TELECOM Nancy 2ème année

Janvier 2015
Module PROJET COMPILATION

Analyse syntaxique descendante - Prise en main du logiciel ANTLR.

⋆ Exercice 1. Premiers pas.

Pour commencer avec ANTLR, on vous propose de prendre l'exemple de la grammaire d'expressions simplifiée (voir fichiers `Expr.g` et `Test.java`) et de rajouter dans la grammaire, les opérateurs / et *modulo* que l'on symbolisera par le caractère %, symboles représentants respectivement la division entière et le reste de la division entière.

Avant de commencer à utiliser Antlr, suivre les indications suivantes (à adapter par la suite si vous utilisez Antlr sur votre ordinateur personnel) :

- à la fin de votre fichier `.bashrc`, ajouter la ligne suivante :
`export CLASSPATH=/home/depot/2A/PROJET_COMPILATION/2013-2014/antlr-3.3-complete.jar:.:$CLASSPATH`
- dans un nouveau terminal, passer en `bash`.

Vous êtes prêts pour exécuter ANTLR sur un fichier contenant une grammaire.

Pour exécuter ANTLR sur un fichier contenant une grammaire, ici `Expr.g`, faire :
`java org.antlr.Tool Expr.g` ou encore `java -jar antlr-3.3-complete.jar Expr.g`
Cette commande produit les deux fichiers `ExprLexer.java` et `ExprParser.java` ainsi que le fichier `Expr.tokens`.
La compilation suivante : `javac ExprLexer.java ExprParser.java Test.java` permet d'obtenir les `.class` correspondants.
Pour exécuter sur un exemple, tapez la commande `java Test` dans un terminal, puis entrez des valeurs. Par exemple :

```
x=1
y=2
3*(x+y)
<EOF>   (soit Ctrl_D dans une fenêtre shell)
```

⋆ Exercice 2. Les arbres et listes.

On reprend l'exercice vu en tavaux dirigés sur les arbres et listes.
On rappelle que l'on note `(1.2)` l'arbre binaire composé des sous-arbres `1` et `2`. Si un arbre est réduit à une feuille, il est noté par sa valeur ; s'il est vide, il est noté `nil`.
On traite également des listes que l'on note sous la forme `(1,2,...,5)` où `1,2,...,5` sont les éléments successifs de la liste. Les éléments de la liste peuvent être soit des listes, soit des arbres, donc aussi des valeurs entières ou `nil`. On reprendra pour cet exercice la grammaire LL(1) vue en travaux dirigés.

On vous demande d'écrire un programme d'analyse syntaxique,

- qui vérifie qu'un texte source est correct : pour cela, écrivez la grammaire correspondante dans un fichier `Arbre_Liste.g`, et testez sur des exemples,
- qui imprime la notation pointée d'une liste initialement écrite avec des virgules et des points (cf. ce qui a été vu en TD).

Fichier Expr.g contenant l'exemple d'une grammaire d'expressions simplifiée.

```
grammar Expr;

@header {
import java.util.HashMap;
}

@members {
/** Map variable name to Integer object holding value */
HashMap<String,Integer>  memory = new HashMap<String,Integer>();
}

prog:   stat+ ;

stat:   expr NEWLINE {System.out.println($expr.value);}
    |   ID '=' expr NEWLINE
        {memory.put($ID.text, new Integer($expr.value));}
    |   NEWLINE
    ;

expr returns [int value]
    :   e=multExpr {$value = $e.value;}
        (   '+' e=multExpr {$value += $e.value;}
        |   '-' e=multExpr {$value -= $e.value;}
        )*
    ;

multExpr returns [int value]
    :   e=atom {$value = $e.value;} ('*' e=atom {$value *= $e.value;})*
    ;

atom returns [int value]
    :   INT {$value = Integer.parseInt($INT.text);}
    |   ID
        {
        Integer v = (Integer)memory.get($ID.text);
        if ( v!=null ) $value = v.intValue();
        else System.err.println("undefined variable "+$ID.text);
        }
    |   '(' expr ')' {$value = $expr.value;}
    ;


ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t')+  {$channel=HIDDEN;};  /* pour la version 3.3 */
/* -> skip ; pour la version 4.1 */
```

Fichier Test.java contenant la fonction main().

```
import org.antlr.runtime.*;

public class Test {
    public static void main(String[] args) throws Exception {
        ANTLRInputStream input = new ANTLRInputStream(System.in);
        ExprLexer lexer = new ExprLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        ExprParser parser = new ExprParser(tokens);
        parser.prog();
    }
}
```