Khoa Toán - Tin học
Fac. of Math. & Computer Science

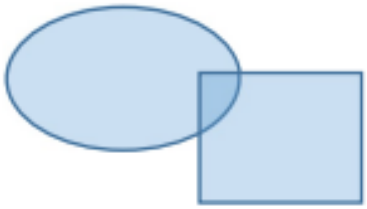
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

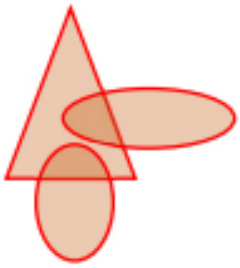
BÁO CÁO BÀI TẬP LỚN

Dự Án 8
Gom Nhóm Hình Liên Thông

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Giáo viên: Nguyễn Ngọc Long





Sinh viên thực hiện:

Nguyễn Xuân Tuấn - 20280112

Đỗ Quốc Việt - 20280115

06/2022

Nội Dung

I. Cấu trúc Class.

II. Thuật toán chương trình.

III. Giao diện đồ hoạ.

IV. Hướng dẫn sử dụng.

I. Cấu trúc Class.

1. Cấu trúc các Class được khai báo và định nghĩa trong bài.

- Point: lớp định nghĩa tọa độ (x,y) cho các hình gồm các phương thức get, set để truy suất và thay đổi thông tin trong quá trình nhập suất

- Geometry: lớp cơ sở cho các hình

- Ellipse: lớp dẫn xuất từ Geometry mang các thuộc tính của hình Ellipse và phương thức ảo được định nghĩa lại cho hình ellipse
    - Circle: lớp dẫn xuất từ Ellipse kế thừa các thuộc tính và phương thức của Ellipse
        - SemiCircle lớp dẫn xuất từ Circle kế thừa các thuộc tính và phương thức của hình tròn và có thuộc tính riêng để xác định chiều bị khuyết, đường thẳng cắt nửa, các phương thức được định nghĩa lại cho phù hợp.
- Polygon: lớp dẫn xuất từ Geometry mang các thuộc tính của hình đa giác (lồi) và các phương thức ảo được định nghĩa lại cho hình đa giác lồi
    - Triangle: lớp dẫn xuất từ Polygon kế thừa các thuộc tính của hình đa giác (lồi) và các phương thức ảo được định nghĩa lại cho hình tam giác
    - Rectangle: lớp dẫn xuất từ Polygon mang các thuộc tính của hình chữ nhật và phương ảo được định nghĩa lại cho hình chữ nhật
        - Square: lớp dẫn xuất từ Rectangle kế thừa các thuộc tính và phương thức ảo của hình chữ nhật

- QuadraticEquationIn2Var: chứa tham số của phương trình bậc hai hai biến và hàm bạn CheckSolution để giải hai phương trình bằng nhau.

- LinearEquationIn2Var: : chứa tham số của phương trình bậc một hai biến và hàm bạn CheckSolution để giải hai phương trình bằng nhau.

2. Thuộc tính và phương thức của các Class.

2. 1. Point.

- Lớp định nghĩa tọa độ gồm các biến x, y chứa thông tinh là vị trí theo trục hoành và trục tung của đối tượng mang kiểu dữ liệu int và được định nghĩa là các thuộc tính private.

- Hàm tạo và các phương thức get, set được định nghĩa là … để truy cập và sửa đổi thông tin các thuộc tính một cách gián tiếp, đảm bảo tính đóng gói của hướng đối tượng.

```
// CPoint
class CPoint
{
private:
    int x, y;
public:
    CPoint(int a = 0, int b = 0);
    void SetCPoint(int a, int b);
    int getX() const;
    int getY() const;
};
```

2.2. Geometry.

- Lớp cơ sở cho các lớp hình, chứa vector các phương trình bậc một và bậc hai, các phương thức ảo được định nghĩa bằng 0 đảm bảo tính đa hình cho các lớp kế thừa:

+ Phương thức ảo getCentroid dùng để lấy thông tin tâm của hình

+ Phương thức ảo getArea dùng để lấy thông tin diện tích của các hình

+ Phương thức ảo IsInside dùng để kiểm tra một điểm (CPoint) có nằm trong hình đó hay không

+ Phương thức ảo draw …

+ Phương thức ảo fill …

+ Hàm bạn checkBelong …

```
class Geometry
{
protected:
    vector<LinearEquationIn2Var> VLinearEqua;
    vector<QuadraticEquationIn2Var> VQuadraEqua;
public:
    virtual CPoint getCentroid() = 0;
    virtual float getArea() = 0;
    virtual bool IsInside(CPoint) = 0;
    virtual void draw(HDC hdc) = 0;
    virtual void fill(HDC hdc, int R, int G, int B) = 0;
    friend set<int> checkBelong();
};
```

2.3. Lớp CEllipse

- Lớp định nghĩa cho hình Ellipse, là lớp dẫn suất từ lớp cơ sở Geometry mang các thuộc tính có thể kế thừa như Ra (bán kình dài), Rb (bán kính rộng), Center (tâm), hàm tạo và các phương thức ảo được thiết kế riêng cho hình Ellipse như getCentroid, getArea, …

```
class CElipse : public Geometry
{
protected:
    int Ra, Rb;
    CPoint Center;

public:
    CElipse(CPoint center = CPoint(0, 0), int a = 0, int b = 0);
    void setCElipse(CPoint center = CPoint(0, 0), int a = 0, int b = 0);
    virtual CPoint getCentroid();
    virtual float getArea();
    virtual bool IsInside(CPoint);
    virtual void draw(HDC hdc);
    virtual void fill(HDC hdc, int R, int G, int B);
};
```

2.4. Lớp CCirlce

- Lớp định nghĩa cho hình Cirlce, là lớp dẫn suất từ lớp Cellipse kế thừa các thuộc tính của Ellipse như Ra, Rb (Ra = Rb) và có thuộc tính riêng biệt: limit = 0 dùng để xác định hình tròn không bị giới hạn, hàm tạo và các phương thức ảo được kế thừa từ Elipse như: getCentroid, getArea, draw, fill. Vì có thuộc tính limit riêng biệt với hình Ellipse nên thiết kế lại hàm ảo IsInside để phù hợp và kế thừa lại cho lớp SemiCCircle

```
// CCircle
class CCircle : public CElipse
{
protected:
    int limit;
public:
    int getLimit() { return limit; };
    CCircle(CPoint center = CPoint(0, 0), int r = 0, int lim = 0);
    void setCCircle(CPoint center = CPoint(0, 0), int r = 0, int lim = 0);
    virtual bool IsInside(CPoint);
};
```

