EXEMPLE D'UTILISATION DE LEX

11 mars 2009

lex ou flex permettent d'effecturer l'analyse lexicale ; ils génèrent un programme en C qui découpe un flux de caractères en unités lexicales : des entiers, des flottants, des noms, des signes spéciaux (+, −, etc) ; cette séquence de lexèmes est ensuite traitée par un anaylyseur syntaxique (parser).

1 basic.lex

```
%{
#include<string.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

enum KindToken {INT,FLOAT,STRING,LP,RP,LB,RB,PLUS,MOINS,MULT,DIV,PV,EGAL};
struct Token { KindToken kind;
     union { int the_int; float the_float; char * the_string; } Value; };
int nblex= 0;
Token lexemes[1000];
%}
digit [0-9]
lettre [a-zA-Z]
%%
" "|\t|\n {;}
{lettre}({lettre}|{digit})* {
      lexemes[nblex].kind=STRING;
      int l=strlen( yytext) + 1;
      char *copy = new char[l];
      strcpy( copy, yytext);
      lexemes[nblex].Value.the_string=copy;
      /*printf("coucou ici yytext=%s\n", yytext);*/
      nblex++; }
{digit}+      { lexemes[nblex].kind=INT;
    lexemes[nblex].Value.the_int = atoi(yytext) ;
    nblex++; }
{digit}+\.{digit}* { lexemes[nblex].kind=FLOAT;
    lexemes[nblex].Value.the_float = atof(yytext) ;
    nblex++; }
"\-"    { lexemes[nblex].kind=MOINS; nblex++; }
```

```
"\/"    { lexemes[nblex].kind=DIV; nblex++; }
"="     { lexemes[nblex].kind=EGAL; nblex++; }
"+"     { lexemes[nblex].kind=PLUS; nblex++; }
"*"     { lexemes[nblex].kind=MULT; nblex++; }
"("     { lexemes[nblex].kind=LP; nblex++; }
")"     { lexemes[nblex].kind=RP; nblex++; }
;   { lexemes[nblex].kind=PV; nblex++; }
.   { printf( "non reconnu: %s\n", yytext); }
%%
main( void )
{
      yylex();
      for( int i=0; i<nblex; i++)
      switch (lexemes[i].kind)
      {
      case STRING: printf( "STRING %s\n", lexemes[i].Value.the_string); break;
      case INT: printf( "INT %d\n", lexemes[i].Value.the_int); break;
      case FLOAT: printf( "FLOAT %f\n", lexemes[i].Value.the_float); break;
      case PLUS: printf( "PLUS +\n"); break;
      case MOINS: printf( "MOINS -\n"); break;
      case MULT: printf( "MULT *\n"); break;
      case DIV: printf( "DIV /\n"); break;
      case PV: printf( "PV ;\n"); break;
      case LP: printf( "LP (\n"); break;
      case RP: printf( "RP )\n"); break;
      default : printf( "heu\n"); break;
      }
      return 0;
}
```

2 makefile

```
ok : lexical basic
lexical : lexical.lex lexical.cpp
    lex lexical.lex
    g++ -o lexical lexical.cpp   -lfl -lc
basic : basic.lex
    lex -obasic_lex.cpp basic.lex
    g++ basic_lex.cpp -o basic -lfl -lc
```

3 Variante

3.1 decla.h

```
#ifndef DECLA_INCLUDED
#define DECLA_INCLUDED
enum KindToken { INT, FLOAT, STRING, LP, RP, LB, RB,
                 PLUS,MOINS,MULT,DIV,PV,EGAL};
struct Token { KindToken kind;
```

```
     union { int the_int;
       float the_float;
       char * the_string;
         } Value;
};
extern Token lexemes[];
extern int nblex;
#endif
```

3.2 lexical.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include"lex.yy.c"
int nblex= 0;
Token lexemes[1000];
main()
{
    yylex();
    for( int i=0; i<nblex; i++)
    switch (lexemes[i].kind)
    {
    case STRING: printf( "STRING %s\n", lexemes[i].Value.the_string); break;
    case INT: printf( "INT %d\n", lexemes[i].Value.the_int); break;
    case FLOAT: printf( "FLOAT %f\n", lexemes[i].Value.the_float); break;
    case PLUS: printf( "PLUS +\n"); break;
    case MOINS: printf( "MOINS -\n"); break;
    case MULT: printf( "MULT *\n"); break;
    case DIV: printf( "DIV /\n"); break;
    case PV: printf( "PV ;\n"); break;
    case LP: printf( "LP (\n"); break;
    case RP: printf( "RP )\n"); break;
    default : printf( "heu\n"); break;
    }
}
```

3.3 lexical.lex

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

chiffre [0-9]
lettre [a-zA-Z]

%%
" "|\t|\n {}
```

```
{lettre}({lettre}|{chiffre})* {
      lexemes[nblex].kind=STRING;
      int l=strlen( yytext) + 1;
      char *copy = new char[l];
      strcpy( copy, yytext);
      lexemes[nblex].Value.the_string=copy;
      //printf("coucou ici yytext=%s\n", yytext);
      nblex++; }
[0-9]+ { lexemes[nblex].kind=INT;
         lexemes[nblex].Value.the_int = atoi(yytext) ;
         nblex++; }
[0-9]+\.[0-9]* { lexemes[nblex].kind=FLOAT;
                 lexemes[nblex].Value.the_float = float( atof(yytext)) ;
                 nblex++; }
"\-"    { lexemes[nblex].kind=MOINS; nblex++; }
"\/"    { lexemes[nblex].kind=DIV; nblex++; }
"="     { lexemes[nblex].kind=EGAL; nblex++; }
"+"     { lexemes[nblex].kind=PLUS; nblex++; }
"*"     { lexemes[nblex].kind=MULT; nblex++; }
"("     { lexemes[nblex].kind=LP; nblex++; }
")"     { lexemes[nblex].kind=RP; nblex++; }
;   { lexemes[nblex].kind=PV; nblex++; }
.   { printf( "non reconnu: %s\n", yytext); }
```

