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

**Devoir Surveillé du 12 mars 2021**

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

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

On s'intéresse à un langage $\mathcal{L}$ permettant de définir un texte (c'est-à-dire une séquence de mots) dont certains mots peuvent être affichés (par un interpréteur) soit en caractères normaux, soit en caractères **gras**, soit en caractères *italiques*. Dans ce langage :

- les mots sont des séquences non vides de lettres de l'alphabet, en minuscules ;
- les portions de texte à afficher en **gras** s'écrivent entre caractères `'*'` ;
- les portions de texte à afficher en *italique* s'écrivent entre caractères `'#'` ;

Ainsi, le texte :

```
la nuit * tous * les # chats * sont gris * # et les #souris *aussi *#
```

sera affiché comme :

la nuit **tous** les *chats **sont gris*** et les *souris **aussi***

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

$$V_t = \{MOT, ETOILE, DIESE, FDS\}$$

où $ETOILE$ et $DIESE$ représentent les caractères `'*'` et `'#'`, $MOT$ une séquence non vide de **lettres minuscules** (entre `'a'` et `'z'`) et $FDS$ le caractère de fin de séquence.

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

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

$$\begin{aligned}
texte &\rightarrow portion_texte\ suite_texte\ FDS \\
portion_texte &\rightarrow ETOILE\ portion_texte\ suite_texte\ ETOILE \\
portion_texte &\rightarrow DIESE\ portion_texte\ suite_texte\ DIESE \\
portion_texte &\rightarrow MOT \\
suite_texte &\rightarrow portion_texte\ suite_texte \\
suite_texte &\rightarrow \varepsilon
\end{aligned}$$

**Q2.** Combien d'erreurs **lexicales** contient la phrase suivante :

```
# Pourquoi les flamands * roses * sont-ils * # roses * ? #
```

**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 ?

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

**Q5.** 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**. Un exemple d'arbre abstrait est donné ci-dessous, sachant que :

- les noeuds `N_SEP` servent à séparer les portions de texte, ils ont toujours un fils droit et un fils gauche
- les noeuds `N_GRAS` (resp. `N_ITAL`) ont uniquement un fils droit qui correspond à une portion de texte en gras (resp. en italique).

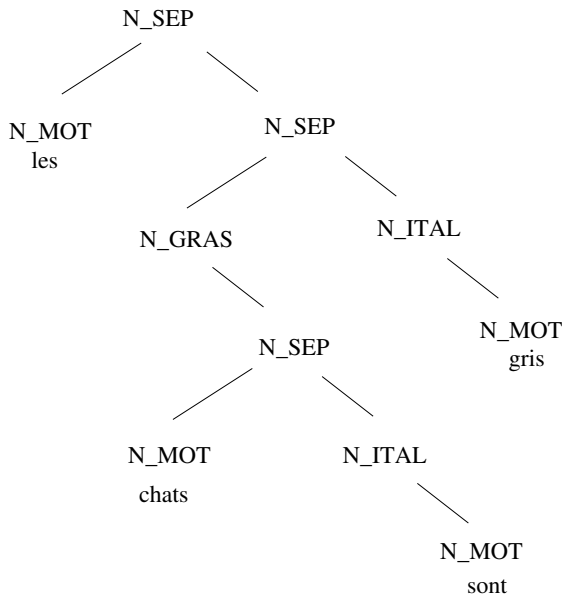


Figure 1: l'arbre abstrait du texte `les * chats # sont # * # gris #`

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

**Q6.** On souhaite désormais rejeter des textes contenant des *MOTS* affichés **à la fois** en gras **et** en italique. Par exemple, la phrase "les **chats** ***sont*** *gris*" (correspondant à l'arbre abstrait de la Figure 1) contient le mot "***sont***" affiché à la fois en gras **et** en italique, elle est donc incorrecte.

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

Ecrire la fonction `verifie_gi` spécifiée ci-dessous :

```
int verifie_gi(Ast A) ;
// e.i. : A est l'arbre abstrait d'une phrase du langage L
// e.f. : verifie_gi(A) renvoie vrai si et seulement si cette phrase
           ne contient pas de mots en gras et italique
```

**Indication :** on pourra utiliser (en les écrivant) deux fonctions récursives auxilliaires de la forme `int contient_gras (Ast A)` et `int contient_ital (Ast A)` qui indiquent respectivement si l'arbre abstrait `A` contient ou non un sous-arbre correspondant à une portion de texte en gras ou en italique.

## Annexe A : le fichier `analyse_lexicale.h`

```
typedef enum {MOT, ETOILE, DIESE, FDS} 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
```

## Annexe B : le fichier `type_ast.h`

```
typedef enum {N_MOT, N_GRAS, N_ITAL, N_SEP} TypeAst ;

typedef struct noeud {
      TypeAst nature ;   // nature du noeud
      struct noeud *gauche, *droit ;  // fils gauche et droit
      } NoeudAst ;

typedef NoeudAst *Ast ;
```