2.5. Lớp SemiCCirlce

- Lớp định nghĩa cho hình SemiCirlce, là lớp dẫn suất từ lớp CCirlce kế thừa các thuộc tính của Circle và thuộc tính limit có thể bằng 0 hoặc 1, dùng để xác định hình tròn chỉ giới hạn ở trên hay ở dưới, hàm tạo và các phương thức ảo được kế lại để phù hợp với thuộc tính giới hạn. Phương thức ảo IsInside được kế thừa từ Ccircle.

```
// SemiCCircle
class CSemiCirlce : public CCircle
{
private:
    LinearEquationIn2Var HorizonLine;
public:
    CSemiCirlce(CPoint center = CPoint(0, 0), int r = 0, int slim = 1);
    virtual CPoint getCentroid();
    virtual float getArea();
    virtual void draw(HDC hdc);
    virtual void fill(HDC hdc, int R, int G, int B);
};
```

2.6. Lớp CRectangle

- Lớp định nghĩa cho hình Rectangle, là lớp dẫn suất từ lớp Polygon mang các thuộc tính đặc trưng như LUpper (điểm trái trên), Rlower (điểm phải dưới), width (chiều dài), height (chiều cao) và kế thừa vector chứa các đỉnh từ Polygon. Các phương thức ảo được thiết kế riêng cho hình Rectangle như getCentroid, getArea, …

```
// CRectangle
class CRectangle : public CPolygon
{
protected:
    int width, height;
public:
    CRectangle(CPoint LUpper = CPoint(0, 0), CPoint RLower = CPoint(0, 0));
    void setCRectangle(CPoint LUpper, CPoint RLower);
    virtual void draw(HDC hdc);
    virtual void fill(HDC hdc, int R, int G, int B);
    virtual CPoint getCentroid();
    virtual float getArea();
    virtual bool IsInside(CPoint);
};
```

2.7. Lớp CSquare

- Lớp định nghĩa cho hình Square, là lớp dẫn suất từ lớp CRectangle kế thừa các thuộc tính của và hàm tạo của CRctangle.

```
class CSquare : public CRectangle
{
public:
    CSquare(CPoint LUpper = CPoint(0, 0), CPoint RLower = CPoint(0, 0));
    void setCSquare(CPoint LUpper, CPoint RLower);
};
```

2.8. Lớp CPolygon

- Lớp định nghĩa cho hình Polygon, là lớp dẫn suất từ lớp Geometry mang các thuộc tính có thể kế thừa như VPoint (vector chứa tọa độ các đỉnh), VEqua (vector chứa các phương trình được tạo bởi 2 đỉnh kề nhau theo chiều đồng hồ) và các phương thức ảo được thiết kế riêng cho hình Polygon như getCentroid, getArea, …

```
class CPolygon : public Geometry
{
protected:
    vector<CPoint> VCPoint;
public:
    CPolygon(vector<CPoint> vCPoint = vector<CPoint>{ CPoint(0,0) });
    void setCPolygon(vector<CPoint> vCPoint);
    virtual void draw(HDC hdc);
    virtual void fill(HDC hdc, int R, int G, int B);
    virtual CPoint getCentroid();
    virtual float getArea();
    virtual bool IsInside(CPoint);
};
```

2.9. Lớp CTriangle

- Lớp định nghĩa cho hình Triangle, là lớp dẫn suất từ lớp CPolygon kế thừa các thuộc tính của và hàm tạo của CPolygon.

```
// CTriangle
class CTriangle : public CPolygon
{
public:
    CTriangle(vector<CPoint> vCPoint);
    void setCTriangle(vector<CPoint> vCPoint);
    virtual float getArea();
};
```

2.10. Lớp LinearEquationIn2Var

- Lớp định nghĩa cho phương trình bậc một chứa 2 biến x,y. Gồm các thuộc tính là tham số của phương trình và các biến giới hạn theo trục hoành và trục tung của đường thẳng được mô tả bởi phương trình. Các hàm bạn nằm bên trong lớp nhằm mục đích cấp quyền truy suất các thuộc tính để giải phương trình đường thẳng tuyến tính giao với các đường thẳng khác.

```
// Equation
// Linear Equation Class in 2 variables
class LinearEquationIn2Var {
private:
    // Linear form : A*(x - a)  + B*(y - b) = 0
    int A, a, B, b;
    int xlim_l, xlim_r, ylim_d, ylim_u;
public:
    LinearEquationIn2Var(int A0 = 0, int a0 = 0, int B0 = 0, int b0 = 0);
    LinearEquationIn2Var(CPoint p1, CPoint p2);
    void setLinearEquationIn2Var(CPoint p1, CPoint p2);

    friend bool CheckSolution(const LinearEquationIn2Var Equa_1, const QuadraticEquationIn2Var Equa_2);
    friend bool CheckSolution(const LinearEquationIn2Var Equa_1, const LinearEquationIn2Var Equa_2);
};
```

2.11. Lớp QuadraticEquationIn2Var

- Lớp định nghĩa cho phương trình bậc hai chứa 2 biến x,y. Gồm các thuộc tính là tham số của phương trình và biến limit chỉ hướng giới hạn cho phương trình. Các hàm bạn nằm bên trong lớp nhằm mục đích cấp quyền truy suất các thuộc tính để giải phương trình đường phi tuyến giao với các đường khác.

```
// Quadratic Equation Class in 2 variables
class QuadraticEquationIn2Var {
private:
    // Quadratic form : A*(x - a)^2  + B*(y - b)^2 = R
    int Limit;
    int A, a, B, b, R;
public:
    QuadraticEquationIn2Var(int A0 = 0, int a0 = 0, int B0 = 0, int b0 = 0, int r = 0, int limit = 0);
    int getLimit() { return Limit; };
    friend class CElipse;
    friend class CCircle;

    friend bool CheckSolution(const LinearEquationIn2Var Equa_1, const QuadraticEquationIn2Var Equa_2);
    friend bool CheckSolution(const QuadraticEquationIn2Var Equa_1, const QuadraticEquationIn2Var Equa_2);
};
```

3. Giải thích các phương thức trong hàm.
3.1. getCentroid.

Đối với các hình như Ellipse, Circle chỉ cần return về Center (tâm) của các hình:

```
CPoint CElipse::getCentroid()
{
	return Center;
}
```

Đối với SemiCircle, tâm của hình được định nghĩa lại trung điểm của đường trung trực đi qua đoạn thẳng tuyến tính của hình:

```
CPoint CSemiCirlce::getCentroid()
{
	if (limit == 1) {
		return CPoint(Center.getX(), Center.getY() - Rb / 2);
	}
	else {
		return CPoint(Center.getX(), Center.getY() + Rb / 2);
	}
}
```

Đối với Rectangle và Square, tâm của hình được nghĩa là trung điểm của hoành độ và tung độ của đường chéo:

```
CPoint CRectangle::getCentroid()
{
	return CPoint((VCPoint[0].getX() + VCPoint[2].getX()) / 2,
				(VCPoint[0].getY() + VCPoint[2].getY()) / 2);
}
```

Đối với Polygon và Triangle, ta lấy trung bình hoành độ, tung độ của các đỉnh:

```
CPoint CPolygon::getCentroid()
{
	int x = 0, y = 0;
	for (int i = 0; i < VCPoint.size(); i++) {
		x += VCPoint[i].getX();
```

```
                y += VCPoint[i].getY();
        }
        return CPoint(x / VCPoint.size(), y / VCPoint.size());
}
```

3.2. getArea.

