Une calculatrice en CUP/Java

But du TP

Le but de ce TP est d'appliquer un générateur d'analyseur syntaxique de la famille yacc (ici `cup` pour Java) afin d'avoir un premier exemple concret, sur un problème simple, du processus de compilation d'un programme : analyse lexicale (*lexeur*) puis analyse sytaxique (*parseur*) permettant de faire correspondre des instructions à chaque règle de grammaire (ici, en Java, l'évaluation des expressions).

Écrire et Compiler

Les fichiers contenant la description lexicale (code `jflex`) auront des extensions en `.jflex`, ceux contenant la description syntaxique (code `cup`) ont une extenssion en `.cup`. Un fichier `Main.java` est fourni pour appeler simplement l'analyseur sur des fichiers d'exemples.

Dans ce TP, toutes les modifications se feront uniquement au niveau des fichiers d'analyse lexicale et d'analyse syntaxique. Le reste du programme vous est donné. Il vous faudra aussi écrire de petits fichiers d'exemples à analyser (sans extension, ou .txt si vous le souhaitez : `java ... calc.Main tests/fichierExemple.txt`). Vous pouvez utiliser n'importe quel éditeur de texte (par exemple emacs) pour les écrire. Comme pour tout TP, il vous est conseillé de compiler et tester souvent.

Pour compiler avec `jflex`, `cup` et `java`, un fichier `Makefile` vous est fourni. Il fait l'hypothèse que tous les fichiers sources sont dans un répertoire `calc` correspondant au package Java du projet. Les fichiers compilés sont dans un répertoire `bin`. Le fichier contient les instructions suivantes pour la compilation :

```
cd calc  && jflex Lexer.jflex
cd calc  && java -jar /usr/share/java/java-cup-0.11b.jar -package calc Parser.cup
javac -cp /usr/share/java/java-cup-0.11b-runtime.jar:bin:. -d bin calc/*.java
```

Puis exécuter avec :

```
java -cp /usr/share/java/java-cup-0.11b-runtime.jar:bin:. calc.Main exemple1
```

Introduction : une calculatrice très simple

Vous trouverez ci-dessous le programme vu en cours. Créez un répertoire `TP3`, récupérez l'archive des fichiers sources[1] sur le site ecampus. Décompressez l'archive et compilez le projet (commande `make` dans le répertoire `TP3/src`). Testez le programme exécutable sur plusieurs exemples, simples ou plus complexes (avec parenthésages multiples). L'archive contient un répertoire `tests` avec quelques exemples testés par la commande `make tests` que vous pouvez adapter.

1. Attention, ne faites pas de copier-coller à partir du présent document : la copie de caractères depuis le format pdf peut insérer des caractères invisibles qui occasionnent des erreurs de compilation incompréhensibles.

Fichier source CUP : Parser.cup

```
import java_cup.runtime.*;
import java.io.*;
import java.util.*;

init with {: System.out.println("Exemple␣:␣Calculatrice␣"); /* Action initiale */ :}

terminal PLUS, MULT, LP, RP, NL;
terminal Integer CTE ;
nonterminal Integer expr, terme, facteur ;
nonterminal ligne ;

start with ligne;

ligne ::=
     expr:e  NL                 {: System.out.println(e);  :}
;
expr ::=
       expr:e1 PLUS terme:e2    {: RESULT = e1+e2; :}
     | terme:e                  {: RESULT = e; :}
;
terme ::=
       terme:e1 MULT facteur:e2 {: RESULT = e1*e2; :}
     | facteur:e                {: RESULT = e; :}
;
facteur ::=
       CTE:n                    {: RESULT = n; :}
     | LP expr:e RP             {: RESULT = e; :}
;
```

Fichier source jflex : Lexer.jflex

```
package calc;

import java_cup.runtime.*;
import java.util.*;
import static calc.sym.*;

%%
%class Lexer
%unicode
%cup
%line
%column
%yylexthrow Exception

NL = \R
BLANC = [ \t\f]
NOMBRE = [0-9]+
%%
{BLANC}   { /* ignore */ }
{NL}      { return new Symbol(NL); }
\+        { return new Symbol(PLUS); }
\*        { return new Symbol(MULT); }
\(        { return new Symbol(LP); }
\)        { return new Symbol(RP); }
{NOMBRE}  { return new Symbol(CTE,Integer.parseInt(yytext())); }
.  { throw new Exception
```

```
(String.format ("Line␣%d,␣column␣%d:␣illegal␣character:␣'%s'\n",
                yyline, yycolumn, yytext() )); }
```

1. Compréhension de code : En lisant le code, donnez la grammaire du langage reconnu par le fichier cup.
2. Donnez un arbre de dérivation pour l'expression suivante :

   `(NOMBRE + NOMBRE * NOMBRE) * NOMBRE`.

