Lex Yacc, affichage de courbes $f(x, y) = 0$

D. Michelucci

Avril 2012

Cette version améliorée du grapheur fournit les unions, intersections et différences entre objets, ainsi que des transformations affines : translations, affinités (changement d'échelle), rotations.

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: 10;
draw ;
```

\# Agnesi

FICHIER DE DONNÉES : *grapher data21*

```
func: union( x^2+y^2-1, (x-1.5)^2+y^2-1)   ;  x0: -3 ; y0: -3; x1: 3; y1: 3; grid
    : yes; nivrec: 10; draw ;
```

FICHIER DE DONNÉES : *grapher data22*

```
func: inter( x^2+y^2-1, (x-1.5)^2+y^2-1)   ;  x0: -3 ; y0: -3; x1: 3; y1: 3; grid
    : yes; nivrec: 10; draw ;
```

FICHIER DE DONNÉES : *grapher data23*

```
func: differ( x^2+y^2-1, (x-1.5)^2+y^2-1)   ;  x0: -3 ; y0: -3; x1: 3; y1: 3;
    grid: yes; nivrec: 10; draw ;
```

FICHIER DE DONNÉES : *grapher data24*

```
func: union( trans(2, 0, differ( x^2+y^2-1, (x-1.5)^2+y^2-1)),
             differ( x^2+y^2-1, (x-1.5)^2+y^2-1)
           )   ;  x0: -3 ; y0: -3; x1: 3; y1: 3; grid: yes; nivrec: 10; draw ;
```

FICHIER DE DONNÉES : *grapher data25*

```
func: union( rota( 30, trans(2, 0, differ( x^2+y^2-1, (x-1.5)^2+y^2-1))),
             differ( x^2+y^2-1, (x-1.5)^2+y^2-1)
           )   ;  x0: -3 ; y0: -3; x1: 3; y1: 3; grid: yes; nivrec: 10; draw ;
```

FICHIER DE DONNÉES : *grapher data26*

```
func: union( trans(2, 0, scale( 0.5, 0.5, differ( x^2+y^2-1, (x-1.5)^2+y^2-1))),
             differ( x^2+y^2-1, (x-1.5)^2+y^2-1)
           )   ;  x0: -3 ; y0: -3; x1: 3; y1: 3; grid: yes; nivrec: 10; draw ;
```

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.lex grapher.yacc grapher.cpp
        lex grapher.lex
        yacc grapher.yacc
        $(CompileGL) -o grapher grapher.cpp   -lfl -lc
clean:
        rm -f y.tab.c lex.yy.c
```

FICHIER makefile_linux :

```
ok : grapher
LIB_PATHS= -L/usr/X11R6/lib
LIBS=-lglut -lGLU -lGL -lc -lm -L/usr/X11/lib -lXext -lXmu -lXi -lX11 -lstdc++



grapher : grapher.h grapher.lex grapher.yacc grapher.cpp #graphic.cpp
        lex grapher.lex
        yacc grapher.yacc
        #cc graphic.cpp grapher.cpp -o grapher -Wno-deprecated \
        cc -DDEBUG  grapher.cpp -o grapher -Wno-deprecated \
        -I/usr/local/Mesa-5.0.2/include/GL \
        -L/usr/local/Mesa-5.0.2/lib ${LIBS} ${LIB_PATHS}\
        -lglut -lc  -lfl \
        -lc -lm -lstdc++
        echo "export LD_LIBRARY_PATH=/usr/local/Mesa-5.0.2/lib/"

clean:
        rm -f y.tab.c lex.yy.c grapher
```

FICHIER grapher.h :

```
#ifndef DECLA_INCLUDED

#define DECLA_INCLUDED
enum Kind {EXPR_TRANS, EXPR_SCALE, EXPR_ROTA, EXPR_UNION, EXPR_INTER,
    EXPR_DIFFER, 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;
                    double u, v; /* translation(u,v,fg), rotation( u degres
                        , fg), scale (ux, uy, fg) */
                    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 contours des carres
                    consideres ? */
                };
int yylex();;
int yyparse();;
int yyerror( char *s);
Expression *new_exp( Kind kin, Expression *fg, Expression *fd);
Expression *new_exp_trans( double, double,  Expression *);
Expression *new_exp_scale( double, double,  Expression *);
Expression *new_exp_rota( double,  Expression *);
Expression *new_expr_num( double x);
Expression *new_expr_int( int x);

void draw( Grapher &grapher);

extern Expression *expr_x;
extern Expression *expr_y;
extern Grapher grapher;
extern int debug;
#endif
```

FICHIER grapher.lex :

