Université Cadi Ayyad
Faculté des sciences Smlalia Marrakech
Département d'informatique
Module de compilation
2020/2021

Projet :

Réalisation d'un compilateur

En utilisant LEX et YACC

Réaliser par :

Iken samya

EL BOUGA latifa

Encadré par :

Mr QAZDAR aimad

Table de matières

Objectif

L'objectif de ce projet est de réaliser un compilateur d'une version simplifiée du une langage source proche de l'algorithmique en utilisant le générateur d'analyseur lexical LEX et le générateur d'analyseur syntaxique YACC (**Y**et **A**nother **C**ompiler **C**ompiler).

Le travail va être structuré en deux parties :

1. **La première partie** : nous allons développer un générateur d'analyseur lexical pour produire un automate fini déterministe minimal permettant de reconnaitre les unités lexicales.
   L'automate est produit sous la forme d'un programme c.
   Il existe plusieurs versions de LEX, nous utiliserons ici FLEX.
2. **La deuxième partie** : nous allons développer un générateur d'analyseur syntaxique qui prend en entrée la définition d'un schéma de traduction (grammaire).
   Il existe plusieurs versions de YACC, nous utiliserons ici bison.

1. Analyseur lexical

1.1 Spécifications lexicales du langage

Création de fichier « unitesLexiquales.h » qui contient les tokens, leurs valeurs et les variables externes :

```
#define ALGO 257

#define IDENTIF 258
#define VAR 259

#define VIRGULE 260
#define DEUX_POINTS 261

#define ENTIER 262
#define REEL 263
#define CARACTERE 264
#define BOOLEAN 265
#define TYPEDF 266
#define STR 267

#define ACC_O 268
#define ACC_F 269

#define FICHIER 270

#define NOMBRE 272

#define DEBUT 273
```

```
#define COMMENT 274
#define FIN 275
#define AFFICHER 276cap

#define PARTH_O 277
#define PARTH_F 279

#define LIRE 280

#define ADD 281
#define SOUST 282
#define MULT 283
#define DIVS 284
#define MODU 285

#define ET 286
#define NOT 287
#define OR 288

#define AFFECTATION 289
#define EGAL 290
#define DIFF 291

#define INF 292
#define INFEGAL 293
#define SUP 294
#define SUPEGAL 295
```

```
#define INC 296
#define DEC 297

#define ASS_ADD 298
#define ASS_SOUS 299
#define ASS_MULT 300
#define ASS_DIV 301

#define SI 302
#define ALORS 303
#define SINON 304
#define FINSI 305
#define FINSINON 306

#define SWITCH 307
#define CAS 308
#define POINT_VIRGULE 309
#define BREAK 310
#define AUTRE 311
#define FINSWITCH 312

#define POUR 313
#define ALLANT 314
#define TO 315
#define FINPOUR 316
```

```
#define TANTQUE 317
#define FAIRE 318
#define FINTQ 319

#define REPETER 320
#define WHILE 321

#define LOOP 322
#define ENDLOOP 323

#define GOTO 324

#define FUNCTION 325
#define FINFCT 326

#define PROCEDURE 327
#define FINPROC 328
#define TYPE 329
#define AFFICHAGE 330
#define DOUBLE_COTES 331
extern int valNombre;
extern char valIdentif[];
extern int valEntier;
extern float valReel;
extern char valBoolean[];
extern char valFichier[];
extern int LineNumber;
```

```
extern char valAffichage[];
extern char valChar[];
```

