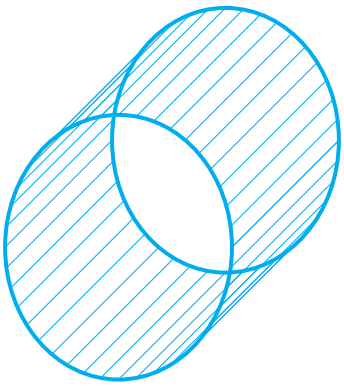
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM

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 phụ trách: Nguyễn Ngọc Long

TP.Hồ Chí Minh - Tháng 7, 2020

Nhóm sinh viên thực hiện

- Đinh Anh Huy - MSSV : 18110103
- Trần Tấn Phong - MSSV : 18110181

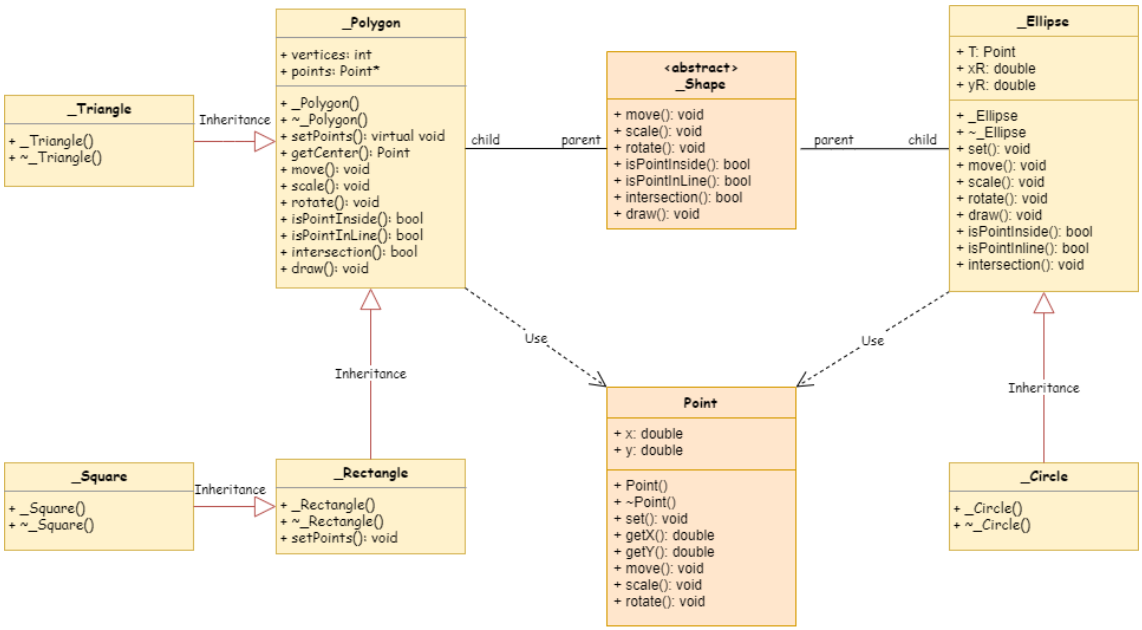
Mục lục

1 Đề bài

Cho các loại đối tượng: hình tròn, hình ellipse, đa giác, hình chữ nhật, hình vuông, hình tam giác. Viết chương trình ứng dụng cho phép tạo (hoặc nhập) hai hình thuộc một trong các hình kể trên. In thông báo cho biết hai hình có giao nhau không, nếu có tô màu phần giao và tô đậm đường biên của phần giao. Người sử dụng có thể bấm các phím mũi tên để di chuyển một trong hai hình, phím +, −, để phóng to thu nhỏ một trong hai hình. Có thể lập trình trong môi trường Window. Sử dụng đa hình để giải quyết vấn đề trên.

