ENSEIRB-Matmeca
2019-2020

1ère année - Filière Info
PG104

TD n°2 - Fonctions et récursivité

Exercice 1: Factorielle et complexité

Dans cette feuille, les définitions de fonctions sont encouragées à être écrites en séparant les différents cas, les faisant ainsi ressembler à des définitions mathématiques, comme par exemple :

$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{si } x < 0 \\ 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases}$$

Ce type de définition induit naturellement un style de programation *déclaratif* pour écrire le code de ses fonctions, à savoir sous forme d'équations.

1. Rappeler la définition mathématique de la fonction factorielle.
2. Écrire l'équation de récurrence vérifiée par sa fonction de complexité. Quelle est la complexité de cette fonction ?

Exercice 2: Écriture de fonctions récursives

1. Soit $\mathrm{sum}(x, y)$ la fonction définie par :

$$\mathrm{sum}(x, y) = \sum_{k=x}^{y} k$$

   Écrire l'équation reliant $\mathrm{sum}(x, y)$ et $\mathrm{sum}(x + 1, y)$, et la compléter de manière à définir entièrement sum par des équations.
2. Écrire une fonction `somme-inter` qui, appelée avec deux entiers a et b comme arguments, renvoie $\mathrm{sum}(a, b)$. Ainsi, `(somme-inter 1 5)` vaut 15, `(somme-inter 5 1)` vaut 0.
3. De même, écrire une fonction `somme-carres` qui calcule la somme des carrés des entiers situés entre ses deux arguments.

   Remarque : penser à utiliser la fonction `sqr` de la bibliothèque standard.

Exercice 3: Calcul du pgcd

Écrire une fonction `pgcd` donnant le plus grand diviseur commun de ses deux arguments (entiers positifs), en se basant sur l'algorithme d'Euclide.

Exercice 4: Calculs de puissance

1. Définir une fonction `puissance` suivant le principe suivant :
   - $x^0 = 1$,

- $x^n = x.x^{n-1}$ si $n > 0$.

4. Écrire la version améliorée de cette fonction de complexité logarithmique.

Rappel : dans tous les cas, penser à tester vos fonctions sur plusieurs (au moins deux) exemples.

Exercice 5: Pair / Impair

Supposons vouloir définir deux fonctions `pair` et `impair` qui déterminent la parité de leur argument suivant le principe suivant :

- 0 est pair (et donc non impair),
- $n > 0$ est pair (respectivement, impair) si $n - 1$ est impair (respectivement, pair).

1. Définir ces fonctions en Scheme grâce à des conditionnelles `if`.
2. Écrire ces fonctions à la main l'aide de formules logiques (utiliser les connecteurs booléens usuels de conjonction, disjonction et négation).
3. Réécrire en Scheme une deuxième version sans instructions conditionnelles.

Exercice 6: Utilisation de trace

La commande `trace`, définie dans la bibliothèque `racket/trace`, est une fonction très utile permettant de visualiser tous les appels à une fonction dont le nom est passé en paramètre. Elle est particulièrement intéressante lorsqu'il s'agit de visualiser le déroulement des appels dans une fonction récursive.

1. Rajouter la commande `(require racket/trace)` au début de votre code.
2. Ajouter `(trace pair)` et `(trace impair)` dans votre code, puis `(pair 10)`.
3. Ajouter `(trace pgcd)`, puis `(pgcd 5 3)`.

Pour retirer le mode trace sur une fonction, il suffit de faire `(untrace ⟨nom-fonction⟩)`. Dans la suite de cette feuille, essayez cette commande sur les différentes fonctions que vous aurez écrites.

Exercice 7: Récursivité arithmétique

1. Écrire une fonction `plus` récursive , qui appelée avec deux entiers a et b comme arguments, effectue la somme de a et b en utilisant l'idée que pour ajouter b, on additionne "b fois" la valeur 1.
2. Écrire une fonction `produit` récursive, qui appelée avec deux entiers a et b comme arguments, effectue le produit de a par b en utilisant l'idée que $a \times b$ revient à faire $a + a + \cdots + a + a$ (b fois).

Pensez à tester vos fonctions, notamment avec des valeurs recouvrant l'ensemble des comportements possibles de vos fonctions.

Exercice 8: Suite de Syracuse

En réutilisant les fonctions `pair` et `impair` écrites dans l'exercice précédent, écrire la fonction qui donne la longueur d'un vol de la suite de Syracuse partant de la valeur n, sachant que cette longueur est donnée par :

- syracuse$(1) = 0$

- $\text{syracuse}(n) = 1 + \text{syracuse}(n/2)$ si n est pair
- $\text{syracuse}(n) = 1 + \text{syracuse}(1 + 3 \times n)$ si n est impair

Pour tester : `(syracuse 7)` renvoie 16.

Pour aller plus loin... **Exercice 9: Chaînes de caractères**

Écrire les fonctions Scheme sur les chaînes de caractères permettant de décider si :

1. une chaîne est un palindrome, de manière itérative ; (utiliser `for`, `set!` et `when`)
2. une chaîne est un palindrome, de manière récursive ;
3. une chaîne est l'anagramme d'une autre, de manière récursive ;
4. une chaîne est l'anagramme d'une autre, de manière itérative.