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
```