1.2 Le fichier « analexe.l »

```
%{
        #include<string.h>
        #include "unitesLexicales.h"
%}

nbr [0-9]
identif [a-zA-Z][0-9a-zA-Z_]*
entier [+-]?{nbr}+
reel [+-]?{nbr}+"."{nbr}+
boolean vrai|faux
fichier {identif}\.{identif}
text [A-Za-z0-9_,; \t\n]
affichage \".*\"
comment "/*"{text}+"*/"
type entier|reel|boolean
car \'.*\'

%%

","  { ECHO; return VIRGULE;}
":"  { ECHO; return DEUX_POINTS; }
"{"  { ECHO; return ACC_O; }
"}"  { ECHO; return ACC_F; }
")"  { ECHO; return PARTH_F; }
"("  { ECHO; return PARTH_O; }
"+"   { ECHO; return ADD; }
"-"   { ECHO; return SOUST; }
"*"   { ECHO; return MULT; }
"/"   { ECHO; return DIVS; }
"%"   { ECHO; return MODU; }
"&"   { ECHO; return ET; }
"!"   { ECHO; return NOT; }
"|"   { ECHO; return OR; }
"="   { ECHO; return AFFECTATION; }
"=="   { ECHO; return EGAL; }
"<>"   { ECHO; return DIFF; }
"<"   { ECHO; return INF; }
"<="   { ECHO; return INFEGAL; }
">"   { ECHO; return SUP; }
">="   { ECHO; return SUPEGAL; }
"++"   { ECHO; return INC; }
"+="   { ECHO; return ASS_ADD; }
```

```
"-="   { ECHO; return ASS_SOUS; }
"*="   { ECHO; return ASS_MULT; }
"/="   { ECHO; return ASS_DIV; }
"\""  { ECHO; return DOUBLE_COTES; }
";"   { ECHO; return POINT_VIRGULE; }
\n { }
\t { }
" " { }

algo   { ECHO; return ALGO; }
var   { ECHO; return VAR; }

caractere    { ECHO; return CARACTERE; }
typedf    { ECHO; return TYPEDF; }
str    { ECHO; return STR; }
debut    { ECHO; return DEBUT; }
fin    { ECHO; return FIN; }
afficher    { ECHO; return AFFICHER; }
lire    { ECHO; return LIRE; }
si    { ECHO; return SI; }
alors    { ECHO; return ALORS; }
sinon    { ECHO; return SINON; }
finsi    { ECHO; return FINSI; }
finsinon    { ECHO; return FINSINON; }
switch { ECHO; return SWITCH; }
cas { ECHO; return CAS; }
break { ECHO; return BREAK; }
autre { ECHO; return AUTRE; }
finswitch { ECHO; return FINSWITCH; }


pour { ECHO; return POUR; }
allant { ECHO; return ALLANT; }
to { ECHO; return TO; }
finpour { ECHO; return FINPOUR; }
tantque { ECHO; return TANTQUE; }
faire { ECHO; return FAIRE; }
fintq { ECHO; return FINTQ; }
repeter { ECHO; return REPETER; }
while { ECHO; return WHILE; }
loop { ECHO; return LOOP; }
endloop { ECHO; return ENDLOOP; }
goto { ECHO; return GOTO; }
function { ECHO; return FUNCTION; }
finfct { ECHO; return FINFCT; }
procedure { ECHO; return PROCEDURE; }
finproc { ECHO; return FINPROC; }
```

```
{boolean}    { ECHO; strcpy(valBoolean, yytext); return BOOLEAN; }
{type} {ECHO; return TYPE;}
{affichage} {ECHO;yytext[strlen(yytext)-1]='\0' ;strcpy(valAffichage,yytext+1); return AFFICHAGE;}
{car} { ECHO; yytext[strlen(yytext)-1]='\0' ; strcpy(valChar, yytext+1); return CARACTERE; }
{comment}  { ECHO; return COMMENT; }
{identif} {ECHO; strcpy(valIdentif, yytext); return IDENTIF; }
{entier}    { ECHO; valEntier = atoi(yytext);return ENTIER; }
{reel}    { ECHO; valReel = atof(yytext);return REEL; }
{fichier}    { ECHO; strcpy(valFichier, yytext);return FICHIER; }
. { ECHO; return yytext[0]; }

%%
int valEntier;
float valReel;
char valBoolean[6];
char valIdentif[256];
char valFichier[256];
char valAffichage[256];
char valChar[256];

int yywrap(void) {
       return 1;
}
```

1.3 Le fichier « principal.c »

```
#include<stdio.h>
#include "unitesLexicales.h"

int main(void) {
        int unite;

        do{
                unite = yylex();
                printf(" (unite: %d", unite);
                if(unite == ENTIER)
                        printf(" val Entier: %d", valEntier);
                else if (unite == REEL)
                        printf(" val Reel: %f", valReel);
                else if (unite == BOOLEAN)
                        printf(" val Boolean: %d", valBoolean);
                else if(unite == IDENTIF)
                        printf(" val Identif: '%s'", valIdentif);
                else if(unite == AFFICHAGE)
                        printf(" val Affichage: '%s'", valAffichage);
                else if(unite == CARACTERE)
                        printf(" val Char: '%s'", valChar);
                printf(")\n");
        }while(unite != 0);
        return 0;
}
```