Amélioration : une calculatrice plus réaliste

1. Ajoutez à ce programme les opérations de soustraction et division (entière).
   Indice : il suffit de compléter très simplement le lexer puis la grammaire du programme cup.
2. Jusqu'à présent la calculatrice ne peut gérer qu'une expression à la fois, il faut relancer le programme si l'on en veut plusieurs. Modifiez les fichiers sources afin que le programme puisse calculer plusieurs expressions dans un même fichier (une expression par ligne).

   Par exemple, si on lance le programme avec le fichier ci-dessous en entrée standard,

   ```
   (20*((50+30)*40))*2
   1+1
   45*3
   10*(1+1)*2+(1+1)*5
   10/2
   10/2+3
   10/(2+3)

   12+3
   ```

   on doit obtenir la sortie suivante :

   ```
   128000
   2
   135
   50
   5
   8
   2
   15
   ```

   Notez que la ligne vide dans le fichier d'entrée n'est pas prise en compte.
   Indice : il faut compléter la grammaire du fichier cup pour pouvoir prendre en compte plusieurs lignes. Le retour à la ligne doit aussi être traité dans le fichier jflex.
3. On désire pouvoir ajouter des commentaires dans les fichiers d'entrée, qui ne sont bien sûr pas pris en compte par le programme. Un commentaire commence avec deux "slash" (//) et se termine par une fin de ligne. Par exemple, le programme doit fonctionner si on le lance avec le fichier ci-dessous en entrée :

   ```
   (20*((50+30)*40))*2 // réponse : 128000
   1+1                 // réponse : 2
   45*3                // réponse : 135
   10*(1+1)*2+(1+1)*5  // réponse : 50
   10/2                // réponse : 5
   10/2+3              // réponse : 8
   ```

```
10/(2+3)            // réponse : 2
                    // ligne non prise en compte
12+3                // réponse : 15
```

Indice : il suffit de modifier le programme source jflex.

4. On désire pouvoir utiliser une variable nommée A dans les fichiers d'entrée. Par exemple, le programme doit fonctionner si on le lance avec le fichier ci-dessous en entrée :

```
3+2*5                   // réponse : 13
A                       // réponse : 0
A=10*(1+1)*2+(1+1)*5    // réponse : A=50
A                       // réponse : 50
A+2                     // réponse : 52
(4*A)/((A*2)-50)        // réponse : 4
A=A*2                   // réponse : A=100
A                       // réponse : 100
```

Indice : il suffit de modifier le programme source `cup`. La variable A doit être déclarée en Java dans la première partie du fichier `cup`, entre les déclarations d'import et l'instruction d'initialisation.

```
parser code {:
public static Integer varA = 0;
:}
```

Les caractères 'A' et '=' deviennent des tokens, au même titre que, par exemple, '+' et '('. La grammaire doit être modifiée afin de prendre en compte l'instruction d'affectation de la variable A, et le fait que A peut jouer dans une expression le même rôle qu'un nombre.

5. Si maintenant, au lieu d'une seule variable A, vous voulez que votre calculatrice puisse utiliser 26 varaibles (lettres de A à Z) comment pouvez-vous faire cela ?

Exercice3 . Grammaire alternative

Peut-on partir directement de la grammaire :

$$E ::= \texttt{CTE} \mid A \mid E + E \mid E * E \mid E - E \mid E/E$$

1. Quelle difficulté présente cette grammaire ?
2. Transformez votre fichier de grammaire pour utiliser ces règles
3. Compilez, quels messages d'erreurs observez-vous ?
4. Afin de résoudre les ambiguïtés, il faut expliciter les règles de précédence entre les opérateurs arithmétiques et les conventions d'associativité des opérateurs. Cela se fait en ajoutant des lignes.

```
precedence left OP1, OP2;
precedence right OP3;
```

Les opérateurs `OP1` et `OP2` ont même précédence et associent à gauche. L'opérateur `OP3` a une précédence plus forte que `OP1` et `OP2` et associe à droite.

Ajoutez les bonnes précédences dans votre analyseur pour respecter les conventions standards d'évaluation des expressions arithmétiques et testez votre programme sur des exemples bien choisis.