Ta dùng các công thức toán học có sẵn để tính diện tích của các hình, riêng đối với Polygon, sử dụng công thức Shoelace để tính toán:

```
float Distant(CPoint a, CPoint b) {
        return sqrt(pow(a.getX() - b.getX(), 2) + pow(a.getY() - b.getY(),
                                                                        2));
}


float CElipse::getArea()
{
        return PI*Ra*Rb;
}

float CSemiCirlce::getArea()
{
        return PI * Ra * Rb / 2;
}



float CRectangle::getArea()
{
        return width * height;
}

float CPolygon::getArea()
{
        // Using Shoelace formula
        float area = 0.0;
```

```
            int n = VCPoint.size();
            int j = n - 1;
            for (int i = 0; i < n; i++)
            {
                  area += (VCPoint[j].getX() + VCPoint[i].getX()) *
                              (VCPoint[j].getY() - VCPoint[i].getY());
                  j = i;
            }
            return abs(area / 2.0);
   }

      float CTriangle::getArea()
      {
            float da, db, dc;
            da = Distant(VCPoint[0], VCPoint[1]);
            db = Distant(VCPoint[0], VCPoint[2]);
            dc = Distant(VCPoint[1], VCPoint[2]);
            float p = 0.5 * (da + db + dc);
            return sqrt(p * (p - da) * (p - db) * (p - dc));
   }
```

3.3. IsInside.

Đối với hình Ellipse, ta có phương trình của đường phi tuyến bậc 2 bao bọc hình là:

```
(x - a)^2/Ra^2 + (y - b)^2/Rb^2 = 1
```

nên ta có các điểm (x, y) nằm bên trong hình sẽ thỏa mãn phương trình dưới đây:

```
(x - a)^2/Ra^2 + (y - b)^2/Rb^2 <= 1
```

Từ công thức ta được phương thức:

```
bool CElipse::IsInside(CPoint pcheck)
{
      int ra2 = VQuadraEqua[0].A * VQuadraEqua[0].A;
      int x = (pcheck.getX() - VQuadraEqua[0].a) * (pcheck.getX() -
                                                VQuadraEqua[0].a);
      int rb2 = VQuadraEqua[0].B * VQuadraEqua[0].B;
```

```
	int y = (pcheck.getY() - VQuadraEqua[0].b) * (pcheck.getY() -
                                                VQuadraEqua[0].b);
	return x / ra2 + y / rb2 <= 1;
}
```

Tương tự đối với hình Circle và SemiCricle, ta có điểm (x, y) nằm bên trong hình sẽ thỏa mãn phương trình:

```
A*(x - a) + B*(y - b) <= R^2
```

Ta có thuộc tính limit để giới hạn chiều của hình nên ta sẽ giới hạn trục hoành và trục tung của điểm nằm trong hình thỏa mãn phương trình:

```
bool CCircle::IsInside(CPoint pcheck)
{
	bool check = false;

	check = (VQuadraEqua[0].A * pow(pcheck.getX() - VQuadraEqua[0].a, 2) +
VQuadraEqua[0].B * pow(pcheck.getY() - VQuadraEqua[0].b, 2)) <=
                                                VQuadraEqua[0].R;
	string res = to_string(limit);
	TCHAR* tmp = new TCHAR[res.length()];
	tmp[res.size()] = 0;
	copy(res.begin(), res.end(), tmp);

	if (check && limit != 0)
		if (limit == 1)
			return pcheck.getY() <= Center.getY();
		else
			return pcheck.getY() >= Center.getY();
	else if (check)
		return true;
	return false;
}
```

Đối với hình Rectangle, Square, Polygon và Triangle, ta đặt giả thuyết điểm nằm trong hình kết hợp với một điểm bất kỳ nằm bên ngoài hình sẽ giao với nhiều nhất một đường thẳng tạo hình. Từ giả thuyết ta có phương thức như sau:

```
bool CRectangle::IsInside(CPoint pcheck)
{
	LinearEquationIn2Var lcheck(pcheck, CPoint(pcheck.getX() + 1000000,
							pcheck.getY()));
	int through = 0;
	for (auto i : VLinearEqua) {
		through += CheckSolution(lcheck, i);
	}
	return through == 1;
}
```

3.4. draw .

Vẽ viền các hình nhờ vào API của win32

3.5. fill.

Tô nền các hình nhờ vào thư viện GDI+

4. Các hàm phụ trợ trong bài.

4.1. Distant.

Hàm tính khoảng các từ 2 điểm (CPoint) bất kỳ

```
float Distant(CPoint a, CPoint b) {
	return sqrt(pow(a.getX() - b.getX(), 2) + pow(a.getY() - b.getY(), 2));
}
```

Dựa vào công thức khoảng các giữa 2 điểm trong không gian Euclid mà ta có được.

4.2. IsConvex.

Hàm kiểm tra các điểm nhập có tạo nên một đa giác lồi theo chiều kim đồng hồ hay không:

```
bool isConvex(vector <CPoint> points) {
	bool neg = false;
	bool pos = false;
	int n = points.size();
```

```
	for (int i = 0; i < n; i++) {
		int a = i;
		int b = (i + 1) % n;
		int c = (i + 2) % n;
		int crossProduct = calc(points[a].getX(), points[a].getY(),
points[b].getX(), points[b].getY(), points[c].getX(), points[c].getY());
		if (crossProduct < 0) neg = true;
		else if (crossProduct > 0) pos = true;
		if (neg && pos) return false;
	}
	float sum = 0;
	for (int i = 0; i < points.size() - 1; i++) {
		sum += Distant(points[i], points[i + 1]);
	}
	if (sum == Distant(points[0], points[points.size() - 1]))
		return 0;
	return true;
}

int calc(int ax, int ay, int bx, int by, int cx, int cy) {
	int BAx = ax - bx;
	int BAy = ay - by;
	int BCx = cx - bx;
	int BCy = cy - by;
	return (BAx * BCy - BAy * BCx);
   }
```

Link tham khảo và giải thích chi tiết: https://www.tutorialspoint.com/convex-polygon-in-cplusplus

4.3. SolveSolution.

Hàm bạn của tất cả các phương trình, dùng để kiểm tra 2 hình giao nhau dựa trên các phương trình đặc trưng của 2 hình.

