

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

using namespace std;

#define dimensiuneFereastră 600
#define NO_LINII_DEFAULT 15
#define NO_COLOANE_DEFAULT 15

unsigned char prevKey;

class Culoare{
private:
    float R;
    float G;
    float B;
public:
    Culoare(float R, float G, float B){
        this->R = R;
        this->G = G;
        this->B = B;
    }
    void setR(float R){
        this->R = R;
    }
    void setG(float G){
        this->G = G;
    }
    void setB(float B){
        this->B = B;
    }
    float getR(){
        return this->R;
    }
    float getG(){
        return this->G;
    }
    float getB(){
        return this->B;
    }
};

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*> M;

class Dreapta{
private:
    int fromX;
    int fromY;
    int toX;
    int toY;
    float m;
    float n;
    int dX;
    int dY;
    int a;
    int b;
    int c;
public:
    Dreapta(int fromX, int fromY, int toX, int toY){
        this->fromX = fromX;
        this->fromY = fromY;
        this->toX = toX;
        this->toY = toY;
        dX = abs(toX - fromX);
        dY = abs(toY - fromY);
        a = dY;
        b = -dX;
        c = dX*fromY - dY*fromX;
        m = dY/(dX+0.0f);
        n = fromY - (dY*fromX)/(dX+0.0f);
    }
    float getM() const { return m; }
    void setM(float val) { m = val; }
    float getN() const { return n; }
    void setN(float val) { n = val; }
    int getDX() const { return dX; }
    void setDX(int val) { dX = val; }
    int getDY() const { return dY; }
    void setDY(int val) { dY = val; }
    int getA() const { return a; }
    void setA(int val) { a = val; }
    int getB() const { return b; }

```

```

void setB(int val) { b = val; }
int getC() const { return c; }
void setC(int val) { c = val; }

};

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 = dimensiuneFereastră/(mLinii+1);
        mDeltaPixeliPerColoana = dimensiuneFereastră/(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(0.76,0.76,0.76);
        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 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 draw() {
    glColor3f(0,0,0);
    for(int i = -mLinii-1 ; i<=mLinii; i++){
        glBegin(GL_LINES);{
            glVertex2i(-dimensiuneFereastramDeltaPixeliPerLinie,i*mDeltaPixeliPerLinie);
            glVertex2i(dimensiuneFereastramDeltaPixeliPerLinie,i*mDeltaPixeliPerLinie);
        }glEnd();
    }
    for(int i = -mLinii-1 ; i<=mLinii; i++){
        glBegin(GL_LINES);{
            glVertex2i(i*mDeltaPixeliPerColoana,-dimensiuneFereastramDeltaPixeliPerColoana);
            glVertex2i(i*mDeltaPixeliPerColoana, dimensiuneFereastramDeltaPixeliPerColoana);
        }glEnd();
    }
}

};

void AfisareSegmentDreapta3 (int fromX, int fromY, int toX, int toY, bool bolded) {
    GrilaCarteziana* grila = new GrilaCarteziana();
    grila->draw();
    Dreapta* dreapta = new Dreapta(fromX, fromY, toX, toY);
    int d = 2*dreapta->getDY()-dreapta->getDX();
    int dE = 2*dreapta->getDY();
    int dNE = 2*(dreapta->getDY()-dreapta->getDX());
    int x = fromX, y = fromY;
    M.push_back(new Punct(x,y));
    if(bolded){
        M.push_back(new Punct(x,y+1));
        M.push_back(new Punct(x,y-1));
    }
    while(x<toX){
        if(dreapta->getM()>=0){
            if(d<=0){
                //alegem E
                M.push_back(new Punct(x+1, y));
                if(bolded){
                    M.push_back(new Punct(x+1,y+1));
                    M.push_back(new Punct(x+1,y-1));
                }
                d+=dE;
                x++;
            } else {
                //alegem NE
                M.push_back(new Punct(x+1,y+1));
                if(bolded){
                    M.push_back(new Punct(x+1,y));
                    M.push_back(new Punct(x+1,y+2));
                }
                d+=dNE;
                x++;
                y++;
            }
        }
    }
}

```

```

    }
    } else {

    }
}
grila->writeRedLine(fromX, fromY, toX, toY);
grila->writePixels(M);
//free memory
while(M.size()>0){
    Punct* p = M.back();
    M.pop_back();
    delete(p);

}
delete(dreapta);
delete(grila);
}

void Init(void) {

    glClearColor(1.0,1.0,1.0,1.0);

    glLineWidth(1);

    //    glPointSize(4);

    glPolygonMode(GL_FRONT, GL_FILL);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-dimensiuneFereastr*0.9f, dimensiuneFereastr*0.9f,
        -dimensiuneFereastr*0.9f, dimensiuneFereastr*0.9f);
}

void Display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    switch(prevKey) {
    case '1':
        AfisareSegmentDreapta3(-16,-16,16,16, true);
        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(dimensiuneFereastră, dimensiuneFereastră);  
  
    glutInitWindowPosition(100, 100);  
  
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  
  
    glutCreateWindow (argv[0]);  
  
    Init();  
  
    glutReshapeFunc (Reshape);  
  
    glutKeyboardFunc (KeyboardFunc);  
  
    glutMouseFunc (MouseFunc);  
  
    glutDisplayFunc (Display);  
  
    glutMainLoop();  
  
    return 0;  
}
```