Lex et Yacc, exemples introductifs

D. Michelucci

1 LEX

1.1 Fichier makefile

```
exemple1: exemple1.lex
        flex -oexemple1.c exemple1.lex
        gcc -o exemple1 exemple1.c -lfl -lc
        exemple1 < exemple1.input


# lex example1.l
# cc lex.yy.c -o example1 -ll
# NOTE: If you are using flex, instead of lex,
# you may have to change '-ll' to '-lfl'
# in the compilation scripts. RedHat 6.x and SuSE need this,
# even when you invoke 'flex' as 'lex'!

exemple2: exemple2.lex
        flex -oexemple2.c exemple2.lex
        gcc -o exemple2 exemple2.c -lfl -lc
        exemple2 < exemple2.input

exemple3: exemple3.lex
        flex -oexemple3.c exemple3.lex
        gcc -o exemple3 exemple3.c -lfl -lc
        exemple3 < exemple3.input

p1: p1.lex
        flex -op1.c p1.lex
        gcc -o p1 p1.c -lfl -lc

p2: p2.lex
        flex -op2.c p2.lex
        gcc -o p2 p2.c -lfl -lc
        p2 < p3.input

p3: p3.lex
        flex -op3.c p3.lex
        gcc -o p3 p3.c -lfl -lc
        p3 p3.input
```

1.2 Fichier exemple1.lex

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

%%
stop    printf("Stop command received\n");
start   printf("Start command received\n");
%%
```

1.3 Fichier exemple1.input

```
stop stop start start
```

1.4 Fichier exemple2.lex

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

%%
[0123456789]+           printf("NUMBER:%s\n", yytext);
[a-zA-Z][a-zA-Z0-9]*    printf("WORD:%s\n", yytext);
%%
```

1.5 Fichier exemple3.lex

Exemple de fichier en entrée :

```
logging {
        category lame-servers { null; };
        category cname { null; };
};

zone "." {
        type hint;
        file "/etc/bind/db.root";
};
```

La sortie correspondante :

```
WORD OBRACE
WORD FILENAME OBRACE WORD SEMICOLON EBRACE SEMICOLON
WORD WORD OBRACE WORD SEMICOLON EBRACE SEMICOLON
EBRACE SEMICOLON

WORD QUOTE FILENAME QUOTE OBRACE
WORD WORD SEMICOLON
WORD QUOTE FILENAME QUOTE SEMICOLON
EBRACE SEMICOLON
```

Le fichier exemple3.lex correspondant :

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

%%
[a-zA-Z][a-zA-Z0-9]*    printf("WORD ");
[a-zA-Z0-9\/.-]+        printf("FILENAME ");
\"                      printf("QUOTE ");
\{                      printf("OBRACE ");
\}                      printf("EBRACE ");
;                       printf("SEMICOLON ");
\n                      printf("\n");
[ \t]+                  /* ignore whitespace */;
%%
```

1.6 p2.lex

```
%{
#define _UINT       1
```

```
#define _SINT       2
#define _REAL       3
#define _KEYWORD    4
#define _ID         5
  int lines=0;
%}
uint  ([1-9][0-9]*)
real  ([0-9]?\.[0-9]+)
id    ([_a-zA-Z][_0-9a-zA-Z]*)
ws    ([ \t]+)
%%
{ws}         {; }
\n           { lines++;}
int |
for |
do |
if |
then |
else         { return _KEYWORD;}
{uint}       { return _UINT; }
[+-]{uint}   { return _SINT; }
{id}         { return _ID; }
{real}       { return _REAL; }
.            { printf("Error: Undefined string found! (%s)\n",yytext);
               exit(1); }
%%
int yywrap(void){
  return 1;
}

int main(int argc, char *argv[]) {
  int res;

  if(argc>2) {
    printf("Syntax: %s [infile]\n",argv[0]);
    exit(1);
  }
  if(argc==2)
    if((yyin=fopen(argv[1],"r"))==NULL) {
      printf("Cannot open input file %s.\n",argv[1]);
      exit(1);
    }
  /* otherwise yyin = stdin */
  while(res=yylex()){
    switch(res){
      case _UINT: printf("<unsigned int>"); break;
      case _SINT: printf("<signed int>"); break;
      case _REAL: printf("<real>"); break;
      case _KEYWORD: printf("<keyword>"); break;
      case _ID: printf("<ID>");
    }
    printf("[%s]\n",yytext);
  }
  printf(" %d lines processed.\n",lines);
  fclose(yyin);
  return 0;
}
```

1.7 p3.lex illustre yywrap, les états

```
%{
#define _UINT       1
```

```
#define _SINT       2
#define _REAL       3
#define _KEYWORD    4
#define _ID         5
  int lines=0;
  int filesno=0;
  char **filename=NULL;
%}
                                  /* the state COMMENT is exclusive */
%x COMMENT
uint  ([1-9][0-9]*)
real  ([0-9]?\.[0-9]+)
id    ([_a-zA-Z][_0-9a-zA-Z]*)
ws    ([ \t]+)
%%
{ws}                {; }
"/*"                { BEGIN(COMMENT); /* enter comment eating mode */ }
<COMMENT>"*/"       { BEGIN (INITIAL); /* exit comment eating mode */ }
<INITIAL,COMMENT>\n  { lines++; /* INITIAL is defined as 0 */ }
<COMMENT>.          { ; /* eat comments */ }
int |
for |
do |
if |
then |
else                { return _KEYWORD;}
{uint}              { return _UINT; }
[+-]{uint}          { return _SINT; }
{id}                { return _ID; }
{real}              { return _REAL; }
.                   { printf("Error: Undefined string found!(%s)\n",yytext);
                      exit(1); }
%%

int yywrap(void){
  static int i=1;

  if(filesno<=0) return 1;                  /* no other input file */
  if((yyin=fopen(filename[i],"r"))==NULL) {
    printf("Cannot open input file %s.\n",filename[i]);
    exit(1);
  }
  i++;
  filesno--;
  return 0;
}

int main(int argc, char *argv[]) {
  int res;

  if(argc == 1) {
    printf("Syntax: %s infile1 [infile2 ...]\n",argv[0]);
    exit(1);
  }
  filesno=argc-1;               /* number of input files */
  filename=argv;                /* names of input files  */
  yywrap();                     /* opens the input first file */
  while(res=yylex()){
    switch(res){
      case _UINT: printf("<unsigned int>"); break;
      case _SINT: printf("<signed int>"); break;
      case _REAL: printf("<real>"); break;
```

