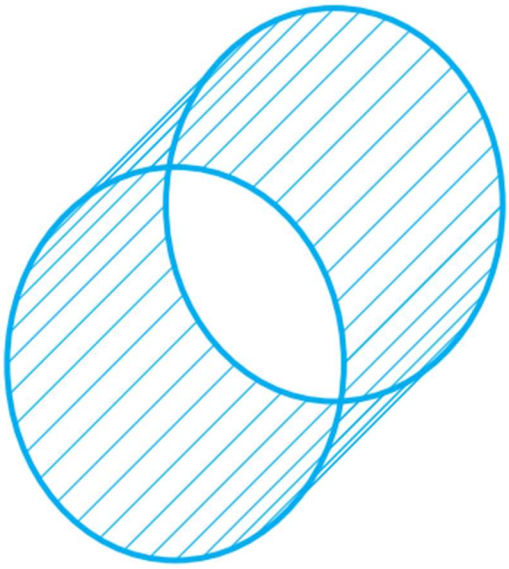
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG – HCM

Khoa Toán - Tin Học

BÁO CÁO BÀI TẬP LỚN MÔN
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG



Khoa Toán - Tin học

Fac. of Math. & Computer Science

Giảng viên: Nguyễn Ngọc Long

Sinh viên thực hiện:

19110370 - Trần Gia Long

19110398 - Lê Đình Nguyên

Đề bài

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. Viết ứng dụng cho phép tạo các hình, xuất thông báo cho biết các hình trên tạo nên mấy nhóm hình liên thông, vẽ các nhóm hình liên thông với màu tô khác nhau, mỗi nhóm một màu tô và xuất tổng diện tích của các hình trong mỗi nhóm (mỗi nhóm có một tổng diện tích). Trong hình minh họa phía dưới có 2 nhóm liên thông.

I. Thiết kế các lớp

- **Thiết kế các lớp cơ bản**

a) Lớp điểm

- ***Xây dựng lớp điểm***

```
class Diem
{
	double x, y;
public:
	Diem(double xx = 0, double yy = 0) : x(xx), y(yy) {} // hàm khởi tạo
	~Diem() {} // hàm huỷ
	//setter
	void set(double xx, double yy) { x = xx; y = yy; }
	//getter
	double layX() const { return x; }
	double layY() const { return y; }
	//operation
	void dichuyen(double dx, double dy) { x += dx; y += dy; }
	void phongto_thunho(Diem tam, double tile);
	void xoay(Diem T, double rad);
	double khoangcach(Diem a) const;
};
```

Ta sẽ bắt đầu với hàm

```
void set(double xx, double yy) { x = xx; y = yy; }
```

để khởi tạo 2 điểm x, y.

Hàm

```
void dichuyen(double dx, double dy) { x += dx; y += dy; }
void phongto_thunho(Diem tam, double tile);
void xoay(Diem T, double rad);
```

để biểu thị thay cho các phép tịnh tiến, vị tự và phép xoay quanh tâm T

Hàm

```
double khoangcach(Diem a) const;
```

để tính khoảng cách giữa hai điểm có tọa độ $(x_1;y_1)$ và $(x_2;y_2)$

- ***Xây dựng các chức năng của đối tượng điểm***

Gồm

❖ ***Tính khoảng cách***

Khoảng cách giữa 2 điểm trong không gian được tính bằng công thức

$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Từ đó ta xây dựng thuật toán

```
double Diem::khoangcach(Diem a) const
{
	double dx = this->layX() - a.layX(), dy = this->layY() - a.layY();
	return sqrt(pow(dx, 2) + pow(dy, 2));
}
```

❖ ***Phóng to thu nhỏ***

```
void Diem::phongto_thunho(Diem tam, double tile)
{
	x = (x - tam.layX()) * tile + tam.layX();
	y = (y - tam.layY()) * tile + tam.layY();
}
```

