Calculette

Dans cette calculette, la priorité entre les opérations ($+, -, *, /$, $\hat{}$, moins unaire) et leur associativité sont gérées par des règles de grammaire dans le fichier yacc, et non par %left, %right.

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
```

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 FLOAT PLUS TIMES LPAREN RPAREN QUIT ERREUR DIV MOINS PV PUISS VAR DEF EGAL
%start top  /* point d'entree */
%%
top : expr PV { $$ = $1; printf("expr = %d \n",$$); } top
| DEF VAR EGAL expr PV { $$ = $4; initialize( $2, $4); printf(" = %d \n",$$); } top
| error {printf("syntax error\n"); } top
| QUIT { return 0;} ;
expr :    sum { $$= $1 ;}
atome : LPAREN expr RPAREN { $$ = $2; } | INT { $$ = $1;}
| VAR { $$ = get_val( $1); } | FLOAT { $$ = $1; } ;
sum: factor { $$=$1; } | sum PLUS factor { $$=$1+$3; } | sum MOINS factor { $$=$1-$3; } ;
factor: puiss { $$=$1; } | factor TIMES puiss { $$=$1*$3;} | factor DIV puiss { $$=$1/$3;}
puiss : puissance { $$=$1;}  | MOINS puissance { $$= 0 - $2; } ;
puissance: atome { $$=$1; }
| atome PUISS puissance { float initial =1; int i;
  for( i = 0 ; i < $3; i++) initial = initial * $1;
  $$ = initial; } ;
%%
#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 variable non initialisee: %s\n", table.tabNom[ indvar]);
     return table.tabVal[ indvar]; }
main() {       init_table(); yyparse(); }
```