

Rapport d'un TP

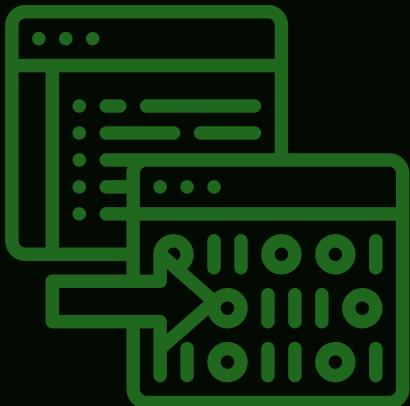
TP Compilation : Programme d'une calculatrice plus avancées avec FLEX & Bison

UNIVERSITÉ DE
BOUMERDES -FS-

DÉPARTEMENT
INFORMATIQUE

FILIÈRE
Informatique (L3)

SPÉCIALITÉ
Système d'informatique



SUPERVISÉ PAR
Dr. Lida ICHALAMEN

PRÉSENTÉ PAR
OUARAS Khelil Rafik

MATRICULE 202031041281

GROUPE 01

JANVIER 2023

I. Introduction :

Flex est un outil pour générer des analyseurs, programmes qui reconnaissent des motifs lexicaux dans du texte. Il lit les fichiers d'entrée donnés, ou bien son entrée standard si aucun fichier n'est donné, pour obtenir la description de l'analyseur à générer. La description est une liste de paires d'expressions rationnelles et de **code C**, appelées règles. En sortie, **Flex** génère un fichier source en **langage C**, appelé « **lex.yy.c** », qui définit une routine « **yylex()** ».

GNU Bison est l'implémentation **GNU** du compilateur de compilateur **Yacc**, spécialisé dans la génération d'analyseurs syntaxiques.

II. Réalisation des étapes du TP :

1. Dans le programme Flex et Bison ci-dessous, on a fait une calculatrice plus avancée :

- Calculatrice simple « +, -, *, /, %, abs »
- Calculatrice scientifique « exp(), ln(), log(), log(base), sqrt(), nombre!, exp »
- Calculatrice trigonométrique « cos, sin, tan, acos, asin, atan, Pi »
- Les Expressions Logiques et de comparaison « &&, | |, >, <, >=, <=, ==, != »
- Calculatrice Conversation (Binaire, Hexadécimal, Octal)

Le Code avec Flex :

```
%{
/* Partie Declaration : */
#include "Declaration.h"
#include "calavanc.tab.h"
#include <stdlib.h>
%}

/* Partie Declaration expression régulière : */
blancs [t]+
chiffre [0-9]
entier {chiffre}+
exposant [eE][+-]?{entier}
reel {entier}(.){entier}?{exposant}?

/* Partie expression régulière : */
%%

{blancs} { /* On ignore */ }

{reel}+ { yyval=atof(yytext); return NOMBRE; }
"+" return PLUS;
"-\" return MOINS;
"**" return FOIS;

"/" return DIVISE;
"^^" return PUISSANCE;
"%%" return MOD;
"abs" return ABS;
"sin" return SIN;
"cos" return COS;
"tan" return TAN;
"asin" return ASIN;
"acos" return ACOS;
"atan" return ATAN;
"sqrt" return SQRT;
"exp" return EXP;
"ln" return LN;
"log" return LOG;
"pi" return PI;
"," return VRLOG; // Virgule dans Log, par exemple :
log(10,2)
"bin" return BINAIRE;
"hex" return HEX;
```

```

"oct" return OCTAL;
"vrai" return VRAI;
"faux" return FAUX;
"||" return OU;
"&&" return ET;
"!" return NON;
 "(" return PG;
 ")" return PD;
 "<" return LT;
 "+" return SOMME;
 "-" return SOUVENT;
 "*" return MULTIPLICATION;
 "/" return DIVISION;
 "%" return RESTE;
 ">" return GT;
 "<=" return LE;
 ">=" return GE;
 "==" return EQ;
 "!=" return NE;
 "\n" return FIN;
 . {printf("Caractère non reconnu : %s\n", yytext);}
%%
```