❖ ***Phép xoay hình***

```
void Diem::xoay(Diem tam, double rad)
{
	double cosa = cos(rad * PI / 180);
	double sina = sin(rad * PI / 180);
	double X = (x - tam.x) * cosa - (y - tam.y) * sina + tam.x;
	y = (x - tam.x) * sina + (y - tam.y) * cosa + tam.y;
	x = X;
}
```

b) Lớp hình

- ***Xây dựng lớp hình***

Lớp hình được xây dựng để hỗ trợ cũng như bổ sung cho lớp điểm

```
class Hinh

{

public:
	virtual void dichuyen(double dx, double dy) = NULL;
```

```
	virtual void phongto_thunho(double s) = NULL;
	virtual void xoay(double rad) = NULL;
	virtual bool DiemTrongHinh(Diem p) const = NULL;
	virtual bool DiemTrenBien(Diem p) const = NULL;
	virtual bool PhanGiao(Hinh* aS, HDC hdc, COLORREF c = RGB(77, 255, 195))
const = NULL;
	virtual void Ve(HDC hdc, COLORREF c = RGB(0, 0, 0)) const = NULL;
};
```

Virtual là một phương thức ảo được kế thừa từ lớp điểm nhằm biểu thị các phép dịch chuyển. Đặc biệt trong đó hàm

```
virtual bool PhanGiao (Hinh* aS, HDC hdc, COLORREF c = RGB(77, 255, 195)) const =
NULL;
```

dùng để kiểm tra xem hai đối tượng trong lớp hình có giao nhau hay không. Và đồng thời tô chúng bằng

không gian màu srgb là (77, 255, 195) tương ứng với màu xanh

- **Thiết kế các lớp đối tượng hình học**

c) **Lớp đa giác**

- ***Xây dựng lớp hình đa giác***

```
class DaGiac : public Hinh
{
protected:
	int dinh;
	Diem* Diems;
public:
	//hàm khởi tạo và hàm huỷ
	static Diem DiemsDefault[];
	DaGiac(int num_vers = 4, Diem* p = NULL) : dinh(num_vers) {
layDiems(num_vers, p); }
	~DaGiac() { delete[] Diems; }
	//setter
	virtual void layDiems(int n, Diem* p);
	//getter
	Diem layTam() const;
	//operation
	void dichuyen(double dx, double dy) override;
	void phongto_thunho(Diem Center, double s);
	void phongto_thunho(double s) override { phongto_thunho(layTam(), s); }
	void xoay(Diem Center, double rad);
	void xoay(double rad) override { xoay(layTam(), rad); }
	//kiemtra
	bool DiemTrongHinh(Diem p) const override;
	bool DiemTrenBien(Diem p) const override;
	//other method
	bool PhanGiao(Hinh* aS, HDC hdc, COLORREF c = RGB(156, 200, 30)) const
override;
	void Ve(HDC hdc, COLORREF c = RGB(0, 100, 100)) const override;
};
```

Trong đó

Lớp đa giác `class DaGiac` là lớp con được kế thừa từ lớp hình `class Hinh`

Các hàm

```
void dichuyen(double dx, double dy) override;
void phongto_thunho(Diem Center, double s);
void phongto_thunho(double s) override { phongto_thunho(layTam(), s); }
void xoay(Diem Center, double rad);
void xoay(double rad) override { xoay(layTam(), rad); }
```

lần lượt là các hàm biểu thị các phương thức di chuyển , phóng to_thu nhỏ dựa trên tâm và xoay hình của đa giác.

Tiếp đó hàm

```
bool DiemTrongHinh(Diem p) const override;
bool DiemTrenBien(Diem p) const override;
```

để kiểm tra điểm nằm trong hình đa giác hay nằm trên đường viền của đa giác đó.

Hàm

```
void Ve(HDC hdc, COLORREF c = RGB(0, 100, 100)) const override;
```

để vẽ ra hình đa giác lên môi trường Windows Application

Về hàm

```
bool PhanGiao(Hinh* aS, HDC hdc, COLORREF c = RGB(156, 200, 30)) const override;
```

là hàm dùng để kiểm tra sự giao nhau giữa các đối tượng. Thuật toán để tìm phần giao cũng như kiểm tra điểm nằm trong hay nằm trên đường biên của hình sẽ được đề cập trong báo cáo này ở phần sau.

