Master 1 – Compilation
Analyse syntaxique d'une grammaire simple

Ce TP sur l'analyse syntaxique correspond à une analyse syntaxique en implémentant une analyse par procédure récursive. Nous allons utiliser les fonctions avancées de jflex pour réaliser ce TP – TP.

Contexte

Nous allons utiliser une grammaire relativement simple et au fur et à mesure nous allons l'améliorer. Notre grammaire va nous permettre de reconnaître, de vérifier et ultérieurement de calculer des expressions arithmétiques simples comme :
2 +(3 * (5 – 2,5))
ou encoure
(2 + 3) * 4
et devra bien évidemment détecter dans la phase d'analyse syntaxique les erreurs comme
(2 + 3)) * 4

Analyse lexicale

La première partie consiste bien évidemment à reconnaître les différents éléments lexicaux du langage. Nous travaillerons avec des réels mais bien évidemment les entiers doivent pouvoir être acceptés dans notre analyseur lexical.

Les différents éléments terminaux de notre langage sont outre les nombres, les parenthèses ouvrantes et fermantes, et les quatre opérateurs classiques (+,*,/,-).

Travail à réaliser

Écrivez votre analyseur lexical avec jflex en utilisant l'option%standalone.
Pour le fichier d'exemple suivant, vous devriez obtenir :
(2 - 3 * (7 + 2.4))

```
LPAR
nombre
MOINS
nombre
FOIS
LPAR
nombre
```

PLUS
nombre
RPAR
RPAR

Dans un deuxième temps, nous allons travailler en utilisant jflex de manière indépendante (c'est-à-dire sans utiliser l'option %standalone).

Dans ce cas, chaque règle devra vous retourner un objet de type Yytoken (**c'est-à-vous de définir la classe Yytoken**), c'est à dire qu'en vis-à-vis de chaque règle vous devrez avoir une ligne de la forme return new Yytoken() avec éventuellement des paramètres, c'est vous qui décidez ce que contient cette classe, vous pouvez passer tout élément que vous souhaitez intéressant.

Voici par exemple, une ligne extraite de mon jflex :

* { return new Yytoken(Yytoken.FOIS);}

Le fichier main que je vous fournis fait appel à chaque fois à la fonction yylex() qui retourne à chaque fois un élément de type Yytoken

```
import java.io.*;

public class Main {


    public static void main(String [] argv) {

        Yytoken token;

        if (argv.length == 0) {
            System.out.println("Usage : java Expressions <inputfile>");
        }
        else {

                /* 1 - Scanner declaration */
                Expressions scanner = null;
                try {

                    scanner = new Expressions( new
java.io.FileReader(argv[0]) );
                    token = scanner.yylex();

                    while ( token != null ){
                        System.out.println(token) ;
                        token = scanner.yylex();
                    }
                }

                catch (java.io.FileNotFoundException e) {
                    System.out.println("File not found : \"" + argv[0] + "\"");
                }
                catch (java.io.IOException e) {
                    System.out.println("IO error scanning file \"" + argv[0] +
"\"");
                    System.out.println(e);
```

```
                }
                catch (Exception e) {
                    System.out.println("Unexpected exception:");
                    e.printStackTrace();

            }
        }
    }
}
```

Travail à réaliser

- bien analyser la classe Main et poser des questions à votre enseignant de compilation ;
- j'ai appelé l'analyseur lexicale engendré par jflex Expressions, mais vous pouvez l'appeler de manière différente ;
- écrire la classe Yytoken. Vous pouvez passer à partir de l'analyseur lexical autant d'éléments que vous souhaitez à cette classe. Pensez à écrire une méthode toString() pour cette classe qui retournera l'élément qu'il contient ;
- compiler l'ensemble, vous devriez obtenir le même résultat que dans la première partie mais cette fois-ci en utilisant votre propre Main.

Analyse syntaxique

Partie TD

L'étape suivante consiste à réaliser l'analyse syntaxique de la grammaire et de vérifier que celle-ci est bien de type LL(1).

Voici la grammaire de notre évaluateur d'expressions :

E → T

E → E + T

E → E - T

P → nombre

P → (E)

T → P

T → T * P

T → T / P

Calculer les ensembles premiers pour E, P, T, E étant l'axiome.

Montrer que cette grammaire est LL(1) en construisant sa table d'analyse

Travail à réaliser

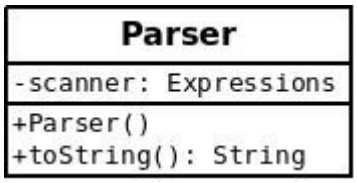
A partir de la table d'analyse, programmer un analyseur syntaxique par procédures récursives, chaque non terminale devant correspondre à une méthode récursive.

- Vous créerez une classe Parser et vous appellerez la méthode parse() de cette classe pour réaliser l'analyse syntaxique. L'analyseur devra vous indiquer si la phrase est correcte syntaxiquement ou non et vous indiquer en cas d'erreur syntaxique sur quel token (lexème) il a buté ;
- cette classe Parser aura un constructeur qui aura comme unique paramètre l'analyser lexical (*cf* exemple de fichier Main ci-dessous). Ceci permettra d'appeler la fonction yylex() qui fournit le prochain lexème directement dans ce fichier.

```
import java.io.*;

public class Main {


    public static void main(String [] argv) {

        Yytoken token;

        if (argv.length == 0) {
            System.out.println("Usage : java Expressions <inputfile>");
        }
        else {

                /* 1 - Scanner declaration */
                Expressions scanner = null;
                try {

                    scanner = new Expressions( new
java.io.FileReader(argv[0]) );
                    Parser p = new Parser(scanner);

                    p.parse();
                }

                catch (java.io.FileNotFoundException e) {
                    System.out.println("File not found : \"" + argv[0] + "\"");
                }
                catch (java.io.IOException e) {
                    System.out.println("IO error scanning file \"" + argv[0] +
"\"");
                    System.out.println(e);
                }
                catch (Exception e) {
```

```
                    System.out.println("Unexpected exception:");
                    e.printStackTrace();

            }
        }
    }
}
```

Diagramme de la classe Parser



Si vous avez tout réussi, appeler votre enseignant pour qu'il vous félicite