# Utilisation de lex et yacc : la calculette

4 décembre 2005

lex et yacc sont utilisées pour générer les fonctions d'analyse lexicale et syntaxique de l'entrée, et programmer une calculette. Le programmeur doit écrire decla.h, makefile, calcul.lex, calcul.yacc, calcul.cpp. L'utilisateur lance le programme et tape par exemple :

```
set a= 2*3+1;
a*10;
quitter
```

## 1 decla.h

```
#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);
```

## 2 Makefile

```
ok : calcul.lex calcul.yacc calcul.cpp
    lex calcul.lex
    yacc calcul.yacc
    g++ -o calcul calcul.cpp  -ll
    #cc calcul.c -ly -ll
```

## 3 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 ; }
```

## 4 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 */
%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         { $$=-$2; }
    | expr MOINS expr     { $$=$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;
}
```

# 5 calcul.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include "y.tab.c"

Table table;

void init_table()
{
    table.tabNb=0;
    for( int i=0; i<MAXNOMS; i++)
    { table.initialized[table.tabNb]=0;
```

```
      table.tabVal[table.tabNb]= 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();
}
```