Feuille d'exercices n°2

Cette feuille d'exercice porte sur les concepts de programmation et les éléments de langage de la section 1.2 des notes de cours (pp 5-10).

Travaux dirigés

Exercice I : Factorielle

La factorielle d'un nombre entier est définie comme :

$$\begin{cases} 0! & = & 1 \\ n! & = & n \times (n-1)! \quad \text{si } n > 0 \end{cases}$$

Q1 – Définir une fonction récursive `fact : int -> int` telle `(fact n)` donne la valeur de `n`!.

Q2 – Donner une version récursive terminale de la fonction factorielle.

Exercice II : Termes d'une suite

Soit u_n définie par: $\begin{cases} u_0 & = & 42 \\ u_{n+1} & = & 3u_n + 4 \end{cases}$

Q1 – Donnez la définition de la fonction `u : int -> int` telle que `(u n)` donne $u_{\mathtt{n}}$. Quelle hypothèse faut-il faire sur `n` ?

Q2 – En utilisant une fonction locale récursive terminale, donnez une nouvelle définition de `u : int -> int`. Cette nouvelle fonction déclenchera également l'exception `(Invalid_argument "u")` si son argument est négatif.

Q3 – On a vu en cours la fonction `iter` définie par

```
let rec iter n f a =
   if (n > 0) then (iter (n-1) f (f a))
   else a
```

Quel est le type le plus général de cette fonction ?

Q4 – Si on pose que $f(n) = 3n + 4$ on a que $u_n = f^n(42)$.

L'itérateur `iter : int -> ('a -> 'a) -> 'a -> 'a` appliqué à une fonction f, une valeur a et un entier n donne la valeur de $f^n(a)$. En utilisant `iter` donnez une nouvelle (et dernière) définition de `u`.

Exercice III : Récurrence sur un intervalle

Q1 – Donnez une définition de la fonction `sum_inter : int -> int -> int` telle que `(sum_inter a b)` donne la somme des entiers compris dans l'intervalle $[\mathtt{a}, \mathtt{b}]$. Faut-il poser des hypothèses sur les arguments ? Élaborez un jeu de tests pour cette fonction.

Q2 – Donnez la définition de la fonction `sumf_inter : (int -> int) -> int -> int -> int` telle que `(sum_inter f a b)` donne la somme `f(a)`$+\ldots+$`f(b)`

Exercice IV : Nombres premiers

Rappels: un nombre entier d est diviseur d'un nombre entier n si et seulement si il existe un nombre entier k tel que $n = d \times k$. On en déduit que $n \mod d = 0$.

Q1 – Donnez la définition de la fonction `less_divider : int -> int -> int` telle que `(less_divider i n)` donne le plus petit diviseur de `n` compris entre `i` (inclus) et `n` (exclu), s'il existe et 0 sinon. Quelles hypothèses faut-il poser pour `i` et `n` ?

Rappel: un nombre entier positif est dit *premier* si et seulement si il n'a pas de diviseur autre que 1 et lui-même. Par convention, on ne considère pas 1 comme un nombre premier.

Q2 – Déduire de la question précédente la définition de la fonction `prime : int -> bool` telle que `(prime n)` vaut `true` si et seulement si `n` est premier.

Q3 – Donnez la définition de la fonction `next_prime : int -> int` telle que `(next_prime n)` donne le plus petit nombre premier supérieur ou égal à `n`.

Cette fonction s'arrêtera-t-elle toujours ?

Q4 – On numérote les nombres premiers de cette manière: 2 a le numéro 0, 3 a le numéro 1, 5 a le numéro 2, 7 a le numéro 3, etc.

Déduire de la question précédente la définition de la fonction `nth_prime: int -> int` telle que `(nth_prime n)` donne le nombre premier de numéro `n`. On suppose `n` positif.

Travaux sur machines

Exercice V : Polynôme d'ordre 2

Q1 – Définissez la fonction `float_of_3int : (int * int * int) -> (float * float * float)` qui convertit les triplets d'entiers en triplets de flottant.

Exemple: `(float_of_3int (8,4,6))` vaut `(8.0, 4.0, 6.0)`

La bibliothèque standard de OCAML contient la fonction `float_of_int : int -> float`.

Q2 – Définissez la fonction `valeur_poly: (int * int * int) -> float -> float` qui, étant donnés entiers a, b, c et un floattnt x donne la valeur du polynôme $ax^2 + bx + c$.

