Đại học Quốc gia Thành phố Hồ Chí Minh
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO BÀI TẬP 2

Môn học: Lập trình hướng đối tượng

Tên chủ đề: báo cáo bài tập 2

GVHD: Nguyễn Hữu Quyền

Ngày báo cáo: 06/04/2023

Nhóm: 02

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: IT002.N28.2

STT	Họ và tên	MSSV	Email
1	Nguyễn Hải Phong	22521088	22521088@gm.uit.edu.vn
2	Hồ Trung Kiên	22520704	22520704@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá	Người đóng góp
1	Câu hỏi 01	100%	Nguyễn Hải Phong
2	Câu hỏi 02	100%	Hồ Trung Kiên
3	Câu hỏi 03	100%	Hồ Trung Kiên

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT BÀI THỰC HÀNH 1

1. Câu hỏi 01

- Tài nguyên: Lớp Point (Điểm) có hai thành phần dữ liệu là hoanhdo và tungdo và các hàm nhập, xuất một điểm, lấy hoành độ, tung độ, hàm tịnh tiến một điểm và hàm vẽ điểm trong chế độ đồ họa.
- Mô tả/mục tiêu: Viết các phương thức thiết lập, các hàm thành phần cho phép thay đổi nội dung của điểm, lấy hoành độ, tung độ, tịnh tiến, nhập, xuất một điểm, hàm vẽ điểm trong chế độ đồ họa.
- Các bước thực hiện/Phương pháp thực hiện:
  - Bước 1: Khai báo thư viện và using namespace std: Khởi tạo thư viện cần thiết gồm iostream (dùng để nhập xuất) và graphics.h cũng như graphics.lib (thư viện dùng để vẽ đồ họa, không có sẵn, phải cài đặt từ Github. Link tải thư viện: https://github.com/githubKYK/Graphics.h)

```
#include <iostream>
#include "graphics.h"
#pragma comment(lib,"graphics.lib")
using namespace std;
```

  - Bước 2: Khởi tạo lớp Point (Điểm) có các dữ liệu và các hàm cần thiết: Các thuộc tính như hoanhdo và tungdo ta sẽ để ở trong phần private trong khi các hàm cần thiết ta sẽ để trong phần public

```
public:
    Point();
    ~Point();
    void NhapDiem();
    void XuatDiem();
    float LayHoanhDo();
    float LayTungDo();
    void TinhTien();
    void VeDiem();
    void VeTrucToaDo();
};
```

Ở hàm khởi tạo (constructor), ta sẽ cho khai báo biến của hoanhdo và tungdo, có thể cho mặc định cả hai đều bằng 0 (hoặc 1 giá trị khác nhưng không nên vượt quá giới hạn cho phép của float vì nếu không sẽ gây lỗi float overflow). Đồng thời trong hàm sẽ khởi tạo màn hình đồ họa có kích thước là 1000 x 800 pixel (27 x 21 cm). Và trong màn hình đồ họa ta sẽ tiến hành gọi đến hàm để vẽ ra trục tọa độ trong màn hình đồ họa

```
Point::Point() {
    initwindow(1000, 800, "Draw a point", 50, 50, false, true);//Khoi tao man hinh do hoa
    tungdo = 0;
    hoanhdo = 0;
    VeTrucToaDo();//Ve truc toa do trong man hinh do hoa
}
```

Hàm vẽ trục đồ họa sẽ vẽ ra 2 đường thẳng chia cắt màn hình làm 4 phần bằng nhau. Ta dùng hàm drawline để vẽ trục Ox từ điểm (0,400) đến (1000,400) và trục Oy từ điểm (500,0) đến (500,800). Sau đó sẽ tạo ra 22 điểm đơn vị bằng

nhau, đánh dấu điểm bằng hàm circle() (đều cách nhau 50 pixel) trên trục Ox và 22 điểm trên trục Oy.

```
void Point::VeTrucToaDo() {
    int TamX = int(getmaxx() / 2) + 1; // toa do x cua tam O
    int TamY = int(getmaxy() / 2) + 1; // toa do y cua tam O
    line(getmaxx() / 2, 0, getmaxx() / 2, getmaxy()); // Ve truc Oy
    line(0, getmaxy() / 2, getmaxx(), getmaxy() / 2); // Ve truc Ox
    // Ve cac cham don vi tam O
    for (int i = TamY; i <= getmaxy(); i += 50) {
        setfillstyle(SOLID_FILL, WHITE);
        circle(getmaxx() / 2, i, 2);
    }
    for (int i = TamY; i >= 0; i -= 50) {
        setfillstyle(SOLID_FILL, WHITE);
        circle(getmaxx() / 2, i, 2);
    }
    for (int j = TamX; j <= getmaxx(); j += 50) {
        setfillstyle(SOLID_FILL, WHITE);
        circle(j, getmaxy() / 2, 2);
    }
    for (int j = TamX; j >= 0; j -= 50) {
        setfillstyle(SOLID_FILL, WHITE);
        circle(j, getmaxy() / 2, 2);
    }
}
```

Hàm vẽ điểm trong chế độ đồ họa sẽ hiển thị điểm được người dùng nhập vào. Đầu tiên hàm sẽ tính gốc tọa độ của trục vừa được vẽ trong hàm vẽ trục đồ họa ứng với tọa độ (500 pixel, 400 pixel) trong chế độ đồ họa. Sau đó sẽ tính toán ứng với tọa độ người dùng nhập vào sẽ là tọa độ nào trong chế độ đồ họa. Công thức:

Tọa độ x' (trong đồ họa) = tọa độ x (trong trục tọa độ) x 50 (do quy định 1 cm = 50 pixel) + tọa độ X của gốc tọa độ (trong đồ họa)

Tọa độ y' (trong đồ họa) = tọa độ y (trong trục tọa độ) x 50 (do quy định 1 cm = 50 pixel) + tọa độ Y của gốc tọa độ (trong đồ họa)

Sau đó ta hiển thị điểm sau khi tính dưới hình dạng hình tròn có kích thước 5 pixel có màu vàng.

```
void Point::VeDiem() {
    int TamX = int(getmaxx() / 2) + 1;
    int TamY = int(getmaxy() / 2) + 1;
    //Thiet lap vi tri diem
    int x = int(hoanhdo * 50 + TamX);
    int y = int(-tungdo * 50 + TamY);
    setfillstyle(SOLID_FILL, YELLOW);
    circle(x, y, 5);
    floodfill(x, y, WHITE);
}
```

Sau đó là hàm nhập hay còn gọi là hàm cập nhật. Hàm này sẽ cho phép người dùng nhập vào giá trị của hoanhdo và tungdo theo ý họ. Đồng thời vì trước đó trên trục Ox và trục Oy tạo ra 22 điểm đơn vị mỗi trục nên hoành độ và tung độ tối đa có thể hiển thị là 11. Cho nên ta sẽ yêu cầu người dùng nhập lại tọa độ điểm nếu vượt quá ngưỡng tối đa đó. Đến khi nhập được tọa độ hợp lệ thì sẽ tiến hành gọi đến hàm vẽ điểm để vẽ điểm đó trên trục.

```
void Point::NhapDiem() {
    do {
        cout << "Nhap hoanh do (x) cua diem: ";
        cin >> hoanhdo;
        cout << "Nhap tung do (y) cua diem: ";
        cin >> tungdo;
        if (hoanhdo > 11 || tungdo > 11) cout << "Nhap lai: " << '\n';
    } while (hoanhdo > 11 || tungdo > 11);
    VeDiem();
}
```