2 Sơ đồ lớp

Dựa vào yêu cầu bài tập lớn, ta xây dựng được các lớp có sơ đồ như sau



3 Thiết kế lớp

3.1 Lớp điểm (class Point)

```
class Point
{
  double x, y;
public:
  //constructor and destructor
  Point(double xx = 0, double yy = 0) : x(xx), y(yy) {}
  ~Point() {}
  //setter
  void set(double xx, double yy) { x = xx; y = yy; }
  //getter
  double getX() const { return x; }
  double getY() const { return y; }
  //operation
  void move(double dx, double dy) { x += dx; y += dy; }
  void scale(Point center, double tile);
  void rotate(Point T, double rad);
  //paint
  void mark(HDC hdc, int size = 5, COLORREF c = RGB(0, 0, 0));
  //other method
  double distance(Point a) const;
  Point setVector(Point b) const;
};
```

Trong đó

- ```
  void move(double dx, double dy);
  void scale(Point center, double tile);
  void rotate(Point T, double rad);
  ```

  là các phương thức thực hiện các thao tác tịnh tiến, vị tự, phép quay tâm T (đơn vị độ) của một điểm.

- ```
  void mark(HDC hdc, int size = 5, COLORREF c = RGB(0, 0, 0));
  ```

  là phương thức thực hiện đánh dấu (+) một điểm trên cửa sổ ứng dụng.

- ```
  double distance(Point a) const;
  ```
  ```

là phương thức thực hiện việc tính khoảng cách giữa hai điểm. Khoảng cách giữa hai điểm $A(x_1, y_1)$ và $B(x_2, y_2)$ được tính thông qua công thức

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

- `Point setVector(Point b) const;`

là phương thức thực hiện lấy vector giữa hai điểm. Vector $\overrightarrow{AB}$ được xác định bằng

$$\overrightarrow{AB} = (x_B - x_A; y_B - y_A).$$

3.2 Lớp đối tượng (class _Shape)

3.2.1 Xây dựng lớp

Ta xây dựng một lớp cơ sở trừu tượng (class _Shape)

```
class _Shape
{
public:
  virtual void move(double dx, double dy) = NULL;
  virtual void scale(double s) = NULL;
  virtual void rotate(double rad) = NULL;
  virtual bool isPointInside(Point p) const = NULL;
  virtual bool isPointInLine(Point p) const = NULL;
  virtual bool intersection(_Shape* aS, HDC hdc, COLORREF c = RGB(156,
   200, 30)) const = NULL;
  virtual void draw(HDC hdc, COLORREF c = RGB(0, 100, 100)) const = NULL
   ;
};
```

Trong đó

```
virtual bool intersection(_Shape* aS, HDC hdc,
    COLORREF c = RGB(156, 200, 30)) const = NULL;
```

là phương thức ảo thuần tuý thực hiện việc kiểm tra hai đối tượng hình học đang xét có giao nhau hay không. Với phương thức này, ta có thuật toán tổng quát tìm phần giao cho hai đối tượng hình học bất kỳ.

3.2.2 Thuật toán tìm phần giao giữa hai đối tượng hình bất kỳ

Để tìm phần giao giữa hai đối tượng bất kỳ, ta sẽ xét các điểm trên hình chữ nhật nhỏ nhất chứa một trong hai đối tượng. Nếu điểm đang xét đều nằm trong cả hai đối tượng thì ta sẽ thay đổi màu của điểm đó bằng hàm `SetPixel()`. Tương tự cho phần biên của phần giao, nếu điểm đang xét thuộc cả ba trường hợp sau thì ta sẽ vẽ một hình tròn bán kính 0.5 tại điểm đó:

- Nếu điểm nằm trên biên của đối tượng 1 và nằm bên trong đối tượng 2,
- Nếu điểm nằm bên trong đối tượng 1 và nằm trên biên của đối tượng 2,
- Nếu điểm nằm trên biên của cả hai dối tượng.

Sau khi xét hết hình chữ nhật ta sẽ có phần giao cần tìm với phần viền được tô đậm.

3.3 Lớp hình Ellipse (class _Ellipse)

3.3.1 Xây dựng lớp

```
class _Ellipse : public _Shape
{
protected:
  Point T;
  double xR, yR;
public:
  //constructor and destructor
  _Ellipse(double xT = 700, double yT = 300, double a = 200, double b =
   100) : T(xT, yT), xR(a), yR(b) {};
  ~_Ellipse() {}
  //setter
  void set(double xT, double yT) { T = Point(xT, yT); xR = 0; yR = 0; }
  //operation
  void move(double dx, double dy) override { T.move(dx, dy); }
  void scale(double s) override { xR *= s; yR *= s; }
  void rotate(double rad) override { if (rad == 90 || rad == -90) {
   double temp = xR; xR = yR; yR = temp; } }
```

