	ANNEE UNIVERSITAIRE 2009/2010 1ERE SESSION DE PRINTEMPS	
	Parcours : CSB6 **Code UE :** INF354 **Epreuve :** Analyse Syntaxique **Date :** 3 Mai 2010 **Heure :** 11H **Durée :** 1H30 Documents : autorisés Epreuve de : Mme Frédérique Carrère	

Les exercices sont indépendants.

Exercice 1

On considère un fichier contenant du code Java commenté. Les commentaires-lignes se situent à droite du motif `//`. Les commentaires ordinaires sont encadrés par les motifs `/*` et `*/`. Les commentaires de documentation sont encadrés par les motifs `/**` et `*/`, ils peuvent se terminer par une liste de motifs spéciaux, appelés `tags`. Le nom d'un `tag` commence par le caractère arobase (`@`) et figure nécessairement en début de ligne (en ignorant les espaces et les `*`). Si des mots commençant par `@` figurent ailleurs qu'en début de ligne, ils ne seront pas considérés comme des `tags`. Chaque `tag` a un texte associé (éventuellement sur plusieurs lignes) qui se termine au `tag` suivant ou au motif final `*/`. Exemple :

```
/**
* Returns the character at the specified index.
*
* @param     index  the index of the desired character.
* @return    the desired character.
* @exception StringIndexOutOfRangeException
*              if the index is not in the range <code>0</code>
*              to <code>length()-1</code>.
*/
```

Ecrire un analyseur lexical, dans le format reconnu par `lex`, qui élimine tous les commentaires du fichier java et affiche dans un fichier nommé `docfile` (on utilisera `fprintf`) les `tags` `@param`, `@return` et `@exception` des commentaires de documentation figurant dans le fichier java, avec leur texte associé.

Exercice 2

Considérons la grammaire suivante :

```
%%
S : T T
;
```

```
T : 'a' S 'b'
  |
;
%%
```

1. Dessiner l'automate des $LR(0)$-items, en indiquant pour chaque état la liste complète de ses items. On pensera à rajouter un marqueur de fin de chaîne et la règle correspondante.
2. L'automate fait-il apparaître des conflits ? Si oui, lesquels et de quel type ?
3. La grammaire est-elle $LR(0)$, $SLR(1)$? Justifiez vos réponses.

Exercice 3

Soit la grammaire G suivante, écrite dans le format reconnu par `yacc` :

```
%token ID FOR DO
%%
instr :  FOR '(' expr ';' expr ';' expr')'  '{' suite_instr '}'
      | expr ';'
;

suite_instr : instr suite_instr
            |
;

expr : ID
     |
;
%%
```

Le token `ID` représente un identificateur. L'automate des $LR(0)$-items est donné en annexe.

1. Calculer pour chaque symbole non terminal de cette grammaire les ensembles *Premier* et *Suivant* correspondants.
2. Donner l'arbre de dérivation du mot `FOR( ; ; ) { ID ; }` dans cette grammaire.
3. D'après l'automate donné en annexe, quel est l'état atteint après la lecture de la première occurrence du caractère `';'` ? Pourquoi `yacc` ne signale-t-il pas de conflit dans cet état ? Quelle est l'action choisie dans le cas de l'exemple ? Donnez le contenu de la pile avant et après cette action.
4. Quel est l'état atteint aprés la lecture de la première occurrence du caractère `'{'` ? Pourquoi `yacc` ne signale-t-il pas de conflit dans cet état ? Quelle est l'action choisie dans le cas de l'exemple ? Donnez le contenu de la pile avant et après cette action.

