```
%option noyywrap
%{
%}

%%

[0-9]+ {yylval = atoi(yytext); return NUMBER;}
[a-z] {yylval = yytext; return ID;}
;  {return POINTV;}
\+ {return PLUS;}
\- {return MOINS;}
\* {return MULT;}
\/ {return DIV;}
\= {return EGAL;}
\n {return 0;}
\( {return PARO;}
\) {return PARF;}

%%
```



```
/* Analyseur syntaxique des expressions arithme'tiques
imprime les nume'ros de regles
a utiliser avec etf.lex qui reconnait les differents terminaux
NUMBER:  nombre entier (ne sert pas ici),
PARO PARF:  parentheses, MULT PLUS multiplication
addition , POINTV point virgule, ID identificateur */

%{
#define YYSTYPE int
%}

%token NUMBER PLUS POINTV MULT PARO PARF ID EGAL

%%


statement   : expression POINTV {printf("regle 0 \n");}
              ;
expression  : expression PLUS terme {printf("regle 1 \n");}
              | terme {printf("regle 2 \n");}
              ;
terme        : terme MULT facteur {printf("regle 3 \n");}
              | facteur {printf("regle 4 \n");}
              ;
facteur     : PARO expression PARF {printf("regle 5 \n");}
              | ID {printf("regle 6 \n");}
              ;

%%

#include "lex.yy.c"

main()
{
    yyparse();
}

yyerror(char *s) {
    printf ("%s\n",s);
}
```

```
/* Calcul d'expressions arithme'tiques on de'termine la valeur
a utiliser avec etf.lex qui reconnait les differents terminaux
NUMBER:  nombre entier,  PARO PARF:  parentheses, MULT PLUS multiplication
addition , POINTV point virgule, ID identificateur (ne sert pas ici) */

%{
#define YYSTYPE int
#include <math.h>
%}

%token NUMBER PLUS POINTV MULT PARO PARF ID EGAL

%%

statement   : expression POINTV {printf("val = %d\n", $1);}
             ;
expression  : expression PLUS terme {$$ = $1 + $3;}
             | terme {$$ = $1;}
             ;
terme       : terme MULT facteur {$$ = $1 * $3;}
             | facteur {$$ = $1;}
             ;
facteur     : PARO expression PARF {$$ = $2;}
             | NUMBER {$$ = $1;}
             ;

%%

#include "lex.yy.c"

main()
{
    yyparse();
}

yyerror(char *s) {
    printf ("%s\n",s);
}
```



```
/* Compilation  d'expressions arithme'tiques on genere du code
a utiliser avec etf.lex qui reconnait les differents terminaux
NUMBER:  nombre entier,  PARO PARF:  parentheses, MULT PLUS multiplication
addition , POINTV point virgule, ID identificateur (ne sert pas ici) */
%{
#define YYSTYPE char*

#define LMAX 1000

trans (char *u){
  /* ajoute 1 a tous les numeros de registres */
  int i;
  for (i = 0; u[i] != '\0'; i++)
    if(isdigit(u[i])) u[i]++;
}

%}

%token NUMBER PLUS POINTV MULT PARO PARF ID EGAL

%%

statement : expression POINTV
          {printf("%s\n", $1);}
          ;

expression : expression PLUS terme {
                      trans($3);
                      strcat(strcat ($1,$3), "add R1 R2\n") ;
                      $$ = $1;}
            | terme {$$ = $1;}
            ;

terme:      terme MULT  facteur {
                       trans($3);
                       strcat(strcat($1,$3), "mult R1 R2\n");
                       $$ = $1;}
            | facteur {$$ = $1;}
            ;

facteur : ID  {$$ = (char *) malloc(LMAX); strcpy($$,"load " );
               strcat($$, $1); strcat($$ ," R1\n");}

         | PARO  expression PARF  {$$ = $2;}
          ;

%%

#include "lex.yy.c"

main()
{
  yyparse();
}

yyerror(char *s)

{
  printf ("%s\n",s);
}
```

```
/* Analyseur syntaxique des expressions arithme'tiques
imprime les nume'ros de regles
a utiliser avec etf.lex qui reconnait les differents terminaux
NUMBER:  nombre entier (ne sert pas ici),
PARO PARF:  parentheses, MULT PLUS multiplication
addition , POINTV point virgule, ID identificateur */

%{
#define YYSTYPE int
int nbRegles = 0;
%}