Hàm lấy hoành độ và hàm lấy tung độ thì ta chỉ việc đơn giản là trả về giá trị của tung độ và hoành độ của giá trị được truyền vào trước đó

```
float Point::LayHoanhDo() {
    return hoanhdo;
}
float Point::LayTungDo() {
    return tungdo;
}
```

Hàm tịnh tiến thì ta sẽ yêu cầu người dùng nhập vào vectơ bất kỳ. Sau đó ta tiến hành tịnh tiến điểm đã được người dùng nhập vào trước đó theo công thức:

x' (hoành độ sau khi tịnh tiến) = x (hoành độ ban đầu) + a (hoành độ của vector tịnh tiến)

y' (tung độ sau khi tịnh tiến) = y (tung độ ban đầu) + b (tung độ của vector tịnh tiến)

Nhưng vì giá trị hiển thị tối đa của cả trục Ox và trục Oy trong chế độ đồ họa là 11 nên ta sẽ yêu cầu người dùng nhập lại vecto tịnh tiến nếu tung độ hoặc hoành độ vượt quá 11. Ta sẽ xuất ra hoành độ và tung độ sau khi đã tịnh tiến trên màn hình khi điểm tịnh tiến đã hợp lệ. Đồng thời gọi đến hàm vẽ trục tọa độ và hàm vẽ điểm để vẽ lại trục tọa độ và vẽ điểm tính tiến trên trục tọa độ.

```
void Point::TinhTien() {
    do {
        float x, y;
        cout << "Nhap vecto de tinh tien: ";
        cin >> x >> y;
        hoanhdo = hoanhdo + x;
        tungdo = tungdo + y;
        if (hoanhdo > 11 || tungdo > 11) cout << "Diem tinh tien da vuot ngoai pham vi hien thi. Vui long nhap lai vecto tinh tien." << '\n';
    } while (hoanhdo > 11 || tungdo > 11);
    cout << "Diem sau khi da tinh tien la: " << '\n';
    clearviewport();
    VeTrucToaDo();
    XuatDiem();
    VeDiem();
}
```

Tiếp theo là hàm xuất dùng để xuất ra tung độ và hoành độ của điểm đã được truyền vào giá trị trước đó.

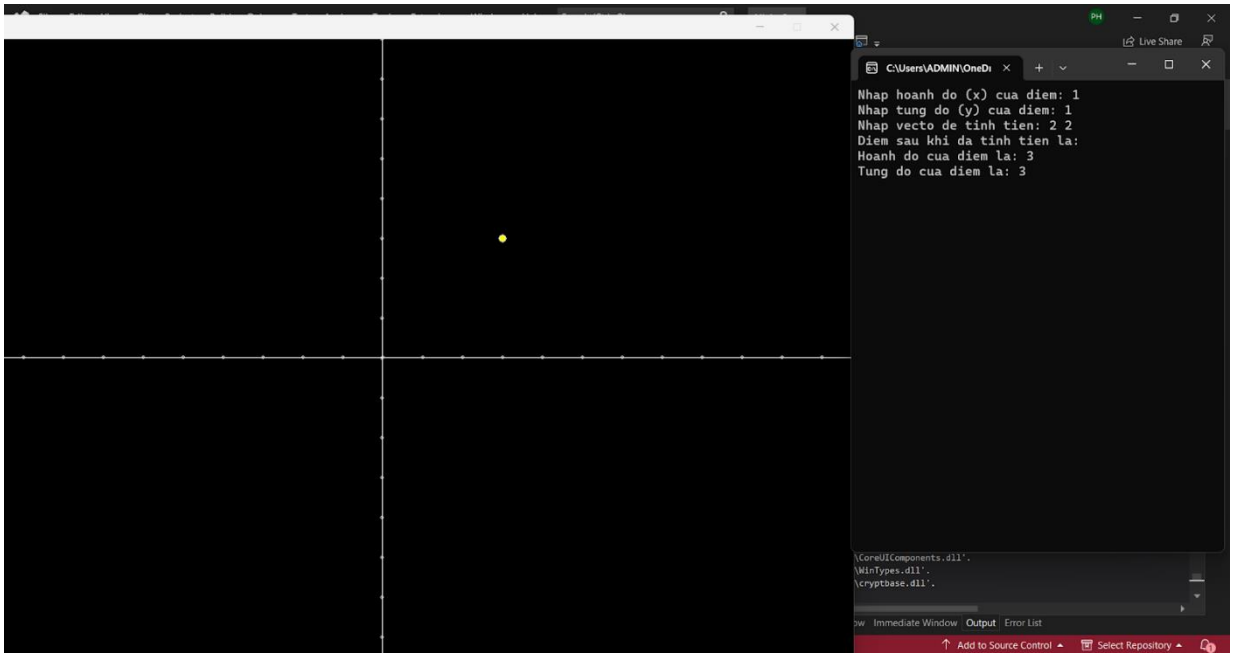
```
void Point::XuatDiem() {
    cout << "Tung do cua diem la: " << tungdo << endl;
    cout << "Hoanh do cua diem la: " << hoanhdo << endl;
}
```

Và cuối cùng làm hàm hủy (hàm destructor) dùng để thoát khỏi chế độ đồ họa sau khi nhập điểm từ người dùng và tịnh tiến điểm. Ta sẽ dùng lệnh getch() và lệnh closegraph() để có thể tắt cửa sổ của chế độ đồ họa và quay lại cửa sổ chính.

```
Point::~Point() {
    getch();
    closegraph();
}
```

- Bước 3: Tạo hàm main. Trong hàm main ta thực hiện khai báo đối tượng a có kiểu dữ liệu là class Point. Ta tiến hành gọi đến các hàm NhapDiem (dùng để nhập tung độ và hoành độ và hiển thị điểm trong chế độ đồ họa), hàm TinhTien (dùng để tịnh tiến điểm đó và hiển thị điểm trong chế độ đồ họa).

Ví dụ demo:



2. Câu hỏi 02:

- Tài nguyên: Viết định nghĩa lớp TamGiac để biểu diễn khái niệm tam giác trong mặt phẳng với các phương thức thiết lập, huỷ bỏ (nếu có).
- Mô tả/mục tiêu: Viết các hàm thành phần nhập, xuất, tịnh tiến, quay, phóng to, thu nhỏ và vẽ tam giác.
- Các bước thực hiện/Phương pháp thực hiện:
  - Bước 1: Khai báo thư viện và using namespace std: Khởi tạo thư viện cần thiết gồm iostream (dùng để nhập xuất), cmath để có thể tính toán các phép toán lượng giác như sin và cos, và graphics.h cũng như graphics.lib (thư viện cần thiết dùng để vẽ tam giác)

```
1  #include <iostream>
2  #include <cmath>
3  #include "graphics.h"
4  #pragma comment(lib,"graphics.lib")
5  using namespace std;
```

  - Bước 2: Khởi tạo lớp Point và khai báo các thuộc tính và các hàm thành phần cần thiết

Về thuộc tính của lớp Point ta khai báo 2 thuộc tính hoành độ (x) và tung độ (y) với kiểu double và để ở phần private, cùng với đó khai báo lớp bạn Triangle (lớp tam giác), ta có thể khai báo lớp bạn ở bất cứ đâu trong lớp Point.