d) Lớp tứ giác

- ***Xây dựng lớp tứ giác***

Lớp tứ giác bao gồm 2 lớp đối tượng là **hình chữ nhật** và **hình vuông**

❖ ***Lớp hình chữ nhật***

Lớp hình chữ nhật `class Hcn` là lớp con được kế thừa từ lớp đa giác `class DaGiac`

```
class Hcn : public DaGiac
{
public:
	//hàm khởi tạo và hàm huỷ
	Hcn(double x = 50, double y = 100, double w = 150, double h = 100) :
DaGiac(4) { setDiems(x, y, w, h); }
	~Hcn() {}
	//setter
	void setDiems(double x, double y, double w, double h);
};
```

❖ ***Lớp hình vuông***

Lớp hình vuông `class Hvuong` là lớp con được kế thừa từ lớp hình chữ nhật `class Hcn`

```
class Hvuong : public Hcn
{
public:
	//hàm khởi tạo và hàm huỷ
	Hvuong(double x = 50, double y = 100, double e = 150) : Hcn(x, y, e, e) {}
	~Hvuong() {}
};

void Hcn::setDiems(double x, double y, double w, double h)
{
	Diems[0] = Diem(x, y);
	Diems[1] = Diem(x + w, y);
	Diems[2] = Diem(x + w, y + h);
	Diems[3] = Diem(x, y + h);
}
```

e) Lớp tam giác

- ***Xây dựng lớp tam giác***

Lớp tam giác `class TamGiac` là lớp con được kế thừa từ lớp đa giác `class DaGiac`

```
class TamGiac : public DaGiac
{
public:
	//hàm khởi tạo và hàm huỷ
	static Diem DiemsDefault_Tri[3];
	TamGiac(Diem* p = NULL) : DaGiac(3) { setDiems(p); }
	~TamGiac() {}
	//setter
	void setDiems(Diem* p);

};
```

f) Lớp elipse

- ***Xây dựng lớp hình elipse***

Lớp hình elipse `class Elipse` là lớp con được kế thừa từ lớp hình `class Hinh`

```
class Elipse : public Hinh

{
protected:
	Diem T;
	double xR, yR;
public:
	//hàm khởi tạo và hàm huỷ
	Elipse(double xT = 700, double yT = 300, double a = 200, double b = 100) :
T(xT, yT), xR(a), yR(b) {};
	~Elipse() {}
	//setter
	void set(double xT, double yT) { T = Diem(xT, yT); xR = 0; yR = 0; }
```

Các hàm

```
void dichuyen(double dx, double dy) override { T.dichuyen(dx, dy); }
void phongto_thunho(double s) override { xR *= s; yR *= s; }
void xoay(double rad) override { if (rad == 90 || rad == -90) { double temp = xR; xR
= yR; yR = temp; } }
```

tương ứng là các hàm biểu thị các phương thức di chuyển tương tự như đối với hình đa giác.

Hàm

```
void Ve(HDC hdc, COLORREF c = RGB(0, 100, 100)) const override { Ellipse(hdc,
T.layX() - xR, T.layY() - yR, T.layX() + xR, T.layY() + yR); }
```

Để vẽ hình elipse lên môi trường Windows Application.

Hàm

```
bool DiemTrongHinh(Diem p) const override;
bool DiemTrenBien(Diem p) const override;
```

để xác định điểm nằm trong hay nằm trên biên của hình elipse đó.

`cuối cùng hàm`

```
bool PhanGiao(Hinh* aS, HDC hdc, COLORREF c = RGB(156, 200, 30)) const;
```

để tìm phần giao của đối tượng

Phần kiểm tra vị trí của 1 điểm so với elipse (nằm trong, nằm ngoài, nằm trên biên) sẽ được đề cập trong phần 2 của bài báo cáo.

g) Lớp tròn

- **Xây dựng lớp hình tròn**

Lớp hình tròn `class Htron` là lớp con được kế thừa từ lớp hình ellipse `class Elipse`

```
class Htron : public Elipse

{
public:
	Htron(double xT = 400, double yT = 400, double a = 150) : Elipse(xT, yT, a,
a) {};
	~Htron() {}
};
```

II. Thuật toán tìm phần giao nhau

a) Với lớp elipse

Ta có phương trình chính tắc cho hình elipse là $\left(\frac{x-x_0}{a}\right)^2 - \left(\frac{y-y_0}{b}\right)^2 = 1$

Với $x_0;y_0$ là tọa độ tâm và a;b là độ dài 2 trục của hình ellipse đó. Từ đó để so sánh vị trị tương đối của 1 điểm bất kì so với elipse, ta chỉ việc thế tọa độ điểm đó vào vế trái phương trình và so sánh với vế phải.

