

DEVOIR COMPILATION

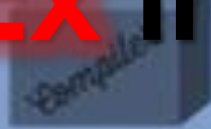
YACC in Bison



&



RegEx in Flex



Réalisation des Analyseurs LEXICALS et SYNTAXIQUE

RÉALISÉ PAR :

Youssef MAHTAT

RESOLUTION DE L'EXERCICE 1 :

→ Fichier analyseur.l :

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "PrintSymbolTableLib.cpp" // Bibliothèque d'affichage réaliser pour l'analyseur
// addOutput(char* lexeme, char* classUnit, int index) : ajout un lexème analysé à l'affichage final (résultat de l'analyse)
// addErrorOutput(char* lexeme, int index) : ajout une erreur trouvée à l'affichage final (résultat de l'analyse)
// printOutput() : affiche le résultat de l'analyse
int index = 1; // indice du 1er mot, puis sera incrémenté à chaque lexème trouvé !
}%

delim [\t ]
bl {delim}+
lettre [a-zA-Z]
chiffre [0-9]
id {lettre}({lettre}|{chiffre})+
signe (\+|\-)?
nbr {chiffre}+(\.{chiffre}+)?((e|E){signe}{chiffre}+)?
%%

\n { /* NE RIEN FAIRE */ }
{bl} { /* IGNORE */ }
"ENTIER" {addOutput(yytext, "MOT-CLE", index); index++;}
"REEL" {addOutput(yytext, "MOT-CLE", index); index++;}
{id} {addOutput(yytext, "IDENTIFICATEUR", index); index++;}
{nbr} {addOutput(yytext, "NOMBRE", index); index++;}
"," {addOutput(yytext, "VIRGULE", index); index++;}
";" {addOutput(yytext, "POINT-VIRGULE", index); index++;}
"=" {addOutput(yytext, "AFFECTATION", index); index++;}
"+" {addOutput(yytext, "ADDITION", index); index++;}
"-" {addOutput(yytext, "SOUSTRACTION", index); index++;}
"*" {addOutput(yytext, "MULTIPLICATION", index); index++;}
"/" {addOutput(yytext, "DIVISION", index); index++;}
"(" {addOutput(yytext, "PARENTHESE OUVRANTE", index); index++;}
")" {addOutput(yytext, "PARENTHESE FERMANTE", index); index++;}
"$" {addOutput(yytext, "FIN", index); index++; printOutput(); exit(0);}
. {addErrorOutput(yytext, index); index++;}
%%

int main()
{
    printf("\nExpressions à analyser (terminees par $) : \n");
    yylex();
    return 0;
}

int yywrap()
{
    // le delimitateur de la lecture
    return 1;
}
```

→ Fichier PrintSymbolTableLib.cpp:

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
using namespace std;
string output = ""; // variable qui contiendra le résultat de l'analyse
char temp[100]; // variable tampon, pour convertir un int to string

// Ajout une ligne à la variable output, qui contient le (lexeme, unité lexical, indice) :
void addOutput(char* lexeme, char* classUnit, int index)
{
    // Preparation de la ligne, et l'ajout partie par partie !
    output += "LEXEME : " ;
    output += lexeme;
    output += " | lexicalUnit : ";
    output += classUnit;
    output += " | index ; ";
    sprintf(temp, "%d", index);
    output += temp;
    output += " ;\n";
}

// Ajout une ligne d'erreur à la variable output, pour un mot qui n'est pas dans le lexique
void addErrorOutput(char* lexeme, int index)
{
    // Preparation de la ligne d'erreur, et l'ajout partie par partie !
    output += "motERREUR : " ;
    output += lexeme;
    output += " | NON INCLU DANS LE LEXIQUE";
    output += " | index ; ";
    sprintf(temp, "%d", index);
    output += temp;
    output += " ;\n";
}

// Affiche la variable output, le résultat de l'analyse
void printOutput()
{
    cout << output;
}
```

Rmq – Pour Compiler et Exécuter l'analyseur :

```
cmd>> flex analyseur.l
cmd>> g++ lex.yy.c -o analyseur
cmd>> analyseur.exe
```