1.4 Tester par un exemple

1.4.1 Exemple

```
j='gjkjkzekl'
pour tantque faire function
pour faire ;
```

1.4.2 Les commandes d'exécution :

```
flex -omonCompilateur.c analexe.l

gcc -o monCompilateur monCompilateur.c principal.c

monCompilateur < algo.txt
```

1.4.3 Après l'exécution :

```
D:\EL BOUGA_IKEN_analyseur lexicale\analyseur lexicale>monCompilateur  0<algo.txt
j (unite: 258 val Identif: 'j')
= (unite: 289)
'gjkjkzekl' (unite: 264 val Char: 'gjkjkzekl')
pour (unite: 313)
tantque (unite: 317)
faire (unite: 318)
function (unite: 325)
pour (unite: 313)
faire (unite: 318)
; (unite: 309)
 (unite: 0)

D:\EL BOUGA_IKEN_analyseur lexicale\analyseur lexicale>
```

2. Analyseur syntaxique

2.1 Le fichier « analexe.l »

```
%{
        extern int lineNumber;
        #include<string.h>
        #include "syntaxeY.h"
%}

%option noyywrap

nbr [0-9]
identif [a-zA-Z][0-9a-zA-Z_]*
entier {nbr}+
reel [-]?{nbr}+"."{nbr}+
boolean vrai|faux
string \".*\"
text [A-Za-z0-9_,; \n\t]
comment "//".*""
comment_lignes "/*"{text}+"*/"
type entier|reel|boolean|caractere
car \'.*\'

%%

","  {  return VIRGULE;}
":"  {  return DEUX_POINTS; }
"{"  {  return ACC_O; }
"}"  {  return ACC_F; }
")"  {  return PARTH_F; }
"("  {  return PARTH_O; }
"+"   {  return ADD; }
"-"   {  return SOUST; }
"*"   {  return MULT; }
"/"   {  return DIVS; }
"%"   {  return MODU; }
"&"   {  return ET; }
"!"   {  return NOT; }
"|"   {  return OR; }
"<-"   {  return AFFECTATION; }
"="   {  return EGAL; }
"<>"   {  return DIFF; }
"<"   {  return INF; }
"<="   {  return INFEGAL; }
">"   {  return SUP; }
">="   {  return SUPEGAL; }
```

```
"++"   {  return INC; }
"--" { return DEC;}
"+="   {  return ASS_ADD; }
"-="   {  return ASS_SOUS; }
"*="   {  return ASS_MULT; }
"/="   {  return ASS_DIV; }
"\""  {  return DOUBLE_COTES; }
";"   {  return POINT_VIRGULE; }
"[" { return CR_O; }
"]" { return CR_F; }

algo   {  return ALGO; }
var   {  return VAR; }
const { return CONST; }
caractere    {  return CARACTERE; }
debut    {  return DEBUT; }
fin    {  return FIN; }
afficher    {  return AFFICHER; }
lire    {  return LIRE; }
si    {  return SI; }
alors    {  return ALORS; }
sinon    {  return SINON; }
finsi    {  return FINSI; }
finsinon    {  return FINSINON; }
selon {  return SELON; }
cas {  return CAS; }
break {  return BREAK; }
autre {  return AUTRE; }
finswitch {  return FINSWITCH; }


pour {  return POUR; }
allant {  return ALLANT; }
to {  return TO; }
finpour {  return FINPOUR; }
tantque {  return TANTQUE; }
faire {  return FAIRE; }
fintq {  return FINTQ; }
repeter {  return REPETER; }
jusqu {  return JUSQU; }
function {  return FUNCTION; }
deb_fct {  return DEB_FCT; }
deb_proc {  return DEB_PROC; }
retourne {  return RETOURNE; }
finfct {  return FINFCT; }
procedure {  return PROCEDURE; }
finproc {  return FINPROC; }
```

```
tableau { return TABLEAU; }
matrice { return MATRICE; }
{boolean}    {  strcpy(yylval.valBoolean, yytext); return BOOLEAN; }
{type} { return TYPE;}
{string} {strcpy(yylval.valAffichage,yytext+1); return STRING;}
{car} {  yytext[strlen(yytext)-1]='\0' ; strcpy(yylval.valChar, yytext+1); return CARACTERE; }
{comment}  {  return COMMENT; }
{comment_lignes} { return COMMENT_LIGNES; }
{identif} { strcpy(yylval.valIdentif, yytext); return IDENTIF; }
{entier}    {  yylval.valEntier = atoi(yytext);return ENTIER; }
{reel}    {  yylval.valReel = atof(yytext);return REEL; }

[ \t] ;
\n ++lineNumber;
. {  return yytext[0]; }

%%
```

