



Báo cáo

BÀI TẬP LỚN HK2 2021-2022

Bài 6

Học phần: Lập Trình Hướng Đối Tượng
Giảng viên: Nguyễn Ngọc Long
Lớp: 20TTH_KHDL

Họ và tên	MSSV
Nguyễn Quốc Bảo	20280006
Nguyễn Minh Hoàng Đạt	20280014

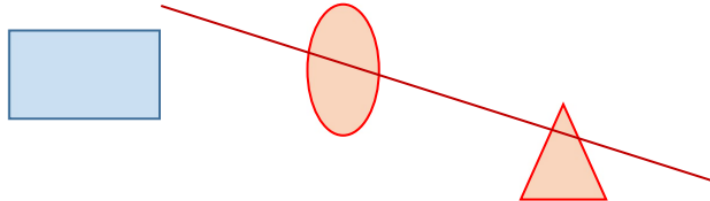
Mục lục

Lời mở đầu	3
I. Định nghĩa các lớp và hàm	4
Point	4
Linear	5
Rectangle	6
Square	9
Triangle	9
Elip	10
Circle	12
Crescent	12
ConvexPolygon	13
Các hàm khác	15
1. Khoảng cách từ 1 điểm đến đường thẳng	15
2. Tìm đỉnh có khoảng cách tới đường thẳng nhỏ nhất	16
3. Xét sự giao nhau giữa 2 đường thẳng	16
4. Tìm giao điểm giữa 2 đường thẳng	17
5. Đánh dấu 1 điểm trên cửa sổ giao diện.	18
6. Kiểm tra 1 điểm thuộc đoạn thẳng	18
7. Tạo điểm mới sau khi Scale.	18
8. Đếm số hình giao nhau với đường thẳng	19
II. Thao tác chương trình	19

Lời mở đầu

Đề bài

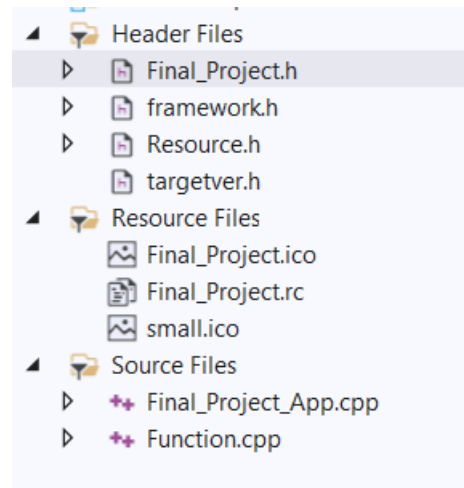
6. Cho một danh sách các đối tượng hình học, mỗi đối tượng thuộc một trong các loại: Hình tròn, hình ellipse, hình bán nguyệt, hình đa giác (lồi), hình chữ nhật, hình vuông, hình tam giác. Cho một đường thẳng ($ax + by = c$) trong mặt phẳng. Viết hàm cho biết đường thẳng cắt các hình nào của danh sách. Viết ứng dụng cho phép tạo các hình và một đường thẳng, vẽ các hình và đường thẳng, xuất thông báo cho biết đường đi qua bao nhiêu hình, tô màu các hình có đường cắt ngang với màu khác các hình còn lại. Trong hình minh họa bên dưới, đường thẳng cắt 2 hình.



Các file nguồn bao gồm

Framework.h, Resource.h, targetver.h xây dựng giao diện hiển thị window

```
#include "targetver.h"
#define WIN32_LEAN_AND_MEAN
// Windows Header Files
#include <windows.h>
// C RunTime Header Files
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
```



Để xây dựng được khái niệm của các đối tượng trong đề bài, Final_Project.h định nghĩa các lớp đối tượng của điểm (Point), đường thẳng (Linear), hình chữ nhật (Rectangle), hình tam giác (Triangle), hình vuông (square), hình Ellipse (Elip), hình tròn (Circle), hình bán nguyệt (Crescent) và hình đa giác lồi (ConvexPolygon). Ngoài ra còn định nghĩa các hàm thuật toán khác để hỗ trợ xử lý chương trình