Về các hàm cần thiết cho lớp Point ta khai báo các hàm như constructor, destructor, TinhTien, Quay, ThuPhong, ThuNho với các truyền đối số cần thiết cho từng hàm

```
8   class Point {
9   private:
10      double x;
11      double y;
12      friend class Triangle;
13  public:
14      Point();
15      ~Point();
16      void TinhTien(double X, double Y);
17      void Quay(int goc);
18      void ThuPhong(double k);
19      void ThuNho(double k);
20  };
```

Tiếp đến ta sẽ đến chi tiết với chức và cách hoạt động của từng hàm:

Đầu tiên là hàm Point() (hàm constructor) và hàm ~Point() (hàm destructor):

Hàm constructor ta sẽ gán cho x và y với giá trị mặc định là 0 ngay khi khởi tạo một đối tượng Point. Hàm destructor ta chưa cần sử dụng nên ta không khai báo gì trong hàm này.

```
22  Point::Point() {
23      x = 0;
24      y = 0;
25  }
26  Point::~Point() { }
```

Tiếp theo là hàm không có kiểu trả về (void) TinhTien() :

Hàm này sẽ có truyền đối số hai biến X và Y có kiểu double và thực hiện phép tính toán để cộng vào x của lớp Point biến X và cộng vào y của lớp Point biến Y.

```
27  void Point::TinhTien(double X, double Y) {
28      x = x + X;
29      y = y + Y;
30  }
```

Tiếp đến là hàm không có kiểu trả về Quay() :

Hàm này sẽ có truyền đối số kiểu int mang tên là goc, mục đích biến này có kiểu định dạng int thay vì double để biểu diễn radiant cho góc quay là vì biến goc có đơn vị là độ thay vì radiant, và góc quay sẽ ngược chiều kim đồng hồ nếu goc là số dương và ngược lại.

Ta sẽ khởi tạo biến double radian để chuyển đổi từ độ sang radian từ biến goc bằng công thức radian = (goc * 3.14159) / 180 với 3.14159 xấp xỉ số pi (π)

Khởi tạo một biến Point mang tên quay để lưu giá trị x và y ban đầu (quay.x=x và quay.y=y) hoặc có thể khởi tạo 2 biến double tuỳ ý để lưu giá trị x và y, từ đó ta có thể thực hiện phép quay một điểm trên toạ độ gốc tâm O với công thức (α có đơn vị là radian) :

$$\begin{cases} x' = x\cos\alpha - y\sin\alpha \\ y' = x\sin\alpha + y\cos\alpha \end{cases}$$

```
31  void Point::Quay(int goc) {
32      Point quay;
33      double radian = (goc * 3.14159) / 180;
34      quay.x = x;
35      quay.y = y;
36      x = quay.x * cos(radian) - quay.y * sin(radian);
37      y = quay.x * sin(radian) + quay.y * cos(radian);
38  }
```

Cuối cùng là 2 hàm ThuPhong() và ThuNho() :

Với hàm ThuPhong() có chức năng phóng đại hay gia tăng khoảng cách từ tâm O đến một điểm có toạ độ x,y k lần với k mang kiểu double và được truyền đối số từ hàm. Ta cài đặt hàm này như sau với x và y là thuộc tính của lớp Point :

```
39  void Point::ThuPhong(double k) {
40      x = x * k;
41      y = y * k;
42  }
```

Tương tự với hàm ThuNho() có chức năng thu nhỏ một điểm có toạ độ x,y so với tâm O k lần với k mang kiểu double và được truyền đối số từ hàm. Ta cài đặt hàm này như sau với x và y là thuộc tính của lớp Point :

```
43  void Point::ThuNho(double k) {
44      x = x / k;
45      y = y / k;
46  }
```

- Bước 3: Khởi tạo lớp Triangle và khai báo các thuộc tính và các hàm thành phần cần thiết :

```
50  class Triangle {
51  private:
52      Point A, B, C;
53  public:
54      Triangle();
55      ~Triangle();
56      void Nhap3DinhTamGiac();
57      void XuatDuLieu3DinhTamGiac();
58      void TinhTienTamGiac();
59      void QuayTamGiac();
60      void PhongToTamGiac();
61      void ThuNhoTamGiac();
62      void VeTrucToaDo();
63      void VeTamGiac();
64  };
```

Về thuộc tính của lớp Triangle ta khai báo 3 thuộc tính mang tên A,B và C có kiểu dữ liệu là Point (vì một tam giác được cấu thành từ 3 điểm không thẳng hàng) và để ở phần private.

```
50  class Triangle {
51  private:
52      Point A, B, C;
```

Về các hàm cần thiết cho lớp Point ta khai báo các hàm như constructor, destructor, VeTrucToaDo, Nhap3DinhTamGiac, XuatDuLieu3DinhTamGiac, TinhTienTamGiac, QuayTamGiac, PhongToTamGiac, ThuNhoTamGiac và VeTamGiac. Ta sẽ không truyền đối số vì quá trình thực hiện của các hàm này sẽ thực hiện nhập song song với tính toán.

```
53  public:
54      Triangle();
55      ~Triangle();
56      void Nhap3DinhTamGiac();
57      void XuatDuLieu3DinhTamGiac();
58      void TinhTienTamGiac();
59      void QuayTamGiac();
60      void PhongToTamGiac();
61      void ThuNhoTamGiac();
62      void VeTrucToaDo();
63      void VeTamGiac();
64  };
```

Tiếp đến ta sẽ đến chi tiết với chức và cách hoạt động của từng hàm:

Đầu tiên với hàm constructor Triangle():

Với hàm constructor ta sẽ tiến hành khởi tạo một cửa sổ trống (window) khác song song với console bằng hàm initwindow() của thư viện graphics.h được truyền vào lần lượt là độ dài chiều ngang (pixel), chiều dài dọc (pixel), tên của window (string), chiều dài ngang khoảng cách của window so với màn hình (pixel), chiều dài dọc khoảng cách của window so với màn hình (pixel). Sau đó là gọi tới hàm VeTrucToaDo(), ta có thể tuỳ chỉnh chiều dài dọc và ngang nếu muốn nhưng để dễ nhìn ta để mặc định như sau:

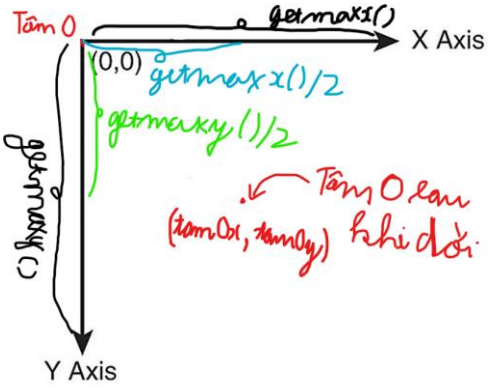
```
69  Triangle::Triangle() {
70      initwindow(1000, 800, "Draw a triangle", 50, 50);
71      VeTrucToaDo();
72  }
```

