Master Informatique **Compilation – J1IN7M13**
Université Bordeaux 1 **2013 – 2014**

TD N^o2 Ecrire une calculette JFlex/CUP

Si vous travaillez avec *Emacs*, créez un nouveau répertoire `Calculette`. Si vous travaillez sous *Eclipse*, créez un nouveau projet `Calculette`. Dans les deux cas, créez l'arborescence suivante :

```
bin/
build.xml
Input.txt
lexer/
lexer/calculette.jflex
lib/
lib/java-cup-11a-runtime.jar
lib/JFlex.jar
lib/java-cup-11a.jar
parser/
parser/calculette.cup
src/
src/Main.java
```

Cette arborescence se trouve dans l'archive
`http ://www.labri.fr/perso/clement/enseignements/compilation/calculette.tar.gz`.

Questions

1. Compiler le projet en exécutant le fichier `build.xml`. Faire fonctionner avec plusieurs exemples.

   Les tokens de l'analyseur lexical sont les terminaux de la grammaire utilisée par CUP. Ils apparaissent sous forme de constantes statiques dans la classe `CalculetteSymbol` générée par l'analyseur.

2. CUP peut associer à chaque terminal et non terminal d'une grammaire un objet appelé attribut. L'attribut associé au non-terminal membre gauche de la règle sera désigné par le mot clé RESULT, les attributs des non terminaux en membre droit sont accessibles en placant des étiquettes directement dans la règle. Par exemple, dans la règle :
   `expr ::= expr:t1 PLUS term:t2`
   `t1` est l'attribut associé au non terminal `expr` et `t2` est l'attribut associé au non terminal `term`. Il faut écrire des règles de calcul d'attributs (bottom-up) dans les actions associées à chaque production de la grammaire.

   Pendant la reconnaissance d'un mot, le parser empile des objets de type `Symbol`. Les attributs correspondent au champ `value` de l'objet `Symbol`. Les attributs des terminaux doivent être initialisés par le lexer. Vous disposez pour cela dans le fichier `calculette.jflex` d'une méthode `symbol (int type, Object value)` qui construit un objet `Symbol` avec l'attribut `value`.

   Dans le fichier de l'archive, les attributs ne sont pas utilisés. Rajouter dans `calculette.jflex` une regexp pour les constantes numériques entières. Rajoutez dans le

fichier `calculette.cup` un calcul d'attributs qui associe aux symboles de la grammaire des attributs de la classe Integer (on remplacera la règle `factor -> ID` par `factor -> NB`). L'attribut du non terminal `axiom` devra être la valeur de l'expression reconnue.
Augmenter la grammaire afin de pouvoir faire des soustractions et des divisions. Comment traitez-vous une division par zéro ?

3. Reconstruire la grammaire en utilisant un seul non terminal pour *expr*, *term*, *factor*. Observer les conflits *Shift/Reduce* dans la table LALR. Résoudre ces conflits en utilisant les priorités des opérateurs.
   Chaque production possède un niveau de priorité : celui du terminal le plus à droite dans sa partie droite. Si la production ne contient pas de terminaux alors elle a le niveau de priorité le plus bas (niveau 0). On peut déclarer la précédence (niveau de priorité) et l'associativité des terminaux en début de fichier de la façon suivante :
   `precedence left <terminal> ;` déclare un opérateur associatif à gauche, `precedence left <terminal>, ... , <terminal> ;` déclare un ensemble d'opérateurs associatifs à gauche de même priorité. `precedence right <terminal> ;` déclare un opérateur associatif à droite, `precedence right <terminal>, ... , <terminal> ;` déclare un ensemble d'opérateurs associatifs à droite de même priorité. Tout terminal non déclaré dans cette partie se voit attribuer le degré de priorité le plus bas.
   Il est possible de aussi de forcer le niveau de priorité d'une production en écrivant à la suite de la production `%prec <terminal>`. La production a alors le niveau de priorité de `<terminal>`.