Introduction à la programmation

Les types composés

Les intervalles

On reprend l'exercice traité sur les intervalles lors des premiers exercices pour illustrer l'intérêt de créer des types composés.

1. Constuire un type `intervalle` autrement dit :
   - créer un type `intervalle` constitué de deux réels ;
   - écrire une fonction qui vérifie si deux réels $x1$ et $x2$ représentent bien un intervalle (c'est à dire que $x < y$), qui lance un message d'erreur si ce n'est pas vrai et qui sinon renvoie un objet de type `intervalle` ;
   - écrire deux fonctions `BorneInf` et `BorneSup` qui, à partir d'un intervalle *[x1,x2]* de type `intervalle` renvoient chacune un réel correspondant respectivement à $x1$ et à $x2$.
2. Concevoir et écrire en *Caml* une fonction qui teste si *[x1,x2]* de type `intervalle` représente bien un intervalle.
3. Concevoir et écrire en *Caml* une fonction qui teste si deux `intervalles` *[x1,x2]* et *[x3,x4]*sont disjoints.

Le type horaire

Cinq minutes plus tard

On peut exprimer un instant dans une journée, à la minute près, par deux entiers, le premier compris entre 0 et 23 et le second compris entre 0 et 59. Par exemple *« Cinq heures et trente minutes »* est représenté par les entiers 5 et 30. Concevoir une fonction qui, à partir d'un instant t exprimé par deux entiers, rend les deux entiers qui représenteront l'heure qu'il sera 5 minutes après t. Par exemple $(5, 30)$ devra renvoyer $(5, 35)$, mais $(3, 56)$ devra renvoyer $(4, 1)$ et enfin $(23, 59)$ devra renvoyer $(0, 4)$.

À la seconde près

À présent, nous voulons manipuler des heures à la seconde près, et étendre les opérations possibles. Pour cela, il sera souvent intéressant de convertir un instant, représenté par trois entiers $(h : m : s)$ (représentant respectivement les heures, les minutes et les secondes) en nombre de secondes depuis le début de la journée (un entier).

1. Construire le type `horaire`, autrement dit :
   - créer le type `horaire` ;
   - écrire une fonction qui à partir de trois entiers h, m et s vérifie si h est compris entre 0 et 23 et si m et s sont compris entre 0 et 59, qui lance un message d'erreur si ce n'est pas vrai et qui renvoie un objet de type `horaire` dont les heures, les minutes et les secondes sont respectivement h, m et s ;
   - écrire trois fonctions `Heures`, `Minutes` et `Secondes` qui, à partir d'un instant t de type `horaire` (de la forme h, m, s)) renvoient chacune un entier correspondant respectivement à h, m et à s.
2. Concevoir et écrire en *Caml* une fonction qui à partir d'un objet de type `horaire` et représentant un instant t, renvoie le nombre de secondes écoulés entre le début de la journée et l'instant t.
3. Concevoir et écrire en *Caml* une fonction qui fait l'opération inverse.
4. Concevoir et écrire en *Caml* une fonction qui à partir de deux instants t_1 et t_2 de type `horaire` indique si t_1 est antérieur ou non à t_2.
5. Concevoir et écrire en *Caml* une fonction qui à partir de deux instants t_1 et t_2 de type `horaire` renvoie un autre objet de type `horaire`, indiquant la durée qui sépare les deux instants.

Les vecteurs

On peut représenter un vecteur par trois réels, le premier étant son abcisse, le deuxième son ordonnée et le troisième sa côte.

