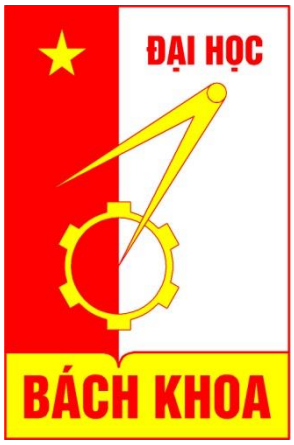
Trường Đại Học Bách Khoa Hà Nội

Viện Điện Tử - Viễn Thông

======o0o======



TỔNG KẾT KHÓA HỌC

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Đề tài:

TRÌNH BÀY CẤU TRÚC CỦA CHƯƠNG TRÌNH SHAPE – EDGE – GRAPH

Sinh viên hướng dẫn: Anh Dương Văn Biên

Chị Nguyễn Thu Quyên

Sinh viên thực hiện: Trần Thu Mai Anh

CTTT Hệ thống nhúng & IoT 01.K64

Hà Nội, 9/2020

Mục lục

Danh mục hình vẽ

Danh mục bảng biểu

1. Giới thiệu

1.1 Tổng quan

Lập trình hướng đối tượng (Object - oriented programming, viết tắt: OOP) là một mẫu hình lập trình dựa trên khái niệm "công nghệ đối tượng", mà trong đó, đối tượng chứa đựng các dữ liệu, trên các trường, thường được gọi là các thuộc tính; và mã nguồn, được tổ chức thành các phương thức. Phương thức giúp cho đối tượng có thể truy xuất và hiệu chỉnh các trường dữ liệu của đối tượng khác mà đối tượng hiện tại có tương tác. Trong lập trình hướng đối tượng, chương trình máy tính được thiết kế bằng cách tách nó ra khỏi phạm vi các đối tượng tương tác với nhau.

Dựa trên nguyên lý kế thừa, trong quá trình mô tả các lớp có thể loại bỏ những chương trình bị lặp, dư. Và có thể mở rộng khả năng sử dụng các lớp mà không cần thực hiện lại. Tối ưu và tái sử dụng code hiệu quả. Đảm bảo rút ngắn thời gian xây dựng hệ thống và tăng năng suất thực hiện.

Sự xuất hiện của 2 khái niệm mới là lớp và đối tượng chính là đặc trưng của phương pháp lập trình hướng đối tượng. Nó đã giải quyết được các khuyết điểm của phương pháp lập trình hướng cấu trúc để lại. Ngoài ra 2 khái niệm này đã giúp biểu diễn tốt hơn thế giới thực trên máy tính.

1.2 Mục tiêu và phạm vi

Mục này tóm tắt nội dung báo cáo, gồm mục đích của báo cáo và các giới hạn nội dung của báo cáo.

1.2.1 Mục tiêu

Báo cáo bao gồm các phần:

a. Tổng quan: Định nghĩa lập trình hướng đối tượng và các phần của một chương trình Shape – Edge – Graph.

b. Mục tiêu và phạm vi: Tóm tắt các phần của báo cáo, mục tiêu của từng phần cũng như phạm vi giới hạn của kiến thức qua chương trình Shape – Edge – Graph.

c. Tổng quan chương trình: Trình bày chi tiết các chức năng, đầu vào, đầu ra của các hàm trong chương trình, UML và sơ đồ khối.

d. Tiêu chí thiết kế: Làm rõ các chức năng, đầu vào, đầu ra cụ thể (định dạng, cấu trúc dữ liệu) của chương trình, môi trường được sử dụng để biên dịch, thực hiện chương trình.

e. Thiết kế chương trình: Trình bày chi tiết các phần đã nêu ở Tổng quan chương trình, các cấu trúc dữ liệu được sử dụng/ xử lý trong các hàm.

f. Triển khai thực hiện: Trình bày chi tiết về cấu trúc các hàm sử dụng trong chương trình, lưu ý một số đoạn mã quan trọng.

g. Kết quả thực nghiệm: Trình bày các kết quả thu được khi chạy thử chương trình

h. Kết luận: Nêu các kết luận

1.2.2 Các giới hạn nội dung của báo cáo

1.2.2.1 Lớp và đối tượng trong C++

- Định nghĩa lớp là định nghĩa một blueprint cho một kiểu dữ liệu, những gì một đối tượng của lớp đó sẽ bao gồm và những hoạt động nào có thể được thực hiện trên một đối tượng đó.
- Một đối tượng được tạo từ một lớp. Khai báo các đối tượng của một lớp giống đúng như chúng ta khai báo các biến của kiểu cơ bản.

