```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <list>
#include <vector>
#include "glut.h"

using namespace std;

#define dimensiuneFereastra 600
#define NO_LINII_DEFAULT 15
#define NO_COLOANE_DEFAULT 15

unsigned char prevKey;

class Punct{
private:
    int X;
    int Y;
public:
    Punct(int x, int y){
        this->X = x;
        this->Y = y;
    }
    void setX(int x){
        this->X = x;
    }
    void setY(int y){
        this->Y = y;
    }
    int getX(){
        return this->X;
    }
    int getY(){
        return this->Y;
    }
};
list<Punct*> MPixeliDreapta;
list<Punct*> MPixeliCerc;
list<Punct*> MPixeliElipsa;
class Cerc{
private:
    Punct* centru;
    int raza;
public:
    Cerc(Punct* pCentru, int pRaza){
        this->centru = pCentru;
        this->raza = pRaza;
    }
    Punct* getCentru() const { return centru; }
    void setCentru(Punct* val) { centru = val; }
    int getRaza() const { return raza; }
```

```
    void setRaza(int val) { raza = val; }
};
class Elipsa{
private:
    Punct* centru;
    int raza1;
    int raza2;
public:
    Elipsa(Punct* pCentru, int pRaza1, int pRaza2){
        this->centru = pCentru;
        this->raza1 = pRaza1;
        this->raza2 = pRaza2;
    }
    Punct* getCentru() const { return centru; }
    void setCentru(Punct* val) { centru = val; }
    int getRaza1() const { return raza1; }
    void setRaza1(int val) { raza1 = val; }
    int getRaza2() const { return raza2; }
    void setRaza2(int val) { raza2 = val; }
};
class SegmentOrizontal{
public:
    int xMin;
    int xMax;
    int y;
    SegmentOrizontal(int y,int xmin, int xmax){
        this->xMax = xmax;
        this->xMin = xmin;
        this->y = y;
    }
};
list<SegmentOrizontal*> MSegmente;
class GrilaCarteziana{
private:
    int mLinii;
    int mColoane;
    int mDeltaPixeliPerLinie;
    int mDeltaPixeliPerColoana;
protected:
public:
    GrilaCarteziana(){
        this->mLinii = NO_LINII_DEFAULT;
        this->mColoane = NO_COLOANE_DEFAULT;
        this->initializari();
    }
    GrilaCarteziana(int pLinii, int pColoane){
        this->mLinii = pLinii;
        this->mColoane = pColoane;
        this->initializari();
    }
    void initializari(){
        mDeltaPixeliPerLinie = dimensiuneFereastra/(mLinii+1);
        mDeltaPixeliPerColoana = dimensiuneFereastra/(mColoane+1);
```

```
}
void writePixel(int atX, int atY){
    float x,y;
    float PI = 4*atan(1.0);
    float radius = 10;
    float delta_theta = 0.01;
    glColor3f(0,0,0);
    glPolygonMode(GL_FRONT, GL_FILL);
    glBegin( GL_POLYGON );{
        for( float angle = 0; angle < 2*PI; angle += delta_theta )
            glVertex2f( atX+radius*cos(angle),atY+radius*sin(angle));
    }glEnd();
}
void writePixels(list<Punct*> m){
    list<Punct*>::const_iterator iterator;
    for(iterator = m.begin(); iterator!=m.end(); iterator++){
        this->writePixel((*iterator)->getX()*mDeltaPixeliPerLinie, (*iterator)->getY()*
        mDeltaPixeliPerColoana);
    }
}
void writeLinePixels(list<SegmentOrizontal*> segments){

    list<SegmentOrizontal*>::const_iterator iterator;
    for(iterator = segments.begin(); iterator!=segments.end(); iterator++){
        int xMax, xMin, aux;
        if( (*iterator)->xMin < (*iterator)->xMax){
            xMin = (*iterator)->xMin;
            xMax = (*iterator)->xMax;
