INF2010 – Structures de données et algorithmes
Hiver 2017
Travail Pratique 2
Structures de données séquentielles

Objectifs :

- Implémenter une file à partir d'un tableau et d'une liste chainée.
- Implanter un algorithme pour la vérification des parenthèses d'une s-expression Lisp.
- Implanter un algorithme pour de résolution des s-expressions Lisp à l'aide d'une pile.

1. Implémentation d'une File (2.5 points).

Vous devez compléter l'implémentation d'une file. Pour rappel, une file est une structure FIFO (first-in-first-out). Un élément peut donc seulement être ajouté à la fin de la file et seul l'élément en tête de file peut être retiré.
Vous allez implémenter une file en utilisant les deux structures de données suivantes :

1.1. File à partir d'un tableau (1.25):

Un tableau est une séquence de cases auxquelles on accède par leur index et qui contiennent les données de la file à raison d'une donnée par case.

1.2. File à partir d'une liste chainée (1.25):

Une liste chaînée est une séquence de nœuds chaînés, c'est-à-dire d'objets distincts qui contiennent les données de la liste, à raison d'une donnée par nœud. Les implantations de listes chaînées requièrent donc généralement l'utilisation d'une classe interne `Node` contenant un champ destiné aux données et un autre destiné au chaînage.

- **Remarque :**

Vous trouverez dans les fichiers `ArrayQueue.java` et `LinkedListQueue.java` la description des méthodes à compléter. La méthode `main` du fichier `QueueMain.java` vous permettra de tester vos deux classes.

2. Manipulation de piles (2.5 points)

Une pile est un conteneur qui implémente le protocole dernier entré, premier sorti (LIFO, last in, first-out). C'est l'une des structures de données les plus utilisées en informatique, notamment pour la récursivité, le traitement des expressions arithmétiques et les parcours d'arbres et de graphes.

Dans cette partie on souhaite utiliser une pile pour évaluer des s-expressions[1] (symbolic expression) du langage de programmation Lisp[2]. Dans une expression Lisp, on écrit les opérateurs avant les opérandes, comme l'indique les exemples ci-dessous. Une telle expression est nommée « s-expression ». Notez que dans le cadre ce TP, les opérateurs considérés sont : +, -, * et /.

- **Exemple :**

  (3+4) *7 **s'écrit** (*(+3 4)7)

  (5+5)*((4/2) - (4-2))*3 **s'écrit** (*(+5 5)(-(/4 2)(-4 2))3)

- **Remarque :**

Dans les exercices qui suivent, vous allez compléter les méthodes de la classe « `Lisp.java` ». La méthode `main` du fichier `LispMain.java` vous permettra de tester vos méthodes.

2.1. Exercice 1 (1):

Dans cet exercice nous souhaitons vérifier si une s-expression Lisp est équilibrée avant de l'évaluer. Complétez la méthode « isEquilibre » en utilisant une pile `java.util.Stack` pour implanter un algorithme qui permet de vérifier si une s-expression Lisp (avec des parenthèses) est équilibrée. C'est-à-dire que pour chaque parenthèse ouverte, il doit y'a voir une parenthèse fermante. N'utilisez qu'une **seule** pile et les méthodes suivantes : `pop()`, `push()`, `peek()`, `empty()`.

- **Exemple :**

-L'expression (* 3 (+ 4 5 6)2) est équilibrée car toute parenthèse ouverte est fermée.

-L'expression (* 3(- (+ 4 5 6)(/ 6 3)))) n'est pas équilibrée car on 4 parenthèses ouvertes et 5 parenthèses fermantes.

2.2. Exercice 2 (1.5):

Complétez la méthode « solve » en n'utilisant qu'une **seule** pile (`java.util.Stack`) pour implanter un algorithme de résolution des s-expressions Lisp, et que les méthodes suivantes : `pop()`, `push()`, `empty()`, `peek()`. Pour simplifier on ne considère que les nombres réels positifs.

[1] https://fr.wikipedia.org/wiki/S-expression

[2] https://en.wikipedia.org/wiki/Lisp_(programming_language)

Instructions pour la remise

Le travail doit être fait par équipe de 2 personnes et doit être remis via Moodle au plus tard :

- 14 Février avant 23h59 pour le groupe Mercredi (B2).
- 21 Février avant 23h59 pour groupe Mercredi (B1).

Veuillez envoyer vos fichiers .java **seulement**, dans un **seul répertoire**, le tout dans une archive de type *.**zip** (et seulement **zip**, pas de **rar**, **7z**, etc) qui portera le nom : **inf2010_lab1_MatriculeX_MatriculeY.zip**, où MatriculeX < MatriculeY.

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.