Introduction à la programmation

La récursivité

La fonction **`trace`**

La fonction `trace`

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

prend en argument un nom de fonction (type `string`). Lorsque 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 les fonctions récursives des questions suivantes.

Élever un nombre flottant à une puissance entière

1. Écrire en *Caml* une fonction qui, à partir d'un entier positif n et d'un flottant x, renvoie la valeur de x^n. Avant d'écrire cette fonction, élaborer un algorithme qui calcule cette valeur de façon récursive.
2. Même question, mais en utilisant le filtrage de n.

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 deux fonctions *Caml* dont l'une est récursive et qui, étant donné un entier positif n, renvoient la somme des n premiers entiers.

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.

Multiplication par un entier p

Dans cette question, nous cherchons à réaliser une multiplication sans avoir recours à l'opérateur de multiplication de *Caml*.

1. Écrire une fonction qui à partir de deux entiers positifs n et p calcule le produit $n \times p$ par une suite d'additions.
2. Est-ce que la fonction peut être utilisée pour deux entiers négatifs ? Est-ce qu'elle peut être utilisée avec un entier positif et un entier négatif ? Est-ce que l'algorithme peut être généralisé pour la multiplication entre flottants, ou entre un entier et un flottant ?

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 calcul de $(\cos(nx), \sin(nx))$

Écrire une fonction qui prend en entrée un entier n et une paire de valeurs réelles qui sont en fait les valeurs du cosinus et du sinus d'un certain angle x, et qui renvoie la paire $(cos(nx), sin(nx))$. Autrement dit, le deuxième argument de la fonction est une paire (a,b) telle que $a = cosx$ et $b = sinx$. Le schéma de calcul doit bien évidemment être récursif. On pourra se servir des formules de trigonométrie suivantes :

$$\begin{aligned}
\cos(nx) &= \cos((n-1)x)\cos(x) - \sin((n-1)x)\,sin(x) \\
\sin(nx) &= \sin((n-1)x)\cos(x) + \cos((n-1)x)\,sin(x)
\end{aligned}$$

Renverser un nombre quelconque

Ecrire et concevoir une fonction qui, à partir d'un entier positif n ne contenant aucun 0 dans ses chiffres, renverse les chiffres de cet entier. Par exemple l'image de 356457 par cette fonction serait 754653.

Un nombre *palindrome*

Rappelons qu'un palindrome est un nombre qui, lorsqu'on inverse l'ordre de ses chiffres ne change pas de valeur. Par exemple, 43134 est un palindrome. Nous avons vu en cours un algorithme permettant d'indiquer si un entier positif n ne contenant aucun 0 dans ses chiffres est un palindrome ou non. Cet algorithme ne s'appuie pas sur celui de la fonction renverser vue plus haut. Écrire en *Caml* une fonction qui, étant donnée un tel entier n indique s'il s'agit d'un palindrome ou non.

La suite factorielle

Définir la suite factorielle sous la sorme d'une suite récurrente et en déduire une fonction en *Caml* qui, à partir d'un entier n, renvoie la valeur de $n!$.

Les combinaisons C_n^p

Le nombre de combinaisons de p objets pris dans un ensemble de n objets différents est de :

$$C_n^p = \frac{n!}{p!(n-p)!}$$

Dans la suite, nous allons essayer plusieurs manières d'écrire une fonction qui, à partir de deux entiers positifs n et p, renvoie ce nombre de combinaisons.

1. On peut écrire cette fonction en utilisant la fonction factorielle ci-dessus.
2. En calculant C_{14}^7 par exemple, calculer le nombre de fois où le produit $7 \times 6 \times \ldots \times 2$ est calculé.
3. Exprimer C_n^p en fonction de C_{n-1}^{p-1} (sans faire intervenir C_{n-1}^p).
4. En déduire une fonction capable de calculer C_n^p à partir de p et de n.
5. Essayer cette fonction pour calculer C_4^3 : est-ce qu'on obtient la valeur attendue ?
6. Enfin on essaie d'écrire cette fonction grâce au triangle de Pascal :

$$\left\{ \begin{array}{lllr} (\forall n \in \mathbb{N}) & & & C_n^0 = 1 \\ (\forall n \in \mathbb{N}) & & & C_n^n = 1 \\ (\forall (n,p) \in \mathbb{N}^2) & | & (n > 1) and (p > 1) and (n > p) & C_n^p = C_{n-1}^{p-1} + C_{n-1}^p \end{array} \right.$$

La suite de *Fibbonacci*

La suite de *Fibonacci* est définie de la manière suivante :

$$\left\{ \begin{array}{rl} & u_0 = 1 \\ & u_1 = 1 \\ \forall n > 1 & u_n = u_{n-1} + u_{n-2} \end{array} \right.$$

1. Écrire une fonction en *Caml* qui, étant donné un entier n, calcule la valeur de la suite de *Fibbonacci* en n. Pour quelles valeurs de n est-ce que la fonction que vous avez écrite converge (aboutit à un résultat et s'arrête) ?
2. Pour la fonction qui calcule la suite de *Fibbonacci*, combien de fois la fonction *Fibbonacci* est-elle appelée sur les entiers $N-1$, $N-2$, ... $N-5$, lors du calcul de la valeur de la fonction pour un entier N ?
3. Soit la suite V_n définie de la façon suivante :

$$\left\{ \begin{array}{rl} & V_0 = (1,1) \\ & V_1 = (1,2) \\ \forall n > 1 & V_n = (u_n, u_{n+1}) \end{array} \right.$$

   où (u_n) est la suite de *Fibbonacci*. Exprimer V_{n+1} en fonction de V_n. On pourra utiliser les fonctions de projection $premier$ et $second$ qui à partir de (x, y) renvoient respectivement x ou y.
4. On cherche une nouvelle fonction qui calcule la suite de *Fibbonacci* mais où, pendant le calcul de la valeur de cette suite pour un entier N, la fonction n'est appelée qu'une seule fois sur chacune des valeurs inférieures à N. En particulier, si (u_n) est la suite de *Fibbonacci*, pour le calcul de u_n, il est inutile de calculer u_{n-1} **ET** u_{n-2} puisque u_{n-2} a déjà été déterminé pendant le calcul de u_{n-1}. Donc il est intéressant de pouvoir conserver certaines valeurs, au lieu de les recalculer. Pour cela, on commencera par écrire une fonction $FibboAux$ qui à partir d'un entier n calcule la valeur de la suite V_n.
5. En déduire une fonction qui calcule la suite de *Fibbonacci* en ne calculant chacun des termes de la suite qu'une seule fois.
6. Nous avons donc vu deux manières différentes de calculer la suite de *Fibbonacci*. Calculer la valeur du 30-ième terme de la suite de *Fibbonacci* avec les deux fonctions.
7. 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 ?