Trong các lớp đó đã đặt các thuộc tính, xây dựng các phương thức để giải quyết vấn đề. Ứng với mỗi lớp khác nhau có các thuật toán giao với đường thẳng và tô màu khác nhau

Function.cpp xây dựng cấu trúc các lớp được định nghĩa trong Final_Project.h

Final_Project_App.cpp xử lý chương trình chính

I. Định nghĩa các lớp và hàm

Khai báo các thư viện

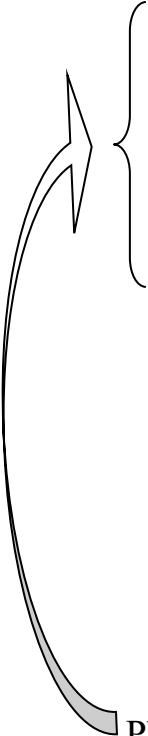
```
#pragma once
#include "resource.h"
#include <iostream>
#include <math.h>
#include <windows.h>

using namespace std;
```

Point

- Định nghĩa lớp Point (điểm)
- Định nghĩa chất điểm có tọa độ (x;y) đặt thuộc tính protected

```
class Point {
protected:
    double x, y;
public:
    Point(double x = 0, double y = 0);
    Point(const Point& k);
    void set(double x, double y);
    void setX(double xx);
    void setY(double yy);
    double GetX() const;
    double GetY() const;
    double Distance(Point other);
    Point MidPoint(Point other);
    virtual void print();
    virtual void draw(HDC hdc, HGDIOBJ original);
    virtual void Move(double dx, double dy);
    virtual void Scale(double T);
    virtual bool isIntersecting(HDC hdc, Point A, Point B);
};
```



Phương thức thiết lập: Khởi tạo mặc nhiên với điểm mặc định là (0, 0). Copy sâu, phương thức setter, getter.

Hàm tính toán

- double Distance(Point other) là hàm trả về kiểu số thực khoảng cách giữa 2 điểm theo công thức khoảng cách

$$MN = \sqrt{(x_N - x_M)^2 + (y_N - y_M)^2}$$

- Point MidPoint (Point other) là hàm trả về kiểu Point tọa độ trung điểm giữa 2 điểm

$$x_P = \frac{x_M + x_N}{2} ; y_P = \frac{y_M + y_N}{2}.$$

Lập phương thức ảo để giải quyết vấn đề đa hình cho việc chọn kiểu các hình

- Virtual void Print() là hàm in ra màn hình tọa độ của điểm
- Virtual void draw(...) là hàm vẽ
- Virtual void Move(double x, double y) là hàm di chuyển chất điểm (hay gọi là thay đổi tọa độ) theo số pixel được truyền vào
- Virtual void Scale(double T) là hàm phóng to (thu nhỏ) theo số pixel truyền vào
- Virtual bool isIntersecting(...) là hàm giao điểm

Linear

- Định nghĩa lớp Linear là đường thẳng có phương trình $ax + by = c$
- Lớp Linear kế thừa từ lớp Point

```

class Linear : public Point {
private:
    double a, b, c; //ax + by = c;
    Point M, N;
public:
    Linear(Point M, Point N);
    Linear(double x1, double y1, double x2, double y2);
    double get_a();
    double get_b();
    double get_c();
    void set(Point M, Point N);
    void set(double Mx, double My, double Nx, double Ny);
    void draw(HDC hdc, HGDIOBJ original);
};

```

- Đặt thuộc tính private cho biến a, b, c là hệ số phương trình $ax + by = c$, M và N là 2 điểm thuộc phương trình

Phương thức thiết lập: Phương thức setter, getter.