```
      case _KEYWORD: printf("<keyword>"); break;
      case _ID: printf("<ID>");
    }
    printf("[%s]\n",yytext);
  }
  printf(" %d lines processed.\n",lines);
  fclose(yyin);
  return 0;
}
```

2 Yacc : calculatrice en notation polonaise

2.1 makefile

```
ok : polish.y
        #lex calcul.lex
        yacc polish.y -o polish.cpp
        g++ -o polish polish.cpp   -lfl -lc
```

2.2 Fichier yacc : polish.y

```
/* Reverse polish notation calculator. */
%{
#define YYSTYPE double
#include <math.h>
#include <stdio.h>
int yylex();
int yyerror( char *);
%}
%token NUM
%% /* Grammar rules and actions follow */
input : /* empty */
   | input line
;
line : '\n'
   | exp '\n'  { printf ("\t%.10g\n", $1); }
;
exp : NUM             { $$ = $1;         }
   | exp exp '+'     { $$ = $1 + $2;    }
   | exp exp '-'     { $$ = $1 - $2;    }
   | exp exp '*'     { $$ = $1 * $2;    }
   | exp exp '/'     { $$ = $1 / $2;    }
   /* Exponentiation */
   | exp exp '^'     { $$ = pow ($1, $2); }
   /* Unary minus    */
  | exp 'n'         { $$ = -$1;        }
;
%%
/* Lexical analyzer returns a double floating point
   number on the stack and the token NUM, or the ASCII
   character read if not a number.  Skips all blanks
   and tabs, returns 0 for EOF. */
#include <ctype.h>
int yylex ()
{ int c;
  /* skip white space  */
  while ((c = getchar ()) == ' ' || c == '\t')
    ;
  /* process numbers   */
  if (c == '.' || isdigit (c))
    {
```

```
      ungetc (c, stdin);
      scanf ("%lf", &yylval);
      return NUM;
    }
  /* return end-of-file  */
  if (c == EOF)
    return 0;
  /* return single chars */
  return c;
}
main ()     /* The ``Main'' function to make this stand-alone  */
{
  yyparse ();
}
#include <stdio.h>
int yyerror (char *s)  /* Called by yyparse on error */
{
  printf ("%s\n", s);
}
```

Exemple d'entrée :

```
$ polish
2 1 +
       3
2 3 5 * +
       17
```

3 Calculatrice

3.1 makefile

```
ok : calcul scanner toupper preprocesser cours_lex_yacc.pdf cours_lex_yacc2.pdf

calcul : calcul.lex calcul.yacc calcul.cpp
        lex calcul.lex
        yacc calcul.yacc
        g++ -o calcul calcul.cpp   -lfl -lc
clean:
        rm -f y.tab.c lex.yy.c

scanner: scanner.lex
        lex -oscanner.c scanner.lex
        gcc -o scanner scanner.c  -lfl -lc

toupper: toupper.lex
        lex -otoupper.c toupper.lex
        gcc -o toupper toupper.c -lfl -lc

preprocesser: preprocesser.lex
        lex -opreprocesser.c preprocesser.lex
        gcc -o preprocesser preprocesser.c -lfl -lc

cours_lex_yacc.pdf: cours_lex_yacc.tex
        latex cours_lex_yacc
        dvipdf cours_lex_yacc
cours_lex_yacc2.pdf : cours_lex_yacc2.tex
        latex cours_lex_yacc2
        dvipdf cours_lex_yacc2
```

3.2 Fichier decla.h

```
#ifndef DECLA_INCLUDED

#define DECLA_INCLUDED
#define MAXNOMS 50
#define MAXLENGTH 20
struct Table
{
        char tabNom[MAXNOMS][MAXLENGTH];
        int tabVal[MAXNOMS];
        int initialized[MAXNOMS];
        int tabNb;
};

extern Table table;

void init_table();
int get_var( char *nom);
int new_var( char *nom);
void initialize( int indicevar, int value);
int get_val( int indicevar);
int yylex();;
int yyparse();;
int yyerror( char *s);

#endif
```

3.3 Fichier calcul.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include"decla.h"
/* the file y.tab.c is generated with: yacc calcul.yacc */
#include "y.tab.c"

Table table;

void init_table()
{
        table.tabNb=0;
        for( int i=0; i<MAXNOMS; i++)
        { table.initialized[i]=0;
          table.tabVal[i]= 0; }
}
int get_var( char *nom)
{
        for ( int i=1; i<=table.tabNb; i++)
                if(!strcmp( table.tabNom[i], nom)) return i;
        return 0;
}
int new_var( char *nom)
{

        table.tabNb ++ ;
        assert( table.tabNb < MAXNOMS );
        strcpy( table.tabNom[table.tabNb], nom);
        table.tabVal[table.tabNb]=0;
        table.initialized[table.tabNb]=0;
        return table.tabNb ;
}
void initialize( int indvar, int valeur)
{
```

```
        if (indvar<=0 || indvar>=MAXNOMS)
        {       printf( "initialize bug: indvar=%d\n", indvar);
        }
        table.tabVal[ indvar] = valeur;
        table.initialized[ indvar] = 1;
}
int get_val( int indvar)
{
        if (indvar<0 || indvar>=MAXNOMS)
        {       printf( "get_val bug: indvar=%d\n", indvar);
        }
        if( table.initialized[ indvar]==0)
        {       printf( "acces a une variable non initialisee: %s\n",
                table.tabNom[ indvar]); }
        return table.tabVal[ indvar];
}



main()
{
        init_table();
        yyparse();
}
```

