TDP2 : Récursivité et chaînes récursives

TAPS : Techniques and Algorithms for Problem Solving

★ **Exercice 1:** Code mystère.

▷ **Question 1:** Calculez les valeurs renvoyées par la fonction f pour n variant entre 1 et 5.

▷ **Question 2:** Quelle est la fonction mathématique vue en cours que f() calcule ?

▷ **Question 3:** Quelle est la complexité algorithmique du calcul ?

```
def f(n:Int):Int = {
  def lambda(n:Int, a:Int, b:Int):Int = {
    if (n == 0) {
      return a;
    } else {
      return lambda(n-1, b, a+b);
    }
  }
  return lambda(n, 0, 1);
}
```

Notez que tout le travail est fait par la fonction interne lambda, et la fonction f ne sert qu'à donner une valeur initiale aux arguments a et b, qui servent d'accumulateur. Il s'agit là d'une technique assez classique en récursivité.

★ **Exercice 2:** Soit le type `List[Char]` muni des opérations suivantes :

Nil	La liste vide
head **::** *tail*	Construit une liste constituée de head, suivi de la liste tail
list.**head**	Récupère le premier caractère de la liste (pas défini si list est la chaîne vide)
list.**tail**	Récupère la liste privée de son premier élément (idem)

Écrire les fonctions suivantes en précisant les préconditions nécessaires. Notez que ces exercices sont aussi accessibles dans la PLM.

▷ **Question 1:** *longueur* : `List[Char]` $\mapsto$ `Int`
retourne le nombre de lettres composant la chaîne

▷ **Question 2:** *est_membre* : `List[Char]` $\times$ `Char` $\mapsto$ `Boolean`
retourne `true` ssi le caractère fait partie de la chaîne

▷ **Question 3:** *occurence* : `Char` $\times$ `List[Char]` $\mapsto$ `Int`
retourne le nombre d'occurences du caractère dans la chaîne

▷ **Question 4:** *tous_differents* : `List[Char]` $\mapsto$ `Boolean`
retourne `true` ssi tous les membres de la chaine sont différents

▷ **Question 5:** *supprime* : `Char` $\times$ `List[Char]` $\mapsto$ `List[Char]`
retourne la chaine privée de toutes les occurences du caractère.
Si le caractère ne fait pas partie de la chaîne, celle-ci est inchangée.

▷ **Question 6:** *deuxieme* : `List[Char]` $\mapsto$ `Char`
retourne le deuxième caractère de la chaîne

▷ **Question 7:** *dernier* : `List[Char]` $\mapsto$ `Char`
retourne le dernier caractère de la chaîne

▷ **Question 8:** *saufdernier* : `List[Char]` $\mapsto$ `List[Char]`
retourne la chaine privée de son dernier caractère

▷ **Question 9:** *nieme* : `List[Char]` $\times$ `Int` $\mapsto$ `Char`
retourne le nieme caractère de la chaîne

▷ **Question 10:** *npremiers* : `List[Char]` $\times$ `Int` $\mapsto$ `List[Char]`
retourne les n premiers caractères de la chaîne

▷ **Question 11:** *nderniers* : `List[Char]` $\times$ `Int` $\mapsto$ `List[Char]`
retourne les n derniers caractères de la chaîne

▷ **Question 12:** *retourne* : `List[Char]` $\mapsto$ `List[Char]`
retourne la chaine lue en sens inverse

▷ **Question 13:** *concat* : `List[Char]` $\times$ `List[Char]` $\mapsto$ `List[Char]`
retourne les deux chaines concaténées

▷ **Question 14:** *min_ch* : `List[Char]` $\mapsto$ `Char`
retourne le caractère le plus petit de la chaîne

On considère l'ordre lexicographique, et on suppose l'existance d'une fonction min(a,b).

▷ **Question 15:** *croissante* : $\begin{cases} \texttt{List[Char]} \mapsto \textit{booléen} \\ \text{retourne si la chaine est croissante (dans l'ordre lexicographique)} \end{cases}$

▷ **Question 16:** *nnaturels* : $\begin{cases} \texttt{Int} \mapsto \texttt{List[Int]} \\ \text{retourne une chaine formée des n premiers entiers naturels} \end{cases}$

Dans un premier temps, on construira $\{n, n-1, n-2, \ldots, 3, 2, 1\}$ avant de construire $\{1, 2, 3, \ldots, n\}$.

▷ **Question 17:** *palindrome* : $\begin{cases} \texttt{List[Char]} \mapsto \textit{booléen} \\ \text{retourne VRAI si la chaine est un palindrome} \end{cases}$

Un palindrome se lit indifféremment de droite à gauche ou de gauche à droite. Exemple : « Esope reste et se repose ». On peut ignorer les espaces.

▷ **Question 18:** *anagramme* : $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \textit{booléen} \\ \text{retourne VRAI si les chaines sont des anagrammes l'une de l'autre} \end{cases}$

Une anagramme d'un mot est un autre mot obtenu en permutant les lettres. Exemples : «chien» et «niche» ; «baignade» et «badinage» ; «Séduction», «éconduits» et «on discute».

▷ **Question 19:** *union* : $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne une chaîne formée de toutes les lettres de ch1 et ch2, sans doublons} \end{cases}$

On peut supposer dans un premier temps que ch1 et ch2 ne contiennent pas de doublons.

▷ **Question 20:** $difference$: $\begin{cases} \texttt{List[Char]} \times \texttt{List[Char]} \mapsto \texttt{List[Char]} \\ \text{retourne toutes les lettres de ch1 ne faisant pas partie de ch2} \end{cases}$

★ **Exercice 3:** En respectant les noms et constructions récursives définies dans l'exercice précédent, implémentez chacune des fonctions en Python en utilisant les structures de listes (List). La transformation d'une chaîne de caractères en une liste se fait simplement par l'appel de la fonction *list*(). Pour chaque fonction, écrivez et lancez un ensemble de tests unitaires, comptez le nombre d'appels récursifs et le nombre d'opérations réalisées. Les plus téméraires pourront mesure les temps d'exécution.