Exercice 1 : Listes à la Lisp

On s'intéresse à la grammaire suivante G d'axiome L, de terminaux {`a`, `(`, `)`}, décrivant des listes à la *Lisp* (L signifie « liste » et S signifie « suite d'éléments ») :

$$L \rightarrow \texttt{(}\ S\ \texttt{)} \mid \texttt{a}$$
$$S \rightarrow L\ S \mid \epsilon$$

On donne pour cette grammaire la table d'analyse suivante :

	`(`	`a`	`)`	#
L	$L \rightarrow$ `(` S `)`	$L \rightarrow$ `a`	erreur	erreur
S	$S \rightarrow L\ S$	$S \rightarrow L\ S$	$S \rightarrow \epsilon$	erreur

Soit m le mot `(a())`.

Q 1.1 : Donner la suite des piles résultant de l'analyse de m par l'automate LL(1) associé à G. Construire en même temps l'arbre syntaxique et la dérivation gauche pour m. □

Q 1.2 : Donner un analyseur récursif descendant pour G. □

Exercice 2 : Itinéraire et analyse syntaxique descendante

On s'intéresse à une grammaire G_I des instructions (très simplifiées) délivrées par un logiciel de calcul d'itinéraire, du style «avancer_100m, au_panneau_Lille tourner_à_gauche, avancer_20m, tourner_à_droite». On utilise pour ce faire les symboles `GO` (avancer_Xm), `TL` (turn left), `TR` (turn right) et `PAN` (panneau).

On a $G_I = \{V_T, V_N, \mathit{route}, P\}$ avec $V_T = \{$ `GO`, `TL`, `TR`, `PAN` $\}$, $V_N = \{$ *route*, *inst*, *suite*, *panneau*, *turn* $\}$ et P contient les productions :

$$\mathit{route} \rightarrow \mathit{inst} \mid \mathit{route}\ \mathit{inst}$$
$$\mathit{inst} \rightarrow \texttt{GO} \mid \mathit{panneau}\ \mathit{turn}$$
$$\mathit{turn} \rightarrow \texttt{TL} \mid \texttt{TR}$$
$$\mathit{panneau} \rightarrow \epsilon \mid \texttt{PAN}$$

Q 2.1 : Cette grammaire n'est pas LL(1) : pourquoi ? □

Q 2.2 : Donner une grammaire G'_I équivalente à G_I et qui vous semble LL(1). □

Q 2.3 : Calculer les ensembles *Premier* et *Suivant* pour G'_I. □

Q 2. 4 : Donner la table d'analyse LL(1) de G'_I. Justifier en utilisant cette table que G'_I est une grammaire LL(1). □

Q 2. 5 : En suivant les conventions du cours, donner les méthodes (signature et corps) de l'analyseur récursif descendant qui reconnaissent respectivement le non-terminal *suite* et le non-terminal *inst*. □

Q 2. 6 : G'_I est-elle une grammaire ambiguë ? Le langage $L(G'_I)$ est-il algébrique ? régulier ? Justifier vos réponses. □

Exercice 3 : Pour bien comprendre les calculs de *Premier* et *Suivant*

Soit la grammaire suivante (légèrement alambiquée) d'axiome S et de terminaux {`a`, `b`, `e`, `d`, `f`} :

$$S \rightarrow ABC \mid DAD$$
$$A \rightarrow \texttt{a}A \mid \epsilon$$
$$B \rightarrow \texttt{b}B \mid \epsilon$$
$$C \rightarrow \texttt{e}C \mid \epsilon$$
$$D \rightarrow \texttt{d}D \mid \texttt{f}$$

Q 3.1 : Construire la table d'analyse de cette grammaire et montrer qu'elle est LL(1). □

Exercice 4 : Grammaire non LL(1)

On considère la grammaire de terminaux {`a`, `+`, `*` }, d'axiome E et de productions :
$E \rightarrow E+C \mid C \quad C \rightarrow CS \mid S$
$S \rightarrow S* \mid X \quad X \rightarrow$ `a`

Q 4.1 : Quel est le langage engendré par cette grammaire ? □

Q 4.2 : Donner une grammaire LL(1) équivalente et justifier en donnant sa table d'analyse LL(1). □

Exercice 5 : Un langage de requêtes

On considère un langage de requêtes et la grammaire G telle que $V_T =$ { `id`, `(`, `)`, `select`, `join`, `cond` } et l'ensemble des productions est :
req → *req* `join` *req* | `select cond (` *req* `)` | `id`

Q 5.1 : Cette grammaire est-elle LL(1) ? □

Q 5.2 : L'opérateur `join` est associatif à droite. Donner une grammaire G_1 non ambiguë équivalente à G. Elle-t-elle LL(1) ? □

Q 5.3 : Donner une grammaire G_2 équivalente à G et LL(1). □

Q 5.4 : On ajoute l'opérateur `union` entre requêtes qui est associatif à gauche. Donner une grammaire LL(1) qui intègre cet opérateur au langage, sachant que le `join` est prioritaire sur l'`union`. □

Exercice 6 : Dangling else

On considère la grammaire suivante engendrant des fragments de programmes à base de *si alors sinon* :

S	⟶	`si` E `alors` S S'	|	`a`
S'	⟶	`sinon` S	|	ε
E	⟶	`b`		

La table d'analyse LL de cette grammaire contient une (et une seule) case contenant plus d'une règle (elle contient 2 règles). La grammaire n'est donc pas $LL(1)$. Habituellement, pour résoudre le problème des `si` sans `sinon`, on fait correspondre tout `sinon` au dernier `si` sans `sinon`.

Q 6.1 : Laquelle des deux règles en conflit dans la table d'analyse LL faut-il choisir pour appliquer ce principe ? □

Q 6.2 : En déduire un analyseur récursif descendant pour le langage associé. □

Exercice 7 : Grammaire LL(k) ?

Soit la grammaire d'axiome S, de terminaux $\{a, b, c, d\}$ et de productions :
$S \rightarrow A \mid B \quad A \rightarrow aAb \mid c \quad B \rightarrow aBbb \mid d$

Q 7.1 : Cette grammaire est-elle LL(1) ? LL(2) ? LL(k) pour k donné ? □