Exercice 4

Soit la grammaire suivante.

```
%token ID NUM PLUS MULT
%%
expr : expr PLUS expr
     | expr MULT expr
     | '(' expr ')'
     | ID
     | NUM
;
%%
```

1. La grammaire est ambigüe. Que faut-il rajouter dans le fichier `yacc` pour que `yacc` ne signale pas de conflits ?
2. Ajouter des règles de grammaire pour pouvoir inclure des appels de fonctions dans un calcul arithmétique. Les fonctions peuvent avoir un nombre quelconque d'arguments.
3. Ajouter un calcul d'attributs (on indiquera TOUTES les modifications à apporter au fichier `yacc` ET éventuellement au fichier `lex`) qui permette d'afficher le nom de toutes les fonctions utilisées ainsi que leur nombre d'arguments.
   Par exemple : `f(g(x)+1, y, h(x, 2*z), z))` produira l'affichage :

   ```
   fonction g: 1 argument
   fonction h: 2 arguments
   fonction f: 4 arguments
   ```

Annexe :

```
   0  $accept : suite_instr $end
   1  suite_instr : instr suite_instr
   2              |
   3  instr : FOR '(' expr ';' expr ';' expr ')' '{' suite_instr '}'
   4        | expr ';'
   5  expr : ID
   6       |

state 0
$accept : . suite_instr $end  (0)
suite_instr : .  (2)
expr : .  (6)

ID  shift 1
FOR  shift 2
$end  reduce 2
';'  reduce 6

suite_instr  goto 3
instr  goto 4
expr  goto 5
```

```
state 1
expr : ID .  (5)

.  reduce 5

state 2
instr : FOR . '(' expr ';' expr ';' expr ')' '{' suite_instr '}'  (3)

'('  shift 6
.  error

state 3
$accept : suite_instr . $end  (0)

$end  accept

state 4
suite_instr : instr . suite_instr  (1)
suite_instr : .  (2)
expr : .  (6)

ID  shift 1
FOR  shift 2
$end  reduce 2
';'  reduce 6
'}'  reduce 2

suite_instr  goto 7
instr  goto 4
expr  goto 5

state 5
instr : expr . ';'  (4)

';'  shift 8
.  error

state 6
instr : FOR '(' . expr ';' expr ';' expr ')' '{' suite_instr '}'  (3)
expr : .  (6)

ID  shift 1
';'  reduce 6

expr  goto 9
```

```
state 7
suite_instr : instr suite_instr .  (1)

.  reduce 1

state 8
instr : expr ';' .  (4)

.  reduce 4

state 9
instr : FOR '(' expr . ';' expr ';' expr ')' '{' suite_instr '}'  (3)

';'  shift 10
.  error

state 10
instr : FOR '(' expr ';' . expr ';' expr ')' '{' suite_instr '}'  (3)
expr : .  (6)

ID  shift 1
';'  reduce 6

expr  goto 11

state 11
instr : FOR '(' expr ';' expr . ';' expr ')' '{' suite_instr '}'  (3)

';'  shift 12
.  error

state 12
instr : FOR '(' expr ';' expr ';' . expr ')' '{' suite_instr '}'  (3)
expr : .  (6)

ID  shift 1
')'  reduce 6

expr  goto 13

state 13
instr : FOR '(' expr ';' expr ';' expr . ')' '{' suite_instr '}'  (3)

')'  shift 14
.  error
```

```
state 14
instr : FOR '(' expr ';' expr ';' expr ')' . '{' suite_instr '}'  (3)

'{'  shift 15
.  error

state 15
instr : FOR '(' expr ';' expr ';' expr ')' '{' . suite_instr '}'  (3)
suite_instr : .  (2)
expr : .  (6)

ID  shift 1
FOR  shift 2
';'  reduce 6
'}'  reduce 2

suite_instr  goto 16
instr  goto 4
expr  goto 5

state 16
instr : FOR '(' expr ';' expr ';' expr ')' '{' suite_instr . '}'  (3)

'}'  shift 17
.  error

state 17
instr : FOR '(' expr ';' expr ';' expr ')' '{' suite_instr '}' .  (3)

.  reduce 3

10 terminals, 4 nonterminals
7 grammar rules, 18 states
```