1.2.2.2 Tính kế thừa trong C++

- Tính kế thừa cho phép định nghĩa một lớp trong điều kiện một lớp khác, làm cho nó dễ dàng để tạo và duy trì một ứng dụng. Điều này cũng cung cấp cơ hội để tái sử dụng tính năng code và thời gian thực thi nhanh hơn.
- Khi tạo một lớp, thay vì viết toàn bộ các thành viên dữ liệu và các hàm thành viên mới thì có thể kế thừa các thành viên của một lớp đang tồn tại. Lớp đang tồn tại được gọi là Base Class - lớp cơ sở, và lớp mới được xem là Derived Class – lớp thừa kế.

1.2.2.3 Tính đa hình trong C++

- Đa hình xuất hiện khi có một cấu trúc cấp bậc của các lớp và chúng là liên quan với nhau bởi tính kế thừa, là một lời gọi tới một hàm thành viên sẽ làm cho một hàm khác thực thi phụ thuộc vào kiểu của đối tượng mà triệu hồi hàm đó.
- Một hàm virtual là một hàm trong một lớp cơ sở mà được khai báo bởi sử dụng từ khóa virtual. Việc định nghĩa trong một lớp cơ sở một hàm virtual, với phiên bản khác trong một lớp kế thừa, báo cho compiler rằng: chúng ta không muốn Static Linkage cho hàm này. (Static Linkage: sự liên hợp tĩnh – lời gọi hàm được sửa trước khi chương trình được thực thi).

1.2.2.4 Tính bao đóng trong C++

- Là một kỹ thuật bao đóng dữ liệu, và các hàm mà sử dụng chúng và trừu tượng hóa dữ liệu là một kỹ thuật chỉ trưng bày tới các Interface và ẩn chi tiết trình triển khai tới người sử dụng.
- C++ hỗ trợ các thuộc tính của đóng gói và ẩn dữ liệu thông qua việc tạo các kiểu tự định nghĩa gọi là classes. Một lớp có thể chứa các thành viên private, protected và public. Theo mặc định, tất cả thành phần được định nghĩa trong một lớp là private.

1.2.2.5 Interface trong C++

- Interface được triển khai bởi sử dụng các Lớp trừu tượng, là một khái niệm của việc giữ Implementation Detail phân biệt với dữ liệu được liên kết.
- Một lớp được tạo là abstract bằng việc khai báo ít nhất một lần các hàm của nó là hàm pure virtual.
- Mục đích của một Lớp trừu tượng là cung cấp một lớp cơ sở thích hợp để từ đó các lớp khác có thể kế thừa. Các lớp trừu tượng không thể được sử dụng để khởi tạo các đối tượng và chỉ phục vụ như là một Interface.

1.2.2.6 Factory Pattern

- Là một design pattern thuộc nhóm khởi tạo. Bản chất của mẫu thiết kế Factory là "Định nghĩa một giao diện (interface) cho việc tạo một đối tượng, nhưng để các lớp con quyết định lớp nào sẽ được tạo. "Factory method" giao việc khởi tạo một đối tượng cụ thể cho lớp con."

1.2.2.7 File I/O và Stream

- Không có phần mềm nào là không thao tác với file.
- Sử dụng các thư viện <ifstream> cho việc đọc file, <ofstream> cho việc ghi file hoặc <fstream> cho cả 2 mục đích

1.2.2.8 Bộ nhớ động

- Heap: cấp phát bộ nhớ động khi chương trình chạy.
- Có thể cấp phát bộ nhớ tại run time bên trong Heap cho biến đó với một kiểu đã cho bởi sử dụng một toán tử new mà trả về địa chỉ của không gian đã cấp phát.
- Sử dụng toán tử delete để giải phóng bộ nhớ đã được cấp phát trước đó bởi toán tử new.

