Grammaire en lex et yacc, 2017

Dominique Michelucci

Le programme présenté ici lit la description d'une grammaire. Les fichiers G1, G2, Gexp, Gexpression décrivent des grammaires. Le programme grammaire (grammaire.lex, grammaire.yacc) lit ces fichiers et construit une structure de donnée représentant la grammaire lue.

Remarques : il n'y a qu'une seule grammaire par fichier ; la partie gauche de chaque règle ne contient qu'un seul symbole (non terminal) ; chaque règle n'a qu'une seule partie droite. Un seul type de liste est gérée : les listes de symboles. La règle décrite par la chaîne de caractères "rule : t -> A b t ;" dans le fichier d'entrée est représentée par la liste de symboles : ["t", "->", "A", "b", "t"]. Ce programme est donc plus simple que votre projet de 2016–2017.

Makefile

```
ok :
        lex -o grammaire_lexee grammaire.lex
        yacc -o grammaire.c grammaire.yacc
        cc -o grammaire grammaire.c -lfl -lc
demo:
        ./grammaire < Gexpression
grammaire.pdf: grammaire.tex
        pdflatex grammaire.tex
grammaire2.pdf: grammaire2.tex
        pdflatex grammaire2.tex
```

Fichier de la grammaire G1

```
{ ELT }
{ list }
list
{
rule: list -> ;
rule: list -> ELT list ;
}
```

Fichier de la grammaire G2

```
{ MULT, NB }
{ factor }
factor
{
rule: factor -> NB ;
rule: factor -> NB MULT factor ;
}
```

Fichier de la grammaire Gexp

```
{ PLUS, MULT, NB }
{ sum, factor }
sum
{
rule: factor -> NB ;
rule: factor -> NB MULT factor ;
rule: sum -> factor ;
rule: sum -> factor PLUS sum ;
}
```

Fichier de la grammaire Gexpression

```
{ PLUS, MULT, NB, VARIABLE }
{ atom, sum, factor }
sum
{
rule: atom -> NB ;
rule: atom -> VARIABLE ;
rule: atom -> PO sum PF ;
rule: factor -> atom ;
rule: factor -> atom MULT factor ;
rule: sum -> factor ;
rule: sum -> factor PLUS sum ;
}
```

Demonstration

```
$ ./grammaire <G1
Le premier ensemble de symboles est: ELT,
Le deuxieme ensemble de symboles est: list,
L'axiome de la grammaire est: list
Il y a 2 regles:
rule: list ->
rule: list -> ELT list

$ ./grammaire <G2
Le premier ensemble de symboles est: MULT, NB,
Le deuxieme ensemble de symboles est: factor,
L'axiome de la grammaire est: factor
Il y a 2 regles:
rule: factor -> NB
rule: factor -> NB MULT factor

$ ./grammaire < Gexp
Le premier ensemble de symboles est: PLUS, MULT, NB,
Le deuxieme ensemble de symboles est: sum, factor,
L'axiome de la grammaire est: sum
Il y a 4 regles:
rule: factor -> NB
rule: factor -> NB MULT factor
rule: sum -> factor
rule: sum -> factor PLUS sum
```

```
$ ./grammaire < Gexpression
Le premier ensemble de symboles est: PLUS, MULT, NB, VARIABLE,
Le deuxieme ensemble de symboles est: atom, sum, factor,
L'axiome de la grammaire est: sum
Il y a 7 regles:
rule: atom -> NB
rule: atom -> VARIABLE
rule: atom -> PO sum PF
rule: factor -> atom
rule: factor -> atom MULT factor
rule: sum -> factor
rule: sum -> factor PLUS sum
```

Fichier grammaire.lex

```
%{
%}
minuscule [a-z]
majuscule [A-Z]
%%
" "|\t|\n   { /* skip */ ; }
","      { return VG;}
"rule:" { return KEYWRD_RULE; }
{majuscule}+  { yylval.chaine= strdup( yytext); return ST; }
{minuscule}+  { yylval.chaine= strdup( yytext); return SNT; }
"{"     { if(debug) printf( "lex a lu un AO\n"); return AO; }
"}"     { if(debug) printf( "lex a lu un AF\n"); return AF; }
"->"    { return FLECHE; }
";"     { if(debug) printf( "lex a lu un PV\n"); return PV ; }
.       { printf("caractere inconnu: [%s]\n", yytext); exit(0); return (-1) ; }
```

Fichier grammaire.yacc

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
typedef struct LIST { char * symbol; struct LIST * queue ; } *list ;
list cons( char *, list);
list tab_rules [1000]; /* contient un tableau de regles (1 regle=1 list) */
int nb_rules=0; /* nombre de regles de la grammaire */
int debug=0;
%}
%union{ char * chaine;  list liste ; }
%type <liste> ensemble
%type <liste> liste_symbols
%type <liste> liste_symbols_non_vide
%type <liste> suite_symbols
%type <liste> regle
%type <chaine> symbol
%token <chaine> NG
%token <chaine> SNT
%token <chaine> ST
```

```
%token PO PF AO AF PIPE FLECHE EGAL KEYWRD_RULE PV VG VIDE
%start grammaire
%%
grammaire: ensemble ensemble SNT AO regles AF { list l;
        printf("Le premier ensemble de symboles est: ");
        for( l=$1; l != 0; l=l->queue) printf( "%s, ", l-> symbol);
        printf("\n"); printf("Le deuxieme ensemble de symboles est: ");
        for( l=$2; l != 0; l=l->queue) printf( "%s, ", l-> symbol);
        printf("\nL'axiome de la grammaire est: %s\n", $3);
        printf("Il y a %d regles:\n", nb_rules);
        {int i; for(  i=0; i<nb_rules; i++)
        { for(l=tab_rules[i]; l!=0; l=l->queue) printf("%s ", l->symbol);
          printf("\n"); } } return 0; } ;
regles: regle {};
regles: regle regles {};
symbol: ST { $$=$1; };
symbol: SNT { $$=$1; };
ensemble : AO liste_symbols AF { $$=$2; } ;
liste_symbols :  { $$=  0; };
liste_symbols : liste_symbols_non_vide { $$=$1; } ;
liste_symbols_non_vide : symbol VG liste_symbols_non_vide { $$=cons($1,$3);}
liste_symbols_non_vide : symbol  { $$=  cons( $1, 0); };
regle: KEYWRD_RULE SNT FLECHE suite_symbols PV { list l;
        $$=cons( "rule:", cons( $2, cons( "->", $4)));
        tab_rules[ nb_rules] = $$; nb_rules++ ; };
suite_symbols: { $$=  0; };
suite_symbols: symbol suite_symbols { $$=cons($1, $2); };
%%
#include "grammaire_lexee"
int yyerror( char *s) { printf( "%s\n", s); return 0; }
list cons( char * hd, list q) { list l= malloc( sizeof( struct LIST));
    l->symbol= strdup( hd); l->queue= q; return l; }
int main( int argc, char * argv[]) { yyparse(); }
```