机器人第二次作业

姓名：曾宇杰

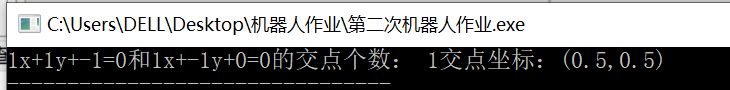
学号2019218164

运行示例：

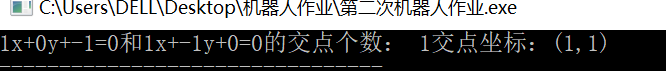
# 直线和直线交点：

## 有一个交点：

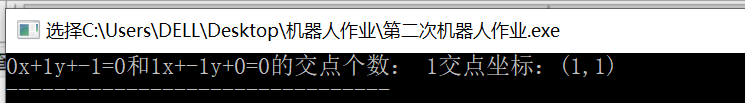
输入：直线x+y=1和直线x=y



输入：直线x=1和直线x=y

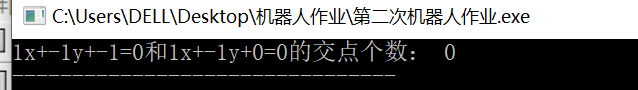


输入：直线y=1和直线x=y



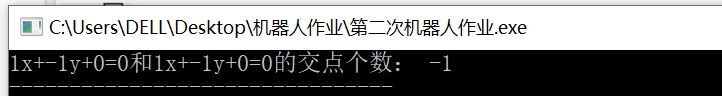
## 没有交点：

输入：直线x=1+y和直线x=y



## 直线重合：

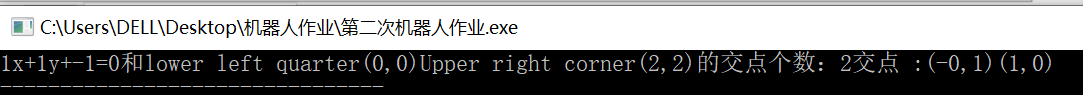
输入直线x=y和直线x=y



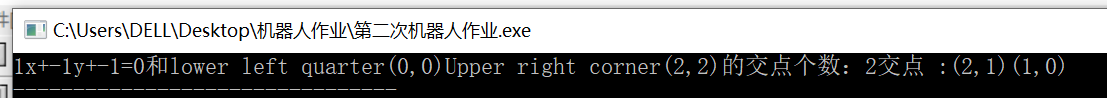
# 直线和矩形交点：

## 有两个交点：

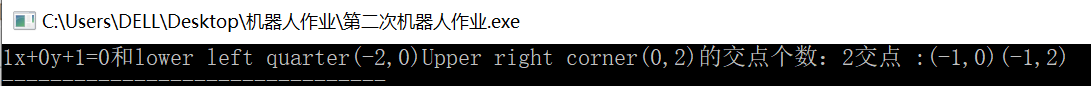
输入：矩形左下角点（0,0）右上角点（2,2），直线x+y=1



输入：矩形左下角点（0,0）右上角点（2,2），直线x-y=1

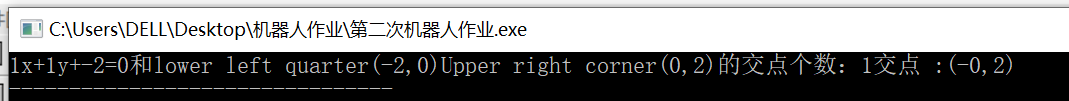


输入：矩形左下角（-2,0）右上角（0,2），直线x=-1

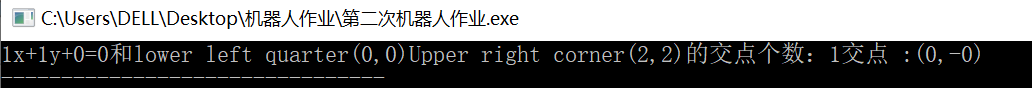


## 有一个交点：

输入：矩形左下角（-2,0）右上角（0,2），直线x+y=2

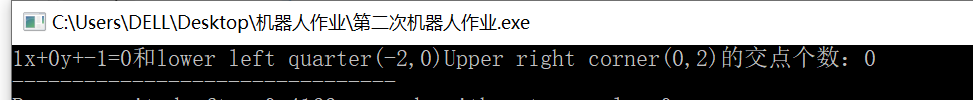


输入：矩形左下角点（0,0）右上角点（2,2），直线x+y=0



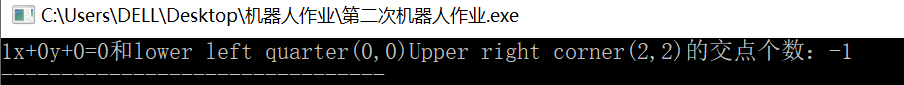
**没有交点：**

输入：矩形左下角（-2,0）右上角（0,2），直线x=1



## 有无数个交点：

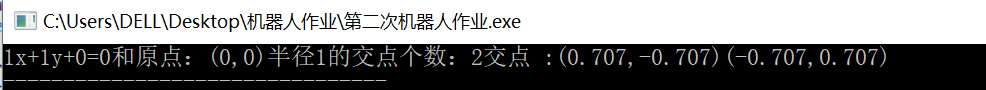
输入：矩形左下角点（0,0）右上角点（2,2），直线x=0



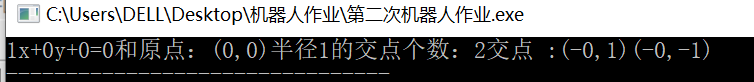
# 直线和圆

## 有两个交点：

输入：直线x+y=0，圆（0,0）半径1

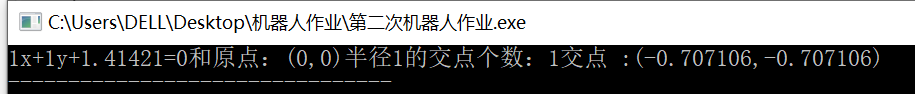


输入：直线x=0，圆（0,0）半径1

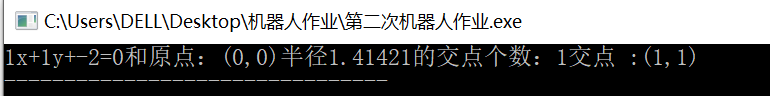


## 有一个交点：

输入：直线x+y+1.414213=0，圆（0,0）半径1

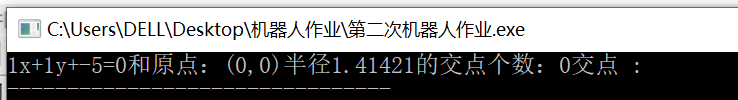


输入：直线x+y-2=0，圆（0,0）半径1.414213



## 没有交点：

输入：直线x+y-5=0，圆（0,0）半径1.414213



#include <iostream>

#include <math.h>

#ifndef ROBOCUB\_SECOND\_HOMEWORK

using namespace std;

class point

{

private:

    double x;

    double y;

public:

    point(double xValue, double yValue);

    point();

    ~point();

    point & operator = (const point &p);

    inline void print(){

        cout << "(" << x << "," << y << ")";

    }

    friend class line;

    friend class rectangle;

    friend class circle;

};

point::point()

{

};

point::point(double xValue, double yValue):x(xValue),y(yValue)

{

};

point::~point()

{

};

point & point::operator = (const point &p)

{

    x = p.x;

    y = p.y;

    return \*this;

}

class line

{

private:

    double A;

    double B;

    double C;

public:

    line(double AValue, double BValue, double CValue);

    line();

    ~line();

    friend int intersection\_line(const line & FirstLine, const line & SecondLine, point &t);

    friend class circle;

    line & operator = (const line & l);

    inline void print()

    {

        cout << A << "x+" << B << "y+" << C << "=0";

    }

    /\*\*

     \* 求点到直线的距离

     \*/

    inline double disOfPointToLine(const point &p)

    {

        return fabs((A\*p.x + B\*p.y + C)/sqrt(A\*A+B\*B));

    }

};

/\*\*

 \* return : -1代表无数个交点，0代表没有交点，1代表一个交点

 \*/

int intersection\_line(const line & FirstLine, const line & SecondLine, point &t)

{

    double m = FirstLine.A \* SecondLine.B - FirstLine.B \* SecondLine.A;

    if (m == 0)

    {

        if (FirstLine.C == SecondLine.C && FirstLine.A == SecondLine.A && FirstLine.B == SecondLine.B)

        {

            return -1;

        }

        else

        {

            return 0;

        }

    }

    else

    {

        t = point(((-FirstLine.C) \* SecondLine.B - FirstLine.B \* (-SecondLine.C))/m, (FirstLine.A \* (-SecondLine.C) - (-FirstLine.C) \* SecondLine.A)/m);

        return 1;

    }

}

line & line::operator = (const line & l)

{

        A = l.A;

        B = l.B;

        C = l.C;

}

line::line()

{

};

line::line(double AValue, double BValue, double CValue)

{   if (A == 0&&B == 0)

    {

        cout << "This line don't exit!";

    }

    else

    {

        A = AValue;

        B = BValue;

        C = CValue;

    }

}

line::~line()

{

}

////////////////////////////////////////////////////////////////////////////////////

class rectangle

{

private:

    point lLim;

    point rLim;

public:

    line leftX;

    line rightX;

    line topY;

    line floorY;

    rectangle(point l,point r);

    rectangle(double xL, double yL, double xR, double yR);

    ~rectangle();

    inline void print()

    {cout<<"lower left quarter";lLim.print();cout<<"Upper right corner";rLim.print();}

    int intersection\_rectangle(const line & line, point &p1, point &p2);

};

//左下右上两个点

rectangle::rectangle(point l, point r)