Le Code avec Bison :

```

%{
// Partie declaration globale et les bibliothèques :
#include "Declaration.h" // fichier externe
"Declaration" variable
#include <stdio.h>
#include <stdlib.h>

#define Pi 3.14159265358979323846
#define E 2.71828182845904523536

// Les Fonctions :
double puissance(double x, int n){
    double result = 1;
    int i;
    for(i = 1; i <= n ; i++){
        result = result * x ;
    }
    return result;
}

double racine(double x){
    double i;
    double diff;
    if(x == 0 || x == 1){
        return(x);
    }else{
        i = x;
        do{
            diff = i;
            i= 0.5 * (i + x / i);
        }while(i != diff);
        return (i);
    }
}

double sin_calc(double x) {{// en radian
    double result = x;
    int i;
    for (i = 3; i < 10; i+=2) {
        result += puissance(-1, i) * puissance(x, i) /
factorial(i);
    }
    return result;
}}
```

```

double cos_calc(double x) {{// en radian
    double result = 1;
    int i;
    for (i = 2; i < 10; i+=2) {
        result += puissance(-1, i) * puissance(x, i) /
factorial(i);
    }
    return result;
}}
```

```

double tan_calc(double x) {{// en radian
    return sin_calc(x) / cos_calc(x);
}

double arcsin_calc(double x) {// en radian
    double result = x;
    int i;
    for (i = 3; i < 10; i+=2) {
        result += puissance(x, i) / (i * (i+1));
    }
    return result;
}

double arccos_calc(double x) {{// en radian
    return (Pi/2 - arcsin_calc(x));
}

double arctan_calc(double x) {{// en radian
    double result = x;
    int i;
    for (i = 3; i < 10; i+=2) {
        result += puissance(-1, (i-1)/2) * puissance(x, i) / i;
    }
    return result;
}

int factorial(int x) {
    if (x == 0) {
        return 1;
    } else {
        return x * factorial(x-1);
    }
}

// Partie declarations des tokens :
%token NOMBRE
%token PLUS MOINS FOIS DIVISE PUISSANCE
%token ABS SIN COS TAN ASIN ACOS ATAN
%token SQRT EXP LN LOG
%token VRAI FAUX OU ET NON LT GT LE GE EQ
%token NE
%token PI
%token VRLOG
%token BINAIRE HEX OCTAL
%token PG PD
%token FIN
%left PLUS MOINS
%left FOIS DIVISE
%left MOD
%left NEG
%right PUISSANCE

%start Input //Axiome "Input"
%%
//Partie Grammaire :
Input:
/* Vide */
|Input Ligne
;

Ligne:
FIN
|BINAIRE PG Expression PD {
    int num = (int) $1;
    char result[32];
    int i = 0;
    while (num > 0) {
        result[i] = (num % 2) + '0';
        num /= 2;
        i++;
    }
    result[i] = '\0';
    printf("Resultat: ");
    for (i = strlen(result)-1; i >= 0; i--) {
        printf("%c", result[i]);
    }
}

```

```

    }
    }

printf("\n");
}

|HEX PG Expression PD {
int num = (int) $1;
char result[32];
int i = 0;
while (num > 0) {
    int rem = num % 16;
    if (rem < 10) {
        result[i] = rem + '0';
    } else {
        result[i] = (rem-10) + 'A';
    }
    num /= 16;
    i++;
}
result[i] = '\0';
printf("Resultat: ");
for (i = strlen(result)-1; i >= 0; i--) {
    printf("%c", result[i]);
}
printf("\n");
}

|OCTAL PG Expression PD {
int num = (int) $1;
char result[32];
int i = 0;
while (num > 0) {
    result[i] = (num % 8) + '0';
    num /= 8;
    i++;
}
result[i] = '\0';
printf("Resultat: ");
for (i = strlen(result)-1; i >= 0; i--) {
    printf("%c", result[i]);
}
printf("\n");
}

}
}
}

|PI { printf("Resultat: %f\n",Pi); }
|EXP { printf("Resultat: %f\n",E); }
|Expression FIN { printf("Resultat: %f\n", $1); }
|ExpressionLogique FIN {
if($1==1){
    printf("Resultat: vrai\n");
}else{
    printf("Resultat: faux\n");
}
}
|error FIN {yyerror();}
;

Expression:

NOMBRE {$$=$1;}
|Expression PLUS Expression {$$=$1+$3;}
|Expression MOINS Expression {$$=$1-$3;}
|Expression FOIS Expression {$$=$1*$3;}
|Expression DIVISE Expression {
if($3 != 0){
    $$=$1/$3;
}else{
    printf("Impossible de diviser sur 0\n");
    $$=0;
}
}
|Expression MOD Expression {$$=(int)$1%(int)$3;}
|MOINS Expression %prec NEG {$$=-$2;}
|Expression PUISSANCE Expression
{$$=puissance($1,$3);}
|ABS PG Expression PD {$$=fabs($3);}
|SIN PG Expression PD {$$=sin_calc($3);}
|COS PG Expression PD {$$=cos_calc($3);}
|TAN PG Expression PD {
if(cos_calc($3) == 0) {
    printf("Erreur, division par 0");
    $$ = 0;
} else {
}
}