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

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

%%
" "|\t|\n     {/* sauter les espaces */}
","   { if (debug) fprintf(stderr,"lecture de ,\n") ; return VIRGULE; }
"inter"   { if (debug) fprintf(stderr,"lecture de inter\n") ; return INTER;
    }
"union"   { if (debug) fprintf(stderr,"lecture de union\n") ; return UNION;
    }
"differ"   { if (debug) fprintf(stderr,"lecture de differ\n") ; return
   DIFFER; }
"trans"    { if (debug) fprintf(stderr,"lecture de trans\n") ; return TRANS
   ; }
"rota"    { if (debug) fprintf(stderr,"lecture de rota\n") ; return ROTA; }
"scale"    { if (debug) fprintf(stderr,"lecture de scale\n") ; return SCALE
   ; }
"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 INTER UNION DIFFER SCALE ROTA TRANS
%token PLUS TIMES
%token LPAREN RPAREN VIRGULE
%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)); };
expr: UNION LPAREN expr VIRGULE expr RPAREN { $$= new_exp( EXPR_UNION, $3,
    $5); };
expr: INTER LPAREN expr VIRGULE expr RPAREN { $$= new_exp( EXPR_INTER, $3,
    $5); };
expr: DIFFER LPAREN expr VIRGULE expr RPAREN { $$= new_exp( EXPR_DIFFER, $3
    , $5); };
expr: TRANS LPAREN nb VIRGULE nb VIRGULE expr RPAREN { $$= new_exp_trans(
    $3, $5, $7); }
expr: ROTA LPAREN nb VIRGULE expr RPAREN { $$= new_exp_rota( $3, $5); }
expr: SCALE LPAREN nb VIRGULE nb VIRGULE expr RPAREN { $$= new_exp_scale(
    $3, $5, $7); }


%%
#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<iostream>
#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

double pi= 4. * atan( 1.) ;

Grapher grapher;
Expression *expr_x, *expr_y;

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_exp_trans( double tx, double ty, Expression* g)
{       Expression* expr= new Expression;
        expr->kind= EXPR_TRANS; expr->fg= g; expr->u = tx; expr->v = ty;
           return expr;
}
Expression* new_exp_scale( double kx, double ky, Expression* g)
{       Expression* expr= new Expression;
        expr->kind= EXPR_SCALE; expr->fg= g; expr->u = kx; expr->v = ky;
           return expr;
}
Expression* new_exp_rota( double thetadeg, Expression* g)
{
        Expression* expr= new Expression;
        expr->kind= EXPR_ROTA; expr->fg= g; expr->u = thetadeg ; 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 oppose_interval( Interval a) { return interval( 0. - a.max, 0. - a
   .min ); }
Interval max_interval( Interval a, Interval b) { return interval( max( a.
   min, b.min), max( a.max, b.max)); }
Interval min_interval( Interval a, Interval b) { return interval( min( a.
   min, b.min), min( a.max, b.max)); }
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_UNION : { Interval g, d;
                g=evaluer_expr( fonction->fg, x, y);
                d=evaluer_expr( fonction->fd, x, y);
                return min_interval( g, d) ;
                break; }
        case EXPR_INTER : { Interval g, d;
                g=evaluer_expr( fonction->fg, x, y);
                d=evaluer_expr( fonction->fd, x, y);
                return max_interval( g, d) ;
                break; }
        case EXPR_DIFFER : { Interval g, d;
                g=evaluer_expr( fonction->fg, x, y);
```

```
                d=evaluer_expr( fonction->fd, x, y);
                return max_interval( g, oppose_interval( d)) ;
                break; }
        case EXPR_TRANS : { Interval g;
                g=evaluer_expr( fonction->fg, moins_interval( x, interval(
                    fonction->u, fonction->u)), moins_interval( y, interval(
                    fonction->v, fonction->v)));
                return g;
                break; }
        case EXPR_SCALE : {
                double invu= 1. / fonction->u; double invv= 1. / fonction->
                    v;
                Interval uu= interval( invu, invu); Interval vv= interval(
                    invv, invv);
                return evaluer_expr( fonction->fg, mult_interval( x, uu),
                    mult_interval( y, vv));
                break;
                }
        case EXPR_ROTA : { double theta = (0. - fonction->u) / 180. * pi ;
                           double co = cos ( theta); double si = sin( theta
                               );
                           Interval isi = interval( si, si); Interval ico=
                               interval( co, co);
                           Interval x2= moins_interval( mult_interval( x,
                               ico), mult_interval( y, isi));
                           Interval y2= plus_interval( mult_interval(x, isi
                               ), mult_interval(y, ico));
                           return evaluer_expr( fonction->fg, x2, y2);
                           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 affichage( );
void clavier(unsigned char touche,int x,int y);
void mouse( int button, int state,int x,int y);
void reshape(int x,int y);
void usage( int argc, char **argv);
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);


        /* Entree dans la boucle principale glut */
        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: 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
   );


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( );
        }
}

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)
{
        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);
//      draw( grapher);
        return 0;
}
```