+ Giữa bậc 1 và bậc 1:

```
bool CheckSolution(const LinearEquationIn2Var Equa_1, const
LinearEquationIn2Var Equa_2)
{
	config cfg;
	cfg.set("auto_config", true);
	context c(cfg);
	solver s(c);

	// Linear form: A*(x - a) + B*(y - b) = 0

	expr x = c.real_const("x");
	expr y = c.real_const("y");
	s.add(Equa_1.A * (x - Equa_1.a) + Equa_1.B * (y - Equa_1.b) == 0);
	s.add(Equa_2.A * (x - Equa_2.a) + Equa_2.B * (y - Equa_2.b) == 0);

	s.add(x >= Equa_1.xlim_l && x <= Equa_1.xlim_r);
	s.add(y >= Equa_1.ylim_d && y <= Equa_1.ylim_u);

	s.add(x >= Equa_2.xlim_l && x <= Equa_2.xlim_r);
	s.add(y >= Equa_2.ylim_d && y <= Equa_2.ylim_u);
	if (!s.check())
		return false;
	else
		return true;
}
```

+ Giữa bậc 1 và bậc 2:

```
bool CheckSolution(const LinearEquationIn2Var Equa_1, const
QuadraticEquationIn2Var Equa_2)
{
	config cfg;
	cfg.set("auto_config", true);
	context c(cfg);
```

```
		expr x = c.real_const("x");
		expr y = c.real_const("y");
		solver s(c);
		s.add(Equa_1.A * (x - Equa_1.a) + Equa_1.B * (y - Equa_1.b) == 0);
		if(Equa_2.Limit == 3)
			s.add((x - Equa_2.a) * (x - Equa_2.a)/(Equa_2.A* Equa_2.A) + (y
	- Equa_2.b) * (y - Equa_2.b)/(Equa_2.B* Equa_2.B) == 1);
		else
			s.add(Equa_2.A * (x - Equa_2.a) * (x - Equa_2.a) + Equa_2.B * (y
	- Equa_2.b) * (y - Equa_2.b) == Equa_2.R);


		// add Linear intersect limit
		s.add(x >= Equa_1.xlim_l && x <= Equa_1.xlim_r);
		s.add(y >= Equa_1.ylim_d && y <= Equa_1.ylim_u);


		// case semiCCircle in Equa_2		<limit = 1 or -1>
		if (Equa_2.Limit != 0)
			if (Equa_2.Limit == 1) {
				s.add(y <= Equa_2.b);
			}
			else if (Equa_2.Limit == -1) {
				s.add(y >= Equa_2.b);
			}


		if (!s.check())
			return false;
		else
			return true;
	}
```

+ Giữa bậc 2 và bậc 2:

```
	bool CheckSolution(const QuadraticEquationIn2Var Equa_1, const
	QuadraticEquationIn2Var Equa_2)
	{
		config cfg;
```

```
	cfg.set("auto_config", true);
	context c(cfg);

	expr x = c.real_const("x");
	expr y = c.real_const("y");

	solver s(c);
	if (Equa_1.Limit == 3 and Equa_2.Limit == 3) {
		s.add((x - Equa_1.a) * (x - Equa_1.a) / (Equa_1.A * Equa_1.A) +
(y - Equa_1.b) * (y - Equa_1.b) / (Equa_1.B * Equa_1.B) == 1);
		s.add((x - Equa_2.a) * (x - Equa_2.a) / (Equa_2.A * Equa_2.A) +
(y - Equa_2.b) * (y - Equa_2.b) / (Equa_2.B * Equa_2.B) == 1);
	}
	else if (Equa_1.Limit == 3) {
		s.add((x - Equa_1.a) * (x - Equa_1.a) / (Equa_1.A * Equa_1.A) +
(y - Equa_1.b) * (y - Equa_1.b) / (Equa_1.B * Equa_1.B) == 1);
		s.add(Equa_2.A * (x - Equa_2.a) * (x - Equa_2.a) + Equa_2.B * (y
- Equa_2.b) * (y - Equa_2.b) == Equa_2.R);
	}
	else if(Equa_2.Limit == 3) {
		s.add((x - Equa_2.a) * (x - Equa_2.a) / (Equa_2.A * Equa_2.A) +
(y - Equa_2.b) * (y - Equa_2.b) / (Equa_2.B * Equa_2.B) == 1);
		s.add(Equa_1.A * (x - Equa_1.a) * (x - Equa_1.a) + Equa_1.B * (y
- Equa_1.b) * (y - Equa_1.b) == Equa_1.R);
	}
	else {
		s.add(Equa_1.A * (x - Equa_1.a) * (x - Equa_1.a) + Equa_1.B * (y
- Equa_1.b) * (y - Equa_1.b) == Equa_1.R);
		s.add(Equa_2.A * (x - Equa_2.a) * (x - Equa_2.a) + Equa_2.B * (y
- Equa_2.b) * (y - Equa_2.b) == Equa_2.R);
	}

	// case semiCCircle in Equa_1    <limit = 1 or -1>
	if (Equa_1.Limit != 0)
		if (Equa_1.Limit == 1)
			s.add(y <= Equa_1.b);
```

```
                else if (Equa_1.Limit == -1)
                        s.add(y >= Equa_1.b);


        // case semiCCircle in Equa_2    <limit = 1 or -1>
        if (Equa_2.Limit != 0)
                if (Equa_2.Limit == 1)
                        s.add(y <= Equa_2.b);
                else if(Equa_2.Limit == -1)
                        s.add(y >= Equa_2.b);


        if (!s.check())
                return false;
        else
                return true;
    }
```

Sử dụng API của Microsoft để giải hệ 2 phương trình bằng nhau qua các form phương trình chung tự định nghĩa cho các hình và phương trình như:

+ Phương trình bậc 1: A*(x - a) + B*(y - b) = 0
+ Phương trình bậc 2: A*(x - a) + B*(y - b) = R
+ Hình Ellipse: (x - a)*(x - a)/Ra*Ra + (y - b)*(y - b)/Rb*Rb = 1

+ Hình Circle: (x - a)^2 + (y - b)^2 = R^2

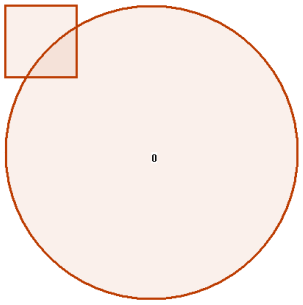
Với điều kiện limmit = 0, hoặc 1 và -1 giới hạn hướng của các phương trình bậc 2 và điều kiện xlim_r, xlim_l, ylim_u, ylim_d của các phương trình bậc 1, ta ràng buộc được các điều kiện bằng nhau để giải các phương trình tìm ra nghiệm phù hợp với ngữ cảnh.

