

Université Paris 7 — IUP 1<sup>ère</sup> année  
Examen de compilation  
13 septembre 2002 — Durée : 2h00

*La notation tiendra compte de la clarté des justifications fournies.*

*Le barème n'est donné qu'à titre indicatif.*

*Documents de cours, TD, TP autorisés.*

**Exercice 1 (5 points)** Répondre par vrai ou faux aux questions suivantes. Barème : 0,5 pour une réponse correcte, -0,5 pour une réponse incorrecte et 0 en cas d'absence de réponse. Dans cet exercice, il n'est pas demandé de justification. Vrai signifie « toujours vrai » et Faux signifie « pas toujours vrai ».

	Vrai	Faux
1. Toute grammaire LR(0) est LL(1).....	<input type="checkbox"/>	<input type="checkbox"/>
2. Toute grammaire LL(1) est LR(1).....	<input type="checkbox"/>	<input type="checkbox"/>
3. Une grammaire récursive droite ne peut pas être LR(0).....	<input type="checkbox"/>	<input type="checkbox"/>
4. Le premier état de l'automate LR(0) d'une grammaire $G$ contient toutes les règles de $G$ , le marqueur étant au début de chaque règle. ....	<input type="checkbox"/>	<input type="checkbox"/>
5. Une grammaire non factorisée à gauche ne peut pas être LL(1).....	<input type="checkbox"/>	<input type="checkbox"/>
6. Si une grammaire factorisée n'est pas récursive gauche, elle est LL(1).....	<input type="checkbox"/>	<input type="checkbox"/>
7. L'ensemble des mots bien parenthésés sur l'alphabet $\{(,)\}$ peut être décrit par une expression <code>lex</code> utilisant seulement les opérateurs concaténation, $*$ , et $ $ . ....	<input type="checkbox"/>	<input type="checkbox"/>
8. Tout langage construit à partir des caractères en appliquant un nombre fini de fois la concaténation et les opérateurs $*$ , $ $ est engendré par une grammaire LL(1). ....	<input type="checkbox"/>	<input type="checkbox"/>
9. Il est possible de calculer tous les attributs d'une grammaire S-attribuée lors d'un parcours en profondeur de l'arbre de dérivation. ....	<input type="checkbox"/>	<input type="checkbox"/>
10. Il est possible de calculer tous les attributs d'une grammaire L-attribuée lors d'un parcours en profondeur de l'arbre de dérivation. ....	<input type="checkbox"/>	<input type="checkbox"/>

**Exercice 2 (5 points)** Écrire un source `lex` permettant d'afficher dans un programme `C` toutes les lignes contenant un commentaire complet, et uniquement celles-ci. On rappelle que :

- un commentaire `C` commence par `/*`, se termine par `*/` ;
- les commentaires `C` ne s'emboîtent pas ;
- un commentaire se trouve hors de toute chaîne de caractères constante (délimitée par les caractères `"`).

Attention, si une ligne contient un commentaire complet, le programme devra afficher la ligne tout entière, et pas seulement le commentaire.

**Exercice 3 (10 points)** On s'intéresse dans cet exercice aux expressions arithmétiques utilisant les opérateurs binaires  $+$ ,  $-$ ,  $*$  et  $/$  avec leur signification usuelle. Une telle expression peut s'écrire en notation *infixe* habituelle, dans laquelle on écrit d'abord le premier opérande, puis l'opérateur ( $+$ ,  $-$ ,  $*$  ou  $/$ ), et enfin le second opérande.

La notation *postfixe* pour les expressions arithmétiques consiste à écrire le premier opérande, puis le second, puis l'opérateur. Cette notation n'utilise pas les parenthèses. Par exemple, l'expression en

notation infixe suivante :  $a * (b + c)$  s'écrit en notation postfixe :  $a b c + *$ .

1. Écrire un programme C utilisant `lex` et `yacc` qui lit sur l'entrée standard une expression arithmétique en notation infixe et écrit sur la sortie standard l'expression en notation postfixe. Les lexèmes reconnus sont
  - les quatre opérateurs ci-dessus (avec leur priorité habituelle) ;
  - les parenthèses ;
  - les nombres entiers et les identificateurs du langage C ;
  - les commentaires, commencés par la chaîne `//` et finissant en fin de ligne ;
  - les blancs, qui seront supprimés par l'analyseur lexical.

Expliquer comment seront résolus les conflits éventuels du source `yacc`.

2. La grammaire utilisée dans le programme écrit à la question précédente est-elle LR(0)? SLR(1)? Justifier.
3. La grammaire utilisée dans le programme écrit à la question 1 est-elle LL(1)? Justifier.
4. Modifier le programme pour traduire les expressions arithmétiques en notation infixe en expressions arithmétiques en notation préfixe, c'est-à-dire dans lesquelles l'opérateur précède ses opérandes.
5. On considère maintenant une machine virtuelle disposant d'un nombre infini de registres  $R_i$  ( $i \geq 0$ ), et le langage intermédiaire comprenant les instructions suivantes :

- `load <entier_ou_variable>, Ri ;`
- `add Ri, Rj ;`
- `sub Ri, Rj ;`
- `mul Ri, Rj ;`
- `div Ri, Rj ;`

Chacune des instructions `load`, `add`, `sub`, `mul`, `div` a deux opérandes. Le premier opérande de `load` représente un entier ou un nom de variable. Les instructions arithmétiques s'effectuent avec comme premier opérande (le contenu de)  $R_i$ , comme second opérande  $R_j$  et rangent le résultat de l'opération dans  $R_i$ .

On demande d'écrire un traducteur d'expressions arithmétiques vers un programme sur ce langage, de telle sorte qu'à l'exécution du programme généré, le registre `R0` contienne la valeur calculée.

À titre d'exemple, l'expression  $10 * a + b$  pourra être traduite en :

```
load 10, R0
load a, R1
mul R0, R1
load b, R1
add R0, R1
```