Hàm tính toán, vẽ

- Void draw là hàm vẽ đường thẳng

```

void Linear::draw(HDC hdc, HGDIOBJ original) {
    original = SelectObject(hdc, GetStockObject(DC_PEN));
    SetDCPenColor(hdc, RGB(255, 0, 0));
    SetDCBrushColor(hdc, RGB(255, 0, 0));

    MoveToEx(hdc, M.GetX(), M.GetY(), NULL);
    LineTo(hdc, N.GetX(), N.GetY());
    SetDCPenColor(hdc, RGB(0, 255, 0));
    SelectObject(hdc, GetStockObject(DC_BRUSH));
    SetDCBrushColor(hdc, RGB(0, 0, 255));
}

```

Dùng hàm MoveToEx và LineTo có sẵn trong thư viện để vẽ đoạn thẳng nối 2 điểm M và N

Rectangle

- Định nghĩa lớp Rectangle (hình chữ nhật)

```
class RecTangle : public Point {
private:
    Point A;
    double d, r;
public:
    RecTangle(double xa, double ya, double d, double r);
    void set(double xa, double ya, double d, double r);
    void draw(HDC hdc, HGDIOBJ original);
    void Move(double dx, double dy);
    void Scale(double T);
    bool isIntersecting(HDC hdc, Point N, Point M);
};
```

- Đặt thuộc tính private cho

- điểm A đỉnh đầu tiên của hình chữ nhật
- d là chiều dài,
- r là chiều rộng

Phương thức thiết lập: Phương thức setter, getter.

Hàm tính toán, vẽ

- void draw(...)

```
void RecTangle::draw(HDC hdc, HGDIOBJ original) {
    original = SelectObject(hdc, GetStockObject(DC_PEN));
    SetDCPenColor(hdc, RGB(0, 255, 0));
    SelectObject(hdc, GetStockObject(DC_BRUSH));
    SetDCBrushColor(hdc, RGB(0, 0, 255));
    Linear L(Point(300, 100), Point(500, 300));
    L.draw(hdc, original);

    //
    if (isIntersecting(hdc, Point(300, 100), Point(500, 300))) {
        SelectObject(hdc, GetStockObject(DC_BRUSH));
        SetDCBrushColor(hdc, RGB(255, 0, 0));
    }
    Rectangle(hdc, A.GetX(), A.GetY(), A.GetX() + d, A.GetY() + r);
}
```

Câu lệnh if là kiểm tra hình chữ nhật có giao với đường thẳng tạo từ 2 điểm có tọa độ (300;100) và (500;300), nếu có sẽ tô màu hình

Hàm Rectangle(hdc, ...) là để vẽ hình chữ nhật

- void Move(...) là di chuyển hình chữ nhật theo số pixel đầu vào
- void Scale(double T) để tăng diện tích hình bằng cách tăng chiều dài d và chiều rộng r nhân với số pixel T
- bool isIntersecting(HDC hdc, Point N, Point M) là hàm kiểm tra hình chữ nhật có giao với đường thẳng tạo từ 2 điểm N, M không

```
Point B(A.GetX() + d, A.GetY());
Point C(A.GetX() + d, A.GetY() + r);
Point D(A.GetX(), A.GetY() + r);
```

✚ Tạo 3 đỉnh còn lại của hình chữ nhật (từ đỉnh gốc là A) lần lượt là B, C, D

```
// Find the minimum distance from 4 vertices of rectangle ABCD to this line
double distance_Point_to_Line[4];
char list_vertices[] = { 'A', 'B', 'C', 'D' };
Point p[] = { A, B, C, D };
for (int i = 0; i < 4; i++) {
    distance_Point_to_Line[i] = DistancePointToLine(NM, p[i]);
    // if the distance of one of each vertex to this line is 0, it will lie on this line.
    if (distance_Point_to_Line[i] == 0) {
        return true;
    }
}
```

✚ Lưu 4 ký tự A, B, C, D vào trong một mảng ký tự tạm để đổi chiều, mảng số thực `distance_Point_to_Line` dùng tạm để tính khoảng cách từ 4 đỉnh hình chữ nhật đó với đường thẳng

