

★ **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 :

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

Écrire les fonctions suivantes. Vous préciserez les préconditions nécessaires.

- ▷ **Question 1:** `longueur` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{Int} \\ \text{retourne le nombre de lettres composant la chaîne} \end{array} \right.$
- ▷ **Question 2:** `est_membre` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{Char} \mapsto \text{Boolean} \\ \text{retourne true ssi le caractère fait partie de la chaîne} \end{array} \right.$
- ▷ **Question 3:** `occurrence` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{Char} \mapsto \text{Int} \\ \text{retourne le nombre d'occurrences du caractère dans la chaîne} \end{array} \right.$
- ▷ **Question 4:** `tous_différents` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{Boolean} \\ \text{retourne true ssi tous les membres de la chaîne sont différents} \end{array} \right.$
- ▷ **Question 5:** `supprime` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{Char} \mapsto \text{List[Char]} \\ \text{retourne la chaîne privée de la première occurrence du caractère.} \end{array} \right.$   
Si le caractère ne fait pas partie de la chaîne, celle-ci est inchangée.
- ▷ **Question 6:** `deuxieme` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{Char} \\ \text{retourne le deuxième caractère de la chaîne} \end{array} \right.$
- ▷ **Question 7:** `dernier` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{Char} \\ \text{retourne le dernier caractère de la chaîne} \end{array} \right.$
- ▷ **Question 8:** `sauf_dernier` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{List[Char]} \\ \text{retourne la chaîne privée de son dernier caractère} \end{array} \right.$
- ▷ **Question 9:** `nieme` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{Int} \mapsto \text{Char} \\ \text{retourne le nieme caractère de la chaîne} \end{array} \right.$
- ▷ **Question 10:** `npremiers` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{Int} \mapsto \text{List[Char]} \\ \text{retourne les n premiers caractères de la chaîne} \end{array} \right.$
- ▷ **Question 11:** `nderniers` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{Int} \mapsto \text{List[Char]} \\ \text{retourne les n derniers caractères de la chaîne} \end{array} \right.$
- ▷ **Question 12:** `retourne` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{List[Char]} \\ \text{retourne la chaîne lue en sens inverse} \end{array} \right.$
- ▷ **Question 13:** `concat` :  $\left\{ \begin{array}{l} \text{List[Char]} \times \text{List[Char]} \mapsto \text{List[Char]} \\ \text{retourne les deux chaînes concaténées} \end{array} \right.$
- ▷ **Question 14:** `min_ch` :  $\left\{ \begin{array}{l} \text{List[Char]} \mapsto \text{Char} \\ \text{retourne le caractère le plus petit de la chaîne} \end{array} \right.$

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

- ▷ **Question 15:** *croissante* :  $\begin{cases} \text{List}[\text{Char}] \mapsto \text{booléen} \\ \text{retourne si la chaîne est croissante (dans l'ordre lexicographique)} \end{cases}$
- ▷ **Question 16:** *nnaturels* :  $\begin{cases} \text{Int} \mapsto \text{List}[\text{Char}] \\ \text{retourne une chaîne formée des } n \text{ premiers entiers naturels} \end{cases}$   
 Dans un premier temps, on construira  $\{n, n-1, n-2, \dots, 3, 2, 1\}$  avant de construire  $\{1, 2, 3, \dots, n\}$ .
- ▷ **Question 17:** *palindrome* :  $\begin{cases} \text{List}[\text{Char}] \mapsto \text{booléen} \\ \text{retourne VRAI si la chaîne est un palindrome} \end{cases}$   
 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* :  $\begin{cases} \text{List}[\text{Char}] \times \text{List}[\text{Char}] \mapsto \text{booléen} \\ \text{retourne VRAI si les chaînes sont des anagrammes l'une de l'autre} \end{cases}$   
 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* :  $\begin{cases} \text{List}[\text{Char}] \times \text{List}[\text{Char}] \mapsto \text{List}[\text{Char}] \\ \text{retourne une chaîne formée de toutes les lettres de ch1 et ch2, sans doublons} \end{cases}$   
 On peut supposer dans un premier temps que ch1 et ch2 ne contiennent pas de doublons.
- ▷ **Question 20:** *difference* :  $\begin{cases} \text{List}[\text{Char}] \times \text{List}[\text{Char}] \mapsto \text{List}[\text{Char}] \\ \text{retourne toutes les lettres de ch1 ne faisant pas partie de ch2} \end{cases}$