```
16  void draw(HDC hdc, COLORREF c = RGB(0, 100, 100)) const override {
     Ellipse(hdc, T.getX() - xR, T.getY() - yR, T.getX() + xR, T.getY() +
     yR); }
17  //check
18  bool isPointInside(Point p) const override;
19  bool isPointInLine(Point p) const override;
20  //intersection
21  bool intersection(_Shape* aS, HDC hdc, COLORREF c = RGB(156, 200, 30))
      const override;
22 };
```

Trong đó

- `class _Ellipse` là lớp con được kế thừa từ lớp cha `class _Shape`.
- `void move(double dx, double dy) override;`

  là phương thức tịnh tiến hình Ellipse bằng cách tịnh tiến *tâm* T của hình.
- `void scale(double s) override;`

  là phương thức phóng to (thu nhỏ) hình Ellipse bằng cách nhân độ dài hai bán trục với tỷ lệ s.
- `void rotate(double rad) override;`

  là phương thức thực hiện quay hình Ellipse với góc quay có đơn vị độ. Ở phương thức này, ta chỉ xét phép quay với góc 90° vì ta sử dụng hàm vẽ Ellipse mặc định của *môi trường Windows* nên chỉ có thể vẽ các hình Ellipse có hai bán trục song song với hai trục toạ độ.
- `bool isPointInside(Point p) const override;`

  là phương thức kiểm tra một điểm nằm bên trong hình Ellipse (không tính phần biên).
- `bool isPointInLine(Point p) const override;`

  là phương thức kiểm tra một điểm nằm trên đường biên của hình Ellipse.

- ```
  bool intersection(_Shape* aS, HDC hdc,
      COLORREF c = RGB(156, 200, 30)) const override;
  ```

  là phương thức kiểm tra hình Ellipse và một đối tượng hình học giao nhau.

3.3.2 Thuật toán kiểm tra vị trí tương đối giữa một điểm và hình Ellipse

Để kiểm tra một điểm $P(x_P, y_P)$ có nằm trong hình Ellipse đang xét hay không, ta sử dụng phương trình chính tắc của hình Ellipse

$$\left(\frac{x - x_0}{a}\right)^2 + \left(\frac{y - y_0}{b}\right)^2 = 1 \tag{1}$$

trong đó (x_0, y_0) là toạ độ tâm của hình Ellipse, a, b là độ dài hai bán trục.
Thay toạ độ của điểm P vào vế trái của phương trình (1) ta được giá trị e, ta so kết quả vừa tính được với 1.

- Nếu $e < 1$ thì P nằm trong hình Ellipse.
- Nếu $e = 1$ thì P nằm trên biên của hình Ellipse.
- Nếu $e > 1$ thì P nằm ngoài hình Ellipse.

Áp dụng thuật toán này, ta xây dựng được hai hàm `isPointInside()` và `isPointInLine()`

```
bool _Ellipse::isPointInside(Point p) const
{
  double r = 0.025;
  double dx = p.getX() - T.getX(), dy = p.getY() - T.getY();
  return (pow(dx / xR, 2) + pow(dy / yR, 2) - 1) < r;
}
bool _Ellipse::isPointInLine(Point p) const
{
  double r = 0.03;
  double dx = p.getX() - T.getX(), dy = p.getY() - T.getY();
  double x = pow(dx / xR, 2) + pow(dy / yR, 2) - 1;
  return (fabs(x) <= r);
}
```

Vì các giá trị `dx, dy, x` ở trên thuộc kiểu `double` nên khi so sánh, ta so sánh với khoảng nhỏ vừa đủ chứa giá trị cần so sánh để tăng độ chính xác của thuật toán.
```

3.4 Lớp hình Tròn (class _Circle)

```
class _Circle : public _Ellipse
{
public:
  _Circle(double xT = 400, double yT = 400, double a = 150) : _Ellipse(
   xT, yT, a, a) {};
  ~_Circle() {}
};
```

Trong đó `class _Circle` được kế thừa từ `class _Ellipse`.

