ENSEIRB-Matmeca
2019-2020

1ère année - Filière Info
PG104

TD n°6 - Fonctionnelles

Exercice 1: Premières fonctionnelles

Dans cet exercice, on vérifie que les fonctions sont des citoyens *de première classe* en Scheme :

- Nommage : de lui donner un nom et, de la stocker dans une liaison ;
- Transmission : de le passer comme paramètre à une fonction ;
- Construction : d'en renvoyer comme retour d'une fonction.

1. Écrire la fonction qui, étant donnée une fonction f passée en paramètre, renvoie $f(1)$. L'appliquer à la fonction $f(x) = 3x^2 + 4.7$ définie sous forme de lambda-expression.
2. Écrire la fonction qui, à un nombre x donné, renvoie la fonction $y \mapsto x * y$.

Exercice 2: Composition de fonction : Scheme vs C

1. Écrire en Scheme une fonction `compose` qui prenne en argument deux fonctions f et g et qui renvoie la composition de f et g
2. Essayer de faire la même chose en langage C.

Exercice 3: Utilisation de letrec

1. En utilisant deux fonctions, écrire l'expression conduisant au calcul de :

$$(1+\sqrt{2})(1+\sqrt{2}+\sqrt{3})(1+\sqrt{2}+\sqrt{3}+\sqrt{4})$$

2. Faites la même chose que précédemment en définissant une seule fonction au top-level et en utilisant une fonction annexe définie localement avec `letrec`.

Exercice 4: Sommes d'entiers

1. Écrire la fonction `sigma (f n p)` qui calcule $\sum_{i=n}^{p} f(i)$, pour une fonction $f : \mathbb{R} \to \mathbb{R}$.
2. Tester votre fonction avec des fonctions nommées et des fonctions anonymes.

Exercice 5: Généralisation des sommes d'entiers

1. Écrire $\mathrm{OP}_{i=n}^{p} f(i)$, pour une fonction $f : X \subseteq \mathbb{R} \to Y \subseteq \mathbb{R}$ et une opération OP binaire quelconque. Faire une version récursive.
2. Définir la fonction factorielle à l'aide de la fonction précédente.
3. Définir une fonction qui approxime e en utilisant le fait que :

$$\lim_{n\to\infty} (1 + \sum_{i=1}^{n} 1/i!) = e.$$

Exercice 6: Racines - formules de Newton

La récurrence suivante utilise la méthode de Newton pour approximer la racine carrée d'un nombre x, en construisant une suite (a_n) :

$$\begin{cases} a_0 & = 1 \\ a_n & = \frac{1}{2}\left(a_{n-1} + \frac{x}{a_{n-1}}\right) \end{cases}$$

Pour écrire `racine-carree`, on écrira quatre fonctions :

1. la fonction `test-arret?` testant la convergence du calcul par la formule $|a_n^2 - x| < \epsilon$;
2. la fonction `suivant` calculant le terme suivant de la série ;
3. la fonction `racine-rec` lançant le calcul avec comme paramètres `start`, `x` et `eps` ;
4. la fonction `racine-carree` correspondant au calcul avec comme unique paramètre `x`.

Pour calculer la racine cubique de x, il existe un schéma de facture équivalente :

$$\begin{cases} a_0 & = 1 \\ a_n & = \frac{1}{3}\left(2a_{n-1} + \frac{x}{a_{n-1}^2}\right) \end{cases}$$

5. Utiliser la même technique d'écriture que pour la fonction `racine-carree` pour écrire la fonction `racine-cubique`.
6. Quel est l'intérêt d'une telle décomposition en sous-fonctions ?

Exercice 7: Tri générique

Écrire une fonction de tri (en reprenant la fonction réalisée sur les listes au TD précédent, `sort-numbers`) prenant en paramètre une liste à trier et la fonction permettant de comparer deux valeurs de la liste. On se limitera au tri par insertion.

Exemple : `(sort-gen '(8 6 3 5 1 2) >= ) ;; → (1 2 3 5 6 8)`

Pour aller plus loin... **Exercice 8: Méthode du point fixe générique**

1. Écrire une fonction `fixpoint` généralisant le calcul de la racine carrée vu en TD2 par la méthode du point fixe. Cette fonction doit admettre 3 paramètres : une fonction de $\mathcal{D} \to \mathcal{D}$, un prédicat sur $\mathcal{D} \times \mathcal{D}$ indiquant l'arrêt du calcul, et un élement de départ du domaine $\mathcal{D}$.
2. Écrire à nouveau la fonction racine carrée en utilisant `fixpoint`.

Pour aller plus loin... **Exercice 9: Sagesse de l'Antiquité**

Considérons une base de données de personnages fournis avec leurs noms, dates de naissance et de mort. Pour simplifier [1], nous nous limiterons à une simple liste de triplets stockée dans un fichier accessible à l'adresse `http://www.labri.fr/perso/renault/working/teaching/schemeprog/files/higher.txt`

1. Sauvegarder le fichier en question, et le charger à l'intérieur de DrRacket à l'aide de la fonction `file→list`.

```
(define db (file→list "path_to_higher.txt"))
```

1. La bibliothèque standard de Racket possède des connecteurs pour les bases de données classiques (c.f. par exemple `https://docs.racket-lang.org/db/using-db.html`).

Dans l'état actuel, les données de ce fichier ne sont pas structurées autrement que des triplets `(string number number)`. Pour leur donner une sémantique, nous allons utiliser des structures équivalentes à celles vues dans le langage C :

```
(struct person (name birth death))
```

La fonction `struct` définit un ensemble de fonctions permettant de manipuler des valeurs structurées de type `person`, dans ce cas des personnes possédant 3 champs.

```
(person "Celestin" 1666 1794) ;; → #<person> / Constructor of a person

(define duck (person "Celestin" 1666 1794))
(person-name  duck) ;; → "Celestin" / Name accessor
(person-birth duck) ;; → 1666       / Birth year accessor
(person-death duck) ;; → 1794       / Death year accessor
```

2. Transformer la base de données `db` en une base de données `dbp` contenant uniquement des valeurs de type `person`.
3. Extraire de cette base de données la liste des noms dans l'ordre alphabétique.
4. Extraire de cette base la personne née le plus tôt chronologiquement.
5. Extraire de cette base la durée de vie la plus longue de toutes ces personnes.