        } else {
            xMin = (*iterator)->xMax;
            xMax = (*iterator)->xMin;
        }
        for(int j =xMin; j<=xMax; j++){
            this->writePixel(j*mDeltaPixeliPerLinie,(*iterator)->y*mDeltaPixeliPerColoana);
        }
    }
}

void writeRedLine(int fromX, int fromY, int toX, int toY){
    glColor3f(1,0,0);
    glBegin(GL_LINES);{
        glVertex2i(fromX*mDeltaPixeliPerLinie, fromY*mDeltaPixeliPerColoana);
        glVertex2i(toX*mDeltaPixeliPerLinie, toY*mDeltaPixeliPerColoana);
    }glEnd();
}
void writeCircle(Cerc* pCerc){
    float PI = 4*atan(1.0);
    float radius = pCerc->getRaza()*mDeltaPixeliPerColoana;
    float delta_theta = 0.01;
    glColor3f(1,0,0);
    glPolygonMode(GL_FRONT, GL_LINE);
    glBegin( GL_POLYGON );{
```

```
            for( float angle = 0; angle < 2*PI; angle += delta_theta )
                glVertex2f( pCerc->getCentru()->getX()*mDeltaPixeliPerLinie+radius*cos(angle),
                            pCerc->getCentru()->getY()*mDeltaPixeliPerColoana+radius*sin(angle));
        }glEnd();
    }
    void writeElipse(Elipsa* pElipsa){
        float PI = 4*atan(1.0);
        float radius1 = pElipsa->getRaza1()*mDeltaPixeliPerLinie;
        float radius2 = pElipsa->getRaza2()*mDeltaPixeliPerColoana;
        float delta_theta = 0.01;
        glColor3f(1,0,0);
        glPolygonMode(GL_FRONT, GL_LINE);
        glBegin( GL_POLYGON );{
            for( float angle = 0; angle < 2*PI; angle += delta_theta )
                glVertex2f( pElipsa->getCentru()->getX()*mDeltaPixeliPerLinie+radius1*cos(angle),
                            pElipsa->getCentru()->getY()*mDeltaPixeliPerColoana+radius2*sin(
                            angle));
        }glEnd();
    }
    void draw(){
        glColor3f(0,0,0);
        for(int i = -mLinii-1 ; i<=mLinii; i++){
            glBegin(GL_LINES);{
                glVertex2i(-dimensiuneFereastra*mDeltaPixeliPerLinie,i*mDeltaPixeliPerLinie);
                glVertex2i(dimensiuneFereastra+mDeltaPixeliPerLinie,i*mDeltaPixeliPerLinie);
            }glEnd();
        }
        for(int i = -mLinii-1 ; i<=mLinii; i++){
            glBegin(GL_LINES);{
                glVertex2i(i*mDeltaPixeliPerColoana,-dimensiuneFereastra+mDeltaPixeliPerColoana
                );
                glVertex2i(i*mDeltaPixeliPerColoana, dimensiuneFereastra+mDeltaPixeliPerColoana);
            }glEnd();
        }
    }

};
void AfisarePuncteCerc3(int x, int y){
    MPixeliCerc.push_back(new Punct(x,y));
    MPixeliCerc.push_back(new Punct(x+1,y));
    MPixeliCerc.push_back(new Punct(x-1,y));
    /*
    MPixeliCerc.push_back(new Punct(-x,-y));
    MPixeliCerc.push_back(new Punct(-x,y));
    MPixeliCerc.push_back(new Punct(x,-y));;
    if(x != y){
        MPixeliCerc.push_back(new Punct(y,x));
        MPixeliCerc.push_back(new Punct(-y,-x));
        MPixeliCerc.push_back(new Punct(-y,x));
        MPixeliCerc.push_back(new Punct(y,-x));
    }*/
}
void AfisareCerc4(Cerc* cerc, bool showGrid){
```

```
    GrilaCarteziana* grila = new GrilaCarteziana();
    if(showGrid){
        grila->draw();
    }
    grila->writeCircle(cerc);
    int raza = cerc->getRaza();
    int x = raza, y = 0;
    int d = 1- raza;
    int dN = 3, dNE = -2*raza+5;
    AfisarePuncteCerc3(x+cerc->getCentru()->getX(),y+cerc->getCentru()->getY());
    while(y!=x){