II. Thuật toán chương trình.

1. Bài toán 2 hình liên thông.

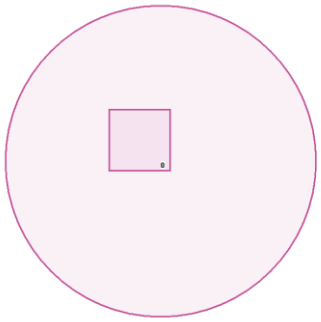
Xác định 2 hình bất kỳ liên thông với nhau xảy ra 2 trường hợp: 2 hình cắt nhau ít nhất tại một điểm hoặc có một hình nằm trong hình còn lại.

1.1. Trường hợp 1: 2 hình cắt nhau ít nhất tại một điểm.



Với mỗi hình ta sẽ chuyển nó thành danh sách các phương trình đường thẳng bậc 1 và bậc 2. Sau đó tiến hành giải từng cặp phương trình của hình này với hình kia. Nếu chỉ cần tồn tại ít nhất một nghiệm thì 2 hình sẽ giao nhau. Như vậy 2 hình liên thông với nhau.

1.2 Trường hợp 2: một hình nằm trong hình còn lại.



Ta dùng trường hợp 2 khi trường hợp 1 không tồn tại nghiệm.

Gọi x, y là 2 hình trong trục toạ độ.

Để kiểm tra x có nằm trong y hay không ta sẽ kiểm tra tra tâm của x có nằm trong y hay không. Và ngược lại để kiểm tra y có nằm trong x hay không ta sẽ kiểm tra tra tâm của y có nằm trong x hay không.

Nếu một trong hai điều kiện trên thoả mãn thì ta xác định đây là trường hợp một hình nằm trong hình còn lại, như vậy 2 hình liên thông với nhau.

```
set<int> checkBelong() {
```

```
    int n = f.size();
    set<int> belong;
    for (int i = 0; i < n - 1; i++) {
        bool ok = 0;
        for (auto u : f[n - 1]->VLinearEqua) {
            if (ok)
                break;
            for (auto v : f[i]->VLinearEqua) {
                if (CheckSolution(u, v)) {
                    belong.insert(i);
                    ok = 1;
                    break;
                }
            }
        }
        if (ok)
            continue;
        for (auto u : f[n - 1]->VLinearEqua) {
            if (ok)
                break;
            for (auto v : f[i]->VQuadraEqua) {
                if (CheckSolution(u, v)) {
                    belong.insert(i);
                    ok = 1;
                    break;
                }
            }
        }
        if (ok)
            continue;
        for (auto u : f[n - 1]->VQuadraEqua) {
            if (ok)
                break;
            for (auto v : f[i]->VLinearEqua) {
                if (CheckSolution(v, u)) {
                    belong.insert(i);
                    ok = 1;
                    break;
                }
            }
        }
        if (ok)
            continue;
        for (auto u : f[n - 1]->VQuadraEqua) {
            if (ok)
                break;
            for (auto v : f[i]->VQuadraEqua) {
                if (CheckSolution(v, u)) {
                    belong.insert(i);
                    ok = 1;
                    break;
                }
            }
        }
        if (!ok) {
            if (f[n - 1]->IsInside(f[i]->getCentroid()) || f[i]->IsInside(f[n - 1]-
>getCentroid())) {
                belong.insert(i);
```

```
            }
        }
    }
    return belong;
}
```

2. Thuật toán phân nhóm.

2.1. Kiến trúc.

Ta thiết kế cấu trúc quản lý hình nào nằm trong một nhóm nào.

Sử dụng một vector có cấu trúc:

```
vector <vector<int>> group;
```

Để lưu lại danh sách các shape mà từng group quản lý. Như vậy trong từng group[i] sẽ có vector chứa id các hình.

2.2. Thuật toán.

Khi thêm một hình mới (đặt là F) vào danh sách ta sẽ tiến hành kiểm tra ngay F liên thông với những hình nào đã có trong danh sách, từ đó biết được F liên thông với những nhóm nào.

Xuất hiện 2 trường hợp: liên thông với 0 nhóm và liên thông với từ 1 nhóm trở lên.

Trường hợp 1: liên thông với 0 nhóm.

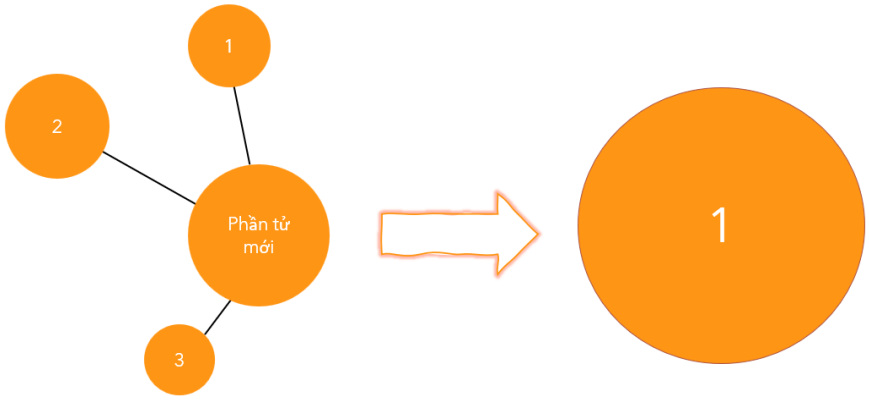
Tạo một nhóm mới rỗng và push F vào nhóm đấy.

Trường hợp 2: liên thông với từ 1 nhóm trở lên.

Ta sẽ có F liên thông với n (n > 0) nhóm.

Tiến hình push danh sách hình từ các nhóm 2, 3…, n vào nhóm 1 sau đó xoá các nhóm 2, 3…, n.