```

```

$$ = tan_calc($3);                                $$=0;
}                                                 }
}                                                 }
|ASIN PG Expression PD {
if(fabs($3) > 1) {
    printf("Erreur, arcsin de |x| > 1 est
impossible\n");
    $$ = 0;
} else {
    $$ = arcsin_calc($3);
}
}

|ACOS PG Expression PD {
if(fabs($3) > 1) {
    printf("Erreur, arccos de |x| > 1 est
impossible\n");
    $$ = 0;
} else {
    $$ = arccos_calc($3);
}
}

|ATAN PG Expression PD {$$=arctan_calc($3);}
|SQRT PG Expression PD {
if($3 >= 0){
    $$=racine($3);
}else{
    printf("Impossible de calculer la racine d'un
nombre négatif\n");
    $$=0;
}
}

|EXP PG Expression PD {$$=exp($3);}
|LN PG Expression PD {
if($3 > 0){
    $$=log($3);
}else{
    printf("Impossible de calculer le logarithme
n'éprien d'un nombre négatif ou nul\n");
}
}

|LOG PG Expression PD {
if($3 > 0){
    $$=log10($3);
}else{
    printf("Impossible de calculer log sur une
valeur négative ou nulle\n");
    $$=0;
}
}

|LOG PG Expression VRLOG Expression PD {
if($3 > 0 && $5 > 0){
    $$=log($5)/log($3);
}else{
    printf("Impossible de calculer log sur une
valeur négative ou nulle\n");
    $$=0;
}
}

|PI {$$=Pi;}
|EXP {$$=E;}
|PG Expression PD {$$=$2;}
|ExpressionLogique {$$=$1;}
;
;

ExpressionLogique:
ComparisonExpression
|Expression ET Expression {$$ = $1 && $3;}
|Expression OU Expression {$$ = $1 || $3;}
|NON Expression {$$=!($2);}
|PG ExpressionLogique PD {$$=$2;}
|VRAI {$$ = 1;}
|FAUX {$$ = 0;}
;

ComparisonExpression:
Expression LT Expression {$$ = $1 < $3;}

```

```
|Expression GT Expression {$$ = $1 > $3;}
|Expression LE Expression {$$ = $1 <= $3;}
|Expression GE Expression {$$ = $1 >= $3;}
|Expression EQ Expression {$$ = $1 == $3;}
|Expression NE Expression {$$ = $1 != $3;}
;

%%
```

```
// Partie C :
void yyerror(){
    printf("Erreur de syntaxe\n");
}

int main(void){
    printf("Entrez une expression à évaluer :\n");
    yyparse();
}
```

Le fichier externe « Declaration.h » :

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

2. Exemples d'exécutions de programme :

Pour faire l'exécution :

```
$ flex calavanc.l
$ bison -d calavanc.y
$ gcc lex.yy.c calavanc.tab.c -ll -lm -w -o calavanc
$ ./calavanc
```

2.a. Exemple calcule simple :

```
Entrez une expression à évaluer :
5+6
Resultat: 11.000000
7-9
Resultat: -2.000000
15*6
Resultat: 90.000000
15%2
Resultat: 1.000000
-5
Resultat: -5.000000
2^3
Resultat: 8.000000
abs(-90)
Resultat: 90.000000
sqrt(4)
Resultat: 2.000000
ln(1)
Resultat: 0.000000
exp(5)
Resultat: 148.413159
log(10,2)
Resultat: 0.301030
```

