Document de travail pour le TD7
- reconnaissance d'expressions parenthésées - recherche avec backtrack -

Question 1- reconnaissance d'expressions parenthésées

Le travail à effectuer consiste à reconnaître des expressions correctement parenthésées. Nous traiterons deux cas distincts.

Premier cas- Les expressions contiennent **un seul type de parenthèses**, par exemple () ou {} ou []. Dans ce cas un compteur des parenthèses restant à fermer suffit.
Ce compteur doit être initialisé à zéro. Lors de la lecture séquentielle de l'expression à analyser, le caractère en cours d'analyse peut être :

- une parenthèse ouvrante, il faut alors incrémenter le compteur ;
- une parenthèse fermante, il faut alors vérifier que le compteur n'est pas nul (si c'était le cas, il manque une parenthèse ouvrante) et décrémenter le compteur ;
- un autre caractère qui est alors ignoré.

En fin de lecture de l'expression, le compteur doit être revenu à zéro.

Second cas- Les expressions peuvent contenir **plusieurs types de parenthèses**. Pour simplifier la comparaison des parenthèses ouvrantes et fermantes (dont les types doivent se correspondre), nous travaillerons en utilisant les lettres minuscules comme parenthèses ouvrantes et les majuscules comme parenthèses fermantes (A ferme a, B ferme b, etc...).

Dans ce cas, il est nécessaire de connaître non seulement le nombre mais le type des parentèses restant à fermer : une pile va permettre de conserver cette information. Pour implémenter la pile, nous utiliserons un tableau de caractères et un indice sommet. Le sommet est initialisé à zéro (indiquant que la pile est vide). Le premier élément sera placé à l'indice 1, etc...

> Une **pile** (imaginer une pile de livres), est une structure de données dans laquelle le premier élément à disposition est celui qui a été placé sur la pile en dernier.
> On parle de structure **LIFO** (Last In Fisrt Out).
> Les opérations de base sur les piles sont :
> `empiler` pour ajouter un élément,
> `estvide` pour tester si au moins un élément est présent sur la pile,
> `sommet` pour consulter l'élément immédiatement disponible sur la pile,
> `dépiler` pour retirer l'élément immédiatement disponible sur la pile.

Lors de la lecture séquentielle de l'expression à analyser, le caractère en cours d'analyse peut être :

- une parenthèse ouvrante, il faut alors empiler cette parenthèse ;
- une parenthèse fermante, il faut alors vérifier que la pile n'est pas vide (si c'était le cas, il manque une parenthèse ouvrante), vérifier que le sommet d epile est du même type que cette parenthèse fermante (si c'était le cas, le parenthésage est incohérent) et dépiler.

En fin de lecture de l'expression, la pile doit être vide.

Pour programmer en Java, vous disposez :

- du type String pour l'expression à analyser :
  `String analyse ;`
  `analyse.length()` pour récupérer le nombre de caractères de la chaîne,
  `analyse.charAt(i)` pour récupérer le ième caractère (ils sont numérotés à partir de 0)
- sur les caractères des méthodes :
  char cc ;
  Character.isLowerCase(cc) résultat booléen indiquant les lettres minuscules
  Character.toUpperCase(cc) résultat de type char, qui est la majuscule correspondant à cc

Question 2- recherche avec backtrack

Le travail à effectuer consiste à rechercher de façon récursive la solution à un problème. Chaque étape consiste à faire un choix (pour une solution partielle) jusqu'à avoir construit une solution globale ou être en échec (aucun choix ne reste possible). L'exemple emblématique des algorithmes de backtrack est le problème du « placement de reines sur un échiquier ». Nous allons traiter l'exemple de la construction d'une solution au « *compte est bon* ». Dans ce jeu, les candidats disposent de *n* valeurs entières strictement positives constituant leur *plaque de jeu* et d'un entier qui est la *cible* à atteindre. Ils peuvent utiliser toute combinaison des entiers de leur plaque par les opérateurs arithmétiques (addition, soustraction, multiplication, division sans reste). Chaque entier de la plaque ne peut être utilisé qu'une seule fois.

Questions préliminaires :

- pour une plaque de *n* entiers, quel est le nombre maximum d'opérations possibles ?
- étant donnés *n* entiers, quelle valeur maximum peut être construite ?