TP2 : introduction à l'analyse lexicale (Lex)

On utilise ici `Jflex`, un générateur d'analyseur lexical en Java (héritier de lex pour C).

Rappel du cours : la génération de l'analyseur se fait en plusieurs étapes

1. le fichier `jflex` à compléter, par exemple `monAnalyseur.jflex`, contient la description de l'analyseur suivant une syntaxe spéciale (à la lex) ;
2. la commande `jflex monAnalyseur.jflex` engendre un fichier Java (`Lexer.java`, le nom du fichier est donné par la directive `%class Lexer`, par défaut le fichier s'appelle `Yylex`). Il contient la fonction d'analyse lexicale et le code pour les actions associées ;
3. la commande `javac Lexer.java` compile cet analyseur et crée le fichier `Lexer.class`
4. la commande `java Lexer fichierTest.txt` exécute l'analyse avec comme entrée la chaîne de caractères du fichier `fichierTest.txt`.

Vous pouvez utiliser votre éditeur java préféré pour créer le fichier `jflex`. Un fichier `Makefile` vous est proposé dans chaque répertoire pour automatiser la compilation (il suffit de taper la commande `make` dans un terminal, en se positionnant dans le répertoire où se trouve le fichier `jflex`).

Comme dans le TP précédent, il faut bien identifier les langages réguliers nécessaires à l'analyse et les écrire correctement dans la syntaxe de jflex. Il est possible et recommandé d'introduire des `abréviations` (noms intermédiaires) pour faciliter l'écriture et la lecture des expressions régulières. Cela se fait avec une déclaration préalable *nom = regexp*. Ensuite dans le corps de l'analyseur on peut utiliser *{nom}* entre accolades comme un équivalent de *regexp*. Attention à ne pas oublier les accolades sinon *nom* sera interprété comme l'expression régulière correspondant au mot `nom`.

Un élément nouveau est la possibilité de déclencher une `action` à la reconnaissance d'un mot du langage associé à l'expression régulière. Cette action est un code arbitraire qui pourra de plus invoquer des fonctions spéciales comme le contenu de la chaîne reconnue ou encore son positionnement (lignes, colonnes) dans l'entrée.

Vous commencerez par récupérer l'archive squelette (disponible sur ecampus ou dans `/public/pil/tp2/`), vous la décompresserez et pourrez ensuite travailler dans les répertoires correspondant à chaque section. Vous disposez d'un fichier de squelette, d'un fichier de test et d'un Makefile que vous pourrez modifier suivant vos besoins.

I) Reconnaître et étiqueter les unités lexicales

Exercice 1

Le but de cet exercice est de construire les prémisses d'un analyseur lexical pour un langage de programmation qui reconnaît les différentes classes d'unité lexicales (identificateurs, entiers, symboles). On illustre ici, l'usage des abréviations, le traitement des caractères non reconnus et l'association d'une action différente pour chaque classe d'unités lexicales.

Récupérez le fichier `UnitesLexicales.zip` et décompressez-le.

On considère le programme `Jflex` suivant disponible dans le répertoire `UnitesLexicales`

```
%%
%class Lexer
%standalone
entier = [0-9]+
%%
{entier} { System.out.println( "entier :" + yytext() );}
\* { System.out.println( "opérateur multiplier" );}
```

1. *Compilez ce fichier et testez-le. Tout d'abord sur un fichier qui ne contient que des entiers et le signe multiplier (on pourra utiliser la commande* `make test0` *qui teste le fichier* `test0.txt`*), puis sur une chaîne de votre choix qui mélange des lettres, des chiffres, des - et des * (comme le fichier* `test.txt`*, à tester avec la commande* `make test`*). Que constatez vous ?*
2. *Rajoutez une action à votre programme pour que les caractères non reconnus (i.e les lettres) n'apparaissent plus à l'écran. Vous utiliserez l'expression régulière étendue consistant en un seul caractère :* `.` *(un seul point) qui correspond à n'importe quel caractère, sauf les fins de ligne.*
   Indice : Il suffit de rajouter une ligne avec une action vide, pour dire que n'importe quel caractère non encore reconnu est ignoré (et pas considéré comme une erreur).
3. *et si vous voulez au lieu de cela afficher "erreur lexicale le caractère . . .n'est pas reconnu" ? Complétez le fichier en conséquence.*
4. *Finissez le programme pour reconnaître les entiers signés (qui peuvent donc être précédés d'un signe moins), et l'opérateur '+' et testez-le.*

II) Utilisation des actions pour des petits calculs

Dans la partie déclarations de `Jflex`, (l'endroit où sont déclarées les expressions régulières nommées), on peut aussi déclarer des variables pour compter des occurrences. Il faut les placer entre `%{` et `%}`, ces deux séquences doivent être **positionnées seules en début de ligne**. On peut ensuite mettre à jour ces variables dans les actions, de facon à calculer des quantités qui nous intéressent.

Exercice 2
Considérons la commande unix `wc` *(*word count*). Cette commande affiche le décompte de lignes, de mots et d'octets pour chaque fichier en entrée.*

```
> wc /etc/passwd
  31   44 1380 /etc/passwd
```

Nous souhaitons écrire un clone simplifié de cette commande `wc`*. On utilisera la fonction* `yylength()` *qui renvoie la longueur de la sous-chaine reconnue.*

1. *Compléter le texte suivant : Le nombre de lignes (dans les fichiers linux) correspond exactement au nombre d'occurrences du caractère . . ..*
2. *Comment reconnaître les mots un par un ? Il faut se mettre d'accord sur la définition d'un mot, ici on acceptera toute suite de caractères qui n'est pas un séparateur (espace, tabulation ou retour à la ligne). Utilisez* `https://regex101.com/` *pour trouver la bonne expression régulière.*
3. *On décide de faire l'analyseur avec trois expressions régulières : une pour les mots, une pour les retours à la ligne et une pour les autres caractères. Comment faire dans les actions associées pour compter le nombre de caractères ?*

4. *Vous êtes mûrs pour écrire le programme* `Jflex` *qui compte et affiche le nombre de caractères, le nombre de mots et le nombre de lignes d'un fichier. Sur le fichier contenant le texte suivant :*

```
a bébé ccc
d e-e fff
g hh iii!
```

Le programme doit afficher `lines=3 words=9 chars=32`*.*
Vous repartirez du squelette qui vous est fourni en modifiant l'expression régulière pour les mots et en complétant les actions. Le code d'affichage doit s'exécuter une seule fois à la fin du fichier (et non pas à chaque reconnaissance d'une unité lexicale). On positionne ce code dans la partie déclaration du fichier jflex entre les balises `%eof{` *et* `%eof}`*, à écrire seules sur la ligne.*

III) Sous-titrage de films en jflex

L'objectif de cette partie est d'écrire quelques programmes aidant à la manipulation de fichiers de sous-titres de films. Un tel fichier contient la suite des sous-titres d'un film, avec des indications sur le moment où chaque sous-titre doit s'afficher et sa durée d'affichage. Il existe plusieurs formats pour ce type de données. Nous nous intéresserons au format .SUB.

Dans un fichier .SUB, chaque sous-titre est défini sur une seule ligne. En début de ligne on trouve le moment de début d'affichage du sous-titre (en nombre d'images depuis le début du film) entre accolades, puis le temps de fin d'affichage entre accolades, enfin le texte du sous-titre. Dans ce texte, un séparateur vertical | indique un saut de ligne.

Voici un exemple de fichier .SUB :
{1138}{1260}La gagnante de la bourse|d'études de 30000 dollars...
{1262}{1369}est Miss Louisiane, Erika Schwarz.
{1371}{1430}Et la nouvelle Miss Amérique...
{1433}{1536}est Miss Kansas, Tara Dawn Holland !

Les squelettes et fichiers de test utilisés dans ce TP sont disponibles dans le répertoire `Stitre`. Vous disposez d'un fichier `test.SUB` qui vous permet de tester vos programmes.

a) Prise en main

Pour commencer, on ne va pas traiter dans les sous-titres le caractère | pour le passage à la ligne qui sera donc vu comme un caractère du texte, (on les gèrera par la suite).

Utilisez le squelette `sous-titre-a.jflex`.

Votre premier programme jflex consistera à lire le fichier et à le réafficher dans un format vous permettant de vérifier que cette lecture est correcte. Sur un fichier contenant la ligne : {1138}{1260}La gagnante de la bourse d'études de 30000 dollars... cela doit afficher sur une seule ligne :

```
Du temps {1138} jusqu'au temps {1260} le sous titre: La gagnante de la bourse
d'études|de 30 000 dollars...
```

Question 1 : Proposez une solution pour distinguer le temps du début du temps de fin d'affichage sachant qu'ils correspondent tous les deux à la même expression régulière.

Question 2 : Ecrivez le programme.
Indication : le contenu d'un sous-titre est une chaîne de caractères qui ne commence pas par "{" et qui ne contient pas de retour à la ligne.

Question 3 : A présent, on vous demande pour chaque sous-titre, d'enregistrer les temps de début et de fin comme des entiers dans un tableau de taille 2 (cela nous sera utile ensuite pour calculer des durées). Le premier élément du tableau est le temps de début, et le deuxième le temps de fin d'affichage.
Indications : il vous faudra utiliser la méthode `substring` de la bibliothèque string, pour extraire la sous-chaine qui contient juste la suite des chiffres sans les accolades. Cette méthode prend deux paramètres entiers qui sont les index où commence et termine la sous-chaîne. Il vous faudra aussi utiliser la méthode statique `Integer.parseInt` qui prend en argument une chaîne de caractères, qui doit contenir seulement des chiffres, et contruit l'entier associé.

b) Le cas des sous titres sur plusieurs lignes

Généralisez ce premier programme d'affichage pour prendre également en compte les sauts de lignes (matérialisés par le caractère `|`). On remplacera l'expression `SUBTITLE` précédente par deux expressions rationnelles différentes : `FSTLINE` qui identifie la première ligne et `LINE` qui identifie les éventuelles autres lignes de sous titres, avec les deux actions suivantes :

```
{FSTLINE} { System.out.println( "soustitre :" + yytext() );}
{LINE} { System.out.println(  yytext() );}
```

c) Affichage avec timing fixe

On ignore à nouveau le caractère `|` de passage à la ligne dans les sous-titres. On souhaite par contre afficher les sous-titres pendant 2 secondes, avec une seconde de blanc. La partie contenant le nombre d'images est ignorée. Vous utiliserez le squelette `sous-titre-c.jflex` qui contient les fonctions `clearConsole()` et `pause()`. Il reste juste à compléter les expressions régulières comme dans les questions précédentes, et à ajouter les actions associées.

d) Affichage avec le bon timing.

A présent, modifiez le programme pour tenir compte du timing spécifique du film. Le sous-titre doit rester affiché exactement le temps spécifié dans le fichier .sub. Attention, ce temps est compté en images et pas en secondes. Par exemple, pour les sous titres

```
{1138}{1260}La gagnante de la bourse|d'études de 30 000 dollars...
{1300}{1369}est Miss Louisiane, Erika Schwarz.
```

Le premier sous-titre doit s'afficher durant $1260 - 1138$ images soit 22 images qui correspondent à $22000/24$ millisecondes. Il y a ensuite une pause entre les images 1260 et 1300 soit $40000/24$ millisecondes.

Le traitement se fera "à la volée" en utilisant uniquement les deux variables d'entiers $t[0]$ et $t[1]$ et en adaptant la fonction `pause` pour qu'elle traite un nombre d'images.

IV) Annales TP noté

a) 2022-23

Répertoire `TP-note-22-23`, fichier `compte.jflex`, commandes `make lex-compil` et `make lex-test` :

Repartir du fichier `compte.jflex` vu en TP qui compte et affiche le nombre de caractères, de mots et de lignes dans un fichier. On a dans ce fichier considéré que le caractère `/` était un séparateur de mots tout comme les espaces, les tabulations et fins de ligne. On a également ajouté la reconnaissance des commentaires qui commencent par `//` et se terminent au passage à la ligne sans action associée.

Questions : Ajouter à cet analyseur l'affichage du nombre de commentaires (sur une seule ligne) ainsi que la longueur cumulée de ceux-ci (sans compter les délimiteurs des commentaires). L'analyseur continuera à afficher le nombre total de caractères, de mots et de lignes par contre il ne comptera que les mots qui apparaissent en dehors des commentaires.

b) 2023-24

Répertoire `TP-note-23-24`, fichier `mail.jflex`, commandes `make lex-compil` et `make lex-test` :

Le fichier `mail.jflex` contient un squelette d'analyseur lexical pour compter et afficher des expressions régulières.

Il s'agit ici de reconnaître dans un texte des url correspondant à des envois de mail c'est-à-dire de la forme `mailto:`*email*.

La partie adresse mail (*email*) devra respecter les conventions suivantes :

— Elle est de la forme *identifiant*`@`*domaine*.*suffixe*
— L'identifiant commence par un caractère alphabétique (de `a` à `z`) en minuscule ou majuscule.
— Le premier caractère de l'identifiant est suivi d'une suite de caractères (possiblement vide) qui peuvent être alphabétiques ou numériques (chiffres de `0` à `9`) ou le tiret `-` ou le point `.`
— Un point ne peut pas être à la fin de l'identifiant ni suivi d'un autre point
— Le domaine commence par un caractère alphabétique puis est suivi d'une suite (possiblement vide) de caractères alphabétiques ou numériques ou le tiret `-`
— Le suffixe est formé de deux ou trois caractères alphabétiques, il doit être suivi d'un caractère quelconque non alphabétique.

Complétez les expressions régulières et les actions du fichier `mail.jflex` pour afficher les url mail telles que spécifiées ci-dessus ainsi que leur nombre total.

Ainsi sur le fichier de test, le résultat attendu est :

```
mailto :christine.paulin@universite-paris-saclay.fr
mailto :Frederic.Gruau@universite-paris-saclay.fr
mailto :mathias.loiseau@universite-paris-saclay.fr
mailto :mickael.mariyanayagam@universite-paris-saclay.fr
mailto :remy.parent@universite-paris-saclay.fr
mailto :thong-thai.van@universite-paris-saclay.fr
mailto :paul.oriat@universite9-paris-saclay.fr
mailto :laurent.guiddir@Universite-Paris-Saclay.fr
mailto :bassel.aleian@universite-paris-saclay.com
mailto :rafik@universite-paris-saclay.fr
mailto :eliott.rigobert@universite-paris-saclay.fr
mailto :mathieu.waharte@u.fr
mailto :a@universite-paris-saclay.fr
mailto :ines.duflos.duflos@universite-paris-saclay.fr
mailto :rayane.rostane@universite-paris-saclay.fr
mailto :justin.moua@universite-paris-saclay.fr
mailto :imane.benbouziane@universite-paris-saclay.fr
mailto :manda.andriamaromanana@universite-paris-saclay.fr
nombre d'URL mailto=18
```