1.2.2.9 Thư viện STL

- Là một tập hợp các lớp Template để cung cấp các lớp và các hàm được tạo theo khuân mẫu cho mục đích lập trình tổng quát, mà triển khai nhiều thuật toán và cấu trúc dữ liệu được sử dụng phổ biến và thông dụng như vector, list, queue và stack. Gồm ba thành phần:
  + Containers
  + Algorithms
  + Iterators

1.3 Các thuật ngữ viết tắt

Mục này trình bày ý nghĩa/định nghĩa về các khái niệm thuật ngữ sẽ được sử dụng trong báo cáo.

Bảng 1: Các thuật ngữ viết tắt.

Từ viết tắt	Tiếng Anh	Tiếng Việt
OOP	Object Oriented Programing	Lập trình hướng đối tượng
UML	Unified Modeling Language	Ngôn ngữ mô hình hóa thống nhất
STL	Standard Tamplate Library	Thư viện Template chuẩn

2. Tổng quan về chương trình

Mục này trình bày rõ chức năng, đầu vào, đầu ra của chương trình. Sơ đồ lớp của chương trình. Các phần trong chương trình.

2.1 Đầu vào và đầu ra của chương trình

2.1.1 Đầu vào

Các hình vẽ (Hình tròn, hình chữ nhật, hình vuông, hình oval, hình tam giác, đoạn thẳng) với những thuộc tính chung và những thuộc tính riêng cho từng hình.

Các đoạn thẳng nối tâm giữa các hình.

2.1.2 Đầu ra:

Kết quả của các phương thức được thực hiện trên các thuộc tính đầu vào.

2.2 UML và sơ đồ lớp của chương trình

2.2.1 UML

Ngôn ngữ mô hình hóa thống nhất (tiếng Anh: Unified Modeling Language, viết tắt thành UML) là một ngôn ngữ mô hình gồm các ký hiệu đồ họa mà các phương pháp hướng đối tượng sử dụng để thiết kế các hệ thống thông tin một cách nhanh chóng.

Cách xây dựng các mô hình trong UML phù hợp mô tả các hệ thống thông tin cả về cấu trúc cũng như hoạt động. Cách tiếp cận theo mô hình của UML giúp ích rất nhiều cho những người thiết kế và thực hiện hệ thống thông tin cũng như những người sử dụng nó; tạo nên một cái nhìn bao quát và đầy đủ về hệ thống thông tin dự định xây dựng. Cách nhìn bao quát này giúp nắm bắt trọn vẹn các yêu cầu của người dùng; phục vụ từ giai đoạn phân tích đến việc thiết kế, thẩm định và kiểm tra sản phẩm ứng dụng công nghệ thông tin. Các mô hình hướng đối tượng được lập cũng là cơ sở cho việc ứng dụng các chương trình tự động sinh mã trong các ngôn ngữ lập trình hướng đối tượng. Các mô hình được sử dụng bao gồm Mô hình đối tượng (mô hình tĩnh) và Mô hình động.