Trong hàm VeTrucToaDo() được gọi có chi tiết như sau:

```
73  void Triangle::VeTrucToaDo() {
74      int tamOx = int(getmaxx() / 2); // toa do x cua tam O
75      int tamOy = int(getmaxy() / 2); // toa do y cua tam O
76      line(tamOx, 0, tamOx, getmaxy()); // Ve truc Oy
77      line(0, tamOy, getmaxx(), tamOy); // Ve truc Ox
78      // Ve cac cham don vi tam O
79      for (int i = tamOy; i <= getmaxy(); i += 50) {
80          setfillstyle(SOLID_FILL, WHITE);
81          circle(tamOx, i, 2);
82      }
83      for (int i = tamOy; i >= 0; i -= 50) {
84          setfillstyle(SOLID_FILL, WHITE);
85          circle(tamOx, i, 2);
86      }
87      for (int j = tamOx; j <= getmaxx(); j += 50) {
88          setfillstyle(SOLID_FILL, WHITE);
89          circle(j, tamOy, 2);
90      }
91      for (int j = tamOx; j >= 0; j -= 50) {
92          setfillstyle(SOLID_FILL, WHITE);
93          circle(j, tamOy, 2);
94      }
95  }
```

Đầu tiên khởi tạo 2 biến kiểu int tamOx và tamOy để chứa hoành độ và tung độ của tâm O sau khi dời tâm từ bên trái góc trên cùng của window về tâm của window bằng cách truyền vào lần lượt là getmaxx()/2 và getmaxy()/2 với 2 hàm getmaxx() và getmaxy() là trả về chiều dài trục x và chiều dài trục y tối đa của window khi khởi tạo (trong trường hợp này là 1000 và 800).

```
74      int tamOx = int(getmaxx() / 2); // toa do x cua tam O
75      int tamOy = int(getmaxy() / 2); // toa do y cua tam O
```



Tiếp đến là 2 dòng vẽ trục toạ độ Ox và Oy bằng hàm line(), hàm line() sẽ vẽ một đường thẳng khi truyền vào toạ độ x,y của 2 điểm cần vẽ đường thẳng.

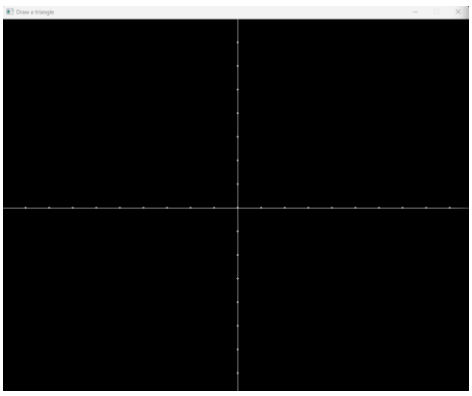
```
76      line(tamOx, 0, tamOx, getmaxy()); // Ve truc Oy
77      line(0, tamOy, getmaxx(), tamOy); // Ve truc Ox
```

Cuối cùng là các lệnh vòng lặp for() để vẽ những chấm tròn bằng hàm circle() truyền vào toạ độ x,y và bán kính (2 pixel) song song với hàm setfillstyle() để vẽ hình tròn đặc.

Tất cả bốn vòng lặp for() này để vẽ những chấm tròn xuất phát từ tâm ra bốn phía khác nhau (mục đích là để dù có tuỳ chỉnh chiều dài hay chiều ngang của window thì đều có thể vẽ chính xác ở tâm màn hình ra bốn góc khác nhau) và mỗi chấm tròn cách nhau 50 pixel tương tự với 1 đơn vị điểm trong hình học (chọn 50 pixel là mặc định cho mỗi đơn vị điểm hoặc có thể tuỳ chỉnh và nâng cấp thêm bằng cách thêm biến pixel trên một đơn vị điểm trong private và người dùng có thể nhập, tương tự với chiều dài và chiều ngang của window,…).

```
78      // Ve cac cham don vi tam O
79      for (int i = tamOy; i <= getmaxy(); i += 50) {
80          setfillstyle(SOLID_FILL, WHITE);
81          circle(tamOx, i, 2);
82      }
83      for (int i = tamOy; i >= 0; i -= 50) {
84          setfillstyle(SOLID_FILL, WHITE);
85          circle(tamOx, i, 2);
86      }
87      for (int j = tamOx; j <= getmaxx(); j += 50) {
88          setfillstyle(SOLID_FILL, WHITE);
89          circle(j, tamOy, 2);
90      }
91      for (int j = tamOx; j >= 0; j -= 50) {
92          setfillstyle(SOLID_FILL, WHITE);
93          circle(j, tamOy, 2);
94      }
95  }
```

Khi gọi hàm VeTrucToaDo() trong hàm constructor Triangle() ta được một cửa sổ như sau:



Tiếp đến là hàm destructor ~Triangle() :

Hàm này ta sẽ gọi tới hai hàm khác là hàm getch() để đợi input của người dùng và sau đó là hàm closegraph() để đóng window vẽ đồ hoạ.

```
96  Triangle::~Triangle() {
97      getch();
98      closegraph();
99  }
```

Theo sau đó là hàm Nhap3DinhTamGiac():

```
100 void Triangle::Nhap3DinhTamGiac() {
101     double AB, AC, BC;
102     int tamOx = int(getmaxx() / 2); // toa do x cua tam O
103     int tamOy = int(getmaxy() / 2); // toa do y cua tam O
104     do {
105         cout << "Nhap toa do diem A: ";
106         cin >> A.x >> A.y;
107         circle(tamOx + A.x * 50, tamOy - A.y * 50, 2);
108         cout << "Nhap toa do diem B: ";
109         cin >> B.x >> B.y;
110         circle(tamOx + B.x * 50, tamOy - B.y * 50, 2);
111         cout << "Nhap toa do diem C: ";
112         cin >> C.x >> C.y;
113         circle(tamOx + C.x * 50, tamOy - C.y * 50, 2);
114         AB = sqrt(pow(A.x - B.x, 2) + pow(A.y - B.y, 2)); // Tinh canh AB
115         AC = sqrt(pow(A.x - C.x, 2) + pow(A.y - C.y, 2)); // Tinh canh AC
116         BC = sqrt(pow(B.x - C.x, 2) + pow(B.y - C.y, 2)); // Tinh canh BC
117         if (AB + AC <= BC || AB + BC <= AC || AC + BC <= AB) {
118             clearviewport();
119             VeTrucToaDo();
120             cout << "Nhap lai toa do 3 diem: " << endl;
121         }
122     } while (AB + AC <= BC || AB + BC <= AC || AC + BC <= AB);
123 }
```

Đầu tiên khai báo 3 biến double chứa độ dài ba cạnh của tam giác sau khi nhập vào toạ độ 3 điểm A,B,C lần lượt là AB,AC,BC và định nghĩa lại tamOx và tamOy.

```
101     double AB, AC, BC;
102     int tamOx = int(getmaxx() / 2); // toa do x cua tam O
103     int tamOy = int(getmaxy() / 2); // toa do y cua tam O
```