2.2 Le fichier « syntaxe.y »

```
%{
        #include <stdio.h>

        extern FILE* yyin;

        int yylex(void);
        void yyerror(const char * msg);

        int lineNumber;
%}

%union{
int valEntier;
float valReel;
char valBoolean[6];
char valIdentif[256];
char valAffichage[256];
char valChar[256];
}


%token ALGO <valIdentif>IDENTIF VAR VIRGULE DEUX_POINTS <valEntier>ENTIER <valReel>REEL <valChar>CARACTERE <valBoolean>BOOLEAN
%token  ACC_O ACC_F NOMBRE DEBUT INFEGAL TABLEAU CR_O CR_F MATRICE
%token COMMENT COMMENT_LIGNES FIN <valAffichage>AFFICHER PARTH_O PARTH_F LIRE ADD SOUST MULT DIVS MODU ET NOT OR AFFECTATION
%token SUP SUPEGAL INC DEC ASS_ADD ASS_SOUS ASS_MULT ASS_DIV SI ALORS SINON FINSI FINSINON SELON CAS POINT_VIRGULE
%token BREAK AUTRE FINSWITCH POUR ALLANT TO FINPOUR TANTQUE FAIRE FINTQ REPETER JUSQU FUNCTION EGAL
%token FINFCT PROCEDURE FINPROC TYPE AFFICHAGE DOUBLE_COTES DEB_FCT RETOURNE DEB_PROC CONST STRING DIFF INF
```

```
%start program
%%
/*---------------------- L'ALGORITHME --------------------------------*/

program : listFonction listProcedure main
                | listProcedure listFonction main
                | listProcedure main
                | listFonction main
                | main { printf("main est correct\n");}

;

/*---------------------- TRAITEMENT DES FONCTIONS ET PROCEDURES ----------------------*/

listFonction : listFonction fonction
                        | fonction {printf("fonction est correcte\n");}
;

listProcedure : listProcedure procedure
                         | procedure {printf("procedure est correcte\n");}
;

fonction : FUNCTION nom_fct PARTH_O listArg PARTH_F DEUX_POINTS TYPE DEB_FCT declaration  listInst RETOURNE ret FINFCT
                |FUNCTION nom_fct PARTH_O PARTH_F DEUX_POINTS TYPE DEB_FCT declaration listInst RETOURNE ret FINFCT
;

procedure : PROCEDURE nom_proc PARTH_O listArg PARTH_F DEB_PROC declaration listInst FINPROC
                 |PROCEDURE nom_proc PARTH_O PARTH_F DEB_PROC declaration listInst FINPROC
;

main : ALGO nom_algo declaration DEBUT listInst FIN
;

nom_fct : IDENTIF
;

nom_proc : IDENTIF
;

nom_algo : IDENTIF
;

/*----- les arguments*/

listArg : listArg arg
                | arg
;

arg : arg VIRGULE args
        | IDENTIF DEUX_POINTS TYPE
;

args : IDENTIF DEUX_POINTS TYPE


/*----- la partie déclaration-------*/

declaration : declaration VAR identifs DEUX_POINTS TYPE POINT_VIRGULE
                        | VAR identifs DEUX_POINTS TYPE POINT_VIRGULE
                        | VAR TABLEAU IDENTIF CR_O ENTIER CR_F DEUX_POINTS TYPE POINT_VIRGULE
                        | declaration VAR TABLEAU IDENTIF CR_O ENTIER CR_F DEUX_POINTS TYPE POINT_VIRGULE
                        | VAR MATRICE IDENTIF CR_O ENTIER CR_F CR_O ENTIER CR_F DEUX_POINTS TYPE POINT_VIRGULE
                        | declaration VAR MATRICE IDENTIF CR_O ENTIER CR_F CR_O ENTIER CR_F DEUX_POINTS TYPE POINT_VIRGULE
                        | CONST IDENTIF EGAL expr POINT_VIRGULE
                        | declaration CONST IDENTIF EGAL expr POINT_VIRGULE


;
expr : ENTIER
;

identifs : identifs VIRGULE IDENTIF
                | IDENTIF
;

ret : expressionArithmetic POINT_VIRGULE

;
```

```
exprSi : instSi FINSI
               | instSi instSinon FINSI
;

instSi : SI PARTH_O condition PARTH_F ALORS listInst
;
instSinon : SINON listInst
;
condition : PARTH_O  co PARTH_F opeLogique condition
                  | co
                  | PARTH_O co PARTH_F
;
opeLogique : ET
                        | OR
;
co : valeur cond valeur
   | NOT condition
;
valeur : ENTIER
                | REEL
                | CARACTERE
                | BOOLEAN
                | IDENTIF
;