3.4 Fichier calcul.lex

```
%{
#include "decla.h"
%}

chiffre [0-9]
lettre [a-zA-Z]

%%
quitter         { return QUIT ; }
" "|\t|\n         ;
set { return DEF ; }
{lettre}({lettre}|{chiffre})* { int var;
                                var=get_var( yytext);
                                if (!var) var=new_var( yytext);
                                yylval = var;
                                return VAR; }
[0-9]+       { yylval = atoi(yytext) ; return INT ; }
"\-"         { return MOINS ; }
"\/"         { return DIV ; }
"="          { return EGAL ; }
"+"          { return PLUS ; }
"*"          { return TIMES ; }
"\^"         { return PUISS ; }
"("          { return LPAREN ; }
")"          { return RPAREN ; }
;            { return PV ; }
.            { return ERREUR ; }
```

3.5 Fichier calcul.yacc

```
%token INT
%token FLOAT
%token PLUS TIMES
%token LPAREN RPAREN
%token QUIT ERREUR
```

```
%token DIV MOINS
%token PV
%token PUISS
%token VAR DEF EGAL

%left MOINS
%left PLUS /* + basse precedence */
%left DIV
%left TIMES /* moyenne precedence */
%nonassoc UMINUS
%right PUISS

%start top  /* point d'entree */
%%
top : expr PV
          { $$ = $1; printf("=%d\n",$$); } top
| DEF VAR EGAL expr PV
          {
            $$ = $4;
            initialize( $2, $4);
            printf("=%d\n",$$);
          }
  top
| error {printf("syntax error\n"); } top
| QUIT { return 0;}
;

expr :    INT                    { $$= $1 ;}
        | VAR                    { $$= get_val( $1);}
        | LPAREN expr RPAREN     { $$=$2 ;}
        | expr PLUS expr         { $$=$1 + $3; }
        | MOINS expr %prec UMINUS { $$= -$2; }
        | expr MOINS expr        { $$=$1 - $3; }
        | expr DIV expr          { $$=$1 / $3; }
        | expr TIMES expr        { $$=$1 * $3; }
        | expr PUISS expr        { int i;int interm=1;
                                   for(i=0;i<$3;i++)
                                     interm = interm *$1;
                                   $$=interm; }
        ;
%%
#include "lex.yy.c"
int yyerror( char *s)
{
        printf( "%s\n", s);
        return 0;
}
```

```
$ calcul
set a=1+2*3;
 = 7
set b= a^3^2;
 = 40353607
2+3*2^10;
 = 3074
2^10;
 = 1024
```

Lex et Yacc, exemple de somme ou produit de liste d'entiers

D. Michelucci

1 fichier makefile

```
ok: exemple_liste liste
liste: liste.lex liste.yacc
        lex liste.lex
        yacc -o liste.cpp liste.yacc
        g++ -o liste liste.cpp -lfl -ly -lc

exemple_liste: exemple_liste.tex
        latex exemple_liste
        dvipdf exemple_liste

clean:
        rm -fr liste.cpp  liste
        rm -fr lex.yy.c
        rm -fr *.aux *.log  *.dvi
```

2 fichier liste.lex

```
%{
#include<stdlib.h>
%}
%%
[ \t\n] { ;}
[0-9]+  { yylval= atoi( yytext); return ENTIER; }
produit { return PRODUIT  ; }
somme { return SOMME  ; }
","     {return  ','; }
"."     {return FIN; }
<<EOF>> {return FIN; }
.       {printf("caractere inconnu: [%c]\n", yytext[0]); }
%%
```

3 fichier liste.yacc

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

```
int res; /* partout, on pourrait remplacer res par $$, ou une autre valeu
extern "C"
{
        int yydebug;
        int yyparse();
        void yyerror(char*);
        int yylex();
        int yywrap() { return 1; }
};
%}
%token SOMME PRODUIT
%token ENTIER
%token FIN
%%
liste: FIN { printf("Adieu, monde cruel\n"); return 0; }
|      SOMME ls FIN { printf("somme=%d\n", $2); } liste
|      PRODUIT lp FIN  { printf("produit=%d\n", $2); } liste ;
ls:    ENTIER { $$=$1; } |   ls ',' ENTIER { $$= $1 + $3; };
lp:    ENTIER { $$=$1; } |    lp ',' ENTIER { $$= $1 * $3;};
%%
#include "lex.yy.c"
extern int yydebug;
/* int main() { yyparse(); } */
main()
{
        yydebug=1;
        yyparse();
}

/* autres regles possibles pour ls et lp :
ls: { $$=0;} |  ls ',' ENTIER { $$= $1 + $3; };
lp: { $$=1;} |  lp ',' ENTIER { $$= $1 * $3; };
elles admettent:
somme .
produit .
*/
```

4 Session

produit 1,2,3,4. somme 1,2,3,4..

Lex et Yacc, exemple 2 de somme ou produit de liste d'entiers

D. Michelucci

1 fichier makefile

```
ok: exemple_liste2.pdf liste2
liste2: liste2.lex liste2.yacc makefile
        lex liste2.lex
        yacc -o liste2.cpp liste2.yacc
        g++ -o liste2 liste2.cpp -lfl -ly -lc

exemple_liste2.pdf : exemple_liste2.tex  liste2.lex liste2.yacc makefile
        latex exemple_liste2
        dvipdf exemple_liste2

clean:
        rm -fr liste2.cpp  liste2
        rm -fr lex.yy.c
        rm -fr *.aux *.log  *.dvi
```

2 fichier liste2.lex

```
%{
#include<stdlib.h>
%}
%%
[ \t\n] { ;}
[0-9]+  { yylval= atoi( yytext); return ENTIER; }
produit { return PRODUIT  ; }
somme { return SOMME  ; }
","     {return  ','; }
"."     {return FIN; }
"("     { return LP; }
")"     { return RP; }
<<EOF>> {return FIN; }
.       {printf("caractere inconnu: [%c]\n", yytext[0]); }
%%
```

