Compilation - N3, S5
TP 1 : Mise en Main de Antlr

1. L'objectif de ce TP c'est de se familiariser avec le générateur de compilateur Antlr
2. Le langage de programmation choisi pour créer le compilateur c'est Java
3. Votre ami pour la documentation de Antlr :
   `https://github.com/antlr/antlr4/blob/master/doc/index.md`

Installation d'Antlr. Commencez d'abord par télécharger depuis le cours en ligne le fichier *installAntlr.txt* et suivez les instructions pour installer Antlr.

Question. Donner une grammaire non-ambigüe dans Antlr que l'on appellera Mot_Eg et qui reconnait le langage $L = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b\}$.

N'oubliez pas que les non-terminaux dans Antlr sont en lettres minuscules. Aussi, l'analyseur lexical construit par Antlr produit, à la fin de la lecture de l'entrée, un lexème égal à la valeur système `EOF`. Pensez par conséquent à terminer chaque règle de l'axiome de départ par `EOF`.

Question. Qu'est-ce qui se passe lorsque l'on compile le fichier Mot_Eg.g4 à l'aide de la commande `antlr4` ? Comment faire pour obtenir le compilateur de notre langage ?

Question. Vous pouvez utiliser l'outil `grun` pour avoir la liste des tokens ou pour visualiser l'arbre syntaxique produit par l'analyseur. On donne le nom du compilateur produit par Antlr (qui sera par défaut le même que le nom de la grammaire), l'axiome dont le langage contient le mot à reconnaître (dans l'exemple ci-dessous on donne l'axiome de début que l'on a appelé `start`) et la liste des options. Ci-dessous deux exemples d'utilisation.

```
grun Mot_Eg 'start' -tokens // donne la liste des jetons
grun Mot_Eg 'start' -gui // produit l'arbre de dérivation du mot à reconnaître
```

Testez votre grammaire avec des mots de L et des mots qui ne sont pas dans L. On termine chaque entrée par `Crt+D`.

Question. Voici une grammaire, appellée Calculette, qui permet de reconnaître les expressions arithmétiques avec addition et produit.

```
grammar Calculette;
start
     : expr EOF;
expr
     : expr '*' expr
     | expr '+' expr
     | ENTIER
;
NEWLINE : '\r'? '\n' -> skip;
WS : (' '|'\t')+ -> skip;
ENTIER : ('0'..'9')+;
UNMATCH : . -> skip;
```

Question. Quelles sont les règles de l'analyseur syntaxique ? Quelles sont les règles de l'analyseur lexical ? Rappelez ce que fait chaque règle de l'analyseur lexical.

Question. Créez le fichier Calculette.g4 qui contiendra la grammaire ci-dessus et compilez pour obtenir le compilateur. Utilisez l'outil `grun` pour tester votre compilateur avec les exemples suivants (avec les options pour obtenir la liste des tokens et l'arbre de dérivation). Pourquoi les espaces sont éliminés ?

```
42
22+ 20
5*8+2*    1
5+2 * 8+ 3
6    *4 + 18
```

Question. Que se passe-t-il si on permute l'ordre dans la règle de expr entre l'addition et la multiplication ? Testez les deux versions avec l'expression $5 + 2 * 8 + 3$.

Question. Enrichir la grammaire pour inclure les opérations de division, soustraction binaire, unaire et expressions parenthésées. La grammaire doit respecter les ordres de priorité. Testez votre grammaire avec les expressions suivantes.

```
42
24+24-6
5*8+2*1
6*4/5+38
42+1+2+-3
5*8+2*-1/-1
(5*6*7*11 + 2)/11*5-1008
(5*6*7*11 + 2)/(11*5)
(5*6*7*11 + 2)/11/5
(5*6*7*11 + 2)/(11/5)-1114
```

Question. Pour le moment, chaque programme de notre langage Calculette est réduit à une seule expression arithmétique. Modifez la grammaire pour qu'un programme puisse être une suite d'expressions qui sont séparées par des sauts de ligne et/ou des `;`.