Université Paris Diderot
L2 Informatique et Math-Info

Analyse de Données Structurées
Année 2014-2015

TP Noté n°6 sujet 2

Analyse lexicale et syntaxique dans Java

Ce TP est divisé en trois sections (section 1 : *échauffement*, section 2 : *on ajoute le branchement et la concaténation* , section 3 : *on calcule*). **À la fin de chaque section**, vous devez soumettre sur Didel dans le travail associé à votre groupe de TD votre solution dans une archive compressée `tar.gz` nommée : *nom-section<numerosection>*`.tar.gz`, où *nom* est votre nom et *section<numerosection>* est le numero de section. Par exemple, si vous vous appelez Ralf Treinen et vous appartenez au groupe Info4, une fois terminée la section 1 vous devrez soumettre sur Didel dans le travail *Info4.Tp note - section1* une archive nommée `treinen-section1.tar.gz`.

Attention, la soumission est permise jusqu'à 13h.
Après ce délai aucune soumission ne sera plus acceptée

Il est possible de mettre à jour (jusqu'à 13h) un travail déjà soumis. On vous conseille donc de soumettre la solution à une section dès que vous l'avez trouvée et ensuite de la mettre à jour dans le cas où vous feriez de modifications.

Le but de ce TP est de construire un analyseur lexical et syntaxique pour un langage avec une variable et des affectations et branchements, défini par la grammaire suivante :

```
   <int> ::= -?[1-9][0-9]* | 0
 <instr> ::= "x :=" <int> | "x +=" <int> | "if x=0 then" <prog> "else" <prog>
  <body> ::=  <instr> |<instr> ; <body>
  <prog> ::= {<body>}
```

FIGURE 1 – un langage avec branchements et une seule variable `x`

1 Échauffement

Les premiers trois exercices sont sur une version simplifiée de la grammaire de Figure 1, ne contenant pas le branchement `if-then-else` et la concaténation `<instr>; <body>`.

Exercice 1 *Complétez le fichier `ifprogr.flex` pour obtenir, à travers `JFlex`, un analyseur lexicale qui découpe l'entrée en des jetons. Les jetons sont spécifiés dans les fichiers `Sym.java` et `Token.java`. Vous ferez attention à tenir compte des espaces ou sauts de lignes pouvant séparer les mots. Faites aussi attention à ce qu'un jeton associé à un entier garde la valeur de l'entier.*

Exercice 2 *Complétez le fichier* `Parser.java` *pour obtenir un analyseur syntaxique qui reconnaît, parmi les flots de jetons générés par* `ifprogr.flex` *, ceux qui appartiennent au langage défini par la grammaire suivante (*`$` *représentant la fin du flot) :*

$$S \mapsto P\$ \qquad P \mapsto \texttt{LPAR}\ I\ \texttt{RPAR} \qquad I \mapsto \texttt{ASSIGN INT} \mid \texttt{INCREM INT}$$

L'analyseur syntaxique traitera le flot de jetons grâce aux méthodes `boolean check(Sym s)` *et* `void eat(Sym s)` *de la classe* `LookAhead1.java`*, dont le code vous est fourni.*

Exercice 3 *La classe* `IfProgram` *contient la méthode* `main` *qui permet de tester votre code sur un fichier. Compilez et exécutez le programme sur les trois exemples* `good*` *et les trois exemples* `bad*` *qui se trouvent sur le sous-répertoire* `Section1` *du répertoire* `TP6_Sujet2` *de Didel.*

2 On ajoute le branchement et la concaténation

Exercice 4 *Modifiez les fichiers* `ifprogr.flex` *,* `Sym.java` *et* `Token.java` *pour que l'analyseur lexicale accepte le symbole de concaténation* `;` *et le branchement* `if x=0 then - else -`*.*

Exercice 5 *Modifiez le fichier* `Parser.java` *pour obtenir un analyseur syntaxique qui reconnaît les flots de jetons qui appartiennent à la grammaire définie en Figure 1 au début de ce TP. Attention : la grammaire de Figure 1 n'est pas LL(1). Il faut donc trouver une grammaire LL(1) équivalente et ensuite l'implémenter. Vous écrirez votre grammaire en commentaire au début de votre fichier* `Parser.java`*. (Suggestion : le problème est dans* `<body>`*. Il faut réécrire* `<body>` *et* `<prog>`*, en particulier* `<prog>` *doit ouvrir l'accolade* `{` *qui sera fermée par une règle* `<body>` $\mapsto$ `}`*). Aussi, rappelez-vous de prendre en compte le symbole* `$` *de fin de flot.*

Exercice 6 *Vérifiez que votre code rejette bien les expressions contenues dans les fichiers* bad* *et accepte ceux des fichiers* good* *qui se trouvent dans le sous-répertoire* `Section2` *du répertoire* `TP6_Sujet2` *de Didel.*

3 On calcule

Exercice 7 *Modifiez l'analyseur syntaxique afin qu'il affiche à la fin de l'analyse la valeur finale de la variable* `x`*. Le branchement* `if x=0 then P else Q` *teste si* `x` *vaut* `0` *(il branche sur* `P`*) ou pas (il branche sur* `Q`*), l'incrémentation* `x += n` *ajoute l'entier* `n` *à la valeur de* `x`*. Si* `x` *est utilisée (dans une incrementation ou branchement) avant toute affectation (*`x := <int>`*) alors le programme doit lever une exception. Par exemple,*

- *il affiche* `-5` *si le programme est*

  `{x := 2; if x=0 then { x += 7 } else { x += -7 }}`

- *il affiche* `666` *si le programme est*

  `{x := 0; if x=0 then {x += 6; x += 60; x += 600 } else {x += -7 }}`

- *et il déclenche une erreur si le programme est :* `{x += 6}`

Attention, il est nécessaire de modifier la classe `LookAhead1` *pour permettre à* `Parser.java` *d'accéder à la valeur d'un jeton associé à un entier.*

Exercice 8 (Bonus) *Traitez les exceptions pour que les messages d'erreur de votre programme soient informatifs.*