Sau đó là vòng lặp do while có chức năng nhập toạ độ x,y của 3 điểm A,B,C cùng với đó vẽ nhừng hình tròn tại vị trí đã nhập để biểu diễn vị trí của các điểm đã nhập vào (tuy trục toạ độ đã được dời về tâm window nhưng trục Oy vẫn còn có trục dương hướng xuống dưới nên với toạ độ y của các điểm A,B,C ta sẽ lấy đối của chúng bằng phép trừ)

```
100  void Triangle::Nhap3DinhTamGiac() {
101      double AB, AC, BC;
102      int tamOx = int(getmaxx() / 2); // toa do x cua tam O
103      int tamOy = int(getmaxy() / 2); // toa do y cua tam O
104      do {
105          cout << "Nhap toa do diem A: ";
106          cin >> A.x >> A.y;
107          circle(tamOx + A.x * 50, tamOy - A.y * 50, 2);
108          cout << "Nhap toa do diem B: ";
109          cin >> B.x >> B.y;
110          circle(tamOx + B.x * 50, tamOy - B.y * 50, 2);
111          cout << "Nhap toa do diem C: ";
112          cin >> C.x >> C.y;
113          circle(tamOx + C.x * 50, tamOy - C.y * 50, 2);
114          AB = sqrt(pow(A.x - B.x, 2) + pow(A.y - B.y, 2)); // Tinh canh AB
115          AC = sqrt(pow(A.x - C.x, 2) + pow(A.y - C.y, 2)); // Tinh canh AC
116          BC = sqrt(pow(B.x - C.x, 2) + pow(B.y - C.y, 2)); // Tinh canh BC
117          if (AB + AC <= BC || AB + BC <= AC || AC + BC <= AB) {
118              clearviewport();
119              VeTrucToaDo();
120              cout << "Nhap lai toa do 3 diem: " << endl;
121          }
122      } while (AB + AC <= BC || AB + BC <= AC || AC + BC <= AB);
123      VeTamGiac();
124  }
```

Từ 3 điểm A,B,C ta tính toán ra độ dài 3 cạnh AB,AC,BC và gán cho 3 biến AB,AC,BC với công thức sau:

Trong mặt phẳng tọa độ, khoảng cách giữa hai điểm $M(x_M;y_M)$ và $N(x_N;y_N)$ là

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

```
114          AB = sqrt(pow(A.x - B.x, 2) + pow(A.y - B.y, 2)); // Tinh canh AB
115          AC = sqrt(pow(A.x - C.x, 2) + pow(A.y - C.y, 2)); // Tinh canh AC
116          BC = sqrt(pow(B.x - C.x, 2) + pow(B.y - C.y, 2)); // Tinh canh BC
```

Sau khi có độ dài 3 cạnh AB,AC,BC ta tiến hành xét liệu 3 cạnh này có phải là 3 cạnh của một tam giác hay không bằng bất đẳng thức tam giác như sau:

Cho tam giác ABC, ta có các bất đẳng thức sau:

- (AB+AC) >BC hay (c+b) >a;
- (AB+BC) >AC hay (c+a) >b;
- (AC+BC) >AB hay (b+a) >c.

Nếu ba cạnh AB,AC,BC không phải 3 cạnh của một tam giác thì tiến hành làm trống window bằng hàm clearviewport() và vẽ lại trục toạ độ bằng hàm VeTrucToaDo(), nhập lại 3 điểm A,B,C cho đến khi nào 3 điểm A,B,C hợp lệ thì dừng. Sau khi nhập xong ta vẽ tam giác bằng hàm VeTamGiac().

```
        if (AB + AC <= BC || AB + BC <= AC || AC + BC <= AB) {
            clearviewport();
            VeTrucToaDo();
            cout << "Nhap lai toa do 3 diem: " << endl;
        }
    } while (AB + AC <= BC || AB + BC <= AC || AC + BC <= AB);
    VeTamGiac();
```

Tiếp theo là hàm XuatDuLieu3DinhTamGiac():

Hàm này đơn giản là xuất dữ liệu của 3 điểm A,B,C ra màn hình console

```
124  void Triangle::XuatDuLieu3DinhTamGiac() {
125      cout << "Toa do diem A la: " << A.x << " " << A.y << endl;
126      cout << "Toa do diem B la: " << B.x << " " << B.y << endl;
127      cout << "Toa do diem C la: " << C.x << " " << C.y << endl;
128  }
```

Hàm VeTamGiac():

Đầu tiên khai báo lại tamOx và tamOy sau đó vẽ các đường thẳng nối giữa các điểm A,B,C lần lượt là AB,AC,BC với toạ độ các điểm được tính toán truyền vào hàm line() với kiểu int (đã được giải thích ở hàm Nhap3DinhTamGiac())

```
129 void Triangle::VeTamGiac() {
130     int tamOx = int(getmaxx() / 2);
131     int tamOy = int(getmaxy() / 2);
132     line(int(A.x * 50 + tamOx), int(-A.y * 50 + tamOy), int(B.x * 50 + tamOx), int(-B.y * 50 + tamOy));
133     line(int(A.x * 50 + tamOx), int(-A.y * 50 + tamOy), int(C.x * 50 + tamOx), int(-C.y * 50 + tamOy));
134     line(int(B.x * 50 + tamOx), int(-B.y * 50 + tamOy), int(C.x * 50 + tamOx), int(-C.y * 50 + tamOy));
135 }
```

Hàm TinhTienTamGiac():

Đầu tiên khai báo hai biến double a và b sau đó tiến hành nhập vào 2 biến a và b từ console và tiến hành tịnh tiến 3 điểm A,B,C bằng cách gọi hàm TinhTien() của từng điểm và truyền vào đối số a và b.

Sau đó gọi thêm các hàm XuatDuLieu3DinhTamGiac() để xuất toạ độ các điểm sau khi tịnh tiến trên console, cùng với hàm clearviewport() để xoá clear màn hình window, sau đó gọi lại hàm VeTrucToaDo() để vẽ trục toạ độ và gọi hàm VeTamGiac() để vẽ lại tam giác từ 3 điểm A,B,C.

```
137 void Triangle::TinhTienTamGiac() {
138     double a, b;
139     cout << "Nhap vecto tinh tien: ";
140     cin >> a >> b;
141     A.TinhTien(a, b);
142     B.TinhTien(a, b);
143     C.TinhTien(a, b);
144     XuatDuLieu3DinhTamGiac();
145     clearviewport();
146     VeTrucToaDo();
147     VeTamGiac();
148 }
```

Hàm QuayTamGiac():

Hàm này sẽ thực hiện việc quay 3 điểm của tam giác xung quanh trục toạ độ O bằng cách nhập vào góc cần quay theo chiều kim đồng hồ nếu góc quay dương và ngược lại.

Đầu tiên khai báo biến int gocquay và thực hiện nhập qua console và tiến hành quay 3 điểm A,B,C theo góc toạ độ tâm O với hàm Quay().

Sau đó gọi thêm các hàm XuatDuLieu3DinhTamGiac() để xuất toạ độ các điểm sau khi tịnh tiến trên console, cùng với hàm clearviewport() để xoá clear màn hình window, sau đó gọi lại hàm VeTrucToaDo() để vẽ trục toạ độ và gọi hàm VeTamGiac() để vẽ lại tam giác từ 3 điểm A,B,C.

```
149  void Triangle::QuayTamGiac() {
150      int gocquay;
151      cout << "Nhap goc quay tam giac (do) : ";
152      cin >> gocquay;
153      A.Quay(gocquay);
154      B.Quay(gocquay);
155      C.Quay(gocquay);
156      XuatDuLieu3DinhTamGiac();
157      clearviewport();
158      VeTrucToaDo();
159      VeTamGiac();
160  }
```

Hàm PhongToTamGiac():

Tương tự với 2 hàm trên nhưng hàm này có chức năng phóng to tam giác bằng cách nhập vào độ phóng to so với góc toạ độ tâm O và biến PhongTo không được bé hơn 1 và nếu bé hơn 1 thì yêu cầu nhập lại.

```
161  void Triangle::PhongToTamGiac() {
162      double phongto;
163      cout << "Nhap do phong to: ";
164      cin >> phongto;
165      while (phongto < 1) {
166          cout << "Nhap lai do phong to: ";
167          cin >> phongto;
168      }
169      A.ThuPhong(phongto);
170      B.ThuPhong(phongto);
171      C.ThuPhong(phongto);
172      XuatDuLieu3DinhTamGiac();
173      clearviewport();
174      VeTrucToaDo();
175      VeTamGiac();
176  }
```

Hàm ThuNhoTamGiac():

Tương tự với PhongToTamGiac() nhưng có chức năng thu nhỏ tam giác và biến thunho cũng không được bé hơn 1.