3.5 Lớp hình Đa giác (class _Polygon)

3.5.1 Xây dựng lớp

```
class _Polygon : public _Shape
{
protected:
  int vertices;
  Point* points;
public:
  //Constructor and Destructor
  static Point pointsDefault[];
  _Polygon(int num_vers = 4, Point* p = NULL) : vertices(num_vers) {
   setPoints(num_vers, p); }
  ~_Polygon() { delete[] points; }
  //setter
  virtual void setPoints(int n, Point* p);
  //getter
  Point getCenter() const;
  //operation
  void move(double dx, double dy);
  void scale(Point Center, double s);
  void scale(double s) { scale(getCenter(), s); }
  void rotate(Point Center, double rad);
  void rotate(double rad) { rotate(getCenter(), rad); }
  //check
```

```
22   bool isPointInside(Point p) const;
23   bool isPointInLine(Point p) const;
24   //other method
25   bool intersection(_Shape* aS, HDC hdc, COLORREF c = RGB(156, 200, 30))
       const;
26   void draw(HDC hdc, COLORREF c = RGB(0, 100, 100)) const;
27 };
```

Trong đó

- `class _Polygon` là lớp con được kế thừa từ lớp cha `class _Shape`.
- ```
  void move(double dx, double dy);
  ```

  là phương thức di chuyển hình đa giác theo vector (dx, dy) bằng cách di chuyển từng đỉnh của hình đa giác.
- ```
  void scale(double s);
  void rotate(double rad);
  ```

  là các phương thức thực hiện các thao tác phóng to (thu nhỏ) và quay hình (góc quay có đơn vị độ) đa giác theo tâm của hình đa giác, với toạ độ tâm được xác định bằng cách lấy trung bình toạ độ của các đỉnh.
- ```
  bool isPointInside(Point p) const override;
  ```

  là phương thức kiểm tra một điểm nằm bên trong hình đa giác.
- ```
  bool isPointInLine(Point p) const override;
  ```

  là phương thức kiểm tra một điểm nằm trên đường biên của hình đa giác.
- ```
  bool intersection(_Shape* aS, HDC hdc,
       COLORREF c = RGB(156, 200, 30)) const override;
  ```

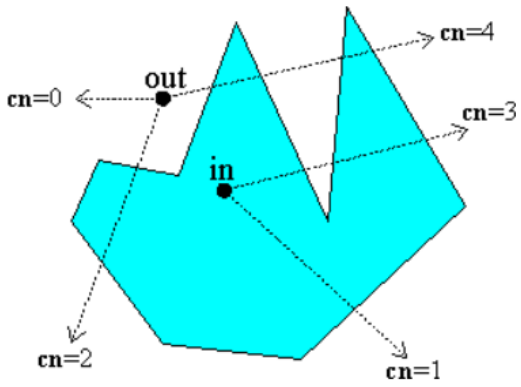
  là phương thức kiểm tra hình đa giác và một đối tượng hình học giao nhau.
  ```

3.5.2 Thuật toán kiểm tra một điểm nằm trong hình đa giác

Thuật toán Crossing Number

Để kiểm tra một điểm nằm bên trong hay bên ngoài một đa giác bất kỳ, ta đi kiểm tra số lần một tia (bắt đầu từ điểm và đi theo bất kỳ hướng cố định nào) giao nhau với các cạnh của đa giác. Nếu điểm nằm ở bên trong của đa giác, số giao điểm sẽ là một số lẻ. Ngược lại, số giao điểm sẽ là số chẵn.

Bằng trực quan, nếu một điểm di chuyển dọc theo một tia từ vị trí đang xét đến vô cùng và nếu nó đi qua giao điểm giữa tia và cạnh đa giác (có thể nhiều lần), sau đó nó luân phiên đi từ bên ngoài vào bên trong và từ bên trong ra bên ngoài, v.v... Cuối cùng điểm đó luôn di chuyển ra khỏi đa giác. Vì vậy, nếu một điểm nằm bên trong đa giác thì dãy các giao điểm phải là: in > out > ... > in > out, và có một số lẻ lần như vậy. Tương tự, nếu một điểm nằm bên ngoài đa giác, ta sẽ có số chẵn lần dãy: out > in > ... > in > out.



Ta xét một trường hợp đơn giản của thuật toán *Crossing Number* là chọn tia bắt đầu tại điểm $P(x_p, y_p)$ cần xét và phương song song với trục hoành và cùng chiều dương với trục hoành. Trong trường hợp này ta dễ dàng tìm được giao điểm giữa tia với các cạnh của đa giác là điểm $P_0(x_0, y_p)$. Để đếm tổng số giao điểm (cn), ta chỉ cần lặp lại thuật toán này qua tất cả các cạnh của đa giác, kiểm tra sự tồn tại của giao điểm và tăng *cn* khi xảy ra. Ta có thể xác định được giá trị của x_0 bằng cách xét hai điểm $P_1(x_1, y_1)$ và $P_2(x_2, y_2)$ là hai đỉnh của một cạnh của hình đa giác. Khi đó ta có phương trình chính tắc của đường thẳng P_1P_2

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}$$

Giả sử tồn tại giao điểm $P(x_0, y_p)$ giữa tia và đoạn P_1P_2, thì khi đó ta có giá trị x_0 được xác định bằng

$$x_0 = \frac{(y_p - y_1)(x_2 - x_1)}{y_2 - y_1} + x_1$$

Ngoài ra, bài toán kiểm tra giao nhao phải xử lý các trường hợp đặc biệt là tia đi qua đỉnh của đa giác hoặc một cạnh song song với trục hoành. Để xử lý các trường hợp này, ta đưa ra các quy tắc:

- Đối với cạnh hướng lên, chỉ lấy điểm đầu, không lấy điểm cuối.
- Đối với cạnh hướng xuống, chỉ lấy điểm cuối, không lấy điểm đầu.
- Không xét các cạnh song song với trục hoành.
- Giao điểm phải nằm bên phải điểm đang xét.

Lưu ý, với quy tắc thứ 4 dẫn đến việc các điểm nằm trên các cạnh bên phải của hình đa giác sẽ được xem là nằm ngoài đa giác và các điểm nằm trên các cạnh bên trái của hình đa giác sẽ được xem là nằm bên trong.

Áp dụng thuật toán này, ta xây dựng được hàm `isPointInSide()` cho `class _Polygon`

