TP n°2

Introduction à `jflex`

Dans ce TP, nous utilisons `jflex` (voir http://jflex.de/docu.html et le cours).

Exercice 1 Échauffement

1. Récupérer sur moodle le fichier `blabla.jflex` et le fichier test `blabla.txt`.

   Examiner le fichier `blabla.jflex` : un certain nombre de commentaires devraient permettre de comprendre la structure du fichier.

   Dans ce fichier, l'option `%standalone` indique que la classe générée par `jflex` fonctionne en mode `standalone`, ce qui signifie qu'elle contient une fonction `main` qui attend un nom de fichier en argument sur la ligne de commande et qui lance l'analyse de ce fichier. Les valeurs renvoyées par l'analyseur sont ignorées et tout texte non couvert par les règles est imprimé sur la sortie standard.

   Pour tester l'analyseur lexical, il faut :

   (a) générer le code java correspondant à l'analyseur : `jflex blabla.jflex`

   (b) compiler le fichier `Lexer` ainsi généré : `javac Lexer.java`

   (c) tester : `java Lexer blabla.txt`

   Comprendre pourquoi le résultat est bien celui-là...

2. L'option `--dot` de `jflex` permet de récupérer les automates produits sous le format `dot`. À partir de tout fichier `.dot` on peut produire un fichier `.pdf` par exemple :

   ```
   dot -Tpdf file.dot > file.pdf
   ```

   Observer les trois automates produits par `jflex` à partir de `chiffres.jflex`. L'automate `nfa` est le plus compréhensible parce qu'il utilise les caractères comme étiquettes. L'automate `dfa-min` est optimisé et on arrive assez bien pour cet exemple à voir quelles lettres et quelles étiquettes correspondent. Le `dfa-big` est nettement moins intéressant.

3. Faisons maintenant quelques modifications :

   (a) ajouter **à la fin** la règle

   ```
   [^]          {}
   ```

   `[^]` représente n'importe quel caractère. Cette règle demande donc de ne rien faire pour les caractères non reconnus par les règles précédentes (même pas les recopier dans la sortie standard). Tester.

   (b) dans la partie droite d'une règle, on peut récupérer le mot reconnu grâce à la méthode `yytext()`. Faire en sorte que les « blabla... » soient imprimés sur la sortie standard, suivi d'un « : » et du numéro du « blabla » dans le texte.

   (c) dans la partie droite d'une règle, on peut aussi récupérer la longueur du mot reconnu grâce à la méthode `yylength()`. Faire en sorte de calculer la longueur moyenne des « zzzz... » et de l'imprimer à la fin de l'analyse (on doit trouver 14).

Exercice 2 Des chiffres et des entiers dans un fichier.

1. Récupérer sur moodle le fichier `chiffres.jflex`. Ce fichier permet de compter le nombre de chiffres (caractères de 0 à 9) dans un fichier. Commencer par tester ce programme sur le fichier de test fourni (`chiffres.txt`).
2. Tester ce qui se passe si on supprime la dernière ligne de `chiffres.jflex`.
3. Remettre la dernière ligne et adapter ce fichier `jflex` de telle manière qu'il permette de calculer la somme de tous les chiffres (on doit trouver 28 chiffres pour une somme de 141) et aussi la moyenne lorsque cela a un sens.
4. Créer un nouveau fichier `jflex` pour faire la même chose avec les entiers (éventuellement négatifs). Si dans le fichier on a « -12.45 », cela sera interprété comme deux nombres entiers -12 et 45, car il ne reconnaît pas (encore) les flottants. Il est recommandé d'afficher les nombres reconnus dans un premier temps pour tester (on doit trouver 10 entiers pour une somme de -201).
5. Maintenant il s'agit de faire la même chose avec des flottants (double), sans exposant. On devra donc reconnaître des nombres comme « 12.34 », « 12. », « -.45 », ... (on doit trouver 8 flottants pour une somme de -1367.641).

Exercice 3 Des nombres romains.

On considère la numération romaine classique. Si besoin, on pourra s'appuyer sur ces descriptions : https://en.wikipedia.org/wiki/Roman_numerals ou https://la.wikipedia.org/wiki/Numeri_Romani en version originale.
Les expressions rationnelles devront être les plus concises possible.
On fournit deux fichiers à des fins de test : `romains.txt` contient les premiers nombres romains, alors que `randrom.txt` contient des mots aléatoires sur l'alphabet { I, V, X, L, C, D, M }.

1. Écrire une spécification `jflex` qui permette de reconnaître toute ligne contenant un nombre romain compris entre un et neuf. L'expression rationnelle ne devra contenir pas plus de deux disjonctions.
2. Éditer cette spécification pour qu'elle permette de reconnaître toute ligne contenant un nombre romain strictement inférieur à cent.
3. Éditer cette spécification pour qu'elle permette de reconnaître toute ligne contenant un nombre romain strictement inférieur à cinq mille.