Grammaires attribuées

Exercice 1 : On considère la grammaire suivante qui décrit des expressions arithmétiques préfixées binaires :

exp	→	chiffre | (+ exp exp) | (∗ exp exp)
chiffre	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Q1 - Attribuer cette grammaire de façon à calculer la valeur de l'expression analysée.

On généralise la grammaire précédente pour décrire maintenant des expressions n-aires (n≥2) :

exp	→	chiffre | (+ exp sexp) | (∗ exp sexp)
sexp	→	sexp exp | exp
chiffre	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Q2 - Donner l'arbre de dérivation de l'expression (+ 1 (∗ 2 3) 4 3)

Q3 - Attribuer cette nouvelle grammaire de façon à calculer là-encore la valeur de l'expression analysée.

Exercice 2 :

On considère la grammaire suivante :

programme	→	bloc
bloc	→	{ dec ; sinst }
dec	→	<u>var</u> suite-ident
suite-ident	→	ident , suite-ident | ident
sinst	→	sinst ; inst | inst
inst	→	opcode (suite-arg) | bloc
suite-arg	→	suite-arg , ident | ident

Remarques :
- pour simplifier, les variables ne sont pas typées mais doivent être déclarées avant d'être utilisées.
- ident et opcode sont des unités lexicales qui désignent respectivement un identificateur quelconque non réservé et un identificateur de fonction prédéfini.

On souhaite associer à chaque occurrence d'identificateur sa *profondeur* (≥ 1) et son *rang* (≥ 1) de déclaration ; ainsi, sur l'exemple suivant, on a fait figurer le couple (*profondeur*, *rang*) correspondant à chacune des occurrences :

```
{ var a, b, c ;          // (1, 1) pour a, (1, 2) pour b et (1, 3) pour c
  f1 (a, c) ;            // (1, 1) pour a et (1, 3) pour c
  { var d, b ;           // (2, 1) pour d et (2, 2) pour b
    f2 (d, b, c) ;       // (2, 1) pour d, (2, 2) pour b et (1, 3) pour c
    { var c, a, d ;      // (3, 1) pour c, (3, 2) pour a et (3, 3) pour d
      f2 (a, b, d) ;     // (3, 2) pour a, (2, 2) pour b et (3, 3) pour d
      f1 (c, a)          // (3, 1) pour c et (3, 2) pour a
    }
  }
  { var e, b, a ;        // (2, 1) pour e, (2, 2) pour b et (2, 3) pour a
    . . .
  }
  . . .
}
```

Q1 - Donner un système d'attributs permettant d'associer, à chaque occurrence de **déclaration** de variable du programme source, la *profondeur* du bloc où elle est déclarée et son *rang* dans sa liste de déclaration.

Q2 - Compléter la solution précédente (introduction de nouveaux attributs et calcul de ceux-ci) afin de pouvoir associer à chaque occurrence d'**utilisation** d'un identificateur la *profondeur* et le *rang* de déclaration de cet identificateur.

Exercice 3 : Soit une grammaire (simplifiée) de déclarations de type

dectypes	→	déclarations
déclarations	→	déclarations type
	|	vide
type	→	<u>union</u> ident { liste_champs }
	|	<u>struct</u> ident { liste_champs }
liste_champs	→	liste_champs ; champ
	|	champ
champ	→	int liste_déclarateurs
	|	float liste_déclarateurs
	|	char liste_déclarateurs
	|	<u>union</u> ident liste_déclarateurs
	|	<u>struct</u> ident liste_déclarateurs
liste_déclarateurs	→	liste_déclarateurs , déclarateur
	|	déclarateur
déclarateur	→	ident

Q1 - On exige que dans une déclaration de type les identificateurs de type référencés aient tous été déclarés avant d'être utilisés. Le langage fournit les types prédéfinis int, float et char. Exemples :

<u>union</u> u1 {int n ; char c}	// construction correcte
<u>struct</u> s1 {int m ; int n}	// construction correcte
<u>struct</u> s2 {int m2 ; <u>struct</u> s1 premier, deuxième}	// construction correcte
<u>struct</u> s3 {int m3 ; <u>struct</u> s4 s}	// construction incorrecte
<u>struct</u> s4 {int m4 ; <u>struct</u> s4 lien1}	// construction incorrecte

Rédiger un système d'attributs permettant de vérifier la bonne construction des types et de calculer l'ensemble des types correctement définis et l'ensemble des types incorrectement définis.

Q2 - On ajoute la production :

déclarateur	→	* ident

et on prévoit une exception pour l'utilisation des types dans une déclaration : afin de construire des définitions récursives, on tolère que le déclarateur d'un champ de type structure soit un pointeur sur le type structure en cours de définition.

Exemples :

<u>struct</u> s4 {int m4 ; <u>struct</u> s4 lien1}	// construction incorrecte
<u>struct</u> cellule {int info ; <u>struct</u> cellule *suiv, *pred}	// construction correcte

Introduire de nouveaux attributs et fournir les modifications à apporter pour prendre en compte cette nouvelle possibilité.