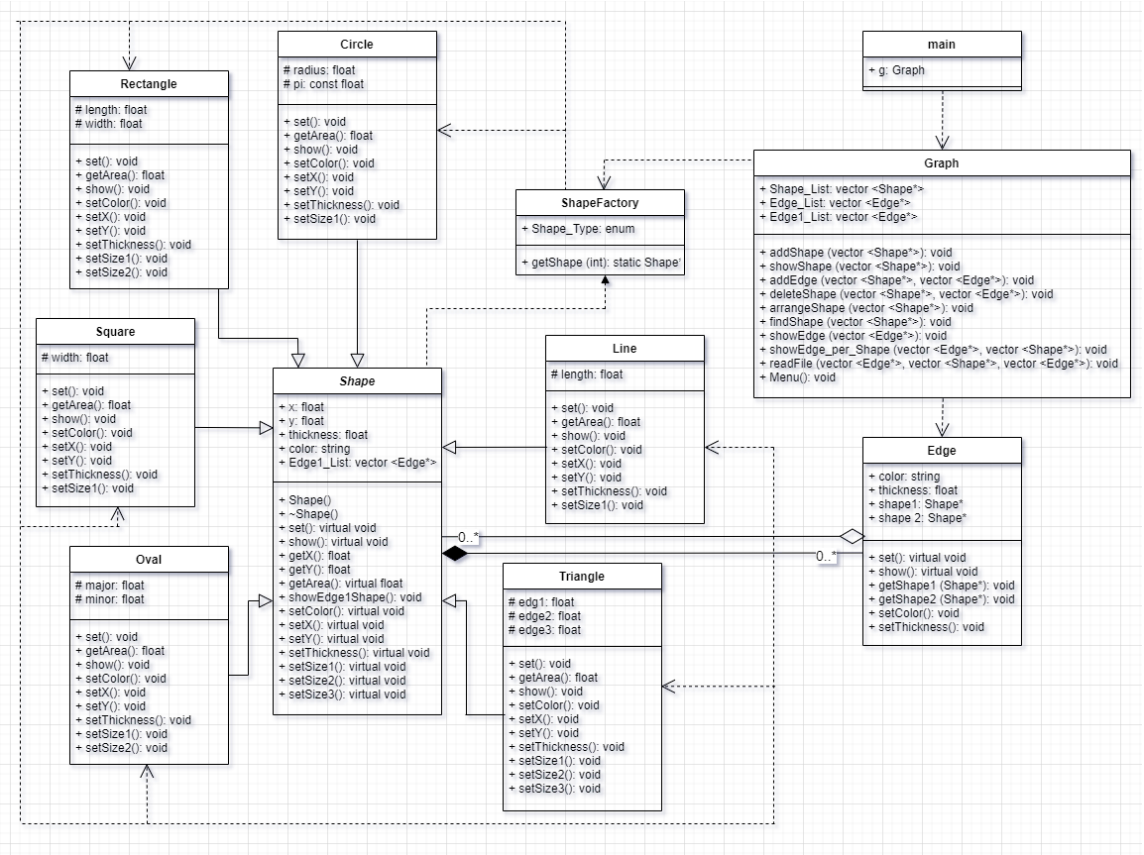
Có các loại sơ đồ UML chủ yếu sau: Sơ đồ lớp, sơ đồ đối tượng, sơ đồ trình tự, sơ đồ tình huống sử dụng,…

2.2.2 Sơ đồ lớp

Một biểu đồ lớp chỉ ra cấu trúc tĩnh của các lớp trong hệ thống. Các lớp là đại diện cho các "đối tượng" được xử lý trong hệ thống. Các lớp có thể quan hệ với nhau trong nhiều dạng thức: liên kết, phụ thuộc, chuyên biệt hóa, đóng gói,…

Tất cả các mối quan hệ đó đều được thể hiện trong biểu đồ lớp, đi kèm với cấu trúc bên trong của các lớp theo khái niệm thuộc tính (attribute) và phương thức (operation). Biểu đồ được coi là biểu đồ tĩnh theo phương diện cấu trúc được miêu tả ở đây có hiệu lực tại bất kỳ thời điểm nào trong toàn bộ vòng đời hệ thống.

Một hệ thống thường sẽ có một loạt các biểu đồ lớp – không phải bao giờ tất cả các biểu đồ lớp này cũng được nhập vào một biểu đồ lớp tổng thể duy nhất – và một lớp có thể tham gia vào nhiều biểu đồ lớp.