✚ Nếu xảy ra trường hợp khoảng cách từ đỉnh tới đường thẳng = 0 tức là đỉnh đó thuộc đường thẳng thì đường thẳng này giao hình chữ nhật qua đỉnh, hàm return true

```
char vertex_nearest_line = list_vertices[FindIndex_of_SmallestValue(distance_Point_to_Line, 4)];
switch (vertex_nearest_line) {
case 'A': {
    // consider the intersection of this line and AB, AD
    Linear AB(A, B);
    Linear AD(A, D);
    if (Check_2_Lines_Intersect(AB, NM) || Check_2_Lines_Intersect(AD, NM)) {
        Point I = Coordinates_of_intersection_of_2_lines(AB, NM);
        mark(hdc, I);
        Point J = Coordinates_of_intersection_of_2_lines(AD, NM);
        mark(hdc, J);
        if (check_inLine(A, B, I) || check_inLine(A, D, J))
            return true;
    }
    //return false;
} break;
```

✚ Ngược lại tìm đỉnh có khoảng cách đến đường thẳng nhỏ nhất từ 2 mảng tạm ở trên. Hàm `FindIndex_of_SmallestValue` là để tìm giá trị nhỏ nhất trả về giá trị thứ tự (Index) được lưu trong mảng

- + Kí tự `vertex_nearest_line` để truy xuất kí tự thứ index trong mảng kí tự
- + Sau đó dùng lệnh switch case để gọi
- + Nếu là kí tự A thì ta xét 2 đường thẳng xuất phát từ đỉnh A là AB và AD
- + Hàm `Check_2_Lines_Intersect` là kiểm tra 2 đường thẳng có giao nhau không. Nếu NM (đường thẳng ban đầu đang xét) giao AB hoặc AD thì gọi I, J là giao điểm 2 đường thẳng đó tính trong hàm `Coordinates_of_intersection_of_2_lines`
- + Nếu I thẳng hàng và nằm giữa AB hoặc J thẳng hàng và nằm giữa AD ở hàm `check_inline` thì hàm return true
- + Tương tự ở đỉnh B ta xét BA, BC. Đỉnh xét C xét CB, CD. Đỉnh D xét DA, DC
- + Nếu không có trường hợp nào xảy ra thì hàm return false

Square

- Định nghĩa lớp Square là hình vuông
- Lớp Square kế thừa từ lớp Rectangle

```
class Square :public RecTangle {
private:
    double s;
public:
    Square(double x, double y, double s);
};
```

Phương thức thiết lập

Triangle

Định nghĩa lớp Triangle là hình tam giác
Lớp Triangle kế thừa từ lớp Point

```

class Triangle : public Point {
private:
    Point A, B, C; // 3 đỉnh tam giác
public:
    Triangle(double xa, double ya, double xb, double yb, double xc, double yc);
    Triangle(Point A, Point B, Point C);
    void set(double xa, double ya, double xb, double yb, double xc, double yc);
    void print();
    void Move(double dx, double dy);
    void draw(HDC hdc, HGDIOBJ original);
    void Scale(double T);
    Point center();
    bool isIntersecting(HDC hdc, Point N, Point M);
};

```

Đặt thuộc tính private 3 điểm Point A, B, C ứng với 3 đỉnh của tam giác

Phương thức thiết lập: Phương thức setter.

Hàm tính toán, vẽ

- void print()
- void move(...)
- void draw(...)
- void Scale(...)
- Point center() là hàm trả về tâm điểm của tam giác (tâm đường tròn nội tiếp tam giác)
- Bool isInterecting(HDC hdc, Point N, Point M)
 - ✚ Tương tự như lớp Rectangle. Tìm đỉnh của tam giác mà có khoảng cách đến đường thẳng NM gần nhất. Sau đó, tính từ 2 đường thẳng xuất phát từ điểm đó giao với đường thẳng NM ta có I và J nếu I hoặc J nằm giữa 2 điểm của đường thẳng thì hình tam giác giao với đường thẳng NM
 - ✚ Ngược lại thì đường thẳng đó không giao với tam giác

Elip

- Định nghĩa lớp Elip là hình Ellipse
- Lớp Elip kế thừa từ lớp Point

```

class Elip : public Point {
private:
    double a, b; // do dai 2 duong cheo
    Point M;
public:
    Elip(double x = 0, double y = 0, double a = 0, double b = 0);
    Elip(Point A, double a = 0, double b = 0);
    double GetA() const;
    double GetB() const;
    void print();
    void draw(HDC hdc, HGDIOBJ original);
    void Move(double dx, double dy);
    void Scale(double T);
    bool isIntersecting(HDC hdc, Point N, Point M);
};

```

Đặt thuộc tính private cho 2 biến a, b là độ dài 2 đường chéo và M là tâm hình ellipse

Phương thức thiết lập: phương thức getter.

