TP n^{o}2 : Fonctions récursives

Q1 Pour la chauffe

Écrire une fonction `somme(n)` qui fait la somme des entiers de 0 jusqu'à n

(a) définissez les cas de bases (ou conditions d'arrêt) pour une ou plusieurs valeurs particulières de n

(b) trouvez une définition récursive (avec un appel de `somme` donc) pour les autres valeurs de n

Q2 Les classiques

- La factorielle

  (a) écrire une fonction récursive non terminale `fac1(n)` qui calcule la factorielle de n ainsi qu'une fonction `fac1_cpt(n)` qui compte le nombre d'appels récursifs effectués

  (b) faites une version récursive terminale des ces deux fonctions : `fac2(n)` et `fac2_cpt(n)`

  (c) mettez ces deux versions dans un module `Factorielle` (la majuscule est importante) : c'est un fichier ne contenant pas de `main` et commençant par

```
module Factorielle where
... definitions de vos fonctions...
```

  (d) créez 2 programmes qui appellent ces 2 versions de factorielles. Un programme est un fichier contenant par exemple

```
import Factorielle
main = do
    print $ "Factorielle non Terminale de 100 = " ++
  show(fac1(100))
```

  puis comparez les performances de ces deux versions en terme de CPU et d'appels récursifs. Que constatez vous ?

- La suite de Fibonnaci : procédez de la même manière avec 2 versions de la fonction qui calcule cette suite et 2 programmes qui les testent.

- Idem avec la fonction `pow2_test` qui teste si un nombre entier est une puissance de 2.

Q3 Shuffle

Le *mélange américain* est une méthode pour mélanger un paquet de cartes : on le partage en deux sous-paquets (pas forcément égaux) et on insère les cartes de l'un dans l'autre pour former le nouveau paquet mélangé. L'imbrication des cartes n'obéït qu'à une seule règle : l'ordre des cartes dans chacun des deux sous-paquets doit subsister dans le paquet final.

(a) Ecrire une fonction qui étant donné les deux nombres de cartes $n1$ et $n2$ dans **deux** sous-paquets P_1 et P_2, calcule le nombre de paquets finaux différents qu'il est possible d'obtenir avec ce mélange.

(b) Même question mais avec **trois** sous-paquets.

Indice :

(a) si un des paquets n'a qu'une seule carte, $n1 = 1$ par exemple dans P_1, alors on peut l'insérer de $n2 + 1$ manières dans le paquet P_2 (*cas1*)

(b) le calcul est symétrique si c'était le paquet P_2 qui n'avait qu'une seule carte (*cas2*)

(c) si les 2 paquets possèdent plus d'une carte alors

  i. on peut faire le calcul pour l'insertion de la première carte d'un paquet dans l'autre, ce qui revient au calcul de *cas1* ou *cas2*

  ii. puis ajouter le calcul de la situation symétrique

  iii. puis faire de même avec les restes des paquets

Q4 Multiplication russe

La technique de multiplication dite *russe* consiste à faire le produit de 2 nombres entiers naturels seulement avec

- des additions
- des multiplications par 2
- des divisions par 2

Son intérêt est que l'on n'est pas obligé de connaitre ses tables de multiplication (à part celle de 2). Elle repose sur la propriété suivante :

$$x \times y \begin{cases} = 0 & si\ x = 0 \quad et \quad y \geqslant 0 \\ = \frac{x}{2} \times (2y) & si\ x > 0\ ,\ y \geqslant 0\ et\ x\ est\ pair \\ = \frac{x-1}{2} \times (2y) + y & si\ x > 0\ ,\ y \geqslant 0\ et\ x\ est\ impair \end{cases}$$

Écrire une fonction qui calcule le produit de deux nombres entiers avec la *multiplication russe.*

Q5 Fonctions rigolotess

(a) Pour planter rapidement votre machine, programmer la fonction d'Ackermann

$$A(m,n)\begin{cases} = n+1 & si\ m=0 \\ = A(m-1,1) & si\ m>0\ et\ n=0 \\ = A(M-1,A(m,n-1)) & si\ m>0\ et\ n>0 \end{cases}$$

C'est une fonction qui croît très vite, elle est notamment utilisée pour tester les implémentations de langage de programmation.

(b) Programmer la fonction $f91$ de McCarthy

$$f91(n)\begin{cases} = n-10 & si\ n>100 \\ = f91(f91(n+11)) & si\ n\leqslant 100 \end{cases}$$

Cette fonction renvoie

- 91 si $n \leqslant 100$
- $n-10$ sinon

(c) Les puissances itérées de Knuth :
En 1976 Donald Knuth a inventé une notation permettant d'écrire de très grands nombres entiers.

- Knut a d'abord défini un opérateur double flèche pour une puissance itérée

$$a \uparrow\uparrow b = \underbrace{a^{a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}}}_{b\ exemplaires\ de\ a}$$

- puis l'opérateur triple flèche :

$$a \uparrow\uparrow\uparrow b = \underbrace{a{\uparrow\uparrow}a{\uparrow\uparrow}a\ldots a{\uparrow\uparrow}a}_{b\ exemplaires\ de\ a}$$

- puis l'opérateur quadruple flèche :

$$a{\uparrow\uparrow\uparrow\uparrow}b = \underbrace{a{\uparrow\uparrow\uparrow}a{\uparrow\uparrow\uparrow}a\ldots a{\uparrow\uparrow\uparrow}a}_{b\ exemplaires\ de\ a}$$

- et ainsi de suite. On peut noter que

$$a{\uparrow^n}b = \underbrace{a{\uparrow^{n-1}}a\ldots a{\uparrow^{n-1}}a}_{b\ exemplaires\ de\ a} = a{\uparrow^{n-1}}\ \underbrace{a{\uparrow^{n-1}}a\ldots a{\uparrow^{n-1}}a}_{b-1\ exemplaires\ de\ a} = a{\uparrow^{n-1}}(a{\uparrow^n}(b-1))$$

Programmer une fonction `fleche` qui calcule $a{\uparrow^n}\ b$. Pour cela vous pouvez suivre la méthode classique de l'écriture d'une fonction récursive :

i. Définissez les cas de base (ou conditions d'arrêt), quand $n=0$ ou $n=1$ ou $b=0$ ...

ii. Ecrivez ensuite la formule récursive de votre fonction