Hình 2.1: Sơ đồ lớp của chương trình

2.3 Tổng quan chương trình

Tổng quan, chương trình bao gồm 6 phần:

- Header files: Khai báo các thư viện lưu trữ các hàm sử dụng trong chương trình.
- Class Shape và các class kế thừa: Định nghĩa các lớp bằng các thuộc tính mỗi hình và các hàm tương ứng.
- Class Edge: Định nghĩa lớp bằng các thuộc tính của cạnh và các hàm tương ứng.
- Class Factory (Pattern Design): Định nghĩa phương thức cho việc tạo một Shape mới.
- Class Graph: Định nghĩa lớp bằng các thuộc tính của cạnh và các hàm thực hiện với Shape và Edge.
- Hàm Main(): Chạy chương trình.

3. Tiêu chí thiết kế

Mục này cần làm rõ các tiêu chí thiết kế của phương pháp/thuật toán.

3.1 Trừu tượng hóa dữ liệu

- Trừu tượng hóa dữ liệu phân biệt code thành Interface và Implementation. Vì thế, trong khi thiết kế thành phần, phải giữ Interface độc lập với Implementation, để nếu thay đổi underlying Implementation thì Interface sẽ vẫn còn tồn tại như cũ.
- Trong trường hợp này, bất kỳ chương trình nào đang sử dụng các Interface này, chúng sẽ không bị ảnh hưởng và sẽ cần một sự tái biên dịch với Implementation mới nhất này.

3.2 Tính bao đóng

- Hầu hết đều đặt các thành viên lớp là private theo mặc định, trừ khi thực sự cần thiết phải trưng bày chúng. Đó là tính đóng gói tốt.

- Điều này được áp dụng thường xuyên nhất cho các thành viên dữ liệu, nhưng nó áp dụng như nhau cho tất cả thành viên, bao gồm cả các hàm virtual trong C++.

3.3 Interface

- Một hệ thống hướng đối tượng có thể sử dụng một lớp cơ sở để cung cấp một Interface chung và chuẩn hóa thích hợp cho tất cả các ứng dụng ngoại vi. Vì thế, thông qua kế thừa từ lớp cơ sở trừu tượng đó, các lớp kế thừa được thiết lập theo cách tương tự.
- Các khả năng (ví dụ: các hàm public), được cung cấp bởi các ứng dụng ngoại vi, được cung cấp ở dạng các hàm pure virtual trong lớp cơ sở trừu tượng. Trình triển khai của các hàm pure virtual này được cung cấp trong các lớp kế thừa tương ứng với các kiểu ứng dụng cụ thể.
- Cấu trúc này cũng cho phép các ứng dụng mới được thêm vào hệ thống một cách dễ dàng, ngay cả sau khi hệ thống đó đã được định nghĩa.

4. Thiết kế chương trình

4.1 Kết cấu chi tiết các phần

Mục này trình bày chi tiết các phần đã nêu ở Tổng quan chương trình.

4.1.1 Tệp tiêu đề

```
#include <iostream>
#include <vector>
#include <fstream>
#include <math.h>
#include <string>

using namespace std;
```

Tệp tiêu đề khai báo các thư viện chứa các hàm sử dụng trong chương trình. Cú pháp khai báo: **#include <Tên thư viện>**.

Dòng 1: Khai báo thư viện **iostream**. Đây là thư viện input/ output của C++ hỗ trợ việc nhập xuất, hỗ trợ các lệnh như **cin** và **cout** trong chương trình.

Dòng 2: Khai báo thư viện **vector**. Thư viện này hỗ trợ việc thao tác với mảng động an toàn và dễ dàng, cho phép tạo các mảng động mà không cần cấp phát và thu hồi vùng nhớ bằng cách sử dụng toán tử **new** và **delete**. Trong chương trình khai báo mảng các hình (**Shape_List**), các cạnh (**Edge_List**) và các cạnh xuất phát từ mỗi hình (**Edge1_List**) bằng kiểu dữ liệu **vector**.

Dòng 3: Khai báo thư viện **fstream**. Thư viện này cho phép các kỹ thuật thao tác với file trong chương trình như: khai báo file **file_input** kiểu **ifstream**, sử dụng lệnh **bool fail()** để kiểm tra đối tượng **file_input** đã liên kết được với file cần mở hay không, **bool eof()** để kiểm tra việc đã duyệt tới cuối file hay chưa, ngoài ra còn có lệnh **void close()** để đóng file sau khi sử dụng.