3 fichier liste2.yacc

```
%{
#include<stdio.h>
#include<stdlib.h>
int res; /* partout, on pourrait remplacer res par $$, ou une autre valeu
extern "C"
{
        int yydebug;
        int yyparse();
        void yyerror(char*);
        int yylex();
        int yywrap() { return 1; }
};
%}
%token SOMME PRODUIT
%token ENTIER
%token FIN
%token LP RP
%%

programme: FIN { printf("Adieu, monde cruel\n"); return 0; }
| expr FIN { printf("resultat=%d\n", $1); } programme ;
expr: ENTIER { $$=$1; }
|     SOMME ls { $$=$2;}
|     PRODUIT lp { $$=$2;}
|     LP expr RP { $$=$2; } ;
ls: expr { $$=$1; }
|   expr ',' ls {$$=$1+$3;} ;
lp: expr { $$=$1;}
|   expr ',' lp {$$=$1*$3;} ;


%%
#include "lex.yy.c"
extern int yydebug;
/* int main() { yyparse(); } */
main()
{
        yydebug=1;
        yyparse();
}

/* autres regles possibles pour ls et lp :
ls: { $$=0;} |  ls ',' ENTIER { $$= $1 + $3; };
lp: { $$=1;} |  lp ',' ENTIER { $$= $1 * $3; };
elles admettent:
somme .
produit .
*/
```

4 Session

```
produit 1,2,3,4.
somme 1,2,3,4.
produit 2, (somme 1, 2, 3), 4.
.
```

Lex et Yacc, exemple $a^n b^n \$$

1 fichier makefile

```
ok: anbn exemple_anbn
anbn: anbn.y
	yacc -o anbn.cpp anbn.y
	#gcc anbn.cpp -o anbn  -lc -lm -lstdc++
	g++  anbn.cpp -o anbn  -lc -lm
exemple_anbn: exemple_anbn.tex
	latex exemple_anbn
	dvipdf exemple_anbn
clean:
	rm -fr *.log *.aux *.dvi
	rm -fr anbn.cpp  anbn
```

2 fichier anbn.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex(); void yyerror(char*s);
int n=0; /* this program recognizes words: a^nb^n$ : $ ab$ aabb$ aaabbb$
*/
%}
%%
input: | mot input ;
mot: S '$' { printf("Le mot a^nb^n$ a ete accepte, n=%d\n", n); } ;
S : { n=0; } | 'a' S 'b'  { n++; } ;
%%
int yylex() { char c; for( c=getchar() ; c==' '|| c=='\t' || c=='\n'; ) c
	if (c==EOF) exit(1);
	if (c=='a' || c=='b' || c=='$') return c;
	else printf("erreur, caractere non reconnu: [%c]\n", c); }
void yyerror(char *s) { printf("errorr %s\n", s); }
int main() { yyparse(); }
```

3 Session

```
$ ab$ aabb$ aaabbb$
```

Lex et Yacc, exemples introductifs

D. Michelucci

1 Compter le nombre de lignes, mots

Compilation par :

```
$ lex -oscanner.c scanner.lex
$ gcc -o scanner scanner.c  -lfl -lc
```

```
%{
    int nchar, nword, nline;
/*
Here is a
scanner that counts the number of characters,
words, and lines in a file (similar to Unix wc):
*/
%}
%%
\n          { nline++; nchar++; }
[^ \t\n]+   { nword++, nchar += yyleng; }
.           { nchar++; }
%%
int main(void) {
    yylex();
    printf("%d\t%d\t%d\n", nchar, nword, nline);
    return 0;
}
```

2 Tout mettre en majuscule : toupper.lex

Compilation par :

```
      $ lex -otoupper.c toupper.lex
$ gcc -o toupper toupper.c -lfl -lc
```

```
%{
#include <ctype.h>
/*
 Tout mettre en majuscule
*/
%}
%%
.    {printf( "%c", toupper( *yytext)); }
```

```
%%
/*
int main(void) {
    yylex();
    printf("%d\t%d\t%d\n", nchar, nword, nline);
    return 0;
}
*/
```

3 Un début de préprocesseur : preprocesser.lex

Compilation par :

```
$ lex -opreprocesser.c preprocesser.lex
$ gcc -o preprocesser preprocesser.c -lfl -lc
```

```
%{
    int nchar, nword, nline;
/*
un debut de preprocesseur
il se contente d'enlever les commentaires // blabla en debut
   de ligne
et d'inclure les fichiers
*/
%}
%x READFILENAME
nom [^"]*
%%
^"//".* { ; /* skip comments until end of line */ }
<INITIAL>^#include\"      {BEGIN(READFILENAME); }
<READFILENAME>{nom}\"\n  {
  BEGIN(INITIAL);
  { char namefile[1000]; int c; FILE *fd;
    sprintf( namefile, "%s", yytext);
    for (c=0; namefile[c]!='"'; c++);
    namefile[c]= '\0';
    printf( "NOM FICHIER=[%s]\n", namefile);
    fd= fopen( namefile , "r");
    if (!fd) fprintf(stderr, "file not found: %s\n", namefile)
        ;
    else { for (c=fgetc( fd); c != EOF; c=fgetc( fd)) printf
       ("%c", c);
           fclose( fd);
         }
  }}
%%
```

Lex et Yacc, exemple du miroir

1 fichier makefile

```
ok: miroir exemple_miroir
miroir: miroir.y
	yacc -o miroir.cpp miroir.y
	#gcc miroir.cpp -o miroir  -lc -lm -lstdc++
	g++  miroir.cpp -o miroir  -lc -lm
exemple_miroir: exemple_miroir.tex
	latex exemple_miroir
	dvipdf exemple_miroir
clean:
	rm -fr miroir.cpp miroir
	rm -fr *.log *.dvi *.aux
```

2 fichier miroir.y

