Initiation à YACC

1 Présentation

YACC permet de générer des analyseurs basés sur des grammaires (voir Table 1 et Table 2).

```
ligne  : ligne expr '\n'
       | ligne '\n'
       |
       ;
terme  : terme '*' facteur
       | terme '/' facteur
       | facteur
       ;
```

```
expr   : expr '+' terme
       | expr '-' terme
       | terme
       ;
facteur : '-' facteur
        | '(' expr ')'
        | NOMBRE
        ;
```

Table 1 – Grammaire ETF

```
ligne  : ligne expr '\n'
       | ligne '\n'
       |
       ;
expr   : expr '+' expr
       | expr '-' expr
       | expr '*' expr
       | expr '/' expr
       | '(' expr ')'
       | '-' expr
       | NOMBRE
       ;
```

Table 2 – Grammaire EEE

2 Format d'un fichier YACC

```
%{
Déclarations C
%}
Déclarations YACC
%%
Règles de Traduction
%%
Partie Code utilisateur
```

Table 3 – Structure d'un fichier YACC

2.1 Partie Définitions

Déclarations C :

— *include* + déclarations de fonctions, variables et constantes globales du programme C, partie encadrée par %{ et %}.

— changement du type des valeurs des non-terminaux : #define YYSTYPE montype (avec montype = double par exemple). Les non-terminaux ont des valeurs entières par défaut.

Déclarations YACC :

— symboles terminaux (%token pour les terminaux sauf s'ils sont confondus avec une unité lexicale de longueur 1, FLEX retournant un int)

— symboles non terminaux (s'il est nécessaire de préciser certains éléments qui leur sont relatifs) avec propriétés et attributs, ainsi que le type des attributs.

2.2 Partie Règles de Traduction

Une règle de traduction est de la forme :

$$\begin{array}{rl} \text{Non}-\text{Terminal}: & \text{partiedroite 1 } \{\text{action 1}\} \\ | & \text{partiedroite 2 } \{\text{action 2}\} \\ \ldots & \\ | & \text{partiedroite n } \{\text{action n}\} \\ ; & \end{array}$$

Les non-terminaux possèdent une valeur, stockée dans des variables particulières ; la valeur du non-terminal de la partie gauche est stockée dans la variable $$, et la liste $1, ... $k les valeurs des symboles de la partie droite dans leur ordre d'apparition. La règle par défaut est : $$ = $1. Par exemple, pour la règle (expr1 : expr2 '+' expr3), $$ contient la valeur de expr1, $1 celle de expr2 et $3 celle de expr3.

2.3 Partie Code utilisateur

Cette partie est optionnelle tout comme le %% qui la précède. Le main par défaut est :

void main(...){ yyparse()}.

yyparse est l'appel à l'analyseur défini par les règles de traduction. Chaque fois que nécessaire, il appelle yylex lui retournant une unité lexicale et place la valeur (ou attribut) associée à cette unité dans la variable yylval.

3 Compilation d'un fichier YACC

$$\begin{array}{ccc} & \text{flex monfichier.l} & \\ \text{monfichier.l} & \longrightarrow & \text{lex.yy.c} \\ & \text{yacc monfichier.y} & \\ \text{monfichier.y} & \longrightarrow & \text{y.tab.c} \\ & \text{gcc y.tab.c -ly -lfl} & \\ \text{y.tab.c} & \longrightarrow & \text{a.out} \end{array}$$

Dans la suite du TP, nous allons implanter une calculatrice. Pour cela :

— nous utiliserons un fichier Flex déterminant les différents lexèmes (fichier calc.lex)

— nous utiliserons un fichier YACC déterminant les combinaisons de ces lexèmes (fichier calc.y)

Les symboles non-terminaux seront définis par la commande %token dans calc.y. Ainsi, pour que tout token déclaré soit utilisable dans le fichier calc.lex, il faut inclure le fichier calc.tab.h qui contient sa déclaration. Ce fichier est généré par l'option -d lors de la compilation *via* yacc.

Pour plus de lisibilité, la définition de la directive YYSTYPE peut être reporté dans le fichier yystype.h (à inclure dans calc.y).

Enfin, les différentes fonctions définies dans calc.lex peuvent être utilisées dans calc.y si leur déclaration est connue ; pour cela, on peut générer le fichier lex.yy.h contenant leurs déclarations par l'option –header-file=lex.yy.h de flex.

Exercice 1 (Implantation ETF)

(Question 1) Trouver une expression rationnelle décrivant les nombres flottants positifs et l'intégrer au fichier FLEX. (5, 5.5, .5 et $5e-10$ sont des réels).

(Question 2) Implanter dans un fichier YACC la grammaire ETF. Ajouter les actions manquantes en s'inspirant de l'action de la somme.

Exercice 2 (Implantation EEE)

(Question 1) Implanter dans un fichier YACC la grammaire EEE (Table 2). Que se passe-t-il ?

La grammaire EEE est ambiguë. Des conflits *shift/reduce* sont présents. Par exemple, le motif "$5+3+4$" peut se voir décomposé de deux façons différentes ($(5+3)+4$ ou $5+(3+4)$). La définition de l'associativité d'un opérateur permet de fixer une unique décomposition. Cependant, cette définition ne suffit pas pour implanter la calculatrice ; on doit également définir un ordre de priorité (par exemple $5+3*4 \neq (5+3)*4$).
Pour régler ces problèmes, on déclare dans la partie *Déclarations Yacc* l'associativité des opérateurs avec les outils %left, %right et %nonassoc. Par exemple, en définissant %left $'+'$, on appliquera le $+$ le plus à gauche (le "+" est déclaré comme associatif à gauche). De plus, les opérateurs doivent apparaître selon l'ordre croissant de priorité, et deux opérateurs équivalents sont déclarés en même temps. Ici, $+$ et $-$ apparaissent avant $*$ et /.
Ainsi, il conviendra de rajouter les lignes suivantes pour la grammaire EEE :

```
%left '-' '+'
%left '*' '/'
%right MOINSU
```

Cette dernière ligne, celle du motif MOINSU permet de surcharger la priorité du $-$ unaire, prioritaire sur tout autre opérateur (ce qui n'est pas le cas pour un - binaire). Cette règle permet de surcharger l'associativité du symbole $-$. Pour utiliser cette priorité, il faut remplacer la ligne de la grammaire définissant le - unaire par :

```
        | '-' expr %prec MOINSU
```

Ainsi la priorité du $-$ unaire sera celle de MOINSU, prioritaire sur les autres opérateurs.

(Question 2) Après avoir implanté les opérations de la bibliothèque *math.h*, gérer les priorités et les associativités des opérateurs en utilisant les outils %left, %right et %prec.