Nếu

Vế trái > vế phải

⇨ Vị trị của điểm nằm bên ngoài hình

Vế trái < vế phải

⇨ Vị trị của điểm nằm bên trong hình

Vế trái = vế phải

⇨ Vị trị của điểm nằm trên biên

Từ đó ta xây dựng 2 biến DiemTrongHinh và DiemTrenBien.

```
bool Elipse::DiemTrongHinh(Diem p) const
{
	double dx = p.layX() - T.layX(), dy = p.layY() - T.layY();
```

```
	return (pow(dx / xR, 2) + pow(dy / yR, 2)) < 1;
}

bool Elipse::DiemTrenBien(Diem p) const
{
	double r = 0.015;
	double dx = p.layX() - T.layX(), dy = p.layY() - T.layY();
	double x = pow(dx / xR, 2) + pow(dy / yR, 2) - 1;
	return (fabs(x) <= r);
}
```

Dựa vào việc xác định vị trí tương đối của các điểm trên elipse
Ta tiến đến việc xây dựng thuật toán để tìm phần giao

```
bool Elipse::PhanGiao(Hinh* aS, HDC hdc, COLORREF c) const
{
	bool check = 0;
	for (double i = T.layX() - xR; i <= T.layX() + xR; i++)
		for (double j = T.layY() - yR; j <= T.layY() + yR; j++)
		{
			Diem t(i, j);
			if (this->DiemTrongHinh(t) && aS->DiemTrongHinh(t))
			{
				SetPixel(hdc, i, j, c);
				check = 1;
			}
			if ((this->DiemTrenBien(t) && aS->DiemTrongHinh(t)) || (this-
>DiemTrongHinh(t) && aS->DiemTrenBien(t)) || (this->DiemTrenBien(t) && aS-
>DiemTrenBien(t)))
			{
				Ellipse(hdc, i - 0.5, j - 0.5, i + 0.5, j + 0.5);
				check = 1;
			}
		}
	return check;
```

b) Với lớp đa giác

- **Kiểm tra điểm có nằm trong hình đa giác hay không**

Đề bài cho đa giác là đa giác lồi. Nên tính lồi là thứ ta cần tận dụng để giải quyết vấn đề. Vì là đa giác lồi nên nó chỉ giao với một đường thẳng bất kỳ tại không quá hai điểm (chính xác thì là 2 hoặc là 0).

Chú ý rằng là, "giao" ở đây là khi ta đi vòng quanh đa giác, ta đi từ một bên của đường thẳng sang bên kia của nó (tham khảo Chuyên đề hình học - Đỗ Mạnh Dũng - VNOI).

Nếu điểm cần xét không nằm trên biên thì sẽ có 2 điều xảy ra

+Điểm sẽ nằm ngoài đa giác nếu tổng số giao điểm là số chẵn

+Điểm sẽ nằm trong đa giác nếu tổng số giao điểm là số lẻ

_Để dễ hình dung ta giả sử điểm cần kiểm tra là $P_0(x_0, y_0)$. Gọi các đỉnh của đa giác là $P_1 \dots P_n$ có tọa độ tương ứng là $(x_1, y_1) \dots (x_n, y_n)$ với n là số đỉnh của đa giác.

Phương trình đường thẳng chính tắc đi qua 2 đỉnh của P_m, P_l bất kì của đa giác là

