	ANNEE UNIVERSITAIRE 2008/2009 1ERE SESSION DE PRINTEMPS **Parcours :** CSB6 **Code UE :** INF354 **Epreuve :** Analyse Syntaxique **Date :** 7 Mai 2009 **Heure :** 11H **Durée :** 1H30 Documents : autorisés Epreuve de : Mme Frédérique Carrère	

Les exercices sont indépendants.

Exercice 1

Dans un document au format html, une balise pour une image à la forme suivante :

```
<IMG SRC="images/image.gif"
        ALIGN=center
        ALT="Texte remplaçant l'image"
        TITLE="Mon image"
        WIDTH=150
        HEIGHT=70>
```

Les mot-clés désignant des attributs (SRC, TITLE, ...) peuvent être écrits en minuscules. Les attributs ne sont pas forcément donnés dans cet ordre.
Ecrire un analyseur lexical, dans le format reconnu par `lex`, qui prend en entrée un document HTML et effectue les traitements suivants :

1. supprime toutes les images du document,
2. compte le nombre d'images au format gif.

Attention : le document peut contenir d'autres balises délimitées par les signes < et >.

Exercice 2

Considérons la grammaire suivante :

```
%%
S : A B
;

B : 'b' A B
   | /* epsilon*/
;

A : '(' S ')'
  | 'a'
;
%%
```

1. Construire 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. Calculer pour chaque symbole non terminal de cette grammaire les ensembles *Premier* et *Suivant* correspondants.
4. 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 WHILE DO
%%
instr :  WHILE '(' expr ')' DO '{' suite_instr '}'
      | expr
;

suite_instr : instr ';' suite_instr
            | instr ';'
;

expr : ID
;
%%
```

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

1. Donner l'arbre de dérivation du mot $WHILE$ (ID) DO { ID ; ID ; } dans cette grammaire.
2. D'après l'automate donné en annexe, quel est l'état atteint aprés la lecture de la première occurrence de ID ; ?
3. Pourquoi `yacc` ne signale-t-il pas de conflit dans cet état ? Quelle est l'action choisie dans le cas de l'exemple ?
4. Quel est l'état atteint aprés la lecture de la deuxième occurrence de ID ; ? Détailler l'évolution du haut de la pile au cours de l'action qui est alors effectuée (on fera figurer les numéros d'états dans la pile).

Exercice 4

Soit la grammaire suivante qui permet de calculer des sommes d'entiers, y compris des sommes de fonctions entières sur un intervalle. Par exemple :
somme (i = 1 .. 3, f(i)).
On considère seulement deux fonctions : $carre(i)$ et $cube(i)$. La grammaire est donnée dans le format reconnu par `yacc` :

```
%token ID  NUM INT SOMME CARRE CUBE
%left PLUS

%%
axiom : E ';'
;

E :  E  PLUS E
  |  S
  |  INT
;

S : SOMME '(' ID '=' INT '.''.' INT ',' F '(' ID ')' ')'
;

F : CARRE
  | CUBE
;

%%
```

Ajouter pour chaque règle de grammaire, un calcul d'attributs qui permette d'évaluer les expressions. Le résultat sera afficher lors de l'action associée à la règle $axiom : E \ ';' \ .$

Annexe :

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

state 0
        $accept : . instr $end  (0)

        ID  shift 1
        WHILE  shift 2
        .  error

        instr  goto 3
        expr  goto 4

state 1
        expr : ID .  (5)

        .  reduce 5
```

```
state 2
        instr : WHILE . '(' expr ')' DO '{' suite_instr '}'  (1)

        '('  shift 5
        .  error

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

        $end  accept

state 4
        instr : expr .  (2)

        .  reduce 2

state 5
        instr : WHILE '(' . expr ')' DO '{' suite_instr '}'  (1)

        ID  shift 1
        .  error

        expr  goto 6

state 6
        instr : WHILE '(' expr . ')' DO '{' suite_instr '}'  (1)

        ')'  shift 7
        .  error

state 7
        instr : WHILE '(' expr ')' . DO '{' suite_instr '}'  (1)

        DO  shift 8
        .  error

state 8
        instr : WHILE '(' expr ')' DO . '{' suite_instr '}'  (1)

        '{'  shift 9
        .  error

state 9
        instr : WHILE '(' expr ')' DO '{' . suite_instr '}'  (1)

        ID  shift 1
        WHILE  shift 2
```

```
        .  error

        instr  goto 10
        expr  goto 4
        suite_instr  goto 11

state 10
        suite_instr : instr . ';' suite_instr  (3)
        suite_instr : instr . ';'  (4)

        ';'  shift 12
        .  error

state 11
        instr : WHILE '(' expr ')' DO '{' suite_instr . '}'  (1)

        '}'  shift 13
        .  error

state 12
        suite_instr : instr ';' . suite_instr  (3)
        suite_instr : instr ';' .  (4)

        ID  shift 1
        WHILE  shift 2
        '}'  reduce 4

        instr  goto 10
        expr  goto 4
        suite_instr  goto 14

state 13
        instr : WHILE '(' expr ')' DO '{' suite_instr '}' .  (1)

        .  reduce 1

state 14
        suite_instr : instr ';' suite_instr .  (3)

        .  reduce 3

10 terminals, 4 nonterminals
6 grammar rules, 15 states
```