Hàm tính toán, vẽ

- Void print()
- Void draw()
- Void Move()
- Void Scale()
- Bool isIntersecting(HDC hdc, Point N, Point M)

$$\frac{(x - h)^2}{a^2} + \frac{(y - k)^2}{b^2} = 1$$

Ta có phương trình hình ellipse là

- + Với (h;k) là tọa độ tâm hình ellipse. Ta xét phương trình đường thẳng NM là $ax + by = c \Rightarrow y = (-a/b)*x + c/a$ (**)
- + Thay (**) vào phương trình ellipse thì ta được phương trình bậc 2 ẩn là x. Ta biến đổi phương trình đó thành dạng $A*x^2 + B*x + C = 0$
- + Xét phương trình bậc 2 có nghiệm khi $B^2 - 4*A*C \geq 0$ thì đường thẳng giao với hình ellipse

Circle

- Định nghĩa lớp Circle là hình tròn
- Lớp Circle kế thừa từ lớp Elip

```
class Circle :public Elip {  
private:  
    double r;  
public:  
    Circle(double x = 0, double y = 0, double r = 0);  
};
```

Phương thức thiết lập

Crescent

- Định nghĩa lớp Crescent là hình bán nguyệt
- Lớp Crescent kế thừa từ lớp Point

```
class Crescent : public Point {  
private:  
    Point Center;  
    double r;  
public:  
    Crescent(double x_c = 0, double y_c = 0, double r = 0);  
    void draw(HDC hdc, HGDIOBJ original);  
    void Move(double dx, double dy);  
    void Scale(double T);  
    bool isIntersecting(HDC hdc, Point N, Point M);  
};
```

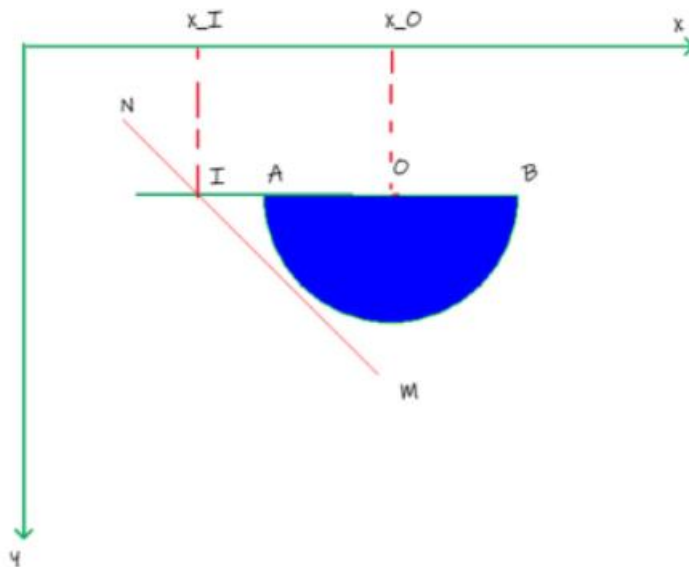
- Đặt thuộc tính private cho biến Center là trung điểm của 2 đầu mút đường kính và r là bán kính

Phương thức thiết lập

Hàm tính toán, vẽ

- Void draw()
- Void move()

- Void Scale()
- Bool isIntersecting(HDC hdc, Point N, Point M)
 - + Do lớp Crescent định nghĩa Point là trung điểm đường kính nên ta khai báo đường thẳng tạo từ Point A, B là 2 đầu của đường kính (tính từ tâm)
 - + TH1: Nếu 2 đường thẳng NM, AB giao nhau thì gọi I là giao điểm 2 đường thẳng đó. Nếu I nằm giữa AB thì đường thẳng giao với hình bán nguyệt
 - + TH2: Ta xét tiếp nửa cung tròn của hình bán nguyệt. Ta gọi lại hàm isIntersecting của hình tròn để xét tính giao nhau của phần cung tròn bán nguyệt với đường thẳng
 - + Nếu đúng thì đường thẳng giao với hình bán nguyệt. Ngược lại trả về false