Dòng 4: Khai báo thư viện **math.h**. Thư viện này cung cấp một số hàm toán học cơ bản. Trong chương trình sử dụng hàm **sqrt((p-edge1) * (p-edge2) * (p-edge3))** trong công thức tính diện tích tam giác với độ dài 3 cạnh là **edge1**, **edge2** và **edge3**, nửa chu vi tam giác là **p**.

Dòng 6: Khai báo **using namespace std** mang ý nghĩa "Sử dụng Thư viện chuẩn", thuận tiện hơn trong lập trình. Khi đó, **cout** hay **cin** được sử dụng trực tiếp, không cần qua **std::cout** hay **std::cin**.

4.1.2 Class Shape

4.1.2.1 Thuộc tính

```
class Shape //Lop co so: Shape
{
    public:
        int index; // Thu tu cua hinh
        float x, y, thickness; // Toa do tam va do rong net ve
        string color; // Mau sac hinh ve
        vector <Edge*> Edge1_List; // Danh sach cac canh 1 hinh
```

Thuộc tính của **Shape** gồm **index, x, y, thickness, color và Edge1_List**.

index là thứ tự của hình vẽ, khai báo kiểu số nguyên **int**.

x, y là tọa độ tâm của hình vẽ, khai báo kiểu số thực **float**.

thickness là độ dày nét vẽ, khai báo kiểu số thực **float**.

color là màu sắc hình vẽ, khai báo kiểu chuỗi **string**.

Edge1_List là mảng các cạnh xuất phát từ mỗi hình, khai báo kiểu **vector <Edge*>**, truy nhập bằng con trỏ.

4.1.2.2 Phương thức

Các phương thức của Shape gồm **set()**, **show()**, **getX()**, **getY()**, **getArea()**, **showEdge1Shape()**, **setColor()**, **setX()**, **setY()**, **setThickness()**, **setSize1()**, **setSize2()**, **setSize3()**.

```
        public:
            Shape() // Ham constructor cua Shape
            {
                x = 0;
                y = 0;
                thickness = 0;
                color = "";
            }

            ~Shape() // Ham destructor cua Shape
            {
                x = 0;
                y = 0;
                thickness = 0;
                color = "";
            }

            virtual void set() // Ham nhap thong tin hinh ve
            {
                cin.ignore();
                cout << "Hay nhap mau sac cua hinh ve: " << endl;
                getline (cin,color);
                cout << "Hay nhap vi tri cua hinh ve: " << endl;
                cout << "Hay nhap hoanh do cua hinh ve: " << endl;
                cin >> x;
                cout << "Hay nhap tung do cua hinh ve: " << endl;
                cin >> y;
                cout << "Hay nhap do rong net cua hinh ve: " << endl;
                cin >> thickness;
            }

            virtual void show() // Ham xuat thong tin hinh ve
            {
                cout << "Vi tri cua hinh ve la (" << x << "," << y << ")
    " << endl;
                cout << "Mau sac cua hinh ve la " << color << endl;
```

Hàm **set()** nhập các thông tin hình vẽ qua các thuộc tính. **x, y, thickness** kiểu **float** nhập qua **cin**. **color** kiểu **string** nhập qua **getline**. Lệnh **cin.ignore()** để xóa các kí tự lưu trong bộ nhớ đệm, tránh gặp lỗi.

Hàm **show()** xuất các thông tin hình vẽ đã nhập ra màn hình.

Các hàm **getX()**, **getY()** và **getArea()** để lấy tọa độ và diện tích hình vẽ.