```
%{
#include<stdio.h>
#include<stdlib.h>
int yylex();
void yyerror(char*s);
%}
%%
input: | mot input ;
mot: S '$' { printf("Le mot a ete accepte\n"); } ;
S : 'a' S 'a'  | 'b' S 'b' | 'c' ;
%%
int yylex() {
	char c;
	for( c=getchar() ; c==' '|| c=='\t' || c=='\n'; ) c=getchar();
	if (c==EOF) exit(1);
	if (c=='a' || c=='b' || c=='c' || c=='$') return c;
	else printf("erreur, caractere non reconnu: [%c]\n", c);
}
void yyerror(char *s) { printf("errorr %s\n", s); }
int main() { yyparse(); }
```

3 Session

```
c$ aca$ bcb$ abcba$ bbcbb$ ababcbaba$
```

LEX YACC, AFFICHAGE DE COURBES $f(x, y) = 0$

FICHIER DE DONNÉES : *grapher data9*

```
func: ((x + 2)^2 + y^2) * ((x-2)^2 + y^2) - 16 ;
x0:-3; y0:-3; x1:3; y1:3; grid: no;nivrec: 9;draw; #Agnesi
```

FICHIER makefile :

```
ok : grapher
LIB_PATHS= -L/usr/X11R6/lib
LIBS=-lGLUT -lGLU -lGL -lc -lm -L/usr/X11/lib -lXext -lXmu -lXi -lX11 -lstdc++
CompileGL= g++ \
-Wno-deprecated \
-I/Developer/SDKs/MacOSX10.4u.sdk/usr/X11R6/include \
-I/Developer/SDKs/MacOSX10.4u.sdk/usr/X11R6/include/GL \
-I/Developer/Examples/OpenGL/GLUT/ \
-I/Developer/SDKs/MacOSX10.4u.sdk/System/Library/Frameworks/GLUT.framework/
    Versions/A/Headers \
-framework OpenGL -framework GLUT -framework Foundation -lc -lm -lstdc++
grapher : grapher.h grapher.lex grapher.yacc grapher.cpp graphic.cpp
        lex grapher.lex ; yacc grapher.yacc
        $(CompileGL) -lfl -lc -o grapher graphic.cpp grapher.cpp
clean:
        rm -f y.tab.c lex.yy.c grapher
```

FICHIER grapher.h :

```
#ifndef DECLA_INCLUDED
#define DECLA_INCLUDED
enum Kind {EXPR_X, EXPR_Y, EXPR_ADD, EXPR_MULT, EXPR_POW, EXPR_SUB,
   EXPR_DOUBLE, EXPR_INT};
struct Expression { Kind kind; Expression *fg; Expression *fd;
  double number; int entier; /* pour x^2 par exemple */ };
struct Interval { double min, max; };
struct Grapher { Expression *function;
  double x0, x1, y0, y1; /* domaine d'afficahge de la courbe */
  int niv_recursion; int grid; /* affiche t on les carres consideres ? */
     };
int yylex(),  yyparse(), yyerror( char *s);
Expression *new_exp( Kind kin, Expression *fg, Expression *fd);
Expression *new_expr_num( double x), *new_expr_int( int x);
void draw( Grapher &grapher);
extern Expression *expr_x, *expr_y;
extern Grapher grapher;
extern FILE* yyin;
extern int argc; extern char**argv;
void clavier(unsigned char touche,int x,int y),
        mouse( int button, int state,int x,int y),
        reshape(int x,int y),
        affichage(),
        affiche_carre( double x0, double x1, double y0, double y1, int
           remplir),
        init_graphic( int argc, char**argv);
extern int width, height, debug;
#endif
```

FICHIER grapher.lex :

```
%{
#include "grapher.h"
%}
chiffre [0-9]
lettre [a-zA-Z]
%%
" "|\t|\n     {/* sauter les espaces */}
"x0:" { if (debug) fprintf(stderr,"lecture de x0\n") ; return X0 ; }
"x1:" { if (debug) fprintf(stderr,"lecture de x1\n") ; return X1 ; }
"y0:" { if (debug) fprintf(stderr,"lecture de y0\n") ;return Y0 ; }
"y1:" { if (debug) fprintf(stderr,"lecture de y1\n") ;return Y1 ; }
"x" { if (debug) fprintf(stderr,"lecture de x\n"); return X; }
"y" { if (debug) fprintf(stderr,"lecture de y\n"); return Y; }
"grid:" { if (debug) fprintf(stderr,"lecture de grid\n"); return GRID;}
"yes" { if (debug) fprintf(stderr,"lecture de yes\n"); return YES;}
"no"  { if (debug) fprintf(stderr,"lecture de no\n"); return NO;}
"nivrec:"  { return NIVREC;}
"func:" { if (debug) fprintf(stderr,"lecture de func\n");  return FUNC;}
"draw"  { if (debug) fprintf(stderr,"lecture de draw\n"); return DRAW;}
[0-9]+"\."[0-9]* { yylval.double_val = atof(yytext);
    if (debug) fprintf(stderr,"lecture de nombre double=%lf\n",yylval.
       double_val) ;
    return DOUBLE ; }
[0-9]+ { yylval.int_val = atoi(yytext) ;
         if (debug) fprintf(stderr,"lecture de nombre entier: %d\n", yylval
            .int_val);
         return INT ; }
"\-" { if (debug) fprintf(stderr,"lecture de MOINS\n");  return MOINS ; }
"+"  { if (debug) fprintf(stderr,"lecture de ADD\n"); return PLUS ; }
"*"  { if (debug) fprintf(stderr,"lecture de MULT\n"); return TIMES ; }
"\^" { if (debug) fprintf(stderr,"lecture de ^\n"); return PUISS ; }
"("  { if (debug) fprintf(stderr,"lecture de (\n"); return LPAREN ; }
")"  { if (debug) fprintf(stderr,"lecture de )\n"); return RPAREN ; }
;    { if (debug) fprintf(stderr,"lecture de PV\n"); return PV ; }
.    { if (debug) fprintf(stderr,"LEX: caractere inattendu:%c\n", *yytext);
   return ERREUR ; }
```

FICHIER grapher.yacc :

