Licence Sciences et Technologie **UGA**
Parcours Informatique **UE INF404**

Devoir Surveillé du 15 mars 2018

Durée : 1h - Document autorisé : une feuille A4 recto-verso - Barême indicatif

Les programmes demandés peuvent êtres écrits en (pseudo-) C ou en notation algorithmique ...

Introduction

On s'intéresse à un langage L permettant de définir des *ensembles*, éventuellement vides, dont les éléments sont soit des entiers, soit des ensembles d'entiers.

On donne ci-dessous quelques exemples de phrase du langage L :
- $\emptyset$
- $\{\emptyset\}$
- $\{1, 8, 42\}$
- $\{\{1\}, \{8\}, \{42\}\}$
- $\{\{1\}, \{\{8\}, \{42\}\}\}$

Questions

Q1 [1 pt]. En vous aidant de ces exemples donnez le vocabulaire (c'est-à-dire l'ensemble des caractères) sur lequel est défini L.

Q2. [1 pt] D'après vous L est-il un langage régulier ? Pourquoi ? (réponse en une phrase !)

Dans la suite, on considère que le langage L est défini sur l'ensemble de lexèmes suivant :

$$V_t = \{VIDE, ACCO, ACCF, VIRG, ENTIER\}$$

dans lequel $VIDE$ dénote l'ensemble vide ($\emptyset$), $ACCO$ et $ACCF$ les accolades ouvrantes et fermantes, $VIRG$ une virgule et $ENTIER$ un entier (une suite non vide de chiffres).

Q3 [3 pt]. Dessinez un automate permettant de reconnaitre les éléments de V_t.

La syntaxe du langage L est donnée par la grammaire ci-dessous, dont l'axiome est *ens* :

$$\begin{aligned}
ens &\rightarrow VIDE \\
ens &\rightarrow ACCO\ liste_ens\ ACCF \\
liste_ens &\rightarrow elem\ suite_liste_ens \\
suite_liste_ens &\rightarrow VIRG\ elem\ suite_liste_ens \\
suite_liste_ens &\rightarrow \varepsilon \\
elem &\rightarrow ENTIER \\
elem &\rightarrow ens
\end{aligned}$$

Q4 [2 pt]. Donnez un exemple de phrase comportant une erreur lexicale et un exemple de phrase comportant une erreur syntaxique.

Q5 [3 pt]. Dessinez l'*arbre de dérivation* de la phrase $\{\emptyset, 1, \{8\}\}$

Q6 [4 pt]. En utilisant les types et fonctions définies en Annexe A, écrivez la fonction `analyser` spécifiée ci-dessous :

```
void analyser(char *nom_fichier) ;
// e.i. : indifferent
// e.f. : une sequence de lexemes a ete lue dans le fichier nom_fichier,
//         une erreur syntaxique est signalee  si elle ne respecte pas la grammaire
```

Q7 [3 pt]. On suppose que l'analyse syntaxique a permis de produire l'*arbre abstrait* de la Figure 1 pour la phrase $\{\emptyset, 1, \{8\}\}$.

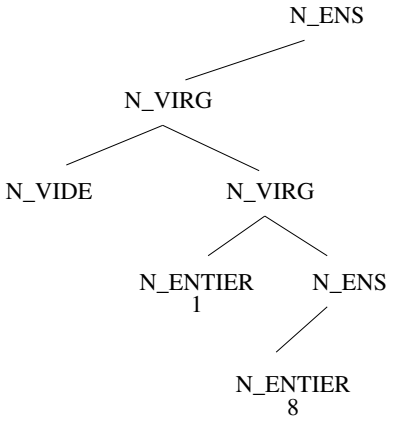

FIGURE 1 – Arbre abstrait de la phrase $\{\emptyset, 1, \{8\}\}$

En vous aidant des types et fonctions définies en Annexe B écrivez la fonction `nbEntiers` définie ci-dessous :

```
int nbEntiers (Ast A) ;
// e.i. : A est l'arbre abstrait d'une phrase du langage L
// e.f. : nbEntiers(A) renvoie le nombre d'entiers present dans l'arbre abstrait A
```

Sur l'arbre abstrait A de la Figure 1, on aurait `nbEntiers(A)` $= 2$.

Q8 (plus difficile !) [4pt].
Pour finir, en vous aidant des types et fonctions définies en Annexe C, ré-écrivez la fonction `analyser` définie ci-dessous :

```
void analyser(char *nom_fichier, Ast *A) ;
// e.i. : indifferent
// e.f. : une sequence de lexemes a ete lue dans le fichier nom_fichier,
//         une erreur syntaxique est signalee  si elle ne respecte pas la grammaire,
//         sinon *A contient l'arbre abstrait de la phrase qui a ete lue.
```

Annexe A : le fichier analyse_lexicale.h

```
   typedef enum {VIDE, ACCO, ACCF, VIRG, ENTIER} Nature_Lexeme ;

   typedef struct {
      Nature_Lexeme nature;    // nature du lexeme
      char chaine[256];           // chaine de caracteres
   } Lexeme ;

   void demarrer(char *nom_fichier);
   // initialise l'analyse lexicale

   void avancer();
   // lit le lexeme suivant

   Lexeme lexeme_courant();       // pourra etre abrege en "LC"
   // valeur du lexeme courant

   int fin_de_sequence();
   // vrai ssi la fin de sequence est atteinte
```

Annexe B : le fichier type_ast.h

```
typedef enum {N_VIDE, N_VIRG, N_ENTIER, N_ENS} TypeAst ;

typedef struct noeud {
    TypeAst nature ;
    struct noeud *gauche, *droit ;
} NoeudAst ;

typedef NoeudAst *Ast ;
```

Annexe C : le fichier ast_construction.h

```
#include "type_ast.h"

Ast creer_ens(Ast astg) ;
// renvoie un arbre abstrait de nature N_ENS,
//    de fils gauche astg

Ast creer_virg(Ast astg, Ast astd) ;
// renvoie un arbre abstrait de nature N_VIRG,
//    de fils gauche astg et de fils droit astd

Ast creer_vide() ;
// renvoie un arbre abstrait "feuille", de nature N_VIDE

Ast creer_entier() ;
// renvoie un arbre abstrait "feuille", de nature N_ENTIER
```