→ Exemple d'exécution :

```
C:\Windows\System32\cmd.exe
D:\ZZ - WorkSpace\ComputingWS\EMSI\4IIR2\COMPILATION WorkSpaces\W - Devoir\Exercice1>analueur.exe

Expressions a analyser (terminees par $) :
ENTIER AB1, AB2, AB3, AB4;
REEL reel1, reel2;
AB1=22; AB2=33;
AB3=AB1+AB2;
reel1=100.34E-2;
$
LEXEME : ENTIER | lexicalUnit : MOT-CLE | index : 1 ;
LEXEME : AB1 | lexicalUnit : IDENTIFICATEUR | index : 2 ;
LEXEME : , | lexicalUnit : VIRGULE | index : 3 ;
LEXEME : AB2 | lexicalUnit : IDENTIFICATEUR | index : 4 ;
LEXEME : , | lexicalUnit : VIRGULE | index : 5 ;
LEXEME : AB3 | lexicalUnit : IDENTIFICATEUR | index : 6 ;
LEXEME : , | lexicalUnit : VIRGULE | index : 7 ;
LEXEME : AB4 | lexicalUnit : IDENTIFICATEUR | index : 8 ;
LEXEME : ; | lexicalUnit : POINT-VIRGULE | index : 9 ;
LEXEME : REEL | lexicalUnit : MOT-CLE | index : 10 ;
LEXEME : reel1 | lexicalUnit : IDENTIFICATEUR | index : 11 ;
LEXEME : , | lexicalUnit : VIRGULE | index : 12 ;
LEXEME : reel2 | lexicalUnit : IDENTIFICATEUR | index : 13 ;
LEXEME : ; | lexicalUnit : POINT-VIRGULE | index : 14 ;
LEXEME : AB1 | lexicalUnit : IDENTIFICATEUR | index : 15 ;
LEXEME : = | lexicalUnit : AFFECTATION | index : 16 ;
LEXEME : 22 | lexicalUnit : NOMBRE | index : 17 ;
LEXEME : ; | lexicalUnit : POINT-VIRGULE | index : 18 ;
LEXEME : AB2 | lexicalUnit : IDENTIFICATEUR | index : 19 ;
LEXEME : = | lexicalUnit : AFFECTATION | index : 20 ;
LEXEME : 33 | lexicalUnit : NOMBRE | index : 21 ;
LEXEME : ; | lexicalUnit : POINT-VIRGULE | index : 22 ;
LEXEME : AB3 | lexicalUnit : IDENTIFICATEUR | index : 23 ;
LEXEME : = | lexicalUnit : AFFECTATION | index : 24 ;
LEXEME : AB1 | lexicalUnit : IDENTIFICATEUR | index : 25 ;
LEXEME : + | lexicalUnit : ADDITION | index : 26 ;
LEXEME : AB2 | lexicalUnit : IDENTIFICATEUR | index : 27 ;
LEXEME : ; | lexicalUnit : POINT-VIRGULE | index : 28 ;
LEXEME : reel1 | lexicalUnit : IDENTIFICATEUR | index : 29 ;
LEXEME : = | lexicalUnit : AFFECTATION | index : 30 ;
LEXEME : 100.34E-2 | lexicalUnit : NOMBRE | index : 31 ;
LEXEME : ; | lexicalUnit : POINT-VIRGULE | index : 32 ;
LEXEME : $ | lexicalUnit : FIN | index : 33 ;
```

RESOLUTION DE L'EXERCICE 2 :

⇒ CONCEPTION & ETUDE INITIALE :

→ Grammaire Régulière :

La syntaxe de la calculatrice scientifique suit la grammaire suivante :

```
R → E $  
E → E + T | E - T | T  
T → T * F | T / F | F  
F → nbr | ( E ) | -F | +F | F ^ F | cosinus F | sinus F | tan F | exp F  
    | log F | valeur_absolu F | racine_carre F
```

SENS DE LA PRIORITE

→ Rmq sur l'ANALYSE LEXICAL :

Un nombre réel est validé par l'expression :

$(\backslash+|\backslash-)? [0-9]+(\backslash.[0-9]+)?((e|E)(\backslash+|\backslash-)?[0-9]+)?$

⇒ IMPLANTATION DE LA CALCULATRICE EN FLEX ET BISON (ET C/C++) :