```
1  bool _Polygon::isPointInside(Point p) const
2  {
3    int count = 0;
4    double xinters;
5    Point p1, p2;
6    for (int i = 0; i < vertices; i++)
7    {
8      p1 = points[i];
9      p2 = points[(i + 1) % vertices];
10     if (p.getY() > MIN(p1.getY(), p2.getY()) && p.getY() <= MAX(p1.getY
     (), p2.getY()))
11       if (p.getX() <= MAX(p1.getX(), p2.getX()))
12         if (p1.getY() != p2.getY())
13         {
14           xinters = (p.getY() - p1.getY()) * (p2.getX() - p1.getX()) / (
     p2.getY() - p1.getY()) + p1.getX();
15           if (p1.getX() == p2.getX() || p.getX() < xinters)
```

```
            count++;
        }
    }
    return (count % 2 != 0);
}
```

3.5.3 Thuật toán kiểm tra một điểm nằm trên biên của hình đa giác

Để kiểm tra một điểm nằm trên biên của hình đa giác, ta đi kiểm tra một điểm có nằm trên một đoạn thẳng hay không. Xét một cạnh bất kỳ của đa giác, ta tìm được khoảng cách giữa điểm đang xét đến hai đầu của đoạn thẳng, sau đó ta so sánh với độ dài cạnh:

- Nếu $AP + BP = AB$ thì điểm P nằm trên đoạn thẳng AB,
- Ngược lại thì điểm P nằm ngoài đoạn thẳng AB,

trong đó A, B là các đỉnh của đa giác, P là điểm đang xét.

Áp dụng thuật toán này, ta xây dựng được hàm `isPointInLine()` cho `class _Polygon`

```
bool _Polygon::isPointInLine(Point p) const
{
    double d;
    double r = 0.04;
    for (int i = 0; i < vertices; i++)
    {
        d = p.distance(points[i]) + p.distance(points[(i + 1) % vertices]) -
        points[i].distance(points[(i + 1) % vertices]);
        if (fabs(d) <= r)
            return true;
    }
    return false;
}
```

3.6 Lớp hình Tam giác (class _Triangle)

```
class _Triangle : public _Polygon
{
```

```
public:
  //constructor and destructor
  static Point pointsDefault_Tri[3];
  _Triangle(Point* p = NULL) : _Polygon(3) { setPoints(p); }
  ~_Triangle() {}
  //setter
  void setPoints(Point* p);
};
```

Trong đó, `class _Triangle` được kế thừa từ `class _Polygon`.

3.7 Lớp Hình Chữ Nhật (class _Rectangle)

```
class _Rectangle : public _Polygon
{
public:
  //constructor and destructor
  _Rectangle(double x = 50, double y = 100, double w = 150, double h =
    100) : _Polygon(4) { setPoints(x, y, w, h); }
  ~_Rectangle() {}
  //setter
  void setPoints(double x, double y, double w, double h);
};
```

Trong đó, `class _Rectangle` được kế thừa từ `class _Polygon`.

3.8 Lớp Hình Vuông (class _Square)