Figure 01 : Capture d'écran sur le Système d'exploitation « linux » l'exécution de programme « Exemple calcule simple »

2.b. Exemple calcule trigonométriques :

```
Entrez une expression à évaluer :  
pi  
Resultat: 3.141593  
sin(5)  
Resultat: -0.958925  
cos(1)  
Resultat: 0.5443080  
tan(1)  
Resultat: 0.534515  
asin(0.5)  
Resultat: 0.511620  
acos(0.333)  
Resultat: 1.234574  
atan(0.58595)  
Resultat: 0.530222  
cos(pi/2)  
Resultat: 2.509153
```

Figure 02 : Capture d'écran sur le Système d'exploitation « linux » l'exécution de programme « Exemple calcule trigonométriques »

2.c. Exemple Expressions Logiques & les comparaisons :

```
Entrez une expression à évaluer :
(1)&&(0)
Resultat: faux
(0)&&(1)
Resultat: faux
(0)&&(0)
Resultat: faux
(1)&&(1)
Resultat: vrai
(1)|||(1)
Resultat: vrai
(1)|||(0)
Resultat: vrai
(0)|||(1)
Resultat: vrai
(0)|||(0)
Resultat: faux
15==15
Resultat: vrai
15==16
Resultat: faux
15!=16
Resultat: vrai
15!=15
Resultat: faux
!(1)
Resultat: faux
!(0)
Resultat: vrai
12<15
Resultat: vrai
12>15
Resultat: faux
12<=12
Resultat: vrai
```

Figure 03 : Capture d'écran sur le Système d'exploitation « linux » l'exécution de programme « Exemple Expressions Logiques & les comparaisons »

2.d. Exemple conversation des nombres (binaire, hexadécimal, octal) :

```
Entrez une expression à évaluer :
30
Resultat: 30.000000
bin(30)
Resultat: 11110
hex(30)
Resultat: 1E
oct(30)
Resultat: 36
```

Figure 04 : Capture d'écran sur le Système d'exploitation « linux » l'exécution de programme « Exemple conversation des nombres (binaire, hexadécimal, octal) »

2.e. Exemple des erreurs de calcul :

```
Entrez une expression à évaluer :
15**5
Erreur de syntaxe
Erreur de syntaxe
15/0
Impossible de diviser sur 0
Resultat: 0.000000
ln(-5)
Impossible de calculer le logarithme népérien d'un nombre négatif ou nul
Resultat: 0.000000
log(0)
Impossible de calculer log sur une valeur négative ou nulle
Resultat: 0.000000
asin(50)
Erreur, arcsin de |x| > 1 est impossible
Resultat: 0.000000
acos(60)
Erreur, arccos de |x| > 1 est impossible
Resultat: 0.000000
atan(66)
Resultat: 2639518721010274.000000
--9
Resultat: 9.000000
bin(5
Erreur de syntaxe
Erreur de syntaxe
(5*5)*9-5(6
Erreur de syntaxe
Erreur de syntaxe
sqrt(-6)
Impossible de calculer la racine d'un nombre négatif
Resultat: 0.000000
log(1,-2)
Impossible de calculer log sur une valeur négative ou nulle
Resultat: 0.000000
```

Figure 05 : Capture d'écran sur le Système d'exploitation « linux » l'exécution de programme « Exemple des erreurs de calcul »

🔗 [Lien de téléchargement les code \(TP Compilation « Calculatrice plus avancées »\) :](#)

<https://mega.nz/folder/cUAXiKTD#5dwO-sNYLf6bwzo1qCTfZA>

III. Conclusion :

Suite à ce TP ont conclu à partir de se familiariser aux commandes **FLEX** et **Bison**, que les deux outils de compilation par défaut sur **les systèmes UNIX** depuis plusieurs décennies. Le premier outil **FLEX** (version **GNU** de la commande **Lex**) construit un analyseur lexical à partir d'un ensemble de règles/actions décrites par des expressions régulières. Le second outil **Bison** est un compilateur de compilateur, version **GNU** de la célèbre commande **Yacc** acronyme de « **yet another compiler of compilers** ».