Licence Sciences et Technologies **Université Grenoble Alpes**
Parcours Informatique **UE INF404**

Devoir Surveillé du 14 mars 2019

Durée : 1h – Documents autorisés : une feuille A4 recto-verso – Barême indicatif

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

On s'intéresse à un langage $\mathcal{L}$ permettant de définir des figures géométriques constituées d'éléments imbriqués et alignés. Ces éléments sont soit des *ellipses* (i.e., des rectangles aux coins arrondis), soit des *rectangles*. La Figure 1 représente un exemple de figure géométrique correspondant à une ellipse contenant 3 éléments imbriqués, de gauche à droite : un rectangle ; une ellipse avec un rectangle imbriqué ; une ellipse.

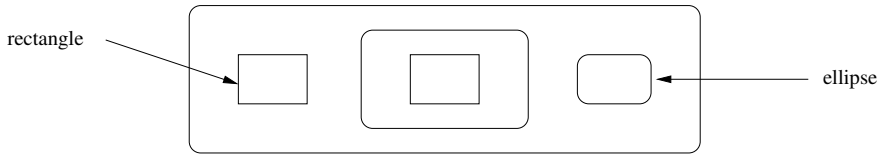

Figure 1: Un exemple de figure géométrique

Le langage $\mathcal{L}$ est défini sur l'ensemble de lexèmes suivants :

$$V_t = \{PARO, PARF, CROCHO, CROCHF, BARRE\}$$

où $PARO$ et $PARF$ représentent les parenthèses ouvrantes `'('` et fermantes `')'`, $CROCHO$ et $CROCHF$ les crochets ouvrants `'['` et fermants `']'`, et $BARRE$ la barre verticale `'|'`. Les caractères séparateurs sont les espaces. Intuitivement, les parenthèses définissent des ellipses, les crochets des rectangles et la barre verticale la juxtaposition de plusieurs éléments sur une même ligne.

Ainsi, la Figure 1 est représentée par la phrase suivante : `([] | ([]) | ())`

Q1. Dessinez un automate permettant de reconnaitre les éléments de V_t.

On donne ci-dessous une grammaire qui définit la syntaxe du langage $\mathcal{L}$.

$$\begin{aligned}
figure &\rightarrow ligne_elems \\
ligne_elems &\rightarrow elem\ suite_ligne_elems \\
suite_ligne_elems &\rightarrow BARRE\ elem\ suite_ligne_elems \\
suite_ligne_elems &\rightarrow \varepsilon \\
elem &\rightarrow PARO\ ligne_elem_imbric\ PARF \\
elem &\rightarrow CROCHO\ ligne_elem_imbric\ CROCHF \\
ligne_elem_imbric &\rightarrow ligne_elems \\
ligne_elem_imbric &\rightarrow \varepsilon
\end{aligned}$$

Q2. Donnez un exemple de phrase du langage $\mathcal{L}$ comportant une **erreur lexicale**.

Q3. Donnez un exemple de phrase du langage $\mathcal{L}$ comportant une **erreur syntaxique**.

Q4. D'après vous $\mathcal{L}$ est-il un langage régulier ? Pourquoi ?

Q5. Dessinez l'**arbre de dérivation** de la phrase `( () | [] )`

Q6. Dessinez la figure géométrique représentée par la phrase `( () | [] )`

On donne en Annexe A la spécification d'un module d'**analyse lexicale** similaire à celui utilisé en TP.

Q7. Ecrire 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
```

Pour effectuer des traitements sur les figures obtenues à partir de phrases du langage $\mathcal{L}$ on étend[1] la procédure `analyser()` pour qu'elle produise un **arbre abstrait**. L'arbre abstrait correspondant à l'exemple de la Figure 1 est donnée ci-dessous[2] :

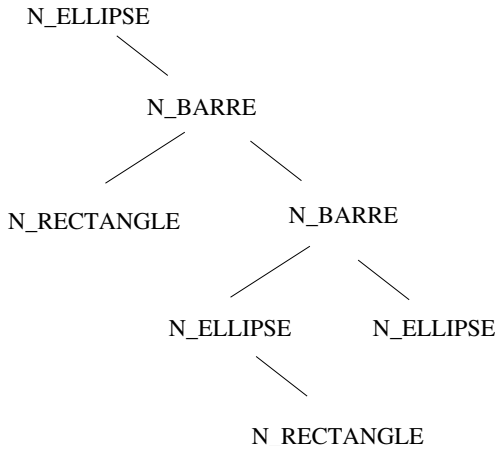

Figure 2: L'arbre abstrait correspondant à la Figure 1

On donne également en Annexe B l'interface du module définissant cet arbre abstrait.

On définit alors la **profondeur** d'une figure géométrique comme le nombre maximal d'éléments imbriqués qu'elle contient. Pour la Figure 1 la profondeur est de 3.

Q8. Ecrire la fonction `profondeur` spécifiée ci-dessous :

```
int profondeur(Ast A) ;
// e.i. : A est l'arbre abstrait d'une phrase du langage L
// e.f. : profondeur(A) renvoie la profondeur de la figure representee par A
```

[1] il n'est pas demandé de programmer cette extension ...

[2] lorsqu'un noeud de l'arbre ne possède qu'un seul fils il s'agit toujours du **fils droit**

A Annexe A : le fichier `analyse_lexicale.h`

```
typedef enum {PARO, PARF, CROCHO, CROCH, BARRE} 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 atteint
```

B Annexe B : le fichier `type_ast.h`

```
typedef enum {N_ELLIPSE, N_RECTANGLE, N_BARRE} TypeAst ;

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

typedef NoeudAst *Ast ;
```