ConvexPolygon

- Định nghĩa lớp ConvexPolygon là hình đa giác lồi

```

class ConvexPolygon :public Point {
private:
    Point* p;
    int size;
public:
    ConvexPolygon(double coor[] = NULL, int len = 0);
    ~ConvexPolygon();
    void draw(HDC hdc, HGDIOBJ original);
    void Move(double dx = 1, double dy = 1);
    void Scale(double T);
    bool isIntersecting(HDC hdc, Point N, Point M);
};

```

- Đặt thuộc tính private cho con trỏ p là tập hợp các đỉnh p[i] với số đỉnh là size

Phương thức thiết lập, phương thức hủy bỏ

Hàm tính toán, vẽ

- Void draw()
- Void Move()
- Void Scale()
- Bool isIntersecting(HDC hdc, Point N, Point M)

Tương tự như lớp Rectangle và Triangle,

```

// Find the smallest element in Point p array to NM
int i = 0;
int index = 0;
double distance = INT_MAX;
while (index < size) {
    if (distance > DistancePointToLine(NM, p[index])) {
        distance = DistancePointToLine(NM, p[index]);
        i = index;
    }
    index++;
}

```

- Tìm đỉnh của hình đa giác có khoảng cách tới đường thẳng ngắn nhất và lưu vị trí i của đỉnh đó

```

int j = 0;
int k = 0;
if (i >= size - 1) {
    j = 0;
}
else {
    j = i + 1;
}
if (i <= 0) {
    k = size - 1;
}
else {
    k = i - 1;
}

```

- J, k là vị trí của 2 đỉnh liền kề với đỉnh có vị trí i ở trên. Nếu I là đỉnh cuối cùng của đa giác thì j quay về đỉnh đầu tiên. Nếu I là đỉnh đầu tiên thì k quay về đỉnh cuối cùng

```

Linear AB(p[i], p[j]);
Linear AD(p[i], p[k]);
if (Check_2_Lines_Intersect(AB, NM) || Check_2_Lines_Intersect(AD, NM)) {
    Point I = Coordinates_of_intersection_of_2_lines(AB, NM);
    mark(hdc, I);
    Point J = Coordinates_of_intersection_of_2_lines(AD, NM);
    mark(hdc, J);
    if (check_inLine(p[i], p[j], I) || check_inLine(p[i], p[k], J))
        return true;
}

```

- Gọi 2 đường thẳng AB, AD từ 3 đỉnh đa giác trên
- Hàm [Check_2_Lines_Intersect](#) là kiểm tra 2 đường thẳng có giao nhau không. Nếu NM (đường thẳng ban đầu đang xét) giao AB hoặc AD thì gọi I, J là giao điểm 2 đường thẳng đó tính trong hàm [Coordinates_of_intersection_of_2_lines](#)
- Nếu I thẳng hàng và nằm giữa AB hoặc J thẳng hàng và nằm giữa AD ở hàm [check_inline](#) thì hàm return true
- Nếu không có trường hợp nào xảy ra thì hàm return false

Các hàm khác

1. Khoảng cách từ 1 điểm đến đường thẳng

```

double DistancePointToLine(Linear L, Point A)
{
    double numerator = abs(A.GetX() * L.get_a() + A.GetY() * L.get_b() - L.get_c());
    double denominator = sqrt(pow(L.get_a(), 2) + pow(L.get_b(), 2));
    return numerator / denominator;
}

```

Trong mặt phẳng tọa độ Oxy, khoảng cách từ điểm $M(X_0; Y_0)$ đến đường thẳng $\Delta: Ax + By + C = 0$

$$d(M, \Delta) = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

Mà trong định nghĩa lớp đường thẳng có dạng $Ax + By = C$ nên ở tử số ta chuyển $Ax + By - C = 0$ và áp dụng công thức

