INF600A: Laboratoire #6
Développement d'une application en ligne de commandes avec `gli` et son DSL : `minised`

Jeudi 22 novembre 2018
13h30–15h30
PK-S1565

Le but de ce labo est de vous familiariser avec l'utilisation du *gem* `gli`, qui définit un DSL pour décrire des applications en ligne de commandes, de style «`git`», avec leurs options, arguments et commandes. L'objectif est aussi de vous familiariser avec l'organisation typique des projets en Ruby — répertoires `bin`, `lib`, `test`, `test_acceptation`, etc.

Pour obtenir les fichiers pour ce laboratoire :

```
$ git clone http://www.labunix.uqam.ca/~tremblay_gu/git/minised.git
```

Note : Première chose à faire après avoir téléchargé les fichiers :

```
$ cd minised
$ bundle install --path vendor/bundle
```

Ce qui est fourni

- `bin/minised` : Le programme principal, qui utilise `gli` et son DSL pour définir une application en ligne de commandes qui émule **une version simplifiée de `sed`**. Ce fichier est un de ceux que vous devez compléter/modifier.
- `lib` : Répertoire contenant divers fichiers utilisés pour la mise en œuvre des fonctionnalités de l'application. Le fichier principal, que vous devez compléter/modifier, est `lib/minised/minised.rb`.
- `test` : Répertoire contenant des tests **unitaires**, spécifiés avec `MiniTest`.
- `test_acceptation` : Répertoire contenant des tests **d'acceptation**, spécifiés eux aussi avec `MiniTest`.
- `Rakefile` : Un fichier qui permet d'automatiser l'exécution des tests. Les principales cibles pour ce labo — **avec la commande «`rake`»** — sont les suivantes :
  - `print_test` et `print_test_acceptation`
  - `sub_test` et `sub_test_acceptation`

Ce que vous devez faire

a. Complétez la mise en œuvre de la méthode `MiniSed.print`.

   Pour ce faire, vous devez **compléter le *squelette* de méthode déjà définie** dans le fichier `lib/minised/minised.rb`.

   Cible pour les **tests unitaires** :

   ```
   rake print_test
   ```

   Rappel : Avec `sed` (et `minised`), toute ligne provenant du flux d'entrée est émise sur le flux de sortie. De plus, une ligne qui *matche* le motif est aussi émise par l'action `print`. Donc, avec l'action `print`, chaque ligne qui *matche* le motif est émise deux (2) fois.[1]

b. Complétez la description de la **commande** `print`, partiellement définie dans le fichier `bin/minised`.

   Ici, l'objectif est que les **tests d'acceptation** associés à cette commande réussissent, tests définis avec `MiniTest` dans le fichier `test_acceptation/print_test.rb`.

   Cible pour les tests d'acceptation :

   ```
   rake print_test_acceptation
   ```

   Suggestion : Pour simplifier le lancement des tests, modifiez la tâche `:default` dans le `Rakefile`.

[1] Dans le «vrai» `sed`, la commande `print` est habituellement utilisée avec l'option «`--quiet`» (ou la forme courte équivalente «`-n`»), qui n'émet rien en sortie, **sauf les lignes qui *matchent* le motif et dont l'action associée est** `p`. Toutefois, pour simplifier le probème, dans le cadre de cet exemple/labo, cette option n'est pas mise en œuvre.

Dans les exemples de `sed` vus en cours, la substitution est présentée comme suit :

```
sed 's/motif/ch/' ...
```

Ceci a pour effet de transformer, dans toutes les lignes, la première occurrence de `motif`, une expression régulière, par la chaine `ch`. Si on indique un `g` à la fin (après le dernier «/»), alors ce sont toutes les occurrences qui sont remplacées ; et aucune substitution n'a lieu si la ligne ne *matche* pas `motif`.

Or, la forme exacte et complète pour la substitution est plutôt la suivante (avec possiblement un `g` à la fin) :

```
sed '/motif/s/motif_a_remplacer/chaine_de_remplacement/' ...
```

Cette opération substitue alors `motif_a_remplacer` par `chaine_de_remplacement`, **mais uniquement dans les lignes qui *matchent*** `motif`. Voici un exemple :

```
$ cat foo.txt
123 abc
abc
3 abc

$ sed 's/[0-9]/X/g' foo.txt
XXX abc
abc
X abc

$ sed '/[1-2]/s/[0-9]/X/g' foo.txt
XXX abc
abc
3 abc
```

c. Ajoutez une méthode `MiniSed.sub`, semblable à `MiniSed.substitute` mais avec le comportement décrit ci-haut.

   Pour ce faire, vous devez ajouter une nouvelle méthode dans le fichier `lib/minised/minised.rb`.

   Cible pour les **tests — unitaires** :

   ```
   rake sub_test
   ```

d. Ajoutez la description de la **commande** `sub` dans le fichier `bin/minised`.

   Cible pour les tests — d'acceptation :

   ```
   rake sub_test_acceptation
   ```

e. Dans `bin/minised`, si vous examinez le code qui définit les commandes `delete` et `print`, vous constaterez… qu'il y a beaucoup de code qui se répète — le code n'est pas **DRY** ☹

Comment pourrait-on éviter cette répétition de code?

Indice : Définissez une méthode qui permet de spécifier/décrire une commande sans option et utiliser **l'envoi dynamique** de message (i.e., `send`) :

```
def commande_sans_option( commande, nb_arguments: 1 )
  ...
end
```