        if(d<0){
            d+=dN;
            dN+=2;
            dNE+=2;
        } else {
            d+=dNE;
            dN+=2;
            dNE += 4;
            x--;
        }
        y++;
        AfisarePuncteCerc3(x+cerc->getCentru()->getX(),y+cerc->getCentru()->getY());
    }
    grila->writePixels(MPixeliCerc);
    //free
    delete(cerc);
    delete(grila);

}
void UmplereElipsa(int x0, int y0, int a, int b){
    int newA = a-1;
    int newB = b-1;
    int xi = 0, x = 0, y = -newB;
    double fxpyp =0.0;
    double deltaV, deltaNV, deltaN;
    GrilaCarteziana* grila = new GrilaCarteziana();
    grila->draw();
    grila->writeElipse(new Elipsa(new Punct(x0, y0), a-1, b-1));
    MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
    while((double)newA*newA*((double)y-0.5)<(double)newB*newB*(x+1)){
        deltaV = (double)newB*newB*(-2*x+1);
        deltaNV = (double)newB*newB*(-2*x+1)+(double)newA*newA*(2*y+1);
        if(fxpyp+deltaV <=0.0){
            fxpyp +=deltaV;
            --x;
            list<SegmentOrizontal*>::const_iterator iterator;
            for(iterator = MSegmente.begin(); iterator!=MSegmente.end(); iterator++){
                if((*iterator)->y == y-y0){
                    (*iterator)->y = y-y0;
                    (*iterator)->xMin = x-x0;
                    (*iterator)->xMax = x0;
                }
```

```
            }
        } else if(fxpyp+deltaNV<=0.0){
            fxpyp += deltaNV;
            --x; ++y;
            MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
        }
    }
    while (y<0){
        deltaNV = (double)newB*newB*(-2*x+1)+(double)newA*newA*(2*y+1);
        deltaN = (double)newA*newA*(2*y+1);
        if(fxpyp+deltaNV<=0){
            fxpyp+=deltaNV;
            --x; ++y;
            MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
        } else {
            fxpyp += deltaN;
            ++y;
        }
        MSegmente.push_back(new SegmentOrizontal(y-y0, x-x0, x0));
    }
    grila->writeLinePixels(MSegmente);
}
void Init(void) {

   glClearColor(1.0,1.0,1.0,1.0);

   glLineWidth(1);

//   glPointSize(4);

   glPolygonMode(GL_FRONT, GL_LINE);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(-dimensiuneFereastra*0.9f, dimensiuneFereastra*0.9f,
        -dimensiuneFereastra*0.9f, dimensiuneFereastra*0.9f);
}
void Display(void) {
   glClear(GL_COLOR_BUFFER_BIT);

   switch(prevKey) {
    case '1':
        AfisareCerc4(new Cerc(new Punct(0,0),10), true);
        break;
    case '2':
        UmplereElipsa(0,0,10,7);
        break;
   default:
      break;
   }

   glFlush();
}

void Reshape(int w, int h) {
```

```
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
}

void KeyboardFunc(unsigned char key, int x, int y) {
   prevKey = key;
   if (key == 27) // escape
      exit(0);
   glutPostRedisplay();
}

void MouseFunc(int button, int state, int x, int y) {
}

int main(int argc, char** argv) {

   glutInit(&argc, argv);

   glutInitWindowSize(dimensiuneFereastra, dimensiuneFereastra);

   glutInitWindowPosition(100, 100);

   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   glutCreateWindow (argv[0]);

   Init();

   glutReshapeFunc(Reshape);

   glutKeyboardFunc(KeyboardFunc);

   glutMouseFunc(MouseFunc);

   glutDisplayFunc(Display);

   glutMainLoop();

   return 0;
}
```