cond : EGAL
         | DIFF
         | INF
         | INFEGAL
         | SUP
         | SUPEGAL
;
expressionArithmetic : expressionArithmetic ADD term
                                          | expressionArithmetic SOUST term
                                          | term
;

term : term MULT factor
     | term DIVS factor
     | factor
;
```

```
/*------ LES INSTRUCTIONS----------*/

listInst : listInst inst
                | inst
                | listInst exprSi
                | exprSi
                | listInst affichage
                | affichage
                | listInst lecture
                | lecture
                | listInst boucle
                | boucle
                | listInst switch
                | switch

;

inst :   /*----- les expressions arith & logiques -----*/
                IDENTIF AFFECTATION expressionArithmetic POINT_VIRGULE
         | expressionSupp
         /*--------- les appels ---------*/
                | appelProc {printf("appel proc correcte\n");}
                | appelFct {printf("appel fct correcte\n");}
                | COMMENT {printf("commentaire correcte\n");}
                | COMMENT_LIGNES {printf("commentaire correcte\n");}
;

lecture : LIRE PARTH_O identifs PARTH_F POINT_VIRGULE
;

affichage : AFFICHER PARTH_O parametres PARTH_F POINT_VIRGULE {printf("affichage correcte\n");}
;

parametres : parametres VIRGULE param
                  | param
;

param : STRING
          | expressionArithmetic
;


 factor : PARTH_O expressionArithmetic PARTH_F
         | IDENTIF
         | ENTIER
         | REEL
         | SOUST ENTIER
         | SOUST REEL
;

appelFct : IDENTIF EGAL IDENTIF PARTH_O PARTH_F POINT_VIRGULE
                 | IDENTIF EGAL IDENTIF PARTH_O identifs PARTH_F  POINT_VIRGULE
;
appelProc : IDENTIF PARTH_O PARTH_F POINT_VIRGULE
                   | IDENTIF PARTH_O identifs PARTH_F POINT_VIRGULE
;
```

```
boucle : pour {printf("pour correct\n");}
               | tantque {printf("tq correct\n");}
               | repeter {printf("repeter correct\n");}

;

pour : POUR IDENTIF ALLANT valeur TO valeur FAIRE listInst FINPOUR
;

tantque : TANTQUE PARTH_O condition PARTH_F FAIRE listInst FINTQ
;

repeter : REPETER listInst JUSQU PARTH_O condition PARTH_F POINT_VIRGULE
;

switch : SELON PARTH_O IDENTIF PARTH_F ACC_O listdesCas ACC_F {printf("switch correct\n");}


listdesCas : listdesCas lesCas
                        | lesCas
;
lesCas : CAS valeur DEUX_POINTS listInst BREAK POINT_VIRGULE
       | AUTRE DEUX_POINTS listInst
;



  expressionSupp : incrementation {printf("instuction d'incrementation \n");}
                             | decrementation {printf(" instuction de decrementation\n");}
                             | assignationAdd
                             | assignationSous
                             | assignationMult
                             | assignationDiv
  ;
  assignationAdd : IDENTIF ASS_ADD val POINT_VIRGULE
  ;
  assignationSous: IDENTIF ASS_SOUS val POINT_VIRGULE
  ;
  assignationMult: IDENTIF ASS_MULT val POINT_VIRGULE
  ;
  assignationDiv : IDENTIF ASS_DIV val POINT_VIRGULE
  ;
  val : ENTIER
          | REEL
          | IDENTIF
  ;
  incrementation : IDENTIF INC POINT_VIRGULE
                             | INC IDENTIF POINT_VIRGULE
  ;
  decrementation : IDENTIF DEC POINT_VIRGULE
                             | DEC IDENTIF POINT_VIRGULE
  ;

  %%
  void yyerror( const char * msg){
          printf("\nline %d : %s", lineNumber, msg);
  }
  int main(int argc,char ** argv){
          if(argc>1) yyin=fopen(argv[1],"r"); // check result !!!
          lineNumber=1;
          if(!yyparse())
          printf("CORRECT!!\n");
  return(0);
  }
```

