Principes d'interprétation des langages
TP noté – 3 mai 2018
Frédéric Gruau et Kaourintin Le Guiban

Les réponses doivent être envoyées à vos chargés de TP habituels par mail **au plus tard à la fin de la séance**. Le mail doit contenir une archive contenant vos différents programme lex et yacc et un fichier contenant des copier-coller de grammaires et sorties de programmes (comme indiqué dans les questions ci-dessous). Format des noms de fichiers : les fichiers correspondant à l'exercice 2 de l'élève Dupont de groupe X, devront s'appeler exo2Dupont.y, exo2Dupont.l et exo2Dupont.txt, et similairement pour les autres exercices. L'objet de mail dans ce cas est DUPONT G X.

On souhaite réaliser un langage qui permet de manipuler des chaînes de caractères, en démarrant le plus simplement possible et en rajoutant progressivement des fonctionnalités.

Rappel :
Pour compiler, utilisez les commandes suivantes :
- Pour les fichiers lex :
  lex exercice1.l
  gcc lex.yy.c -ll
- Pour les fichiers yacc et lex :
  lex exercice2.l
  yacc -d exercice2.y
  gcc y.tab.c lex.yy.c -ll -ly

Exercice 1

Soit le programme helloWord suivant :

```
programme HelloWord
$A = "bonjour " ;
lire($B) ;
ecrire($A) ;
ecrire($B) ;
```

On utilise les classes lexicales suivantes (tokens), dont certaines comportent une valeur indiquée :

- les registres (un dollar plus une lettre majuscule),
- le signe =,
- les chaines de caractères constantes,
- le point virgules,
- les deux parenthèses,
- les identifiants de fonctions tels que lire ou ecrire (un identifant)

Dans l'archive, on vous donne un programme lex qui permet déjà de lire la première ligne de ce programme
$A = "bonjour " ;
Complétez ce programme lex pour générer les expressions régulières et les instructions pour les autres classes de tokens, de façon que le programme lex affiche pour chaque token sa classe, et sa valeur lorsqu'il en a une. Exécutez le programme lex sur le fichier d'exemple HelloWorld (on doit y trouver 19 tokens au total) et inclure la sortie du programme dans le rendu.

On utilisera les noms suivant en lettres majuscules pour nommer les classes de tokens : REG, EGAL, CHAINE, POINTVIRG, PAROUV, PARFERM, IDENT.

Exercice 2

La grammaire suivante permet de générer des programmes comme celui de l'exemple précédent :

$$
\begin{aligned}
instrs &\rightarrow instr \quad POINTVIRG \quad instrs \quad | \quad instr \\
instr &\rightarrow assign \quad | \quad call \\
call &\rightarrow IDENT \quad PAROUVRE \quad expr \quad PARFERME \\
assign &\rightarrow REG \quad EGALE \quad expr \\
expr &\rightarrow REG \quad | \quad CHAINE
\end{aligned}
$$

Le programme yacc n'aura pas d'actions, il se contente de vérifier si la syntaxe est correcte ou non. Notons que la fonction ecrire peut prendre en paramètre un nom de registre ou une chaîne, alors que la fonction lire ne peut prendre qu'un nom de registre. Cependant, ce sont là des vérifications de type qui ne sont pas assurées par la syntaxe. L'utilisation du symbole non terminal expr permettra d'enchaîner facilement sur les exercices suivants.

Dans l'archive, le fichier exercice2.y vous donne déjà la structure du programe yacc et la traduction de l'une des règles. Cela vous permet déjà d'analyser la première ligne du programme
$A = "bonjour " ;

Modifiez le programme lex exercice2.l de la même manière que pour l'exercice 1, en renvoyant cette fois la classe du token à chaque fois qu'un token est reconnu.

Ajoutez dans le programme yacc exercice2.y les règles manquantes pour réaliser l'analyse syntaxique de helloWord et de tout autre fichier qui respecte la même syntaxe.

Exercice 3

On considère le programme HelloWorld2 suivant ou les fonctions peuvent prendre plusieurs paramètres :
programme HelloWord2
lire($B,$C) ;
ecrire("bonjour ", $B," et ",$C) ;

Modifier la grammaire précédente pour qu'elle puisse générer de tels programmes. On utilisera un non-terminal exprs (avec un s), qui peut générer plusieurs expr séparés par des virgules.

Copiez-collez cette grammaire dans votre rendu, puis implémentez-là en Yacc. Recopiez dans votre compte-rendu la sortie correspondant au programme HelloWord2.

Exercice 4

On souhaite introduire deux nouveaux opérateurs : un opérateur de concaténation représenté par un point "." qui permet de concaténer des chaînes de caractères, et un opérateur noté "^" qui permet de répéter une chaîne de caractére. Par exemple $A . "bidule" sera égal a "bonjour bidule", "je l'aime,"^3 sera égal a "je l'aime, je l'aime, je l'aime,".

Quelle(s) autre(s) classe(s) lexicale(s) est/sont est nécessaires pour faire cela ? (Inclure la réponse dans le rendu).

On utilise la grammaire non ambiguë suivante pour générer ces expressions :

$$expr \rightarrow expr2 \quad | \quad expr \quad POINT \quad expr2$$
$$expr2 \rightarrow expr2 \quad PUISS \quad INT \quad | \quad CHAINE \quad | \quad REG \quad | \quad PAROUVRE \quad expr \quad PARFERME$$

L'opérateur de répétition a une précédence plus forte que la concaténation, on doit donc utiliser des parenthèses si on veut répéter une concaténation et pouvoir écrire par exemple : ("je l'aime".$A)^3."je meurs".

Modifiez le lex et le yacc afin que votre programme prenne en compte ces nouveaux opérateurs. Recopier dans votre rendu la sortie de l'analyse du programme suivant intitulé JeLaime.

Programme jeLaime

```
$A="René"
ecrire( (" je l'aime ".$A)^3." je meurs") ;
```