1. Construire un type `vecteur`.
2. Concevoir et écrire en *Caml* une fonction qui prend en entrée trois nombres réels x, y et z qui renvoie le vecteur (x, y, z).
3. Concevoir et écrire trois fonctions `Abcisse`, `Ordonnee` et `Cote` qui, à partir d'un vecteur (x, y, z) de type `vecteur` renvoient chacune un entier correspondant respectivement à x, y et à z.
4. Concevoir et écrire en *Caml* une fonction qui prend en entrée deux vecteurs v_1 et v_2 qui renvoie le vecteur égal à la somme v_1 + v_2.
5. Concevoir et écrire en *Caml* une fonction qui prend en entrée deux vecteurs v_1 et v_2 qui renvoie le produit scalaire.
6. Concevoir et écrire en *Caml* une fonction qui prend en entrée un vecteur v et qui renvoie sa norme.
7. Concevoir et écrire en *Caml* une fonction qui prend en entrée un réel k et un vecteur (x, y, z) et qui renvoie un vecteur de composantes (kx, ky, kz).
8. Concevoir et écrire en *Caml* une fonction qui prend en entrée un vecteur (x, y, z) et qui renvoie un vecteur unitaire de même direction.

Les rationnels

Les nombres flottants ne permettent pas de représenter précisément les nombres rationnels. Par exemple le rationnel 1/3 ne peut etre représenté que sous la forme 0.333333 ce qui est une approximation qui peut se révéler gênante dans certains cas. Pour ne pas faire d'approximation, la solution est de représenter les nombres rationnels par leur numérateur et leur dénominateur, sans chercher à les diviser. Par exemple 1/3 serait représenté en *Caml* par le doublet (1,3).

1. Construire le type `fraction`.
2. Concevoir et écrire en *Caml* une fonction `ProduitFraction` qui, à partir de deux fractions q_1 et q_2, donne une nouvelle fraction égale au produit $q_1 q_2$.
3. Concevoir et écrire en *Caml* une fonction `SommeFraction` qui, à partir de deux fractions q_1 et q_2, donne une nouvelle fraction égale à la somme $q_1 + q_2$.
4. Concevoir et écrire en *Caml* une fonction `SontEgales` qui, à partir de deux fractions q_1 et q_2, renvoie `true` si les deux fractions représentent le même nombre rationnel et qui renvoie `false` sinon.
5. Concevoir et écrire en *Caml* une fonction qui à partir d'une fraction q donne une autre fraction égale à q mais irréductible.

Les complexes

On peut représenter un nombre complexe par deux réels, le premier étant sa partie réelle, et le deuxième sa partie imaginaire.

1. Construire un type `complexe`.
2. Concevoir et écrire en *Caml* une fonction qui prend en entrée deux nombres complexes c_1 et c_2 qui renvoie le nombre complexe égal à la somme $c_1 + c_2$.
3. Concevoir et écrire en *Caml* une fonction qui prend en entrée deux nombres complexes c_1 et c_2 qui renvoie le nombre complexe égal au produit $c_1 c_2$.
4. Concevoir et écrire en *Caml* une fonction qui prend en entrée un complexe c et qui renvoie sa norme.

5. Concevoir et écrire en *Caml* une fonction qui prend en entrée un complexe c et qui renvoie son argument (il s'agit de l'angle θ tel que $c = |c|.e^{i\theta}$).
6. Concevoir et écrire en *Caml* une fonction qui prend en entrée un complexe c et qui renvoie ce complexe sous la forme d'une paire norme-argument.

Les flottants

On peut représenter un nombre décimal d par deux entiers appelés *mantisse* et *exposant* de telle façon que :

$$d = mantisse * 10^{exposant}$$

(Dans cet exercice on se limitera à la base 10).
Par exemple 3,5 peut être représenté par 35 et -1 puisque $3,5 = 35 * 10^{-1}$.

1. Construire un type `flottant` capable de représenter un nombre décimal par deux entiers.
2. Concevoir et écrire en *Caml* une fonction qui étant donnée deux `flottants` d_1 et d_2 renvoie un autre `flottant` représentant le produit $d_1 d_2$.
3. En supposant que vous disposez d'une fonction `puissance` qui, à partir d'un entier x et d'un entier n renvoie l'entier x^n, concevoir et écrire en *Caml* une fonction qui à partir de deux `flottants` d_1 et d_2 renvoie un autre `flottant` représentant la somme $d_1 + d_2$.