L2 Licence Sciences et Technologie **UGA**
Parcours Informatique **UE INF404**

Devoir Surveillé du 16 mars 2022

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

Les programmes peuvent être écrits en C et/ou en notation algorithmique

Introduction

On s'intéresse à un langage d'expressions portant sur des chaînes de caractères. Dans ce langage, une chaîne de caractères est soit la chaîne vide (`[]`), soit une suite de lettres minuscules entre crochets (comme `[bonjour]`). Les opérateurs du langage sont la *concaténation* (noté `@`), le *plus long prefixe commun* entre deux chaînes (noté `#`) et la *suppression du premier caractère* d'une chaîne non vide (noté `!`).

Ainsi :
- `[bon] @ [jour]` est la chaîne `[bonjour]`
- `[bonjour] # [bonsoir]` est la chaîne `[bon]`
- `![bonjour]` est la chaîne `[onjour]`

Nontons que l'opérateur `![]` n'est pas défini sur la chaîne vide.

L'objectif de ce travail est d'écrire un programme sur le même modèle que ce qui a été fait en TP pour la "calculatrice". Ce programme prendra donc en entrée une expression lue au clavier et effectuera :
- une analyse lexicale (partielle) du langage : partie 1 ;
- une analyse syntaxique : partie 2 ;
- un traitement effectué à partir de l'analyse syntaxique : partie 3.

Partie 1 : analyse lexicale

Les lexèmes du langage sont les suivants :
- `CONCAT`, représente l'opérateur de concaténation : `@`
- `PREF`, représente l'opérateur de plus long préfixe commun : `#`
- `SUP`, représente l'opérateur de suppression du 1er caractère : `!`
- `PARO`, représente la parenthèse ouvrante : `(`
- `PARF`, représente la parenthèse fermante : `)`
- `CHAINE`, représente une suite **non vide** de lettres minuscules entre crochets : par exemple `[bonjour]`
- `VIDE`, représente la chaîne vide : `[]`

Les caractères séparateurs sont "espace" et "fin ligne".

Q1. Dessinez un automate reconnaissant les lexèmes du langage. Cet automate lit en entrée une séquence de caractères et il atteint un état final lorsqu'un lexème a été reconnu. Les transitions seront étiquetées uniquement par le caractère courant. On ne fera pas apparaitre le traitement des erreurs lexicales.

Partie 2 : analyse syntaxique

La grammaire du langage est la suivante :

$$
\begin{array}{rcl}
Exp & \rightarrow & Terme\ X \\
X & \rightarrow & \texttt{CONCAT}\ Terme\ X \\
X & \rightarrow & \varepsilon \\
Terme & \rightarrow & Facteur\ Y \\
Y & \rightarrow & \texttt{PREF}\ Facteur\ Y \\
Y & \rightarrow & \varepsilon \\
Facteur & \rightarrow & \texttt{SUP}\ Facteur \\
Facteur & \rightarrow & \texttt{CHAINE} \\
Facteur & \rightarrow & \texttt{VIDE} \\
Facteur & \rightarrow & \texttt{PARO}\ Exp\ \texttt{PARF}
\end{array}
$$

Q2. Dessinez l'*arbre de dérivation* obtenu pour l'expression E suivante :

```
! [xlorem] # [loremipsum] @ [dolor]
```

Donnez un exemple d'expression sans *erreur lexicale* mais comportant une *erreur syntaxique*.

Q3. Classez les opérateurs `@`, `#` et `!` par priorité décroissante (du plus prioritaire au moins prioritaire).

Q4. En utilisant les primitives de `analyse_lexicale.h`, fournies en Annexe A, écrivez le corps de la fonction `analyser` spécifiée ci-dessous. Vous pouvez écrire des fonctions auxilliaires (comme `Rec_Exp`, `Rec_X`, `Rec_Terme`, `Rec_Y` et `Rec_Facteur`). Vous pouvez également utiliser la fonction `Erreur()` pour indiquer qu'une erreur est détectée. Enfin, vous pourrez abréger `lexeme_courant()` par `LC()`.

```
void analyser() ;
// lit une sequence de lexemes et appelle la fonction Erreur()
// si cette sequence est syntaxiquement incorrecte
```

Partie 3 : calcul de la longueur de la chaîne résultat

Q5. Quelle est la longueur (en nombre de caractères) de la chaîne de caractère produite par l'expression E de la `Q2` ?

Q6. Complétez le code de la **Q4** pour que la fonction `analyser` fournisse en résultat la **longueur** de la chaîne de caractères correspondant à l'expression lue (lorsque cette expression ne contient pas d'erreurs) comme spécifié ci-après :

```
int analyser () ;
// lit une sequence de lexemes et renvoie la longueur de la chaine
// de caracteres representee par cette sequence (si elle ne contient
// pas d'erreur syntaxique)
```

Indication :

- il n'est pas nécessaire de construire un arbre pour répondre à cette question ;
- en C la fonction `strlen(s)` renvoie la longueur de la chaîne de caractères `s`.

Annexe A : le fichier `analyse_lexicale.h`

```
typedef enum {
   CONCAT,           // @
   PREF,             // #
   SUP,              // !
   PARO,             // (
   PARF,             // )
   VIDE,             // []
   CHAINE,           // suite non vide  de lettres miniscules entre crochets
   FIN_SEQ           // lexeme de fin de sequence
} Nature_Lexeme;


typedef struct {
   Nature_Lexeme nature;  // nature du lexeme
   char *chaine;          // chaine de caracteres correspondant au lexeme courant
} Lexeme;

void demarrer();
//   debute l'analyse lexicale
//   lexeme courant est le premier lexeme de la sequence

void avancer();
//   lit le lexeme suivant

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

int fin_de_sequence()
//   vaut vrai ssi FIN_SEQ est le lexeme courant

void arreter();
//  arrete l'analyse lexicale
```