%token NUMBER PLUS POINTV MULT PARO PARF ID EGAL

%%


statement  : expression POINTV {
                printf("nombre de règles utilisées = %d\n", nbRegles);}
           ;
expression : expression PLUS terme {nbRegles++;}
           | terme {nbRegles++;}
           ;
terme      : terme MULT facteur {nbRegles++;}
           | facteur {nbRegles++;}
           ;
facteur    : PARO expression PARF {nbRegles++;}
           | NUMBER {nbRegles++;}
           ;

%%

#include "lex.yy.c"

main()
{
    yyparse();
}

yyerror(char *s) {
    printf ("%s\n",s);
}
```



```
%option noyywrap
%{
%}

%%

[0-9]+ {yylval = atof(yytext); return NUMBER;}
; {return POINTV;}
\+ {return PLUS;}
\- {return MOINS;}
\* {return MULT;}
\/ {return DIV;}
\= {return EGAL;}
\n {return 0;}
\( {return PARO;}
\) {return PARF;}

%%
```

```
/* Calcul d'expressions arithme'tiques on de'termine la valeur
a utiliser avec etf.lex qui reconnait les differents terminaux
NUMBER:  nombre entier,  PARO PARF:  parentheses, MULT PLUS multiplication
addition , POINTV point virgule, ID identificateur (ne sert pas ici) */

%{
#define YYSTYPE double
#include <math.h>
%}

%token NUMBER PLUS POINTV MULT PARO PARF ID EGAL MOINS DIV

%%


statement   : expression POINTV {printf("val = %f\n", $1);}
             ;
expression  : expression PLUS terme {$$ = $1 + $3;}
             | expression MOINS terme {$$ = $1 - $3;}
             | terme {$$ = $1;}
             ;
terme       : terme MULT facteur {$$ = $1 * $3;}
             | terme DIV facteur {$$ = $1 / $3;}
             | facteur {$$ = $1;}
             ;
facteur     : PARO expression PARF {$$ = $2;}
             | NUMBER {$$ = $1;}
             ;

%%

#include "lex.yy.c"

main()
{
    yyparse();
}

yyerror(char *s) {
    printf ("%s\n",s);
}
```



```
/* Compilation  d'expressions arithme'tiques on genere du code
a utiliser avec etf.lex qui reconnait les differents terminaux
NUMBER:  nombre entier,  PARO PARF:  parentheses, MULT PLUS multiplication
addition , POINTV point virgule, ID identificateur (ne sert pas ici) */
%{
#define YYSTYPE char*

int numregistre = 0;
%}

%token NUMBER PLUS POINTV MULT PARO PARF ID EGAL MOINS DIV

%%

statement   : expression POINTV
              ;

expression  : expression PLUS terme {
                 printf("add R%d R%d\n", numregistre-1, numregistre);
                 numregistre--;}
              | terme {$$ = $1;}
              ;

terme        : terme MULT facteur {
                 printf("mult R%d R%d\n", numregistre-1, numregistre);
                 numregistre--;}
              | facteur {$$ = $1;}
              ;

facteur     : ID {numregistre++; printf("load %s R%d\n", $1, numregistre);}
              | PARO  expression PARF  {$$ = $2;}
              ;

%%

#include "lex.yy.c"

main()
{
  yyparse();
}

yyerror(char *s)

{
  printf ("%s\n",s);
}
```

```
%option noyywrap
%{
%}

%%

; {return POINTV;}
\n {return 0;}
a {return PARA;}
b {return PARB;}

%%
```



```
/* Analyseur syntaxique des expressions parenthèsées
imprime la hauteur des parenthèses
a utiliser avec exo2-1.lex qui reconnait les differents terminaux
PARA PARB:  parentheses de type (),
POINTV point virgule */

%{
#define YYSTYPE int
int hauteur = 0;
%}

%token POINTV PARA PARB

%%

statement   : expression POINTV {printf("\n");}
             ;
expression  : ouvrir expression fermer expression
             | ouvrir expression fermer
             | ouvrir fermer expression
             | ouvrir fermer
             ;

ouvrir      : PARA {hauteur++; printf("%d", hauteur);}
             ;
fermer      : PARB {hauteur--; printf("%d", hauteur);}
             ;

%%

#include "lex.yy.c"

main()
{
    yyparse();
}

yyerror(char *s) {
