Option informatique : TP 1

Avant toute chose, la référence en ligne se trouve sur `http://caml.inria.fr/pub/docs/manual-caml-light/`. Les instructions d'installation y sont détaillées, et les installeurs (libres) sont sur `http://caml.inria.fr/caml-light/release.en.html`.
Une fois installée, la distribution de Caml-light contient un évaluateur interactif, accessible via la commande `camllight` :

```
$ camllight
# 1+1;;
- : int = 2
# float_of_int;;
- : int -> float = <fun>
```

Cet interpréteur interactif compile et évalue les expressions à la volée. Attention à bien terminer les entrées par `;;`. Le `#` désigne l'invite de commande de Caml, et la réponse comprend d'abord le type de la valeur retournée, puis la valeur elle-même.
Il est recommandé de programmer à l'aide d'un bon éditeur de texte (emacs, vim, notepad++, etc.). Cela permet de sauvegarder facilement son code source, en plus de la coloration syntaxique qui aide à lire le programme. L'extension classique des fichiers Caml est `.ml`.

1. Essayer de deviner le type et le résultat des expressions suivantes lorsque celles-ci sont correctes. Vérifier ensuite à l'aide de l'ordinateur, en corrigeant si nécessaire les erreurs de syntaxe :

   ```
   if 2>3 then 4;;
   1>2 && 5>3;;
   1.>2. || "ab"<="ac";;
   let a=1 in a=2;;
   1.0 + 4e5;;
   74/31;;
   74./.31.;;
   ```

2. Définir deux fonctions « valeur absolue » à l'aide d'un `if ... then ... else ...`, l'une opérant sur les entiers et l'autre sur les réels.
3. Écrire une fonction **somme** qui calcule à l'aide d'une boucle `for` la somme des entiers de 0 à un entier n passé en argument.
4. Écrire une fonction **fact** qui calcule à l'aide d'une boucle `for` la factorielle d'un entier n donné. La tester sur 5 et 6, vérifier qu'on obtient respectivement 120 et 720.
5. Écrire une fonction **puissance** à deux arguments qui calcule à l'aide d'une boucle `for` la puissance n-ième d'un réel x donné (n étant un entier). Vérifier que `puissance 2 10` renvoie 1024.

6. Si $x \in \mathbb{R}_+^*$, alors la suite $(u_n)_{n\in\mathbb{N}}$ définie par
   — $u_0 = 1$
   — $\forall n \in \mathbb{N} \quad u_{n+1} = \frac{1}{2}\left(u_n + \frac{x}{u_n}\right)$
   converge vers $\sqrt{x}$ (algorithme de Héron, en fait déjà connu des Babyloniens). Écrire une fonction **racine** utilisant cette méthode pour calculer une valeur approchée de $\sqrt{x}$ sur les flottants avec une précision ϵ donnée (on stoppera le calcul de u_n dès que $\text{abs}(u_n - \frac{x}{u_n}) < \epsilon$). Tester la fonction sur quelques exemples, dont 2. et 16.
7. Afficher tous les caractères standards visibles (le code ASCII est compris entre 32 et 127) à raison de 32 caractères par ligne ; le passage à la ligne sera obtenu à l'aide de la fonction `print_newline : unit -> unit`. On pourra s'aider de `char_of_int : int -> char`.
8. Écrire une fonction **miroir** qui renvoie le « miroir » d'un entier naturel donné. Par exemple,

```
# miroir 123456;;
- : int = 654321
```

9. Un entier naturel est dit parfait s'il est égal à la somme de ses diviseurs propres (ie distincts de lui-même). Écrire une fonction `parfait : int -> bool` déterminant si un entier est parfait ou non, puis trouver tous les nombres parfaits inférieurs à 1000.
10. Écrire une fonction qui résout l'équation à coefficients réels $ax^2+bx+c = 0$ en retournant le couple de réels formé par ses deux racines lorsqu'elles sont réelles, et sinon affiche un message d'erreur (utiliser la fonction `failwith`) :

```
# solve (1.,-3.,2.);;
-: float * float = 1., 2.
```

11. Écrire une fonction `maximum` qui retourne le plus grand élément d'un tableau donné, ou échoue avec `failwith` si le tableau est vide.
12. On cherche à programmer les opérations usuelles sur les polynômes (à coefficients entiers), ceux-ci étant représentés par des tableaux de longueurs variables. Ainsi, le polynôme $a_nX^n + a_{n-1}X^{n-1} + \ldots + a_0$ (de degré n) sera représenté par le tableau `[|`a_0`;`a_1`;...;`a_n`|]` (de longueur $n+1$). Programmer :
    (a) `affiche : int vect -> unit` : affichage d'un polynôme sous la forme $a_nX^n + a_{n-1}X^{n-1} + \ldots + a_0$.
    (b) `eval : int vect -> float -> float` : calcul de la valeur d'un polynôme P en un réel x.
    (c) `somme : int vect -> int vect -> int vect` : somme de deux polynômes.
    (d) `produit : int vect -> int vect -> int vect` : produit de deux polynômes.