TP 3 d'Introduction à la programmation

Exercices sur les types et la récursivité

L'algorithme d'*Euclide* pour le calcul du PGCD

Pour calculer le PGCD de deux entiers positifs a et b, nous allons utiliser l'algorithme d'*Euclide*. Celui-ci est fondé sur le fait que si l'un des deux entiers est nul le PGCD est égal à l'autre entier et que, sinon, le PGCD de deux entiers positifs a et b est le même que le PGCD de a et de $b - a$ (si toutefois $b \geq a$). Écrire l'algorithme sous forme récursive et écrire en *Caml* une fonction qui, à partir de deux entiers positifs a et b, renvoie la valeur du PGCD de ces deux entiers.

Le type rationnel

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 serait 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. Même question que précédemment, les calculs devant ici être accélérés en traitant séparément, et par filtrage, le cas où l'un des numérateurs est nul.
4. 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 $q_1 + q_2$.
5. 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.
6. Concevoir et écrire en *Caml* une fonction qui à partir d'une fraction q donne une autre fraction égale à q mais irréductible.

La somme des n premiers entiers

On peut calculer la somme des n premier entiers au moins de deux manières différentes. Une de ces deux manières correspond à un algorithme récursif. Implémenter cet algorithme par une fonction *Caml* qui, étant donné un entier positif n, renvoie la somme des n premiers entiers.

La fonction `trace`

La fonction `trace` prend en argument un nom de fonction (type `string`). Lorsque

```
# trace;;
- : string -> unit = <fun>
```

la fonction dont le nom a été donné en argument à `trace` est appelée, alors *Caml* affiche à l'écran l'argument qui lui a été donné et le résultat qui a été calculé. Prenons par exemple la fonction `suivant` définie ci-dessous :

```
# let suivant x = 1+x;;
suivant : int -> int = <fun>

# trace "suivant";;
The function suivant is now traced.
- : unit = ()
```

À présent tous les appels de la fonction `suivant` sont affichés à l'écran :

```
#suivant 5;;
suivant <-- 5
suivant --> 6
- : int = 6
```

Tracer la fonction qui calculait la somme des n premiers entiers (question précédente).

La suite de *Fibbonacci*

1. Nous avons vu deux manières différentes de calculer la suite de *Fibbonacci*. Écrire en *Caml* ces deux fonctions ;
2. Calculer la valeur du 30-ième terme de la suite de *Fibbonacci* avec les deux fonctions.
3. Tracer les deux fonctions et calculer le 5-ième terme de la suite de *Fibbonacci*. Quel est le nombre de fois où chacune des deux fonctions est appelée ?

La division entière

1. Nous cherchons à réaliser une fonction qui, à partir de deux entiers positifs a et b, renvoie le quotient de la division de a par b. On cherche à réaliser cela sans utiliser l'opérateur de division de *Caml*. Dans un premier temps, écrire l'algorithme de cette division sous la forme d'une récursion.
2. Même question, mais pour le reste de la division de a par b.