Push F vào nhóm 1.




```
void groupShape(set <int> belong) {
    int n = f.size();
    if (belong.size() == 0 and f.size() > m) {
        m = f.size();
        group.push_back({ n - 1 });
    }

    else {
        m = f.size();
        vector <int> sGroup;
        for (int i = group.size() - 1; i >= 0; i--) {
            for (int j = 0; j < group[i].size(); j++) {
                if (belong.count(group[i][j])) {
                    sGroup.push_back(i);
                    break;
                }
            }
        }
        for (int i = 0; i < sGroup.size() - 1; i++) {
            for (auto item : group[sGroup[i]]) {
                group[sGroup[sGroup.size() - 1]].push_back(item);
            }
            group.erase(group.begin() + sGroup[i]);
        }
        group[sGroup[sGroup.size() - 1]].push_back(n - 1);
    }
}
```

3. Quyết định màu sắc.

Nhờ quản lý ở dạng nhóm ta có thể dễ dàng cài đặt màu riêng biệt cho từng nhóm cũng như các hình trong nhóm đấy sẽ có 1 màu chung.

```
int colorR[MAXSHAPE] = { 0 }, colorG[MAXSHAPE] = { 0 }, colorB[MAXSHAPE] = { 0 };
```

Sử dụng hệ màu R, G, B ta có thể random ra 256^3 màu (16.777.216 màu). Tuy nhiên những màu sắc ở dãy màu cuối sẽ cho mức độ sáng cao gần như màu trắng dẫn đến hiện tượng khó quan sát vì vậy ta sẽ cài đặt mức loại bỏ các màu sắc có giá trị lớn hơn bằng (180, 180, 180).

```
R = rand() % 256, G = rand() % 256, B = rand() % 256;
```

4. Tính tổng diện tích từng nhóm liên thông.

Nhờ quản lý ở dạng nhóm ta có thể dễ dàng tính được tổng diện tích của các hình trong một nhóm hình. Sau đó lưu vào 1 vector groupArea để thuận tiện cho việc xuất ra sau này.

```
for (auto shape : group[i]) {
     sum += f[shape]->getArea();
}
groupArea.push_back(sum);
```

III. Giao diện đồ hoạ.

1. Tạo chương trình

1.1. Giới thiệu.

Window32API : Là một thư viện các hàm, các định nghĩa được hệ điều hành windows cung cấp cho các lập trình viên có thể lấy ra để sử dụng cho các mục đích viết các ứng dụng của mình trên nền windows. Trong hệ thống windows, đã có rất nhiều các hàm có sẵn để tạo ra các dạng ứng dụng như là :Tạo cửa sổ, tạo button, dialog, message box,…. .

Và vấn đề người lập trình phải biết rằng, nó đã có sẵn, và mình phải tìm cách lấy nó ra để viết cho các ứng dụng của mình. Đây chính là lúc chúng ta nhớ lại ý nghĩa của việc lập trình module và hàm trong c++.

Người ta tạo ra các hàm sẵn có phục vụ cho mục đích của mình, và cho phép người khác sử dụng lại để phục vụ cho mục đích của họ, miễn sao cả hai ứng dụng đều chạy chung trên một nền tảng.

1.2. Sử dụng.

Project được xây dựng bằng Win32api thuần bằng cách tạo Project mới trong Visual Studio với Windows Application App. Sau đó tiếp tục phát triển giao diện và import code các Class.

2. Giao diện chính.

Với cảm hứng bắt nguồn từ văn hoá bắc âu, thiết kế của chương trình được thừa hưởng tinh hoa của phong cách Minimalism (tối giản).

Giao diện chính của chương trình là cửa sổ đồ hoạ nơi mọi giao thoa của hình học được phác hoạ tinh tế.

Hai thẻ Chọn hình và About cho người dùng hoàn toàn đơn giản để có thể bắt đầu và sử dụng chương trình.

3. Giao diện nhập.

Thiết kế giao diện nhập được tạo và lưu trong file *.rc.

Với mỗi hình sẽ quản lý một dialog box.

```
INT_PTR CALLBACK InputHinhTron(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            ICheck = 1;
            GetDlgItemText(hDlg, IDC_EDIT1, px1, 30 - 1);
            ix1 = _ttoi(px1);
            GetDlgItemText(hDlg, IDC_EDIT2, py1, 30 - 1);
            iy1 = _ttoi(py1);
            GetDlgItemText(hDlg, IDC_EDIT3, px2, 30 - 1);
            ix2 = _ttoi(px2);
            GetDlgItemText(hDlg, IDC_EDIT4, py2, 30 - 1);
            iy2 = _ttoi(py2);
            Geometry* FF = new CCircle(CPoint(ix1, iy1), ix2);
            f.push_back(FF);
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

Và dữ liệu được lấy ra và dùng ngay lặp tức cho việc khởi tạo một hình mới, sau đó push hình mới vào danh sách để chờ được xử lý gom nhóm.

4. Giao diện xuất

4.1. Xuất hình.

Khi mỗi hình được thêm vào danh sách, chương trình sẽ luôn bắt đầu cập nhật lại thông tin cho chúng và thao tác vẽ lại cho tất cả.

Trong case `WM_PAINT`, Vòng lặp sẽ được gọi và thực hiện vẽ tất cả các hình có trong danh sách.

```
for (int i = 0; i < f.size(); i++) {
            SelectObject(hdc, CreateHatchBrush(HS_API_MAX, RGB(255, 0, 250)));
```

```
            SelectObject(hdc, CreatePen(PS_INSIDEFRAME, 3, RGB(colorR[i], colorG[i],
colorB[i])));
            f[i]->fill(hdc, colorR[i], colorG[i], colorB[i]);
            f[i]->draw(hdc);
}
```

a. Vẽ viền của hình: sử dụng win32api thuần để vẽ viền cho tất cả các hình, với cài đặt trong suốt 100% ở bên trong.

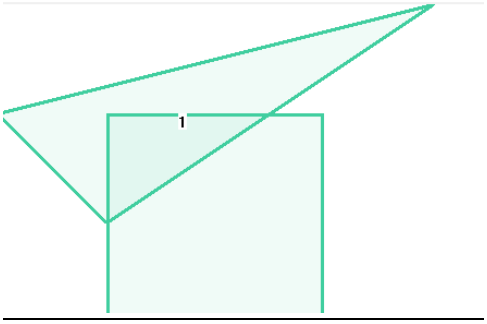
```
void CElipse::draw(HDC hdc)
{
      Ellipse(hdc, Center.getX() - Ra, Center.getY() - Rb, Center.getX() + Ra,
Center.getY() + Rb);
}
```

b. Tô màu bán-trong suốt trong hình: sử dụng api của gdiplus và tận dụng được kênh màu ARGB để có thể tuỳ chỉnh được mức trong suốt mà ta muốn nhờ hệ số Alpha(A).