2. Tìm đỉnh có khoảng cách tới đường thẳng nhỏ nhất

```

int FindIndex_of_SmallestValue(double a[], int n)
{
    int index = 0;
    double min = a[index];
    for (int i = 1; i < n; i++) {
        if (a[i] <= a[index])
            index = i;
    }
    return index;
}

```

Ta có mảng số thực $a[]$ lưu khoảng cách từ các đỉnh đến đường thẳng đang xét ($a[0]$ ứng với đỉnh A, $a[1]$ ứng với đỉnh B...). Ta duyệt các phần tử trong mảng để tìm phần tử nhỏ nhất, ứng với vị trí index của phần tử đó là đỉnh có khoảng cách đến đường thẳng nhỏ nhất

3. Xét sự giao nhau giữa 2 đường thẳng


```

bool Check_2_Lines_Intersect(Linear L, Linear K)
{
    // use cramer's rule to solve it
    // (L): ax + by = c
    // (K): a'x + b'y = c
    double d = L.get_a() * K.get_b() - K.get_a() * L.get_b();
    double dx = L.get_c() * K.get_b() - K.get_c() * L.get_b();
    double dy = L.get_a() * K.get_c() - K.get_a() * L.get_c();
    if (d != 0)
        return true;
    return false;
}

```

Xét 2 phương trình đường thẳng là một hệ 2 phương trình 2 ẩn, sử dụng quy tắc Cramer để biện luận 2 đường thẳng giao nhau khi hệ có nghiệm, tức là d phải khác 0

Phương pháp Cramer

Hệ phương trình tuyến tính 2 biến $\begin{cases} a_1x + b_1y = c_1 \\ a_2x + b_2y = c_2 \end{cases}$

- Tính các định thức:

$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1b_2 - a_2b_1$$

$$D_X = \begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix} = c_1b_2 - c_2b_1$$

$$D_Y = \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} = a_1c_2 - a_2c_1$$

- Công thức Cramer:

$$x = \frac{D_X}{D}$$

$$y = \frac{D_Y}{D}$$

4. Tìm giao điểm giữa 2 đường thẳng

```

Point Coordinates_of_intersection_of_2_lines(Linear L, Linear K)
{
    // use cramer's rule to solve it
    // (L): ax + by = c
    // (K): a'x + b'y = c
    double d = L.get_a() * K.get_b() - K.get_a() * L.get_b();
    double dx = L.get_c() * K.get_b() - K.get_c() * L.get_b();
    double dy = L.get_a() * K.get_c() - K.get_a() * L.get_c();
    return Point(dx / d, dy / d);
}

```

Tương tự hàm kể trên. Khi đã thỏa quy tắc cramer là 2 đường thẳng có nghiệm, thì nghiệm đó có dạng (dx/d;dy/d)

5. Đánh dấu 1 điểm trên cửa sổ giao diện.

```

void mark(HDC hdc, Point I) {
    SetPixel(hdc, I.GetX(), I.GetY(), RGB(255, 0, 0));
}

```

Dùng để đánh dấu tọa độ giao điểm của đường thẳng và 1 hình bất kỳ trong danh sách các hình.

6. Kiểm tra 1 điểm thuộc đoạn thẳng

```

bool check_inLine(Point N, Point M, Point I)
{
    return abs (I.Distance(N) + I.Distance(M) - N.Distance(M) ) <= 0.0000001;
}

```

Giả sử I thuộc đoạn thẳng NM thì $IN + IM = NM$ hay $IN + IM - NM = 0$

Do định nghĩa tọa độ các điểm có kiểu số thực và có thể sẽ xảy ra sai số thập phân cực nhỏ nên trong quá trình tính toán có thêm mất 0.000001 hoặc phát sinh thêm do làm tròn

Nên ta xét trị tuyệt đối $IN + IM - NM \leq 0.000001 \sim 0$. Ngược lại nếu lớn hơn thì I không thuộc đoạn thẳng NM

7. Tạo điểm mới sau khi Scale.

```

Point NewPoint_afterScale(Point midpoint, Point center, double T)
{
    return Point(2*T*(center.GetX() - midpoint.GetX()) + center.GetX(),
        2 * T * (center.GetY() - midpoint.GetY()) + center.GetY());
}

```

Trong hình ban đầu, có trung điểm và trọng tâm là 2 điểm cố định, ta sẽ dùng đó làm chuẩn để tìm ra tọa độ điểm mới theo công thức:

+ Gọi A là điểm cố định, G là trọng tâm, I là trung điểm của đường thẳng xuất phát từ A qua G, và A' là điểm sau khi scale.