$$\frac{x - x_m}{x_l - x_m} = \frac{y - y_m}{y_l - y_m}$$

Nếu có điểm $P_0(x_0, y_0)$ nằm trên đoạn $P_m P_l$

Thì $x_0 = \frac{(y_0 - y_m)(x_l - x_m)}{y_l - y_m} + x_m$

Từ ý tưởng đó ta sẽ xây dựng hàm DiemTrongHinh và thực hiện như bên dưới

```
bool DaGiac::DiemTrongHinh(Diem p) const
{
	int count = 0;
	double x_in;
	Diem p1, p2;
	for (int i = 0; i < dinh; i++) // chạy trên các đỉnh
	{
		p1 = Diems[i]; // đỉnh thứ nhất
		p2 = Diems[(i + 1) % dinh];// đỉnh kế tiếp
		if (p.layY() > MIN(p1.layY(), p2.layY()) && p.layY() <= MAX(p1.layY(),
p2.layY()))
			// kiểm tra xem là cái p có nằm giữ 2 y của p1 và p2
			if (p.layX() <= MAX(p1.layX(), p2.layX()))
				// xét điểm x xem có đang nằm giữ 2 điểm x của p1 và p2
				if (p1.layY() != p2.layY())
					// xét 2 điểm y có bằng không?
				{
					// nếu thỏa các điều kiện trên thì ta sẽ có biến
x_in
					x_in = (p.layY() - p1.layY()) * (p2.layX() -
p1.layX()) / (p2.layY() - p1.layY()) + p1.layX();
					if (p1.layX() == p2.layX() || p.layX() < x_in)
						// xét từ trái qua xem cái điểm x_in có đụng
hình ko?
						count++;
				}
	}
	return (count % 2 != 0);
}
```

- **Kiểm tra điểm có nằm trên biên của hình đa giác hay không**

Ta chỉ cần so sánh tổng độ dài từ điểm đó lần lượt đến 2 đỉnh còn lại của đa giác so với độ dài 1 cạnh tạo bởi 2 điểm đó. Nếu 2 kết quả ta so sánh không bằng nhau thì ta kết luận điểm không nằm trên biên và ngược lại

Từ đó ta xây dựng hàm DiemTrenBien trong lớp đa giác

```
bool DaGiac::DiemTrenBien(Diem p) const
{
	double  d;
	double r = 0.03;
	for (int i = 0; i < dinh; i++)
	{
		d = p.khoangcach(Diems[i]) + p.khoangcach(Diems[(i + 1) % dinh]) -
Diems[i].khoangcach(Diems[(i + 1) % dinh]);
		// d = ap + bp - ab
		if (fabs(d) <= r) //fabs là trị tuyệt đối
			return true;
	}
	return false;
	}
```

III. Thiết kế ứng dụng vẽ bằng Windows Application Environment

Khởi đầu

1) **Tạo đối tượng**

Ta tạo các biến để lưu trữ tọa độ của đối tượng điểm

```
static Diem p[3] = { Diem(200,200), Diem(100, 600), Diem(600,450) };
static Diem p2[3] = { Diem(400,200), Diem(600, 450), Diem(1000,450) };
static Diem plg1[5] = { Diem(400,200), Diem(600, 300), Diem(450, 350), Diem(300,
400), Diem(200, 350) };
static Diem plg2[6] = { Diem(300,200), Diem(750, 250), Diem(800, 300), Diem(600,
400), Diem(400, 450), Diem(200, 350) };
```

Kế tiếp

Ta tạo 2 mảng tương ứng với 2 hình để lưu trữ các đối tượng hình học bao gồm hình đa giác, hình chữ nhật, hình vuông, hình tam giác, hình elipse, hình tròn.

```
static Hinh* aS[] =
{
		new DaGiac(5, plg1),
		new Hcn(200, 100, 300, 200),
		new Hvuong(100, 100, 200),
		new TamGiac(p),
		new Elipse(300,250,200,100),
		new Htron(200, 200, 150),
```

```
};
static Hinh* aS2[] =
{
		new DaGiac(6, plg2),
		new Hcn(600, 100, 300, 200),
		new Hvuong(700, 100, 200),
		new TamGiac(p2),
		new Elipse(800,250,200,100),
		new Htron(600, 200, 150)

};
```

Tạo phương thức để thao tác

2) Thao tác trên cửa sổ Windows Application bằng con trỏ chuột

Để nhận diện được con trỏ chuột trên màn hình. Ta sẽ bắt đầu với ý tưởng kiểm tra vị trí của con trỏ chuột trên cửa sổ Drawing, thông qua hàm DiemTrongHinh().

Sau đó ta sẽ xác định xem vị trí của con trỏ chuột có thuộc bất kì đối tượng nào trong 2 đối tượng hình học ta đã tạo ban đầu hay không.

Nếu có

Con trỏ chuột sẽ được thay đổi và ngay lập tức sẽ vẽ lại đối tượng hình học tương tự như với đối tượng hình học được xác định, đồng thời di chuyển theo vị trí của con trỏ chuột.

