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); }
```