

Programmation applicative – L2

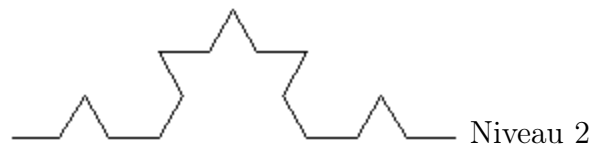
Première série d'exercices supplémentaires pour ceux qui ont tout fini...

H. Chahdi {hatim.chahdi@umontpellier.fr}
T. Lorieul {titouan.lorieul@umontpellier.fr}
J. Thiebaut {jocelyn.thiebaut@umontpellier.fr}

1 Le flocon de Von Koch

Soit la courbe de Von Koch définie de la façon suivante : On considère un triangle équilatéral. C'est le niveau 0. Ensuite, on prend chaque côté et on le divise en trois. Le tiers du milieu est remplacé par deux segments de même longueur. C'est le niveau 1. On réitère le processus sur chaque segment...Voici quelques niveaux illustrant la transformation associée à chaque segment :

_____ Niveau 0





Exercice 1 Soit a_0 la longueur du trait au niveau 0 ; que vaut a_1 la longueur du trait global au niveau 1 en fonction de a_0 ?

De façon générale, que vaut a_i la longueur du trait au niveau i en fonction de a_{i-1} la longueur du trait au niveau $(i - 1)$? puis en fonction de a_0 ? Que se passe-t-il pour a_i quand i tend vers l'infini ?

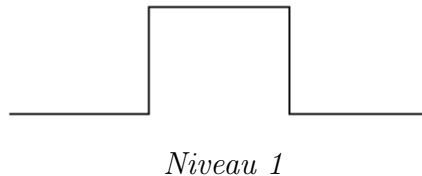
Exercice 2 Ecrire la fonction (`von_koch prof a`) où l'on passera en argument la longueur a du trait de niveau 0 et le niveau de profondeur ou l'on veut s'arrêter. Comme dans toute fonction récursive, il faut réfléchir :

- à la condition d'arrêt. et ce que l'on fait alors
- à la définition du problème au niveau i en fonction du même problème au niveau $(i-1)$

Exercice 3 Écrire une fonction (`prepa-von-koch`) qui nettoie l'écran et positionne la tortue de telle façon à exécuter la courbe de Von Koch sur toute la surface de la fenêtre de dessin.

Exercice 4 En utilisant la primitive `DrScheme (sleep N)` qui met le système en attente pendant N secondes (par exemple (`sleep 3`) met le système en attente pendant 3 s), écrire une fonction (`anim-von-koch n`) qui fasse apparaître successivement la courbe de von koch au niveau 0, 1, 2, jusqu'à n , en laissant 3 s entre les apparitions de chaque niveau.

Exercice 5 En faire de même, avec la courbe de Von Koch modifiée :



Exercice 6 Modifier les fonctions précédentes pour pouvoir commencer le flocon à partir d'un polygone régulier à un nombre quelconque n de côtés, passé en paramètre.

2 Généralisation

Les mathématiciens ont commencé à démontrer des théorèmes de géométrie en dimension 2, puis en dimension 3, puis... en dimension N . La **généralisation** étant une activité importante de la démarche scientifique, pourquoi ne pas s'en servir en programmation ? Souvent, il vous arrivera d'écrire un algorithme adapté à un problème particulier et de vous apercevoir qu'à peu de frais, il vous est possible d'en déduire un algorithme plus général qui pourra être ré-utilisé par la suite dans d'autres situations.

Autre intérêt de la généralisation : résoudre des problèmes particuliers. C'est peut être surprenant, mais il est parfois plus facile [en programmation comme en maths] de résoudre des problèmes généraux que des problèmes particuliers!

Essayons de généraliser la fonction factorielle. Pourquoi se borner à ne faire que des multiplications? Remplaçons celle-ci par une opération binaire associative et commutative `combiner` d'élément neutre `null-value` tel que `(combiner x null-value) = (combiner null-value x) = valeur de x`. La fonction d'accumulation est :

```
(define (accumulate combiner null-value a b)
  (if (> a b)
      null-value
      (combiner a (accumulate combiner null-value (+ a 1) b)))))
```

```
> (accumulate * 1 10 20)      ; calcule le produit:  $10 \times 11 \times \dots \times 20$ 
⇒ 6704425728000
```

Exercice 7

1. Écrivez une définition de la fonction factorielle `fact-accu` qui utilise cette fonction d'accumulation.
2. Écrivez une définition de la fonction `somme-accu` qui à entier n associe la somme des entiers de 1 à n , en utilisant cette fonction d'accumulation.
3. Écrivez une définition de la fonction `liste-accu` qui à entier n associe la liste des entiers de 1 à n , en utilisant cette fonction d'accumulation.

Exercice 8 Cette généralisation est insuffisante pour calculer une somme de carrés par exemple. On veut pouvoir indiquer le terme de la fonction à "accumuler".

1. Introduisez une fonction `term` en paramètre qui s'appliquera à chaque élément de l'intervalle $[a, b]$, et écrivez la fonction plus générale `(accumulate-term combiner null-value term a b)` telle que :

```
> (accumulate-term + 0 square 10 20) ; calcule la somme:  $10^2 + 11^2 + \dots + 20^2$ 
⇒ 2585
```

2. Redéfinissez les fonctions `fact-accu`, `somme-accu`, `liste-accu` avec cette nouvelle version de la fonction d'accumulation.
3. Calculez une valeur approchée de la constante $e = 2.718\dots$ en exprimant le développement limité $1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{15!} + \frac{1}{n!} \dots$. Comparez avec la valeur de `(exp 1)` où `exp` est prédéfini en Scheme. Pour quelle valeur de n approche-t-on e avec une précision inférieure à 10^{-11} ?
4. Calculez le produit de tous les nombres impairs de $[1, 50]$.
5. Calculez le produit de tous les nombres pairs de $[1, 50]$.
6. Calculez le produit de tous les nombres égaux à 1 modulo 3 de $[1, 50]$.

Exercice 9 La question précédente soulève la question : pourquoi se limiter à aller de 1 en 1 dans les appels de la fonction `accumulate` ? Nous pourrions parcourir l'intervalle $[a, b]$ 2 en 2, ou de 5 en 5, etc. Et d'ailleurs, pourquoi avec un pas constant ? Utilisons plutôt une fonction `next` pour calculer le suivant d'un nombre dans l'intervalle.

1. Écrivez (`accumulate-term-next` combiner `null-value` `term` `a` `next` `b`) plus générale encore où `next` est une fonction qui à partir de `a` calcule le suivant de `a` dans l'intervalle $[a, b]$.
2. Redéfinissez les fonctions `fact-accu`, `somme-accu`, `liste-accu` avec cette nouvelle version de la fonction d'accumulation.
3. Calculez une valeur approchée de la somme $S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \dots$. Pour une grande valeur devinez-vous vers quoi tend la limite ?

Exercice 10 Enfin, toujours à l'aide de la fonction d'accumulation, écrivez une fonction `pi-product` qui calcule le produit : $\frac{2 \times 2 \times 4 \times 4 \times 6 \times 6 \times 8 \times \dots \times 2n \times 2n}{1 \times 3 \times 3 \times 5 \times 5 \times 7 \times 7 \times \dots \times (2n-1) \times (2n+1)}$ qui converge vers $\frac{\pi}{2}$. Précisez pour quelle valeur de `n` la convergence s'effectue avec une précision inférieure à 10^{-3} . Comparer avec l'approximation de `e` faite précédemment.