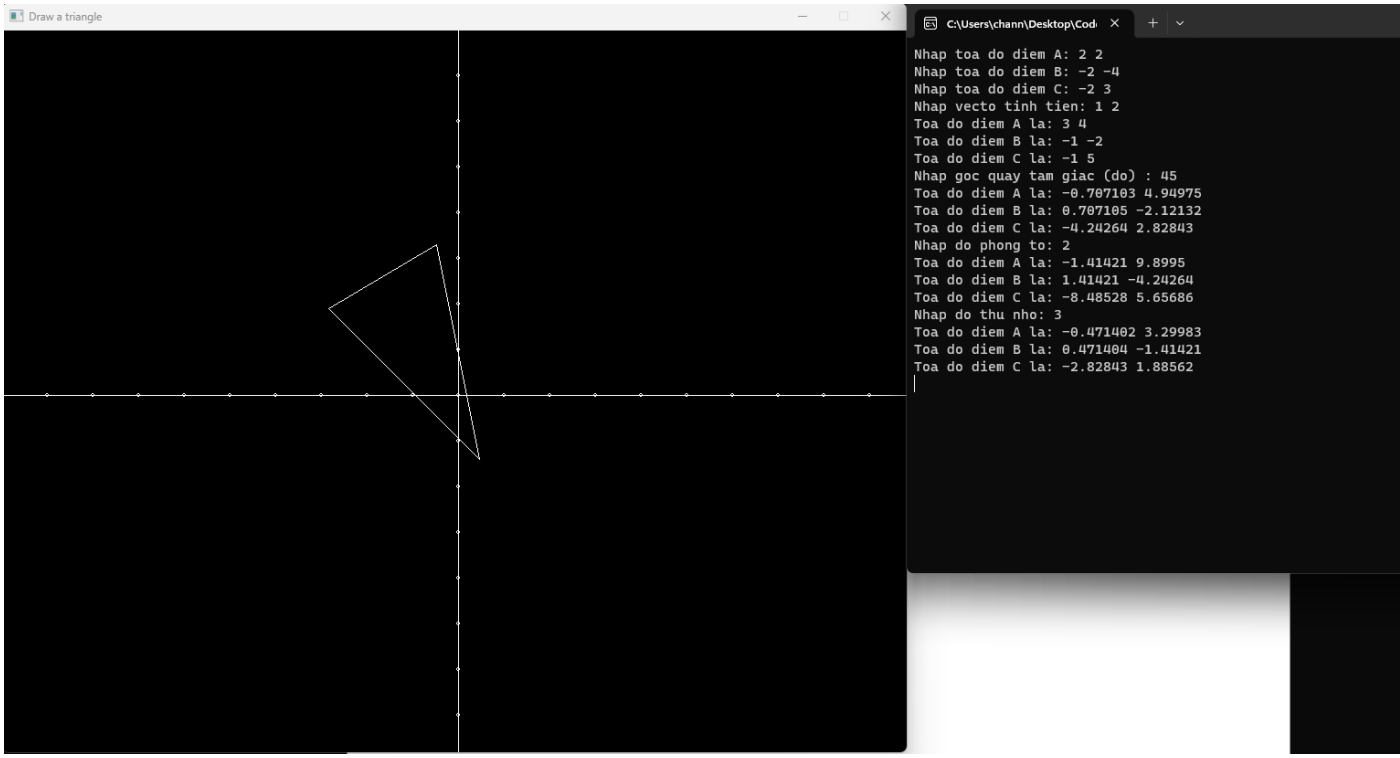
```
177  void Triangle::ThuNhoTamGiac() {
178      double thunho;
179      cout << "Nhap do thu nho: ";
180      cin >> thunho;
181      while (thunho < 1) {
182          cout << "Nhap lai do thu nho: ";
183          cin >> thunho;
184      }
185      A.ThuNho(thunho);
186      B.ThuNho(thunho);
187      C.ThuNho(thunho);
188      XuatDuLieu3DinhTamGiac();
189      clearviewport();
190      VeTrucToaDo();
191      VeTamGiac();
192  }
```

- Bước 4: Khởi tạo hàm main

Hàm main ta tiến hành khởi tạo Triangle A và gọi tới các hàm Nhap3DinhTamGiac(), TienTienTam(), QuayTamGiac(), PhongToTamGiac(), ThuNhoTamGiac().

```
195  int main() {
196      Triangle A;
197      A.Nhap3DinhTamGiac();
198      A.TinhTienTamGiac();
199      A.QuayTamGiac();
200      A.PhongToTamGiac();
201      A.ThuNhoTamGiac();
202      return 0;
203  }
```

Ví dụ demo :



3. Câu hỏi 03:

- Tài nguyên: Viết định nghĩa lớp DaGiac để biểu diễn khái niệm đa giác trong mặt phẳng với các phương thức thiết lập, huỷ bỏ (nếu có).
- Mô tả/mục tiêu: Viết các hàm thành phần nhập, xuất, tịnh tiến, quay, phóng to, thu nhỏ và vẽ đa giác.
- Các bước thực hiện/Phương pháp thực hiện:
  - Bước 1: Khai báo thư viện và using namespace std: Khởi tạo thư viện cần thiết gồm iostream (dùng để nhập xuất), vector cần thiết để xử lý dữ liệu, algorithm để có thể tính toán các phép toán lượng giác như sin và cos,… và graphics.h cũng như graphics.lib (thư viện cần thiết dùng để vẽ đa giác).

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "graphics.h"
#pragma comment (lib,"graphics.lib")
using namespace std;
```

- Bước 2: Khởi tạo lớp Point và khai báo các thuộc tính và các hàm thành phần cần thiết:

  Lớp này sẽ có khai báo và định nghĩa giống như trong bài tam giác.

```
class Point {
private:
    double x, y;
    friend class DaGiac;
public:
    Point();
    ~Point();
    void TinhTien(double X, double Y);
    void Quay(int gocquay);
    void PhongTo(double k);
    void ThuNho(double k);
};
```

- Bước 3: Khởi tạo lớp DaGiac và khai báo các thuộc tính và các hàm thành phần cần thiết:

```
class DaGiac {
private:
    int TongDiem;
    Point* A;
public:
    DaGiac();
    ~DaGiac();
    void VeTrucToaDo();
    void NhapDaGiac();
    void XuatDuLieuDaGiac();
    void TinhTienDaGiac();
    void QuayDaGiac();
    void PhongToDaGiac();
    void ThuNhoDaGiac();
    void VeDaGiac();
    void VeTrucToaDoVaDaGiac();
    void sortPointsByAngle(vector<Point>& points);
};
```

Đầu tiên ta sẽ để các thuộc tính như tổng số điểm của đa giác có kiểu int và tên là TongDiem cùng với con trỏ Point tên A trong vùng private.