2.3 Tester par un exemple

2.3.1 Exemple :

```
|
function somme(a : entier, b : entier):entier deb_fct
        var s : entier;
        s <- a + b;
        s += 5;
        afficher(a,a+h);
        retourne h+1;
finfct

algo test
        const a = 4;
        var a,b,t:entier;
        var tableau lat[12] : entier;
        var matrice lat[12][3] : entier;
debut
        ++t;
        --t;
        si ((a = b) & (v= 3)) alors
                si(a = b) alors
                        a <- a + b;
                        afficher("la valeur de a est :",(V+n)*2,b);
                        lire(g,j);
                        proc();       // appel a une procedure
                finsi
        sinon
        a <- a-b;
        a = fonct(d);   // appel a une fonction
        finsi
        pour a allant 7 to q faire    // boucle imbriquee
                pour a allant q to 8 faire
                        ++p; q<- 8*9+2;
                        pour q allant v to p faire
                                a -= 9;
                        finpour
                finpour
        finpour
                                /* commentaire sur plusieurs
                                lignes */
```

```
        tantque(a <0) faire
                tantque(a > o) faire
                        afficher("la valeur est ",e);
                        a++;
                fintq
        fintq

        repeter
                a++;
        jusqu(q=0);

        selon (a) {
                cas 0 : a++; break;
                cas 1 : afficher("le switch",f, a+9); break;
                autre : a<-8*8/97+55;
                }
        /*fin du
        programme */
fin
```

2.3.2 Les commandes d'exécution :

```
flex -olexiqueL.c analexe.l

bison -d -osyntaxeY.c syntaxe.y

gcc -o prog lexiqueL.c syntaxeY.c

prog < exemple.txt
```

2.3.3 Après l'exécution :

```
D:\EL BOUGA_IKEN_analyseur syntaxique\analyseur syntaxique>flex -olexiqueL.c analexe.l

D:\EL BOUGA_IKEN_analyseur syntaxique\analyseur syntaxique>bison -d -osyntaxeY.c syntaxe.y

D:\EL BOUGA_IKEN_analyseur syntaxique\analyseur syntaxique>gcc -o prog lexiqueL.c syntaxeY.c

D:\EL BOUGA_IKEN_analyseur syntaxique\analyseur syntaxique>prog  0<exemple.txt
affichage correcte
fonction est correcte
instuction d'incrementation
 instuction de decrementation
affichage correcte
appel proc correcte
commentaire correcte
appel fct correcte
commentaire correcte
commentaire correcte
instuction d'incrementation
pour correct
pour correct
pour correct
commentaire correcte
affichage correcte
instuction d'incrementation
tq correct
tq correct
instuction d'incrementation
repeter correct
instuction d'incrementation
affichage correcte
switch correct
commentaire correcte
CORRECT!!

D:\EL BOUGA_IKEN_analyseur syntaxique\analyseur syntaxique>
```