Khi thả chuột, vị trí con trỏ sau đó sẽ được cập nhật, hình thay thế khi di chuyển và đối tượng tại vị trí ban đầu sẽ được xoá và được vẽ lại ở vị trí hiện hành.

Nếu không

Hiển nhiên không có gì xảy ra.

Từ ý tưởng đó ta sẽ thực hiện như bên dưới

```
case WM_LBUTTONDOWN:
		xc = GET_X_LPARAM(lParam); yc = GET_Y_LPARAM(lParam);
		if (aS[h]->DiemTrongHinh(Diem(xc, yc)))
		{
			hdc = GetDC(hWnd);
			SelectObject(hdc, hPenDot);
			SetROP2(hdc, R2_NOTXORPEN);
			aS[h]->Ve(hdc);
			SetCursor(hCursorMove);
			SetCapture(hWnd);
		}
		else
			if (aS2[h2]->DiemTrongHinh(Diem(xc, yc)))
			{
				hdc = GetDC(hWnd);
				SelectObject(hdc, hPenDot);
				SetROP2(hdc, R2_NOTXORPEN);
				aS2[h2]->Ve(hdc);
				SetCursor(hCursorMove);
				SetCapture(hWnd);
			}
```

```
		break;
case WM_MOUSEMOVE:
		if (GetCapture() == hWnd)
		{
			px = xc; py = yc;
			xc = GET_X_LPARAM(lParam); yc = GET_Y_LPARAM(lParam);
			dx = xc - px; dy = yc - py;
			if (pS->DiemTrongHinh(Diem(px, py)))
			{
				hdc = GetDC(hWnd);
				SelectObject(hdc, hPenDot);
				SetROP2(hdc, R2_NOTXORPEN);
				aS[h]->Ve(hdc);

				aS[h]->dichuyen(dx, dy);
				aS[h]->Ve(hdc);
			}
			else
				if (pS2->DiemTrongHinh(Diem(px, py)))
				{
					hdc = GetDC(hWnd);
					SelectObject(hdc, hPenDot);
					SetROP2(hdc, R2_NOTXORPEN);
					aS2[h2]->Ve(hdc);
					aS2[h2]->dichuyen(dx, dy);
					aS2[h2]->Ve(hdc);
				}
		}
		break;
case WM_LBUTTONUP:
		if (GetCapture() == hWnd)
		{
			ReleaseCapture();
			InvalidateRect(hWnd, NULL, TRUE);
		}
		break;
```

3) Thao tác trên cửa sổ Windows Application bằng cách phím chức năng

```
case WM_KEYDOWN:
		switch (wParam)
		{
		case VK_TAB:
			if (GetAsyncKeyState(VK_CONTROL))
			{
				if (++h2 == n2) h2 = 0;
				pS2 = aS2[h2];
				InvalidateRect(hWnd, NULL, TRUE);
				break;
			}
			else
			{
				if (++h == n) h = 0;
```

```
                pS = aS[h];
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
case VK_OEM_PLUS:
        if (GetAsyncKeyState(VK_CONTROL))
        {
                pS2->phongto_thunho(1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
        else
        {
                pS->phongto_thunho(1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
case 0x4C:
        if (GetAsyncKeyState(VK_CONTROL))
        {
                pS2->xoay(-45);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
        else
        {
                pS->xoay(-45);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
case 0x52:
        if (GetAsyncKeyState(VK_CONTROL))
        {
                pS2->xoay(45);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
        else
        {
                pS->xoay(45);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
case VK_OEM_MINUS:
        if (GetAsyncKeyState(VK_CONTROL))
        {
                pS2->phongto_thunho(1 / 1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
        else
        {
                pS->phongto_thunho(1 / 1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
        }
case VK_LEFT:
        if (GetAsyncKeyState(VK_CONTROL))
```

```
		{
			pS2->dichuyen(-10, 0);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
		else
		{
			pS->dichuyen(-10, 0);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
	case VK_RIGHT:
		if (GetAsyncKeyState(VK_CONTROL))
		{
			pS2->dichuyen(10, 0);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
		else
		{
			pS->dichuyen(10, 0);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
	case VK_UP:
		if (GetAsyncKeyState(VK_CONTROL))
		{
			pS2->dichuyen(0, -10);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
		else {
			pS->dichuyen(0, -10);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
	case VK_DOWN:
		if (GetAsyncKeyState(VK_CONTROL))
		{
			pS2->dichuyen(0, 10);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
		else {
			pS->dichuyen(0, 10);
			InvalidateRect(hWnd, NULL, TRUE);
			break;
		}
	}
```