```
void CElipse::fill(HDC hdc, int R, int G, int B)
{
      Graphics g(hdc);
      SolidBrush brush(Color(20 /*A*/, R /*R*/, G /*G*/, B /*B*/));
      g.FillEllipse(&brush, Center.getX() - Ra, Center.getY() - Rb, Ra * 2, Rb *
2);

}
```

c. Vẽ id cho mỗi nhóm: với mỗi nhóm, vẽ id vào tâm của hình đầu tiên của mỗi nhóm.



4.2 Xuất diện tích.

Diện tích được lưu lại tại `groupArea;` với kiểu dữ liệu là số thực, sau đó được chuyển sang kiểu dữ liệu string và xử lý lấy 2 chữ số đằng sau dấu chấm phẩy, tiếp tục chuyển sang kiểu dữ liệu TCHAR* và dùng lệnh DrawText để xuất ra màn hình đồ hoạ.

```
vector <float> groupArea;
        float sum = 0;
        for (int i = 0; i < group.size(); i++) {
            sum = 0;
            for (auto shape : group[i]) {
                sum += f[shape]->getArea();
            }
            groupArea.push_back(sum);
            string num = to_string(sum);
```

```
            string ss = "";
            int index = 0;
            while (index < num.length() and num[index] != '.') {
                ss += num[index];
                index++;
            }
            for (int i = 0; i < 3; i++) {
                ss += num[index + i];
            }
            string res = to_string(i) + ":  " + ss;
            TCHAR* tmp = new TCHAR[res.length()];
            tmp[res.size()] = 0;
