
# let rec jeune_prochain_controle c d =
                        if ((difference_mois (aj_mois d 1) c.dernier_controle >= 36) ||
                            (dist_depuis_controle c (aj_mois d 1) >= 6000))
                                        then d
                                        else jeune_prochain_controle c (aj_mois d 1);;
val jeune_prochain_controle : chameau -> date -> date = <fun>
```

On peut aussi le faire de façon non récursive :

```
# let vieux_prochain_controle c = let temps_limite = 4000/c.kpm in
        if (temps_limite > 24) then aj_annee c.dernier_controle 2
                                else aj_mois c.dernier_controle temps_limite;;
val vieux_prochain_controle : chameau -> date = <fun>
# let jeune_prochain_controle c = let temps_limite = 6000/c.kpm in
        if (temps_limite > 36) then aj_annee c.dernier_controle 3
                                else aj_mois c.dernier_controle temps_limite;;
```

Dans la suite, je conserve néanmoins la première définition.

```
# let prochain_controle c = if (vieux_chameau c today)
                                 then vieux_prochain_controle c {today with jour=1}
                                 else jeune_prochain_controle c {today with jour=1};;
val prochain_controle : chameau -> date = <fun>
```

```
# let cham1 = {nom="cham1"; naissance = {jour=1; mois="mai"; annee=1996};
               kpm=200;
               dernier_controle={jour=16; mois="juin"; annee=1999}};;
val cham1 : chameau =
  {naissance={jour=1; mois="mai"; annee=1996}; kpm=200;
   dernier_controle={jour=16; mois="juin"; annee=1999}}
# let cham2 = {nom="cham2"; naissance = {jour=17; mois="aout"; annee=1992};
               kpm=150;
               dernier_controle={jour=2; mois="octobre"; annee=2000}};;
val cham2 : chameau =
  {naissance={jour=17; mois="aout"; annee=1992}; kpm=150;
   dernier_controle={jour=29; mois="octobre"; annee=2000}}
# let cham3 = {nom="cham3"; naissance = {jour=1; mois="janvier"; annee=20001};
               kpm=300;
               dernier_controle={jour=1; mois="janvier"; annee=2001}};;
val cham3 : chameau =
  {naissance={jour=1; mois="janvier"; annee=2001}; kpm=300;
   dernier_controle={jour=1; mois="janvier"; annee=2001}}
# ecris (prochain_controle cham1);;
- : string = "1 decembre 2001"
# ecris (prochain_controle cham2);;
- : string = "1 octobre 2002"
# ecris (prochain_controle cham3);;
- : string = "1 aout 2002"
```

```
# let sort inf a1 a2 a3 = if (inf a1 a2)
                then if (inf a2 a3) then (a1,a2,a3)
                                    else if (inf a1 a3) then (a1,a3,a2)
                                                        else (a3,a1,a2)
                else if (inf a1 a3) then (a2,a1,a3)
                                    else if (inf a2 a3) then (a2,a3,a1)
                                                        else (a3,a2,a1);;
val sort : ('a -> 'a -> bool) -> 'a -> 'a -> 'a -> 'a * 'a * 'a = <fun>
# let noms (c1,c2,c3) = (c1.nom, c2.nom, c3.nom);;
val noms : chameau * chameau * chameau -> string * string * string = <fun>
# let trie c1 c2 c3 =
        let inf a1 a2 = (inferieur (prochain_controle a1) (prochain_controle a2)) in
                noms (sort inf c1 c2 c3) ;;
val trie : chameau -> chameau -> chameau -> string * string * string = <fun>
# trie cham1 cham2 cham3;;
- : string * string * string = "cham1", "cham3", "cham2"
```

Exercice 7 Chameaux filtres

Commençons par une méthode sans enregistrement :

```
# let next now prev pprev = 2*now + 2*prev - 4*pprev;;
val next : int -> int -> int -> int = <fun>
# let rec nnext now prev pprev i = if i=0 then now
                        else nnext (next now prev pprev) now prev (i-1);;
val nnext : int -> int -> int -> int -> int = <fun>
# let nbcham i = match i with
          0 -> 2
        | 1 -> 3
        | 2 -> 4
        | n -> nnext 4 3 2 (n-2);;
val nbcham : int -> int = <fun>
# nbcham 50;;
- : int = 67108864
```

Cette méthode n'est pas très propre dans le sens où elle masque le fait qu'elle garde en mémoire les trois dernières valeurs. Il en va autrement avec des enregistrements :

```
# type cheptel = {act : int; un_an : int; deux_ans : int};;
type cheptel = { act : int; un_an : int; deux_ans : int; }
# let rec evolution c n = match n with
        0 -> c
        | i when (i>0) -> evolution
              {act = 2*c.act + 2*c.un_an - 4* c.deux_ans;
               un_an = c.act; deux_ans = c.un_an;} (i-1);;
val evolution : cheptel -> int -> cheptel = <fun>
# let cham i = match i with
        0 -> 2
        | 1 -> 3
        | 2 -> 4
        | n when (n>0) -> (evolution
              {act=4; un_an=3; deux_ans = 2} (n-2)).act;;
val cham : int -> int = <fun>
# cham 50;;
- : int = 67108864
```

NB : on aurait pu programmer la fonction de manière complètement récursive, c'est à dire en écrivant une fonction `cham(i)` faisant appel à `cham(i-1)`, `cham(i-2)` et `cham(i-3)`, mais le temps de calcul aurait été calamiteux :

```
# let rec cham i = match i with
          0 -> 2
        | 1 -> 3
        | 2 -> 4
        | i -> 2*cham(i-1) + 2*cham(i-2) - 4*cham(i-3);;
# cham 50;;
Interrupted.
```

J'ai interrompu ce dernier calcul au bout d'une minutes, alors que `nbcham` répondait immédiatement.

Exercice 8 Gaulois(es) filtres

```
# type sit_familiale = Celib | Marie of int;;
type sit_familiale = Celib | Marie of int
# type gaulois = {homme: bool; sit: sit_familiale; poids: int};;
type gaulois = { homme : bool; sit : sit_familiale; poids : int; }
# let impots g = if g.homme then
                match g.sit with
                          Celib -> 3*g.poids + 50
                        | Marie i -> 4*g.poids - 20*(i+2)
                            else
                match g.sit with
                          Celib -> 4*g.poids + 30
                        | Marie i -> 3*g.poids - 25*(i+2);;
val impots : gaulois -> int = <fun>

# let asterix = {homme=true; sit = Celib; poids = 50};;
val asterix : gaulois = {homme=true; sit=Celib; poids=50}
# let obelix = {homme=true; sit = Celib; poids = 300};;
val obelix : gaulois = {homme=true; sit=Celib; poids=300}
# let abraracourcix = {homme=true; sit=Marie 4; poids = 150};;
val abraracourcix : gaulois = {homme=true; sit=Marie 4; poids=150}
# let bonnemine = {homme=false; sit=Marie 4; poids=50};;
val bonnemine : gaulois = {homme=false; sit=Marie 4; poids=50}
# impots asterix;;
- : int = 200
# impots obelix;;
- : int = 950
# impots abraracourcix;;
- : int = 480
# impots bonnemine;;
- : int = 0
```