- **Đối với đối tượng hình màu tím**

_Ta sử dụng phím tab để thay đổi hình

_Phím cộng trừ tương ứng để phóng to lẫn thu nhỏ đối tượng và

_Hệ phím mũi tên để điều hướng các đối tượng hình học di chuyển qua trái hoặc phải , lên hoặc xuống

_Dùng phím L và R để xoay trái và phải một góc 45 độ

- **Đối với đối tượng hình màu vàng**

_Ta chỉ cần giữ thêm phím ctrl và làm tương tự như với đối tượng màu tím

4) **Bước cuối cùng là vẽ hình**

```
hdc = BeginPaint(hWnd, &ps);

		HPEN hpen;
		GetClientRect(hWnd, &rt);
		swprintf_s(szInfor);
		DrawText(hdc, szInfor, lstrlen(szInfor), &rt, DT_LEFT);
		SelectObject(hdc, hbr1);
		SelectObject(hdc, penBorder1);
		pS->Ve(hdc);
		SelectObject(hdc, hbr2);
		SelectObject(hdc, penBorder2);
		pS2->Ve(hdc);
		SelectObject(hdc, penBorder);
		if (pS2->PhanGiao(pS, hdc))
			swprintf_s(szItsMessage, L"Giao nhau");
		else
			swprintf_s(szItsMessage, L"Không Giao nhau");
		DrawText(hdc, szItsMessage, lstrlen(szItsMessage), &rt, DT_CENTER |
DT_TOP);
		EndPaint(hWnd, &ps);
```

Trong đó

Với hàm đã khởi tạo từ trước

```
case WM_CREATE:
		GetClientRect(hWnd, &rt);
		hbr1 = CreateSolidBrush(RGB(204, 153, 255));
		hbr2 = CreateSolidBrush(RGB(255, 255, 128));
		hCursorMove = LoadCursor(NULL, IDC_SIZEALL);
		hCursorCross = LoadCursor(NULL, IDC_CROSS);
		penBorder = CreatePen(PS_SOLID, 2, RGB(0, 0, 0));
		penBorder1 = CreatePen(PS_SOLID, 2, RGB(89, 0, 179));
		penBorder2 = CreatePen(PS_SOLID, 2, RGB(128, 128, 0));
		break;
```

Hàm này được dùng để gán màu cho các đối tượng ta muốn vẽ. Từ đó ta cũng có thể đổi màu các đối tượng hình học tùy theo ý thích. Miễn là màu đó có tồn tại trên không gian màu srgb.

Câu lệnh

```
if (pS2->PhanGiao(pS, hdc))
			swprintf_s(szItsMessage, L"Giao nhau");
		else
			swprintf_s(szItsMessage, L"Không Giao nhau");
```

để xuất ra thông báo " Giao nhau " nếu 2 đối tượng hình học có giao nhau và ngược lại

Và cuối cùng là xuất hình lên cửa sổ ứng dụng

thì

Hình 1 sẽ được đại diện bằng hàm hbr1 và có không gian màu Srgb là (204,152,255) tương đương với màu tím

Hình 2 sẽ được đại diện bằng hàm hbr2 và có không gian màu Srgb là (255,255,128) tương đương với màu vàng

Màu viền tương ứng của 2 hình sẽ được tô bằng hàm `penBorder1` và `penBorder2` cùng với không gian màu sgrb là (89,0,179) và (128,128,0).

Phần giao (nếu có tồn tại) của hai đối tượng được thực hiện tô màu nền ngay trong hàm pS2->phangiao (pS, hdc) cùng với đường viền được tô màu bằng hàm `penBorder` với không gian màu srgb là (0,0,0) tương ứng với màu đen và phần giao thì đã được đề cập ở mục 1b) sẽ được tô bằng màu xanh