Rappels

Le *discriminant* Δ d'un polynôme d'ordre 2 $ax^2 + bx + c$ est défini par $\Delta = b^2 - 4ac$. L'équation $ax^2 + bx + c = 0$ où x est l'inconnue a une solution si $\Delta = 0$, deux solutionss si $\Delta > 0$ et aucune sinon. On parlera, par abus de langage des *solutions d'un polynôme*. Les solutions d'un polynôme sont $\frac{-b+\sqrt{\Delta}}{2a}$ et $\frac{-b-\sqrt{\Delta}}{2a}$.

Q3 – Définissez la fonction `discriminant : (int * int * int) -> int` qui, étant donné trois entiers a, b et c, calcule le discriminant du polynôme $ax^2 + bx + c$.

Q4 – Définissez une fonction `nb_solution : int -> int` qui, étant donné le discriminant Δ d'un polynôme $ax^2 + bx + c$, donne le nombre de solutions de l'équation $ax^2 + bx + c = 0$.

Q5 – Définissez la fonction `solutions` qui donne les solutions du polynôme $ax^2+bx = c$. Les paramètres de cette fonction sont a; b et c. Si le polynome admet deux solutions, la fonction donne la couple formé de ces deux solution; s'il n'en a qu'une, elle donne un couple contenant deux fois la solution; s'il n'en a pas, par convention, elle donne, le couple $(0.0, 0.0)$. Quel est le type de cette fonction ?

Exercice VI : Tester la récursion terminale

Le problème de Bâle consiste à déterminer la valeur exacte de la somme de la série convergente :

$$\sum_{i=1}^{n} \frac{1}{i^2}$$

Ce terme $\mathcal{P}(n)$ peut être approché, au rang n, à l'aide la récurrence :

$$\begin{cases} \mathcal{P}(1) & = & 1 \\ \mathcal{P}(n) & = & \frac{1}{n^2} + \mathcal{P}(n-1) \quad \text{si } n > 1 \end{cases}$$

Q1 – Donnez une fonction récursive calculant le n-ième terme de la série $\mathcal{P}(n)$

- Que vaut $\sqrt{6 * \mathcal{P}(n)}$ pour n = 500, n = 5 000 , n = 50 000 ?

- Que se passe-t-il si n = 500 000?

Q2 – Donnez à présent une définition récursive terminale de cette fonction.

- Que vaut $\sqrt{6 * \mathcal{P}(n)}$ pour n = 500 000? Pouvez-vous justifier ce comportement?

Exercice VII : Approximation de la racine carrée

On peut obtenir une valeur approchée de la racine carrée d'un nombre a en utilisant les termes de la suite

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

Le choix de x_0 est arbitraire. par exemple, on peut prendre 1.

Q1 – Définir la fonction `f` telle que $f(x) = \frac{1}{2}\left(x + \frac{a}{x}\right)$. Quelle est le type de `f` ?

Q2 – Une première manière d'utiliser cette suite pour obtenir une valeur approchée de la racine carrée est de calculer le n-ième terme de la suite x_n.

Définir la fonction `sqrt_n (n: int) (a: float) (x0: float) : float` qui calcule le n-ième terme de la suite x_n en prenant `x0` comme valeur de x_0.

Utilisez cette fonction avec des valeurs croissantes de `n` et observez.

Point fixe

Le point fixe d'une fonction f et un x tel que $f(x) = x$. Cela donne une autre manière d'utiliser la suite des x_n pour déterminer la racine carrée: itérer le calcul de $x_0, x_1, x_2, etc.$ jusqu'à trouver un x_n tel que $x_n = x_{n-1}$. C'est un point fixe puisque qu'on aura que $x_n = (\texttt{f}\ x_{n-1}) = x_{n-1}$. On a alors atteint un *point fixe* et l'on obtiendra pas de meilleure approximation.

Attention: les calculs sur les flottants ne sont pas exacts. On testera donc l'égalité *"à epsilon près"*.

Q3 – Définissez la fonction `eq_eps (e: float) (x: float) (y: float) : bool` qui donne `true` si et seulement si la distance entre `x` et `y` est inférieure (strictement) à `e`.

La fonction de la bibliothèque standard `abs_float : float -> float` donne la valeur absolue de son argument.

Q4 – Définissez la fonction `sqrt_x (e: float) (a: float) (x0: float) : float` qui utilise la méthode du point fixe pour trouver une valeur approchée de la racine carrée de `a`. Le premier terme de la suite sera `x0`. On teste l'égalité *"à* `e` *près"*.

Utilisez cette fonction avec des valeurs décroissantes de `e` et observez.