```
class DaGiac {
private:
    int TongDiem;
    Point* A;
```

Tiếp theo là các hàm cần thiết của lớp DaGiac:

DaGiac(); // constructor: hàm này tiến hành gán cho TongDiem = 0 và khởi tạo mảng con trỏ ứng với số TongDiem, cùng với đó khởi tạo window như đã giải thích ở lớp TamGiac và tiến hành vẽ trục toạ độ bằng hàm VeTrucToaDo()

```
129    DaGiac::DaGiac() {
130        TongDiem = 0;
131        A = new Point[TongDiem];
132        initwindow(1000, 800, "Draw a polygon", 50, 50);
133        VeTrucToaDo();
134    }
```

~DaGiac(); // destructor: hàm này sẽ thực hiện việc chờ input để có thể dừng màn hình window, tránh việc window tắt trước khi cần bằng hàm getch(), sau đó là hàm closegraph() để đóng window và cuối cùng là giải phóng mảng động A có TongDiem phần tử.

```
135    DaGiac::~DaGiac() {
136        getch();
137        closegraph();
138        for (int i = 0; i < TongDiem; i++) {
139            A[i].x = NULL;
140            A[i].y = NULL;
141        }
142        delete[] A;
143    }
```

void VeTrucToaDo(); // vẽ trục toạ độ: tương tự với lớp TamGiac

```
101    void DaGiac::VeTrucToaDo() {
102        int tamOx = int(getmaxx() / 2); // toa do x cua tam O
103        int tamOy = int(getmaxy() / 2); // toa do y cua tam O
104        line(tamOx, 0, tamOx, getmaxy()); // Ve truc Oy
105        line(0, tamOy, getmaxx(), tamOy); // Ve truc Ox
106        // Ve cac cham don vi tam O
107        for (int i = tamOy; i <= getmaxy(); i += 50) {
108            setfillstyle(SOLID_FILL, WHITE);
109            circle(tamOx, i, 2);
110        }
111        for (int i = tamOy; i >= 0; i -= 50) {
112            setfillstyle(SOLID_FILL, WHITE);
113            circle(tamOx, i, 2);
114        }
115        for (int j = tamOx; j <= getmaxx(); j += 50) {
116            setfillstyle(SOLID_FILL, WHITE);
117            circle(j, tamOy, 2);
118        }
119        for (int j = tamOx; j >= 0; j -= 50) {
120            setfillstyle(SOLID_FILL, WHITE);
121            circle(j, tamOy, 2);
122        }
123    }
```

void NhapDaGiac(); // Nhập số điểm và các điểm đa giác: tiến hành nhập số điểm và truyền vào TongDiem, nếu như số điểm bé hơn 3 sẽ tiến hành nhập lại. Sau đó tính tamOx và tamOy, tiến hành nhập toạ độ các điểm cùng với đó vẽ các điểm trên window bằng hàm circle() (ta có thể cải tiến bằng cách thêm vào cách thực hiện nhập lại nếu như có điểm bị trùng hoặc tất cả các điểm của đa giác thẳng hàng). Cuối cùng sau khi nhập xong là tiến hành vẽ đa giác bằng hàm VeDaGiac().

```
165    void DaGiac::NhapDaGiac() {
166        do {
167            cout << "Nhap so luong diem da giac: ";
168            cin >> TongDiem;
169            if (TongDiem < 3) cout << "Vui long nhap lai so luong diem da giac: ";
170        } while (TongDiem < 3);
171        int tamOx = int(getmaxx() / 2);
172        int tamOy = int(getmaxy() / 2);
173        for (int i = 0; i < TongDiem; i++) {
174            cout << "Nhap toa do diem thu " << i + 1 << ": ";
175            cin >> A[i].x >> A[i].y;
176            circle(A[i].x * 50 + tamOx, -A[i].y * 50 + tamOy, 2);
177        }
178        VeDaGiac();
179    }
```

void XuatDuLieuDaGiac(); // Xuất toạ độ các điểm của đa giác:

```
void DaGiac::XuatDuLieuDaGiac() {
    for (int i = 0; i < TongDiem; i++) {
        cout << "Toa do diem thu " << i + 1 << " la: " << A[i].x << " " << A[i].y << endl;
    }
}
```

void TinhTienDaGiac(); // Tịnh tiến đa giác: tiến hành khai báo 2 biến double là x và y và nhập x,y. Sau đó thực hiện tịnh tiến từng điểm trong mảng A bằng vòng lặp for với hàm TinhTien() và truyền vào x,y. Cuối cùng xuất dữ liệu của các điểm trong đa giác sau khi tịnh tiến bằng hàm XuatDuLieuDaGiac() và vẽ lại trục toạ độ và đa giác bằng hàm VeTrucToaDoVaDaGiac().

```
void DaGiac::TinhTienDaGiac() {
    double x, y;
    cout << "Nhap vector tinh tien: ";
    cin >> x >> y;
    for (int i = 0; i < TongDiem; i++)
        A[i].TinhTien(x, y);
    XuatDuLieuDaGiac();
    VeTrucToaDoVaDaGiac();
}
```

void QuayDaGiac(); // Quay đa giác: khai báo biến int tên gocquay và nhập biến gocquay, tiến hành quay từng điểm của đa giác bằng vòng lặp for với hàm Quay() và truyền vào gocquay. Cuối cùng xuất dữ liệu của các điểm trong đa giác sau khi quay bằng hàm XuatDuLieuDaGiac() và vẽ lại trục toạ độ và đa giác bằng hàm VeTrucToaDoVaDaGiac().

```
void DaGiac::QuayDaGiac() {
    int gocquay;
    cout << "Nhap goc quay cua da giac: ";
    cin >> gocquay;
    for (int i = 0; i < TongDiem; i++)
        A[i].Quay(gocquay);
    XuatDuLieuDaGiac();
    VeTrucToaDoVaDaGiac();
}
```

void PhongToDaGiac(); // Phóng to đa giác: tạo biến double tên k và nhập k nếu k<1 thì tiến hành nhập lại, tiến hành gọi hàm PhongTo() với tất cả các điểm trong đa giác và truyền vào k. Cuối cùng xuất dữ liệu của các điểm trong đa giác sau khi phóng to bằng hàm XuatDuLieuDaGiac() và vẽ lại trục toạ độ và đa giác bằng hàm VeTrucToaDoVaDaGiac().

```
void DaGiac::PhongToDaGiac() {
    double k;
    cout << "Nhap do phong to da giac: ";
    cin >> k;
    while (k < 1) {
        cout << "Nhap lai do phong to da giac: ";
        cin >> k;
    }
    for (int i = 0; i < TongDiem; i++)
        A[i].PhongTo(k);
    XuatDuLieuDaGiac();
    VeTrucToaDoVaDaGiac();
}
```

void ThuNhoDaGiac(); // Thu nhỏ đa giác: tương tự với hàm PhongToDaGiac() nhưng tiến hành gọi hàm ThuNho() với tất cả các điểm trong đa giác thay vì PhongTo().