            copy(res.begin(), res.end(), tmp);
            DWORD dwHeight = GetSystemMetrics(SM_CYSCREEN);
            WORD dwWidth = GetSystemMetrics(SM_CXSCREEN);
            int limitHeight = 15;
            RECT rleft = { dwWidth - 200, i * limitHeight, dwWidth, i * limitHeight
+ limitHeight };
            RECT r = { dwWidth - 150, i * limitHeight, dwWidth, i * limitHeight +
limitHeight };
            DrawText(hdc, L"Nhóm ", 8, &rleft, (DT_LEFT) | (DT_SINGLELINE));
            DrawText(hdc, tmp, res.length(), &r, (DT_LEFT) | (DT_SINGLELINE));
}
```

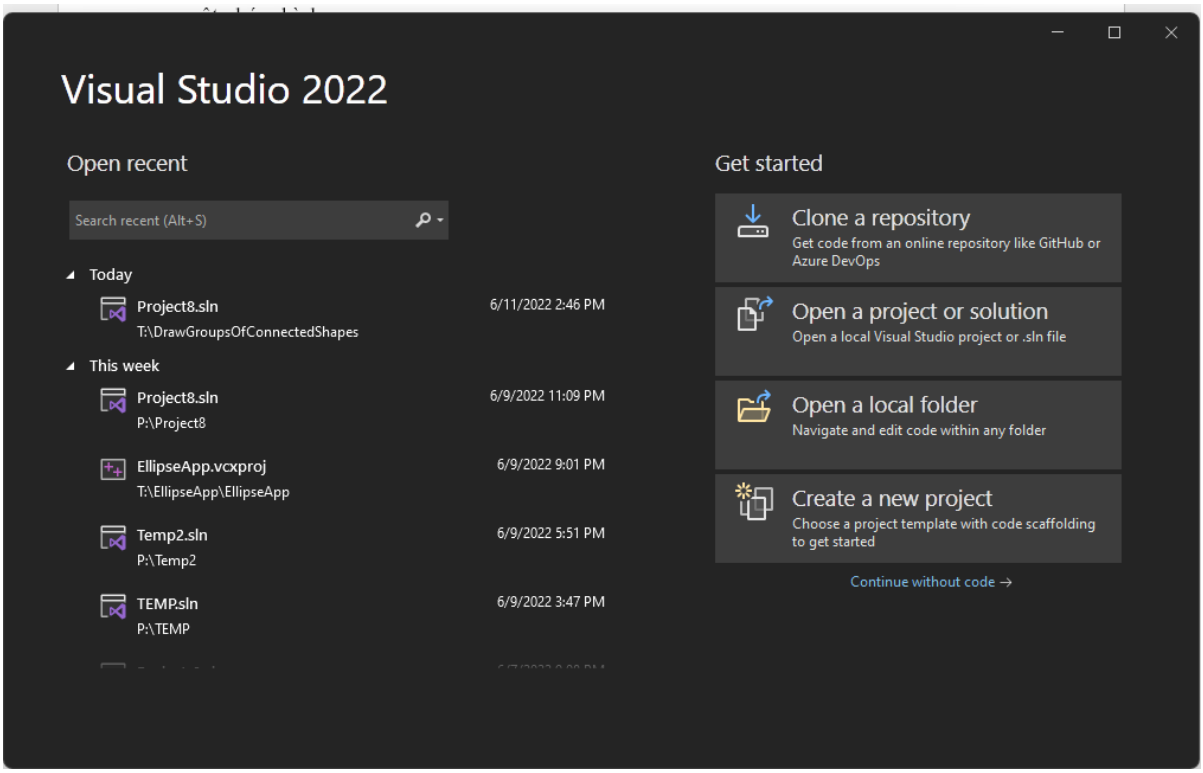
IV. Hướng dẫn sử dụng.

1. Khởi động chương trình.

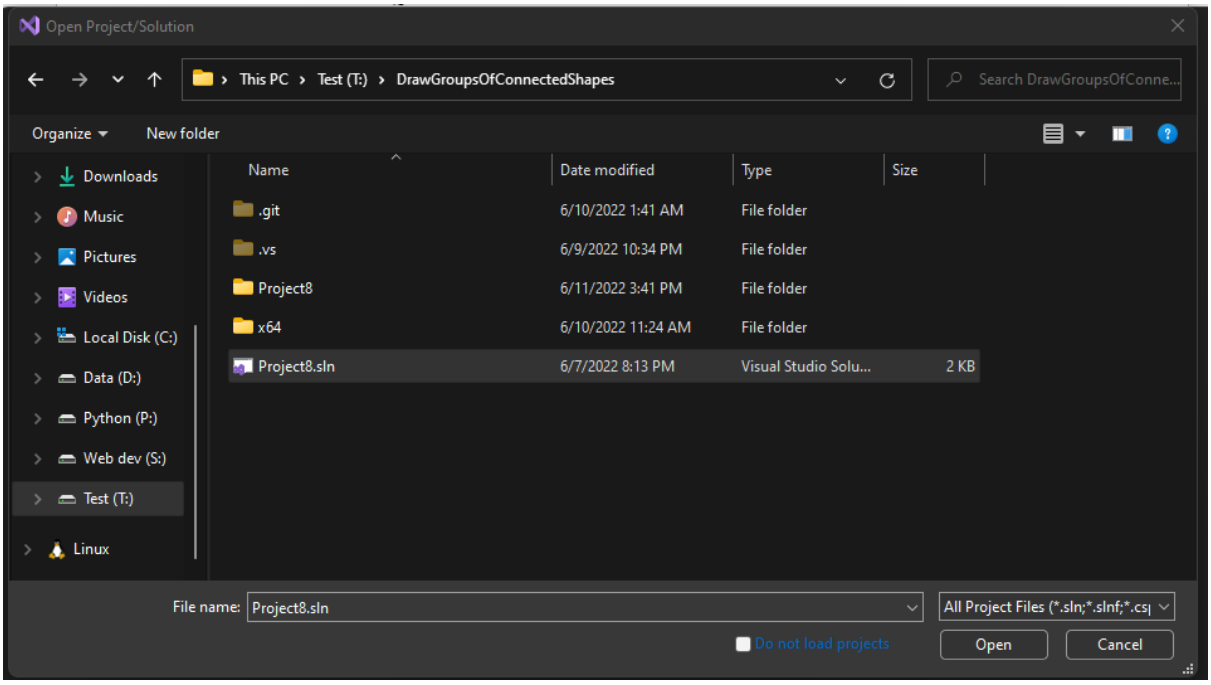
1.1. Build chương trình bằng Visual Studio.

B1. Mở Visual Studio.

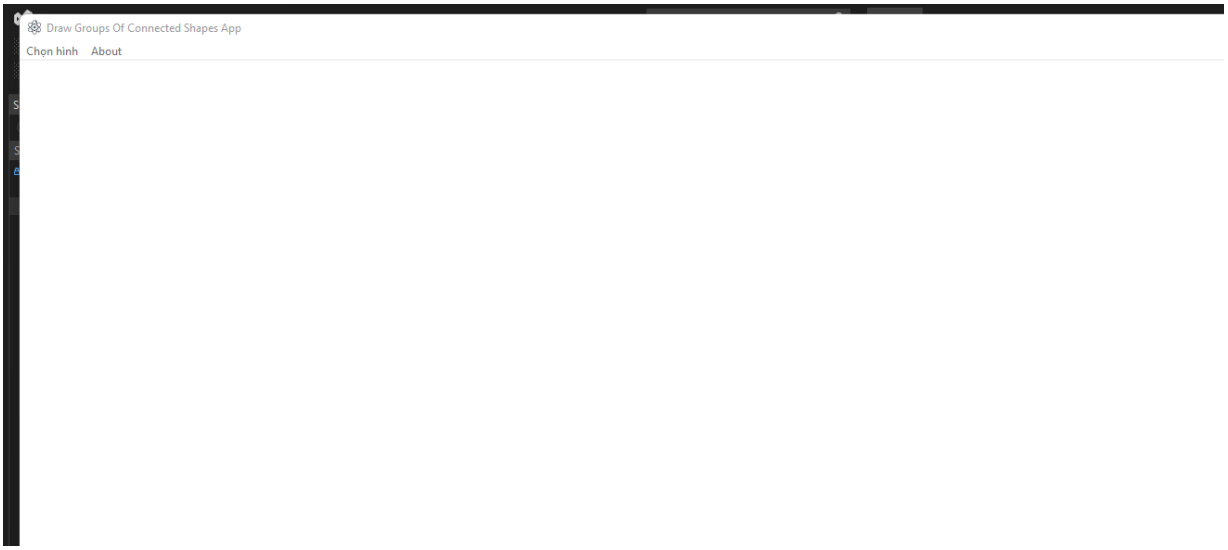
B2. Chọn Open a project or solution.



B3. Chọn đến file Project8.sln.

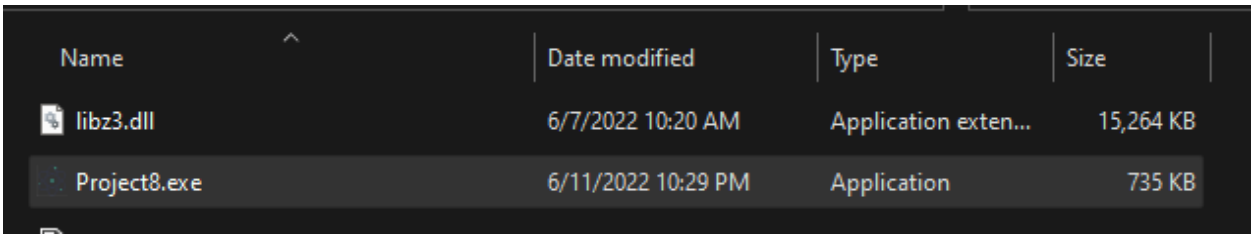


B4. F5 để build chương trình.



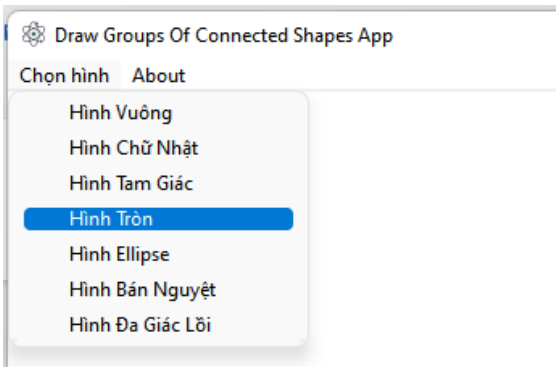
1.2. Chạy trực tiếp bằng Execute file.

Mở trực tiếp file Project8.exe để chạy chương trình.

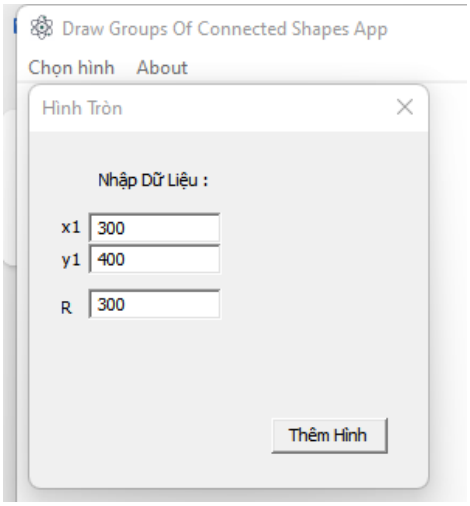


2. Hướng dẫn sử dụng chương trình.

Chọn thẻ chọn hình sau đấy chọn hình muốn nhập.



Điền các thông tin cần thiết sau đấy nhấn vào thêm hình.



Hình sẽ được tự động vẽ ra.