# DM D'INFORMATIQUE (Option) n°1

À rendre le mercredi 02/05/18
Vous êtes encouragés à écrire et tester les fonctions sur ordinateur et à imprimer votre code.

⋆ ⋆ ⋆

Ce problème concerne l'étude des mots de Dyck et de leur lien avec le dénombrement des arbres binaires et les nombres de Catalan.

Dans tout le sujet, les fonctions demandées devront être récursives et n'utiliser aucun élément de programmation impérative (références, types mutables, boucles, ...).

## I Mots de Dyck

Dans tout le sujet, un « mot » sera une suite de parenthèses ouvrantes et fermantes. Pour simplifier l'écriture des programmes, on utilisera un type de parenthèses représentées par les lettres `A` et `B`.

```
type lettre = A | B;;

type mot = lettre list;;
```

Par exemple, `[B; B; A; B]` est un mot correspondant à ))().

**Définition A**

On définit le langage de Dyck, appelé également langage des mots bien parenthésés, de la façon suivante :

- Le mot vide est bien parenthésé. On pourra le noter $\varepsilon$ par la suite.
- Si $u$ et $v$ sont des mots bien parenthésés, alors $AuB$ et $uv$ sont des mots bien parenthésés.

Par la suite, on notera $\mathcal{L}$ ce langage.

**Exemple B**

> Les mots de Dyck correspondent à la suite des parenthèses pouvant intervenir dans une expression mathématique syntaxiquement correcte. Par exemple, l'expression $2 + ((x + y) \times ((x + 2) \times z))$ fournit le mot bien parenthésé (()((()))). Le mot donné en exemple avant la définition n'est pas bien parenthésé.

**Question 1.** Montrer rigoureusement que $ABAABB \in \mathcal{L}$.

Pour $u$ un mot quelconque, on note $|u|_A$ (resp. $|u|_B$) le nombre de $A$ (resp. de $B$) présents dans $u$.

**Question 2.** Écrire une fonction `compteA : mot -> int` qui compte le nombre de $A$ dans un mot donné en argument.

**Question 3.** Montrer que $u \in \mathcal{L}$ est équivalent à ($|u|_A = |u|_B$ et pour tout préfixe $v$ de $u$, $|v|_A \geq |v|_B$).

**Question 4.** En déduire une fonction `dyck : mot -> bool` qui teste si un mot est un mot de Dyck. La fonction ne devra parcourir le mot qu'une seule fois (on ne fera pas nécessairement appel à `compteA`).

**Question 5.** On cherche à compléter un mot mal parenthésé de la manière suivante :

- Une parenthèse fermante non ouverte est complétée par une parenthèse ouvrante juste avant.
- Une parenthèse ouvrante non fermée est complétée par une parenthèse fermante à la fin du mot.

Par exemple, le mot $ABBAAAB$ sera complété en $ABABAAABBB$.
Écrire une fonction `complete : mot -> mot` qui effectue cette complétion.

Pour $n \in \mathbb{N}$, on pose $\mathcal{L}_n$ l'ensemble des mots de Dyck de taille $2n$ et $C_n = \text{Card}(\mathcal{L}_n)$.

**Question 6.** En décrivant complètement $\mathcal{L}_k$ pour $k \in [\![0; 3]\!]$, déterminer $C_0$, $C_1$, $C_2$ et $C_3$.

**Question 7.** Soit $n \in \mathbb{N}^*$ et $u = u_1u_2\dots u_{2n} \in \mathcal{L}_n$. En considérant le plus petit entier $k \geq 2$ tel que $|u_1\dots u_k|_A = |u_1\dots u_k|_B$, Montrer que $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$.

**Question 8.** Cette question a pour objectif de calculer le terme général $C_n$.
On définit un **scrutin** de taille $2n+1$ comme une suite $\alpha = a_1\dots a_{2n+1}$ telle que pour tout $i$, $a_i \in \{-1;1\}$ et $\sum_{i=1}^{2n+1} a_i = 1$. Le terme « scrutin » vient de l'étude d'une élection entre deux candidats. Un scrutin est dit **strict** si de plus, pour tout $k \in [\![1;2n+1]\!]$, $\sum_{i=1}^{k} a_i > 0$. Enfin, on appelle **rotation** d'un scrutin $\alpha = a_1\dots a_{2n+1}$ un autre scrutin de la forme $\rho_k(\alpha) = a_k a_{k+1}\dots a_{2n+1}a_1a_2\dots a_{k-1}$, pour $k \in [\![1;2n+1]\!]$. Pour toute la question, on considère $\alpha$ un scrutin de taille $2n+1$.

a) Montrer que les $2n+1$ rotations de $\alpha$ sont toutes distinctes.

b) Montrer par récurrence sur $n$ qu'il existe une rotation de $\alpha$ qui est un scrutin strict.

c) Montrer qu'aucune autre rotation n'est stricte.

d) Déterminer le nombre de scrutins stricts de taille $2n+1$.

e) En déduire la valeur de $C_n$.

# II Arbres binaires

On considère les arbres binaires quelconques définis par le type usuel :

```
type arbre = Vide | N of arbre * arbre;;
```

**Question 9.** Écrire une fonction `taille : arbre -> int` calculant la taille d'un arbre.

**Question 10.** Dessiner tous les arbres de taille 0, 1, 2 et 3, puis montrer que le nombre d'arbres binaires de taille $n$ est $C_n$.

**Question 11.** En s'inspirant de la question 7, écrire une fonction bijective `arbre_dyck : mot -> arbre` qui construit un arbre binaire de taille $n$ à partir d'un mot de Dyck de taille $2n$.

**Question 12.** Écrire sa fonction réciproque `dyck_arbre : arbre -> mot`. On autorisera dans un premier temps l'opérateur `@` de concaténation, puis on écrira une version sans effectuer de concaténation.

**Question 13.** Déterminer la complexité temporelle de la fonction `dyck_arbre` sans concaténation.

⋆ ⋆ ⋆