```
%{
#include"grapher.h"
%}
%union { double double_val; int int_val; Expression * expr_val; }
%token <double_val> DOUBLE
%token <int_val> INT
%type <expr_val> expr
%type <double_val> nb
%token PLUS TIMES
%token LPAREN RPAREN
%token ERREUR
%token MOINS
%token PV
%token PUISS
%token X Y X0 X1 Y0 Y1 FUNC DRAW GRID YES NO NIVREC
%left MOINS
%left PLUS /* + basse precedence */
```

```
%left DIV
%left TIMES /* moyenne precedence */
%right PUISS
%start top  /* point d'entree */
%%
top: | top instruction {}
instruction: FUNC expr PV { grapher.function=$2; if (debug) printf("
    fonction ok\n");}
| X0   nb  PV { grapher.x0= $2;  if (debug) printf("recu: x0=%lf\n", $2);
    }
| Y0   nb  PV { grapher.y0= $2;  if (debug) printf("recu: x1=%lf\n", $2); }
| X1   nb  PV { grapher.x1= $2;  if (debug) printf("recu: y0=%lf\n", $2); }
| Y1   nb  PV { grapher.y1= $2;  if (debug) printf("recu: y1=%lf\n", $2); }
| NIVREC INT PV { grapher.niv_recursion= $2; }
| GRID YES  PV { grapher.grid= 1; }
| GRID NO   PV { grapher.grid= 0; }
| error {printf("syntax error\n"); }
| DRAW PV { return 0;} ;
nb: INT { $$=(double) $1; }
|   MOINS INT { $$= 0. - (double) $2; }
|   DOUBLE { $$=$1;}
|   MOINS DOUBLE { $$= 0. - $2; }
expr: DOUBLE { $$ = new_expr_num( $1) ;};
expr:  INT   { $$ = new_expr_int( $1);};
expr:  X     { $$ = expr_x; };
expr:  Y     { $$ = expr_y; };
expr: LPAREN expr RPAREN    { $$ = $2 ;}
expr: expr PLUS expr { $$ =new_exp( EXPR_ADD, $1 , $3 ); };
expr: MOINS expr     { $$ =new_exp( EXPR_SUB, new_expr_num(0.), $2 ); };
expr: expr MOINS expr{ $$ =new_exp( EXPR_SUB, $1, $3 ); };
expr: expr TIMES expr{ $$ =new_exp( EXPR_MULT, $1 , $3); };
expr: expr PUISS INT { $$ =new_exp( EXPR_POW, $1, new_expr_int( $3)); };
%%
#include "lex.yy.c"
int yyerror( char *s) { printf( "%s\n", s); return 0; }
```

FICHIER grapher.cpp :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include"grapher.h"
#include<iostream.h>
#include<math.h>
#include<glut.h>
/* the file y.tab.c is generated with: yacc calcul.yacc */
#include "y.tab.c"
#ifdef DEBUG
int debug=1;
#else
int debug=0;
#endif
Grapher grapher;
Expression *expr_x, *expr_y;
int argc; char**argv;
```

```
Expression* new_exp( Kind kin, Expression* g, Expression* d)
{    Expression* expr= new Expression;
     expr->kind= kin; expr->fg= g; expr->fd= d; return expr;}
Expression* new_expr_num( double x)
{   Expression* expr= new Expression; expr->kind= EXPR_DOUBLE;
    expr->number=x; return expr;}
Expression* new_expr_int( int x)
{   Expression* expr= new Expression; expr->kind= EXPR_INT;
    expr->entier=x; return expr;}
double min( double x, double y) { if (x<y) return x; else return y;}
double max( double x, double y) { if (x<y) return y; else return x;}
Interval interval( double a, double b)
{    assert( a <= b); Interval ab; ab.min=a; ab.max=b; return ab;}
Interval plus_interval( Interval a, Interval b)
{     return interval( a.min + b.min, a.max + b.max);}
Interval moins_interval( Interval a, Interval b)
{     return interval( a.min - b.max, a.max - b.min);}
Interval mult_interval( Interval a, Interval b)
{     double a0b0= a.min * b.min; double a0b1= a.min * b.max;
      double a1b0= a.max * b.min; double a1b1= a.max * b.max;
      return interval( min( min( a0b0, a0b1), min( a1b0, a1b1)),
                  max( max( a0b0, a0b1), max( a1b0, a1b1)));}
Interval pow_interval( Interval x, int n)
{   if (n==0) return interval( 1., 1.);
    if (n==1) return x;
    if (n%2 == 1)  /* c'est monotone croissant */
        { double x0n=x.min, x1n= x.max;
          for( int i=1; i<n; i++) { x0n *= x.min; x1n *= x.max;}
          return interval( x0n, x1n);}
    else    return mult_interval( x, pow_interval( x, n-1));}
    // bon, il y a plus rapide, mais ca marche...
Interval evaluer_expr( Expression* fonction, Interval x, Interval y)
{     switch( fonction->kind) {
    case EXPR_INT: return interval( double(fonction->entier), double(
       fonction->entier)); break;
    case EXPR_DOUBLE: return interval( fonction->number, fonction->number);
        break;
    case EXPR_ADD: { Interval g, d; g=evaluer_expr( fonction->fg, x, y);
        d=evaluer_expr( fonction->fd, x, y); return plus_interval( g,d);
           break;}
    case EXPR_SUB: { Interval g, d; g=evaluer_expr( fonction->fg, x, y);
        d=evaluer_expr( fonction->fd, x, y); return moins_interval( g, d);
           break;}
    case EXPR_MULT: { Interval g, d; g=evaluer_expr( fonction->fg, x, y);
        d=evaluer_expr( fonction->fd, x, y); return mult_interval( g, d);
           break;}
    case EXPR_X: return x; break;
    case EXPR_Y: return y; break;
    case EXPR_POW: { Interval g=evaluer_expr( fonction->fg, x, y);
        assert( fonction->fd->kind == EXPR_INT); int n=fonction->fd->entier
           ;
        return pow_interval( g, n); break;}
    default: printf("oups?\n"); exit(0); break;}}
int peut_etre_nul( Interval x) { return x.min <=0 && x.max >= 0.;}

void considere( double x0, double x1, double y0, double y1, int niv,
```

```
    Grapher &g)
{   Interval valf= evaluer_expr( g.function, interval(x0,x1), interval(y0,
    y1) );
    if (g.grid) affiche_carre( x0, x1, y0, y1, 0);
    if( peut_etre_nul( valf))
    { if( niv==0) { affiche_carre( x0, x1, y0, y1, 1);}
        else {  double xm= (x0+x1)/2.; double ym= (y0+y1)/2.;
            considere( x0,xm, y0,ym, niv-1, g);
            considere( x0,xm, ym,y1, niv-1, g);
            considere( xm,x1, y0,ym, niv-1, g);
            considere( xm,x1, ym,y1, niv-1, g);}}}

void draw( Grapher &g) { considere( g.x0, g.x1, g.y0, g.y1, g.niv_recursion
    , g);}

int main(int argc_, char**argv_)
{     argc= argc_; argv= argv_;
    expr_x= new Expression; expr_x->kind= EXPR_X;
    expr_y= new Expression; expr_y->kind= EXPR_Y;
    grapher.niv_recursion = 8;
    grapher.x0= -1.; grapher.x1 = 1.; grapher.y0= -1.; grapher.y1= 1.;
        grapher.grid=1;
    grapher.function = new_exp( EXPR_ADD,
             new_exp( EXPR_POW, expr_x, new_expr_int(2)),
             new_exp( EXPR_ADD, new_exp( EXPR_POW, expr_y, new_expr_int(2))
                 ,
                                new_expr_num( -1.)));
    if (argc==2) yyin= fopen( argv[1], "r"); else yyin= fopen( "data1", "r
        ");
    if (yyin<=0) { fprintf( stderr, "no entry file?\n"); exit(1);}
    yyparse(); init_graphic( argc, argv); return 0;}
```