```
class _Square : public _Rectangle
{
public:
  //constructor and destructor
  _Square(double x = 50, double y = 100, double e = 150) : _Rectangle(x,
    y, e, e) {}
  ~_Square() {}
};
```

Trong đó, `class _Square` được kế thừa từ `class _Polygon`.

4 Thiết kế ứng dụng vẽ hình trong môi trường Windows

4.1 Tạo đối tượng và vẽ hình

4.1.1 Tạo đối tượng

```
static Point p[3] = { Point(200,200), Point(100, 450), Point(500,450) };
static Point p2[3] = { Point(800,200), Point(600, 450), Point(1000,450) };
static Point plg1[5] = { Point(200,200), Point(350, 200), Point(400, 300), Point(300, 300), Point(200, 350) };
static Point plg2[6] = { Point(650,250), Point(750, 250), Point(800, 350), Point(700, 400), Point(600, 350), Point(600, 300) };
static _Square A;
static _Shape* aS[] = {
    new _Polygon(5, plg1),
    new _Ellipse(300,250,200,100),
    new _Rectangle(200, 100, 300, 200),
    new _Circle(200, 200, 150),
    new _Triangle(p),
    new _Square(100, 100, 200)
};
static _Shape* aS2[] = {
    new _Polygon(6, plg2),
    new _Ellipse(800,250,200,100),
    new _Rectangle(600, 100, 300, 200),
    new _Circle(600, 200, 150),
    new _Triangle(p2),
    new _Square(700, 100, 200)
};
static int h = 0, h2 = 0;
const int n = sizeof(aS) / sizeof(aS[0]), n2 = sizeof(aS2) / sizeof(aS2[0]);
```

```
24 static _Shape* pS = aS[h];
25 static _Shape* pS2 = aS2[h2];
```

Trong đó

- `_Shape* aS[]` và `_Shape* aS2[]` là hai mảng con trỏ kiểu `_Shape` chứa các loại đối tượng: hình tròn, hình ellipse, đa giác, hình chữ nhật, hình vuông, tam giác.
- `h, h2` là hai biến dùng để lưu chỉ số hiện hành của hai đối tượng thuộc hai mảng khác nhau khi xuất ra màn hình.
- `n, n2` là hai biến dùng để lưu độ dài hai mảng `aS, aS2`.
- `_Shape* pS` và `_Shape* pS2` là hai biến con trỏ kiểu `_Shape` dùng để trỏ đến đối tượng hiện hành của hai mảng.

4.1.2 Vẽ hình

```
1  //draw object 1
2  SelectObject(hdc, hbr1);
3  SelectObject(hdc, penBorder1);
4  pS->draw(hdc);
5  //draw object 2
6  SelectObject(hdc, hbr2);
7  SelectObject(hdc, penBorder2);
8  pS2->draw(hdc);
9  //intersect
10 SelectObject(hdc, penBorder);
11 if (pS2->intersection(pS, hdc))
12     swprintf_s(szItsMessage, L"Intersect!");
13 else
14     swprintf_s(szItsMessage, L"Not intersect!");
15 DrawText(hdc, szItsMessage, lstrlen(szItsMessage), &rt, DT_CENTER |
   DT_TOP);
```

Trong đó

- Đối tượng 1 được vẽ với màu nền là `hbr1` (màu xanh dương) và màu viền là `penBorder1` .

- Đối tượng 2 được vẽ với màu nền là `hbr2`(màu hồng) và màu viền là `penBorder2`.
- Phần giao (nếu có) của hai đối tượng được thực hiện tô màu nền(màu xanh lá) ngay trong hàm `pS2->intersection(pS, hdc)` và phần viền được vẽ với màu `penBorder`.

4.2 Các thao tác trên cửa sổ ứng dụng

4.2.1 Thao tác di chuyển đối tượng bằng cách kéo thả chuột

Khi chuột trái được nhấn, bằng cách sử dụng hàm `isPointInside()` để kiểm tra xem vị trí con trỏ lúc đó có nằm trong hai đối tượng hay không. Nếu có, con trỏ chuột thay đổi thành `hCursorMove` và một hình tương tự đối tượng được xác định được vẽ ra bằng bút vẽ `hPenDot`, sau đó hình này sẽ di chuyển theo vị trí của con trỏ chuột. Khi thả chuột, vị trí con trỏ hiện hành sẽ được lấy, 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.

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