{

    lLim = l;

    rLim = r;

    line tLeftLim\_x (1, 0, -lLim.x);

    line tRightLim\_x (1, 0, -rLim.x);

    line tLeftLim\_y (0, 1,-lLim.y);

    line tRightLim\_y (0, 1,-rLim.y);

    leftX = tLeftLim\_x;

    rightX = tRightLim\_x;

    topY = tRightLim\_y;

    floorY = tLeftLim\_y;

    /\*

    leftX.print();

    rightX.print();

    topY.print();

    floorY.print(); \*/

};

rectangle::rectangle(double xL, double yL, double xR, double yR)

{

    point l (xL, yL);

    point r (xR, yR);

    lLim = l;

    rLim = r;

    line tLeftLim\_x (1, 0, -lLim.x);

    line tRightLim\_x (1, 0, -rLim.x);

    line tLeftLim\_y (0, 1,-lLim.y);

    line tRightLim\_y (0, 1,-rLim.y);

    leftX = tLeftLim\_x;

    rightX = tRightLim\_x;

    topY = tLeftLim\_y;

    floorY = tRightLim\_y;

    /\*

    leftX.print();

    rightX.print();

    topY.print();

    floorY.print(); \*/

}

rectangle::~rectangle()

{

};

int rectangle::intersection\_rectangle(const line & line, point &p1, point &p2)

{

    point t[4] = {point(65535,65535),point(65535,65535),point(65535,65535),point(65535,65535)};

    if (intersection\_line(line,leftX,t[0]) == -1)

    {

        return -1;

    }

    if (intersection\_line(line,rightX,t[1]) == -1)

    {

        return -1;

    }

    if (intersection\_line(line,topY,t[2]) == -1)

    {

        return -1;

    }

    if (intersection\_line(line,floorY,t[3]) == -1)

    {

        return -1;

    }

    int j = 0;

    int m = 0;

    if (t[0].y >= lLim.y && t[0].y <= rLim.y)

    {

        if (j == 0)

        {

            p1 = t[0];

        }

        else

        {

            p2 = t[0];

        }

        j++;

        m++;

    }

    if (t[1].y >= lLim.y && t[1].y <= rLim.y)

    {

        if (j == 0)

        {

            p1 = t[1];

        }

        else

        {

            p2 = t[1];

        }

        j++;

        m++;

    }

    if (t[2].x > lLim.x && t[2].x < rLim.x)

    {

        if (j == 0)

        {

            p1 = t[2];

        }

        else

        {

            p2 = t[2];

        }

        j++;

        m++;

    }

    if (t[3].x > lLim.x && t[3].x < rLim.x)

    {

        if (j == 0)

        {

            p1 = t[3];

        }

        else

        {

            p2 = t[3];

        }

        j++;

        m++;

    }

    return m;

}

inline double myRound(double x)

{

    return floor(x\*1000+0.5)/1000.0;

}

class circle

{

private:

    point centre;

    double R;

public:

    circle(point c, double r);

    ~circle();

    inline void print()

    {cout<<"圆心坐标：";centre.print();cout<<"半径："<<R;}

    int intersection(line & l,point &p1, point &p2);

};

int circle::intersection(line & l,point &p1, point &p2)

{

    double dis = l.disOfPointToLine(centre);

    //cout << "\*" << dis;

    line m (l.B,-l.A,-l.B\*centre.x + l.A\*centre.y);

    //m.print();

    if (myRound(dis) > myRound(R))

    {

        return 0;

    }

    else if (myRound(dis) < myRound(R))

    {

        point s (0,0);

        point s1 (0,0);

        point s2 (0,0);

        intersection\_line(l,m,s);

        if ( myRound(l.B) == 0)

        {

            s1.x = s.x;

            s2.x = s.y;

            s1.y = sqrt(R\*R - dis\*dis) + s.y;

            s2.y = -sqrt(R\*R - dis\*dis) + s.y;

        }

        else

        {

            double sinValue = sin(atan(-l.A/l.B));

            double cosValue = cos(atan(-l.A/l.B));

            s1.x = myRound(sqrt(R\*R - dis\*dis)\*cosValue + s.x);

            s2.x = myRound(-sqrt(R\*R - dis\*dis)\*cosValue + s.x);

            s1.y = myRound(sqrt(R\*R - dis\*dis)\*sinValue + s.y);

            s2.y = myRound(-sqrt(R\*R - dis\*dis)\*sinValue + s.y);

            /\*

            s2.x = -sqrt(R\*R - dis\*dis)\*cosValue + s.x;

            s1.y = sqrt(R\*R - dis\*dis)\*sinValue + s.y;

            s2.y = -sqrt(R\*R - dis\*dis)\*sinValue + s.y;

            \*/

        }

        p1 = s1;

        p2 = s2;

        return 2;

    }

    else

    {

        intersection\_line(l,m,p1);

        return 1;

    }

}

circle::circle(point c, double r)

{

    centre = c;

    R = r;

}

circle::~circle()

{

}

#endif // !ROBOCUB\_SECOND\_HOMEWORK