FICHIER graphic.cpp :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include"grapher.h"
#include<iostream.h>
#include<math.h>
#include<glut.h>

int width=600; int height=600;
void idle() { glutPostRedisplay();}
void init_graphic( int argc, char**argv)
{ /* initialisation de glut et creation de la fenetre */
        glutInit( &argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowPosition(200,200);
        glutInitWindowSize(width,height);
        glutCreateWindow("visu");
        /* Initialisation d'OpenGL */
        glClearColor(0.0,0.0,0.0,1.0);
        glColor3f(1.,1.,1.); glPointSize(2.0); glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING); float sun0[]={5.,6.,10.,1.};
        glEnable(GL_LIGHT0);
        glLightModeli( GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
        glLightfv(GL_LIGHT0, GL_POSITION, sun0);
        GLfloat ambient[] = {0.4, 0.4, 0.4, 1.};
        GLfloat diffuse[] = {1.0, 1.0, 1.0, 1.0};
        GLfloat specular[] = {1., 1., 1., 1.};
```

```
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
        /* enregistrement des fonctions de rappel */
        glutDisplayFunc(affichage); glutKeyboardFunc(clavier);
        glutMouseFunc( mouse); glutReshapeFunc(reshape);
        glutIdleFunc(idle); glutMainLoop();}
void affichage()
{       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION); glLoadIdentity();
        glOrtho( grapher.x0, grapher.x1, grapher.y0, grapher.y1, -1., 1.);
        glPointSize(2.0); glDisable(GL_DEPTH_TEST);
        glDisable(GL_LIGHTING); glClear(GL_COLOR_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW); glLoadIdentity();
        draw( grapher ); glutSwapBuffers(); glFlush(); glFinish();}
void mouse( int button, int state,int x,int y)
{    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    { /* changer le centre de la fenetre */
        double L= glutGet( GLUT_WINDOW_WIDTH);
        double H= glutGet( GLUT_WINDOW_HEIGHT);
        double dx = ((double)x-L/2.) * (grapher.x1-grapher.x0)/L ;
        double dy = - ((double)y-H/2.) * (grapher.y1-grapher.y0)/H;
        grapher.x0 += dx; grapher.x1 += dx;
        grapher.y0 += dy; grapher.y1 += dy;}
        else if (button == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
        { /* zoom */
        double dx=grapher.x1-grapher.x0;
        double dy=grapher.y1-grapher.y0;
        grapher.x0 += dx/4.; grapher.x1 -= dx/4.;
        grapher.y0 += dy/4.; grapher.y1 -= dy/4.;}
        else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        { /* s'eloigner */
        double dx=grapher.x1-grapher.x0;
        double dy=grapher.y1-grapher.y0;
        grapher.x0 -= dx; grapher.x1 += dx;
        grapher.y0 -= dy; grapher.y1 += dy;}
        glutPostRedisplay();}
void clavier(unsigned char touche,int x,int y)
{ static int num_fonction=0; switch (touche) {
    case 'r' : glutPostRedisplay(); break;
    case 'q' : /*la touche 'q' permet de quitter le programme */ exit(0); break;
    default: if (argc==2) yyin= fopen( argv[1], "r" ); else yyin= fopen( "data1", "r");
         if (yyin>0) { fprintf( stderr, "*** relecture du fichier de donnees ****\n");
               fflush( stderr); yyparse(); glutPostRedisplay();}
            else       { fprintf( stderr, "*** probleme de fichier ***\n");}
            break;}}
void reshape(int x,int y) { glViewport( 0,0, x, y); glutPostRedisplay();}
void affiche_carre( double x0, double x1, double y0, double y1, int remplir)
{ if (remplir) { glEnable( GL_LIGHTING); glColor3f(1.,0.,0.); glBegin( GL_POLYGON);
        glVertex2d( x0, y0); glVertex2d( x1, y0); glVertex2d( x1, y1);
        glVertex2d( x0, y1); glEnd( );}
    else { glDisable( GL_LIGHTING); glColor3f( 1., 1., 0.); glBegin( GL_LINE_LOOP);
        glVertex2d( x0, y0); glVertex2d( x1, y0); glVertex2d( x1, y1);
        glVertex2d( x0, y1); glEnd( );}}
```