DEUG MIAS2. Institut Galilée
Option : Langages formels et compilation. 1999–2000
TP 4.

LEX est un générateur d'analyseur lexical, YACC un générateur d'analyseur syntaxique.

Un analyseur syntaxique a besoin des services d'un analyseur lexical : nous en prendrons un qui est engendré par LEX !

Vous devez consulter 'TP LEX and YACC User Manual' ainsi que le contenu de `LEXLIB.PAS` *et* `YACCLIB.PAS`.

Exercice 1. Expressions aritmétiques.

Ceci est un exercice d'observation du contenu des fichiers `EXPR.Y` et `EXPRLEX.L` et de l'exécution du programme `EXP.EXE` :

- `EXPR.Y` est la description, dans le format YACC, d'un analyseur syntaxique : le programme `EXPR.PAS` lui correspondant est obtenu par la commande
  `YACC EXPR`
- `EXPRLEX.L` est la description, dans le format LEX, d'un analyseur lexical nécessaire au précédent : le programme `EXPRLEX.PAS` lui correspondant est obtenu par la commande
  `LEX EXPRLEX`
- Enfin, on obtient le programme exécutable `EXPR.EXE` par la commande
  `TPC EXPR`

Remarques.

• C'est la directive de compilation `{$I EXPRLEX}` qui figure au début de la troisième section de `EXPR.Y` qui provoquera l'inclusion du contenu du fichier `EXPRLEX.PAS`, lors de la compilation de `EXPR.PAS`.

• Le fichier `EXPR.Y` permettra à YACC de construire une fonction `yyparse` : la troisième section de ce programme fait essentiellement un appel à cette fonction.

• Il revient à la fonction `yyparse` de faire appel à `yylex` mais un programme YACC n'y fait jamais appel explicitement.

• De son cté, la fonction `yylex` rend la main au moyen de l'une des procédures de retour suivantes :

- `return ( n )` : la valeur retournée `n` est celle d'un `token` : en fait, un token est une constante entière, qui désigne une unité lexicale,
  par exemple `return ( NUM )`.
- `returnc ( c )` : la valeur retournée `c` est (le code ASCII d') un caractère, constituant une unité lexicale à lui seul,
  par exemple `returnc ( yytext[1] )`.

Pour comprendre le fonctionnement des analyseurs, vous testerez `EXPR.EXE` sur plusieurs exemples de votre choix (attention, toute tentative de division par `0` sera sanctionnée par une erreur à l'exécution ...).

Exercice 2.

On considère la grammaire inspirée de l'exercice 3 du TD 1 :

$$\begin{array}{ll} 1 : S \to S * S & 4 : T \to T\,1 \\ 2 : S \to S + S & 5 : T \to 1 \\ 3 : S \to T & \end{array}$$

Ecrivez des progammes `BATLEX.L` et `BAT.Y` capables d'engendrer un programme permettant de déterminer si une expression est valide pour cette grammaire et de calculer la valeur numérique correspondante. (pour pallier l'ambiguïté de cette grammaire vous devrez peut–être définir des degrés de précédence et des modes d'association pour les opérations $*$ et $+$).

Vous introduirez un nouveau non terminal `input`, comme il est fait dans `EXPR.Y`, afin de pouvoir effectuer plusieurs essais de calcul sans avoir à relancer `BAT.EXE` plusieurs fois.

Exercice 3.

Reprendre l'exercice précédent pour la grammaire suivante :

$$\begin{array}{l} 1 : S \to S * S \\ 2 : S \to S + S \\ 3 : S \to \mathbf{id} \end{array}$$

où **id** est un terminal représentant l'unité lexicale des chaînes non vides d'occurrences du caractère 1.