INF5171: Laboratoire #4
Répartition statique vs. dynamique des tâches entre les *threads*

5 octobre 2017
13h30–15h30
PK-S1565

Ce laboratoire comporte deux parties :

a. Voir pages suivantes.

b. Exercices 5.7 (p. 23) et 5.12 (p. 47) des notes de cours :
`http://www.labunix.uqam.ca/~tremblay/INF5171/Chapitres/ch5-patrons-programmation.pdf`

Première partie

Le but de ce laboratoire est de mieux comprendre la distinction entre répartition **statique** par blocs des tâches entre les *threads* et répartition **dynamique**.

---

1 Obtenir le code à compléter

Obtenez une copie des fichiers pour le labo en exécutant la commande suivante sur `japet` :

```
$ git clone http://www.labunix.uqam.ca/~tremblay/git/Palindromes.git
```

2 Ce que vous devez faire

2.1 Méthodes à compléter

Vous devez tout d'abord **compléter** diverses méthodes :

a. `String#palindrome?`

   Détermine si une chaine est un palindrome, i.e., un mot ou une suite de mots dont l'ordre des lettres reste le même qu'on lise de gauche à droite ou de droite à gauche, par exemple :

```
>> "kayak".palindrome?
=> true
>> "".palindrome?
=> true
>> "xyz".palindrome?
=> false
```

   De plus, pour permettre des palindromes plus «riches», on va ignorer les caractères non alphabétiques ainsi que la casse des caractères :

```
>> "A man, a plan, a cat, a canal --- Panama?".palindrome?
=> true
```

b. `palindromes_seq?`, `palindromes_par_static?` et `palindromes_par_dynamic?`

   Ces méthodes sont appelées par l'intermédiaire de la méthode (*dispatcher*) `palindromes?`, laquelle reçoit en argument 0, 1 ou plusieurs chaines et retourne un tableau de booléens qui indique si chacune des chaines est un palindrome ou non.

   - `palindromes_seq?` : Version séquentielle.
   - `palindromes_par_static?` : Version parallèle utilisant l'approche de votre choix, mais avec une répartition **statique** des tâches entre les *threads*.
   - `palindromes_par_dynamic?` : Version parallèle utilisant l'approche de votre choix, mais avec une répartition **dynamique** des tâches entre les *threads*.

Les cibles du `makefile` pour les tests sont les suivantes :

```
tests_seq

tests_static_true
tests_dynamic_1
tests_dynamic_10

tests_par         # Toutes les versions.
```

2.2 Mesures et analyse des performances

Une fois les trois (3) méthodes `palindromes_{seq,par_static,par_dynamic}?` mises en oeuvre, vous devez effectuer des mesures de performances pour deux séries de données, puis comparer les résultats obtenus pour les accélérations.

Pouvez-vous expliquer pourquoi on obtient de tels résultats? Pour ce faire, vous devrez examiner le fichier `mesurer_temps.rb` et comprendre le rôle de l'argument passé sur la ligne de commande — utilisé via `ARGV[0]` dans la méthode `generer_chaines`.

Les cibles du `makefile` pour les mesures de performances sont les suivantes :

```
bm1
bm2
```

L'exécution des commandes liées à ces cibles vont générer des graphes d'accélérations, `graphe_1.png` et `graphe_2.png`.

3 Informations additionnelles

- L'appel suivant permet de déterminer si le `i`^{e} caractère de la chaine `ch` est une lettre :

  ```
  ch[i] =~ /[a-zA-Z]
  ```

  Sa négation (le caractère n'est pas une lettre) s'exprime comme suit :

  ```
  ch[i] !~ /[a-zA-Z]
  ```

- Les méthodes `upcase` et `downcase` permettent d'obtenir la version majuscule ou minuscule d'une chaine :

  ```
  >> "abCD00".upcase
  => "ABCD00"
  >> "abCD00".downcase
  => "abcd00"
  ```