→ Fichier global.h :

```
#define YYSTYPE double  
extern YYSTYPE yylval;
```

→ Fichier calculatrice.y :

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"
int yyerror(char* s);
int yylex();
}%

%token NOMBRE PO PF FIN
%left PLUS MOINS
%left FOIS DIV
%nonassoc COSINUS SINUS RACINE ABS LOGARITHME TANGENTE EXPONENTIELLE
%right PUISSANCE
%start R
%%

R : E FIN {printf("Resultat = %f", $1); return 0;};
E : E PLUS T  {$$=$1+$3;}
  | E MOINS T {$$=$1-$3;}
  | T        {$$=$1;}
  ;
T : T FOIS F  {$$=$1*$3;}
  | T DIV F  {if ($3 == 0) {printf ("division par zéro interdite"); exit(1);} else{ $$ = $1 /
$3;}}
  | F
  ;
F : NOMBRE
  | PO E PF {$$=$2;}
  | MOINS F  %prec NEG {$$= -$2;}
  | PLUS F   %prec POS {$$= +$2;}
  | F PUISSANCE F {$$ = pow($1,$3);}
  | COSINUS F {$$ = cos($2);}
  | SINUS F {$$ = sin($2);}
  | TANGENTE F {$$ = tan($2);}
  | LOGARITHME F {$$ = log($2);}
  | EXPONENTIELLE F {$$ = exp($2);}
  | ABS F {$$ = fabs($2);}
  | RACINE F {$$ = sqrt($2);}
  ;
%%

int yyerror(char *s)
{
    printf("%s", s); return 1;
}

int main()
{
    printf("Donnez une expression : ");
    yyparse();
    return 0;
}
```

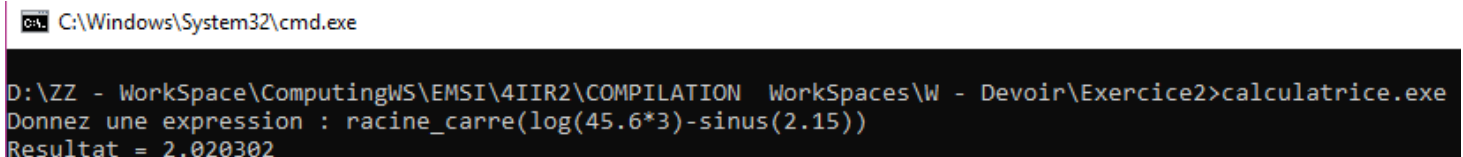
→ Fichier calculatrice.l :

```
%{
#include <math.h>
#include <stdlib.h>
#include "global.h"
#include "calculatrice.tab.h"
}%
nombre [0-9]+(\.[0-9]+)?((e|E)(\+|\-)?[0-9]+)?
%%
{nombre} {yylval=atof(yytext); return NOMBRE;}
"\n"      {return FIN;}
"+"       {return PLUS;}
"_"       {return MOINS;}
"*"       {return FOIS;}
"/"       {return DIV;}
"("       {return PO;}
")"       {return PF;}
"^"       {return PUISSANCE;}
"cosinus" {return COSINUS;}
"sinus"   {return SINUS;}
"log"     {return LOGARITHME;}
"valeur_absolu" {return ABS;}
"racine_carre" {return RACINE;}
"tangente" {return TANGENTE;}
"exp"     {return EXPONENTIELLE;}
.         { }
%%
int yywrap(void)
{
    return 1;
}
```

Rmq – Pour Compiler et Exécuter les analyseurs (CALCULATRICE SCIENTIFIQUE) :

```
cmd>> bison calculatrice.y
cmd>> flex calculatrice.l
cmd>> gcc lex.yy.c calculatrice.tab.c -o calculatrice
cmd>> calculatrice.exe
```

⇒ Exemple d'exécution :



```
C:\Windows\System32\cmd.exe
D:\ZZ - Workspace\ComputingWS\EMSI\4IIR2\COMPILATION WorkSpaces\W - Devoir\Exercice2>calculatrice.exe
Donnez une expression : racine_carre(log(45.6*3)-sinus(2.15))
Resultat = 2.020302
```