    printf ("%s\n",s);
}
```

13 déc 00 11:20 **exo2−2.lex** Page 1/1

```
%option noyywrap
%{
%}

%%

; {return POINTV;}
\n {return 0;}
a {return PARA;}
b {return PARB;}
c {return PARC;}
d {return PARD;}

%%
```

13 déc 00 12:23 **exo2−2.y** Page 1/1

```
/* Analyseur syntaxique des expressions parenthèsées imbriquées
imprime les hauteurs des parenthèses
a utiliser avec exo2-2.lex qui reconnait les differents terminaux
PARA PARB:  parentheses de type (),
PARC PARD:  parentheses de type [],
POINTV point virgule */

%{
#define YYSTYPE int
int hauteurAB = 0;
int hauteurCD = 0;
%}

%token POINTV PARA PARB PARC PARD

%%

statement   : expression POINTV {printf("\n");}
             ;
expression  : ouvrirAB expression fermerAB expression
             | ouvrirCD expression fermerCD expression
             | ouvrirAB expression fermerAB
             | ouvrirCD expression fermerCD
             | ouvrirAB fermerAB expression
             | ouvrirCD fermerCD expression
             | ouvrirAB fermerAB
             | ouvrirCD fermerCD
             ;

ouvrirAB    : PARA {hauteurAB++; printf("(%d,%d)", hauteurAB, hauteurCD);}
             ;

fermerAB    : PARB {hauteurAB--; printf("(%d,%d)", hauteurAB, hauteurCD);}
             ;

ouvrirCD    : PARC {hauteurCD++; printf("(%d,%d)", hauteurAB, hauteurCD);}
             ;

fermerCD    : PARD {hauteurCD--; printf("(%d,%d)", hauteurAB, hauteurCD);}
             ;

%%

#include "lex.yy.c"

main()
{
    yyparse();
}

yyerror(char *s) {
    printf ("%s\n",s);
}
```

```
  0  $accept : statement $end

  1  statement : expression POINTV

  2  expression : expression PLUS terme
  3             | terme

  4  terme : terme MULT facteur
  5        | facteur

  6  facteur : PARO expression PARF
  7          | ID
```



```
state 0
        $accept : . statement $end  (0)

        PARO  shift 1
        ID  shift 2
        .  error

        statement  goto 3
        expression  goto 4
        terme  goto 5
        facteur  goto 6


state 1
        facteur : PARO . expression PARF  (6)

        PARO  shift 1
        ID  shift 2
        .  error

        expression  goto 7
        terme  goto 5
        facteur  goto 6


state 2
        facteur : ID .  (7)

        .  reduce 7


state 3
        $accept : statement . $end  (0)

        $end  accept


state 4
        statement : expression . POINTV  (1)
        expression : expression . PLUS terme  (2)

        PLUS  shift 8
        POINTV  shift 9
        .  error


state 5
        expression : terme .  (3)
        terme : terme . MULT facteur  (4)

        MULT  shift 10
        PLUS  reduce 3
        POINTV  reduce 3
        PARF  reduce 3


state 6
        terme : facteur .  (5)

        .  reduce 5
```

```
state 7
        expression : expression . PLUS terme  (2)
        facteur : PARO expression . PARF  (6)

        PLUS  shift 8
        PARF  shift 11
        .  error


state 8
        expression : expression PLUS . terme  (2)

        PARO  shift 1
        ID  shift 2
        .  error

        terme  goto 12
        facteur  goto 6


state 9
        statement : expression POINTV .  (1)

        .  reduce 1


state 10
        terme : terme MULT . facteur  (4)

        PARO  shift 1
        ID  shift 2
        .  error

        facteur  goto 13


state 11
        facteur : PARO expression PARF .  (6)

        .  reduce 6


state 12
        expression : expression PLUS terme .  (2)
        terme : terme . MULT facteur  (4)

        MULT  shift 10
        PLUS  reduce 2
        POINTV  reduce 2
        PARF  reduce 2


state 13
        terme : terme MULT facteur .  (4)

        .  reduce 4


10 terminals, 5 nonterminals
8 grammar rules, 14 states
```


Table of Contents