```
22 case WM_MOUSEMOVE:
23     if (GetCapture() == hWnd)
24         {
25         px = xc; py = yc;
26         xc = GET_X_LPARAM(lParam); yc = GET_Y_LPARAM(lParam);
27         dx = xc - px; dy = yc - py;
28         if (pS->isPointInside(Point(px, py)))
29         {
30             hdc = GetDC(hWnd);
31             SelectObject(hdc, hPenDot);
32             SetROP2(hdc, R2_NOTXORPEN);
33             aS[h]->draw(hdc);
34             aS[h]->move(dx, dy);
35             aS[h]->draw(hdc);
36         }
37         if (pS2->isPointInside(Point(px, py)))
38         {
39             hdc = GetDC(hWnd);
40             SelectObject(hdc, hPenDot);
41             SetROP2(hdc, R2_NOTXORPEN);
42             aS2[h2]->draw(hdc);
43             aS2[h2]->move(dx, dy);
44             aS2[h2]->draw(hdc);
45         }
46     }
47     break;
48 case WM_LBUTTONUP:
49     if (GetCapture() == hWnd)
50     {
51         ReleaseCapture();
52         InvalidateRect(hWnd, NULL, TRUE);
53     }
54     break;
```

4.2.2 Thao tác di chuyển đối tượng bằng bàn phím

Ta sử dụng các phím mũi tên để di chuyển đối tượng có màu xanh dương. Khi kết hợp với phím *Ctrl*, ta sẽ di chuyển đối tượng còn lại (màu hồng).

```
case WM_KEYDOWN:
    switch (wParam)
    {
        case VK_LEFT:
            if (GetAsyncKeyState(VK_CONTROL))
            {
                pS2->move(-10, 0);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
            else
            {
                pS->move(-10, 0);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
        case VK_RIGHT:
            if (GetAsyncKeyState(VK_CONTROL))
            {
                pS2->move(10, 0);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
            else
            {
                pS->move(10, 0);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
        case VK_UP:
            if (GetAsyncKeyState(VK_CONTROL))
            {
```

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

4.2.3 Thao tác thay đổi đối tượng hình học bằng phím *Tab*

Ta sử dụng phím *Tab* để thay đổi hình dạng đối tượng có màu xanh dương. Kết hợp với phím *Ctrl* để thay đổi đối tượng còn lại (màu hồng).

```
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;
            }
    }
```

4.2.4 Thao tác phóng to, thu nhỏ đối tượng bằng bàn phím

Ta sử dụng phím (+) và (-) trên bàn phím chữ để thực hiện phóng to và thu nhỏ đối tượng có màu xanh dương. Kết hợp với phím *Ctrl* để thực hiện thao tác tương tự trên đối tượng còn lại (màu hồng).

```
case WM_KEYDOWN:
    switch (wParam)
    {
        case VK_OEM_PLUS:
            if (GetAsyncKeyState(VK_CONTROL))
            {
                pS2->scale(1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
            else
            {
                pS->scale(1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
        case VK_OEM_MINUS:
            if (GetAsyncKeyState(VK_CONTROL))
            {
                pS2->scale(1 / 1.2);
```

```
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
            else
            {
                pS->scale(1 / 1.2);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
    }
```

4.2.5 Thao tác xoay đối tượng bằng bàn phím

Ta sử dụng phím chữ (L) và (R) trên bàn phím chữ để thực hiện xoay trái và xoay phải đối tượng có màu xanh dương một góc 90°. Kết hợp với phím *Ctrl* để thực hiện thao tác tương tự trên đối tượng còn lại (màu hồng).

```
case WM_KEYDOWN:
    switch (wParam)
    {
        case 0x4C:
            if (GetAsyncKeyState(VK_CONTROL))
            {
                pS2->rotate(-90);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
            else
            {
                pS->rotate(-90);
                InvalidateRect(hWnd, NULL, TRUE);
                break;
            }
        case 0x52:
            if (GetAsyncKeyState(VK_CONTROL))
            {
                pS2->rotate(90);
```

```
21                  InvalidateRect(hWnd, NULL, TRUE);
22                  break;
23              }
24              else
25              {
26                  pS->rotate(90);
27                  InvalidateRect(hWnd, NULL, TRUE);
28                  break;
29              }
30      }
```