+ Ta có: $\text{vector}(GA') = T \cdot \text{vector}(GA) = 2 \cdot T \cdot \text{vector}(IG)$

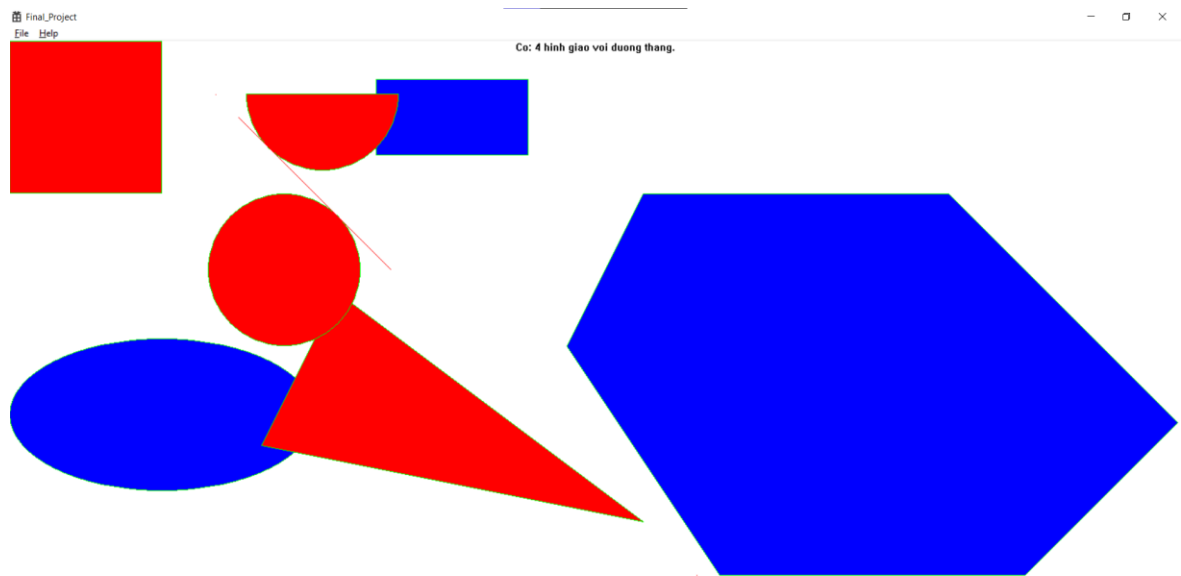
+ Dùng phương pháp tọa độ ta sẽ tìm được điểm A'.

8. Đếm số hình giao nhau với đường thẳng

```
int Count_Intersect(Point** p, int n, HDC hdc, Point N, Point M) {  
    int count = 0;  
    for (int i = 0; i < n; i++) {  
        if (p[i]->isIntersecting(hdc, N, M)) {  
            count++;  
        }  
    }  
    return count;  
}
```

II. Thao tác chương trình

- Khi biên dịch chương trình sẽ mở cửa sổ giao diện với đường thẳng và các hình. Đồng thời, có thể thao tác các hình hiển thị để xét sự giao nhau giữa các hình và đường thẳng
- Nếu hình giao với đường thẳng sẽ được tô màu đỏ. Ngược lại được tô màu xanh
- Thao tác các hình bằng các phím di chuyển trên bàn phím
- Phóng to thu nhỏ các hình bằng nút '+' và '-'
- Chuyển sang các hình khác bằng nút tab
- Khi thao tác các hình xong có thể tab để chuyển đến phần giao diện cuối cùng hiển thị tất cả các hình và đường thẳng trong 1 giao diện



Cảm ơn thầy đã theo dõi