```
216 void DaGiac::ThuNhoDaGiac() {
217     double k;
218     cout << "Nhap do thu nho da giac: ";
219     cin >> k;
220     while (k < 1) {
221         cout << "Nhap lai do thu nho da giac: ";
222         cin >> k;
223     }
224     for (int i = 0; i < TongDiem; i++)
225         A[i].ThuNho(k);
226     XuatDuLieuDaGiac();
227     VeTrucToaDoVaDaGiac();
228 }
```

void VeDaGiac(); // Vẽ đa giác: để vẽ đa giác ta khởi tạo 2 biến int tamOx và tamOy cần thiết để vẽ đa giác nhưng phương pháp vẽ đa giác sẽ phức tạp hơn tam giác.

//Để vẽ các cạnh của đa giác một cách hợp lý thì ta cần sắp xếp các điểm và thứ tự vẽ từ điểm này sang điểm khác bằng một thuật toán sắp xếp các điểm được thực hiện bởi hàm sortPointsByAngle() và truyền tham chiếu vector Point, trước khi sort thì ta khai báo vector kiểu Point là temp và truyền vào tất cả các điểm của đa giác bằng vòng lặp for.

//Sau khi sort ta khởi tạo tiếp một vector int tên Ve và truyền vào toạ độ x,y của các điểm trong vector temp bằng vòng lặp for (với 1 đơn vị là 50 pixel và cách tính toán lại toạ độ đã giải thích ở lớp TamGiac). Sau khi truyền hết các điểm vào vector Ve ta truyền tiếp một điểm vào vector Ve là điểm đầu tiên để có thể vẽ một đa giác khép kín khi vẽ đa giác bằng hàm drawpoly().

//Trước khi vẽ bằng drawpoly() ta sẽ tạo biến con trỏ int arr bằng data() của vector Ve để có thể vẽ với hàm drawpoly(). Cuối cùng là vẽ bằng hàm drawpoly() truyền vào TongDiem+1 là số lần vẽ cạnh và con trỏ arr.

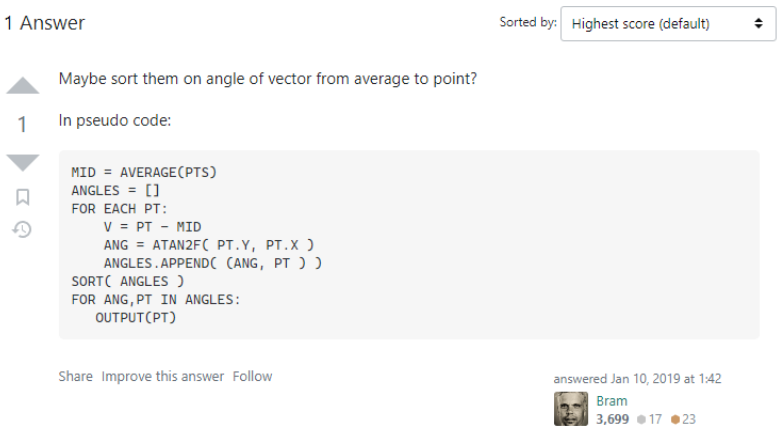
```
144 void DaGiac::VeDaGiac() {
145     int tamOx = int(getmaxx() / 2);
146     int tamOy = int(getmaxy() / 2);
147     vector<Point> temp;
148     // truyền mảng động Point A vào vector Point temp
149     for (int i = 0; i < TongDiem; i++) temp.push_back(A[i]);
150     // sort các điểm trong vector temp bởi góc của chúng với trục toạ độ Oxy
151     sortPointsByAngle(temp);
152     vector <int> Ve;
153     for (int j = 0; j < TongDiem; j++) {
154         int tempx = int(temp[j].x * 50 + tamOx);
155         int tempy = int(-temp[j].y * 50 + tamOy);
156         Ve.push_back(tempx);
157         Ve.push_back(tempy);
158     }
159     Ve.push_back(int(temp[0].x * 50 + tamOx));
160     Ve.push_back(int(-temp[0].y * 50 + tamOy));
161     int* arr = Ve.data();
162     drawpoly(TongDiem + 1, arr);
163 }
```

void VeTrucToaDoVaDaGiac(); // Vẽ trục toạ độ và đa giác: hàm này sẽ clear màn hình trước đó bằng clearviewport() và vẽ trục toạ độ với đa giác bằng 2 hàm VeTrucToaDo() và VeDaGiac().

```
124    void DaGiac::VeTrucToaDoVaDaGiac() {
125        clearviewport();
126        VeTrucToaDo();
127        VeDaGiac();
128    }
```

void sortPointsByAngle(vector<Point>& points); // hàm dùng để sắp xếp thứ tự các điểm được truyền vào từ các điểm của đa giác với truyền tham chiếu từ một vector lớp Point (tránh việc vẽ sai đa giác từ các điểm). Được thực hiện từ gợi ý như sau:

https://gamedev.stackexchange.com/questions/166958/draw-concave-polygon-from-set-of-edge-points



1 Answer

Sorted by: Highest score (default)

1

Maybe sort them on angle of vector from average to point?

In pseudo code:

```
MID = AVERAGE(PTS)
ANGLES = []
FOR EACH PT:
    V = PT - MID
    ANG = ATAN2F( PT.Y, PT.X )
    ANGLES.APPEND( (ANG, PT ) )
SORT( ANGLES )
FOR ANG,PT IN ANGLES:
   OUTPUT(PT)
```

Share Improve this answer Follow

answered Jan 10, 2019 at 1:42

Bram

3,699 ● 17 ● 23

```
69   bool compareAngles(const std::pair<float, Point>& a, const std::pair<float, Point>& b) {
70       return a.first < b.first;
71   }
72
73   void DaGiac::sortPointsByAngle(vector<Point>& points) {
74       // Tinh toan toa do trung binh cua tat ca cac diem
75       Point average;
76       for (const auto& p : points) {
77           average.x += p.x;
78           average.y += p.y;
79       }
80       average.x /= points.size();
81       average.y /= points.size();
82
83       // tinh toan cac goc va luu vao thanh tung cap trong vector
84       std::vector<std::pair<float, Point>> angles;
85       for (const auto& p : points) {
86           double dx = p.x - average.x;
87           double dy = p.y - average.y;
88           double angle = atan2f(dy, dx);
89           angles.emplace_back(angle, p);
90       }
91
92       // sort cac diem boi goc cua chung
93       sort(angles.begin(), angles.end(), compareAngles);
94
95       // luu lai cac diem da sort vao vector ban dau
96       for (size_t i = 0; i < points.size(); i++) {
97           points[i] = angles[i].second;
98       }
99   }
```

- Bước 4: Khởi tạo hàm main
  Hàm main sẽ thực hiện khởi tạo DaGiac a, và gọi tới các hàm của DaGiac gồm NhapDaGiac(), TinhTienDaGiac(), QuayDaGiac(), PhongToDaGiac() và ThuNhoDaGiac(), các hàm như XuatDuLieuDaGiac(), VeDaGiac(),… đã được gọi trong các hàm trên nên không cần phải gọi tới trong main(), kết thúc chương trình khi đã chạy xong.

```
230  int main()
231  {
232      DaGiac a;
233      a.NhapDaGiac();
234      a.TinhTienDaGiac();
235      a.QuayDaGiac();
236      a.PhongToDaGiac();
237      a.ThuNhoDaGiac();
238      return 0;
239  }
```

Ví dụ demo: