L2 Informatique 2022 – 2023
Module *Principes d'Interprétation des Langages* Feuille de TP n° 1

TP1 : introduction à l'analyse lexicale (Lex)

Prise en main avec https ://regex101.com

Exercice 1
Nous allons utiliser le site `https://regex101.com/`. Celui-ci permet de tester, étant données une expression régulière et une chaîne de caractères, où sont les *correspondances* (match en anglais), c'est-à-dire les parties de la chaîne qui correspondent à un mot du langage de l'expression régulière.

Il y a beaucoup de questions, mais elles se font rapidement.

1. Rentrez l'expression régulière `[-]?[0-9]+` qui reconnait les nombres entiers signés : puis testez-la sur une chaîne de votre choix qui mélange des lettres, le caractère tiret '`-`' et des chiffres. Que constatez vous ?
2. Remplacez l'expression précedente pour cette fois reconnaître les identificateurs definis comme une lettre (minuscule ou majuscule) suivie d'une suite de lettres ou de chiffres. Pour testez votre expression, vous construirez une chaîne pour obtenir plusieurs correspondances distinctes séparées par des caractères autres que des blancs.
3. Vous allez à présent trouver une expression régulière pour reconnaître n'importe quelle unité lexicale de l'exercice 1 du TD 1. Il s'agit des constantes entières signées, des identificateurs, mais aussi les symboles spéciaux `:=`, `;`, `*` et `+`. **Attention, pour les caractères déja utilisés pour décrire des expressions régulières comme `*`, il faut les faire précéder du caractère d'échappement \ (antislash).**
4. Tester les correspondances sur l'expression `x 3 := y - 4;` et l'expression `x 3 := y -4;`. Quelle est la différence ? pouvez-vous à présent formuler pourquoi la première expression `x 3 := y - 4;` n'est pas lexicalement bien formée ?

Outil Jflex : reconnaître et étiqueter les unités lexicales

Exercice 2
On va maintenant utiliser `Jflex`, un générateur d'analyseur lexical en Java qui est la version java de lex, plus facile d'emploi grace à de meilleurs messages d'erreurs.

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

1. le fichier `monSuperProg.jflex` contient la description de l'analyseur suivant une syntaxe spéciale (à la lex) ;
2. la commande `jflex monSuperProg.jflex` engendre un fichier Java (`Lexer.java`) contenant la fonction d'analyse lexicale et le code pour les actions associées ;
3. la commande `javac Lexer.java` compile ce lexeur ;
4. la commande `java Lexer fichierInput.txt` exécute ce lexeur avec comme entrée la chaîne de caractères du fichier `fichierInput.txt`.

Vous pouvez utiliser votre interface habituelle `intellij`, l'avantage est d'avoir simultanément la fenêtre d'édition et celle pour la compilation.

On considère le programme `Jflex` suivant disponible dans `/public/pil/tp1/entierSquelette` :

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

1. Copier ce fichier dans un répertoire tp1, compilez-le et testez-le sur une chaine de votre choix qui mélange des lettres des chiffres et des - . 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) correspond à n'importe quel caractère, sauf les fins de ligne.

   Indice : Il suffit de rajouter une ligne avec seulement 4 caractères.
3. et si vous voulez au lieu de cela afficher "erreur lexicale le caractère . . .n'est pas reconnu" ?
4. Finissez le programme pour reconnaître les entiers signés (qui peuvent donc être précédés d'un signe moins), et l'opérateur '+' et testez-le.

Reconnaissance d'expressions avec la négation

Dans l'exercice suivant on utilisera juste le site `https://regexp101.com` Notre objectif se limite à explorer les expressions régulières utilisant la négation.

Exercice 3
On considère le problème relativement courant de reconnaître tout sauf quelque chose en particulier. Par exemple l'expression `[^a]*` correspond à toute chaine qui ne contient pas le caractére 'a'. Testez-là sur une chaine qui contient des 'a'.

Cette syntaxe (tous les caractères sauf . . .) n'est pas nécessaire lorque on étudie les expressions régulières d'un point de vue théorique, mais par contre elle est bien pratique lorsque on les utilise avec des alphabets de grande taille comme dans les programmes ;

Pour reconnaître une constante correspondant à une chaîne de caractères, l'expression n'importe quoi entre deux guillemets `".*"` ne marche pas. Le vérifier sur la chaine `toto "hhh" tata "jjj"` que se passe t'il ? Comment corriger cela ?

Exercice 4
On souhaite analyser un texte et restituer les parties entre crochets [ ] (avec les crochets) en les séparant par des retours à la ligne.
Attention, les crochets ayant une signification dans les expressions régulières, le caractère crochet s'écrit avec un caractère d'échappement.

Exemple : dans la chaine suivante :

```
Ce texte n'est pas matché.
[Ce texte est matché] et [celui-ci aussi ] ]mais pas lui.
Vous ne devez pas matcher cela mais [vous devez matcher
ce retour à  la ligne, et ce crochet tout au bout [ !]
```

Les sous chaines qui doivent être reconnues sont :

```
[Ce texte est matché]
[celui-ci aussi ]
[vous devez matcher
ce retour à  la ligne, et ce crochet [ !]
```

Utilisation des actions pour faire 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. Par exemple, 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 utisera 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 ? Utilisez `https://regex101.com/` pour trouver.
3. Comment finalement compter les 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 bb ccc
   d ee fff
   g hh iii
   ```

   Le programme doit afficher `lines=3 words=9 chars=27`. Vous devrez utiliser l'expression réguliere spéciale `<<EOF>>` qui introduit une action finale à exécuter, lorsque tout le fichier est lu (EOF signifie "End Of File"). Attention, il faut terminer l'action associée par l'instruction `return YYEOF`, sinon cela boucle (le caractère de fin de fichier n'étant pas consommé, l'action est redéclenchée).

   Une alternative est de mettre le code à exécuter dans la partie déclarations du fichier jflex entre les balises `%eof{` et `%eof}`, à positionner seules sur la ligne.