Hàm **showEdge1Shape()** in ra tất cả các cạnh nối tâm một hình bất kì, phụ thuộc vào **Edge** nên được định nghĩa sau sau khi đã định nghĩa class **Edge**.

```
        // Ham lay toa do hinh ve
        float getX()
        {
           return x;
        }

        float getY()
        {
           return y;
        }

        virtual float getArea() // Ham ao lay dien tich hinh ve
        {
            return 0;
        }

        void showEdge1Shape(); // Ham in ra tat ca cac canh noi tam
1 hinh bat ki

        // Cac ham lay thong tin cua Shape tu File
        virtual void setColor (string c)
        {
            color = c;
        }

        virtual void setX (float hd)
        {
            x = hd;
        }

        virtual void setY (float td)
        {
            y = td;
        }
```

```
virtual void setThickness (float tn)
        {
            thickness = tn;
        }

        virtual void setSize1(double size) {}

        virtual void setSize2(double size) {}

        virtual void setSize3(double size) {}

        virtual void writeFile() // Ham ghi file
        {
            fstream file_output ("D:\\MAIANH_K64\\text", ios::app);
            file_output << x << "," << y << "," << color << "," << t
hickness << "}" << endl;
            file_output.close();
        }
};
```

Các hàm **setColor(), setX(), setY(), setThickness(), setSize1(), setSize2() và setSize3()** lấy thông tin của hình từ file. **setSize1(), setSize2()** hay **setSize3()** phụ thuộc vào tính chất từng hình.

Các hàm là hàm ảo (**virtual**) để thông tin lớp kế thừa được in ra.

4.1.3 Class Edge

4.1.3.1 Thuộc tính

```
class Edge // Lop co so: Edge
{
    public:
        string color;
        float thickness;
        Shape* shape1; // Tao bien hinh duoc noi tam dau tien la sha
pe1 kieu Shape
        Shape* shape2; // Tao bien hinh duoc noi tam thu hai la shap
e2 kieu Shape
```

Các thuộc tính của Edge bao gồm **color, thickness, shape1** và **shape2**.

color là màu nét vẽ, khai báo kiểu dữ liệu dạng chuỗi **string**.

thickness là độ dày nét vẽ, khai báo kiểu dữ liệu số thực **float**.

***shape1** và ***shape2** là 2 con trỏ tới 2 hình được nối tâm, khai báo kiểu dữ liệu **Shape**.

4.1.3.2 Phương thức

```
virtual void set() // Ham nhap thong tin canh noi tam
        {
            cin.ignore();
            cout << endl << "Hay nhap mau sac cua canh noi tam: " <<
 endl;
            getline (cin, color);
            cout << "Hay nhap do day net ve cua canh noi tam: " << e
ndl << endl;
            cin >> thickness;
        }

        virtual void show() // Ham xuat thong tin canh noi tam
        {
            cout << endl << "Doan thang noi tam 2 hinh:" << endl;
            shape1-
>show(); // Goi ham xuat thong tin hinh duoc noi tam dau tien
            shape2-
>show(); // Goi ham xuat thong tin hinh duoc noi tam thu hai
            cout << "Mau sac cua canh noi tam la " << color << endl;
            cout << "Do day net ve cua canh noi tam la " << thicknes
s << endl;
        }

        void getShape1(Shape* shape) // Ham lay thong tin hinh can n
oi tam thu nhat
        {
            shape1 = shape;
        }

        void getShape2(Shape* shape) // Ham lay thong tin hinh can n
oi tam thu hai
        {
            shape2 = shape;
        }

        }
```

Các phương thức của Edge bao gồm **set(), show(), getShape1(), getShape2(), setColor()** và **setThickness().**

Hàm **set()** nhập thông tin cạnh nối tâm qua các thuộc tính **color** và **thickness**. **thickness** kiểu **float** nhập qua **cin**. **color** kiểu **string** nhập qua **getline.** Lệnh **cin.ignore()** để xóa các kí tự lưu trong bộ nhớ đệm, tránh gặp lỗi.

Hàm **show()** xuất thông tin cạnh nối tâm ra màn hình. Trong đó có lệnh xuất thông tin 2 hình được nối tâm bởi cạnh tương ứng là **shape1->show()** và **shape2-> show()**.

Hàm **getShape1()** và **getShape2()** lấy thông tin 2 hình được nối tâm bởi cạnh.

Các hàm **setColor(), setThickness()** lấy thông tin cho cạnh từ file.

4.1.4 Các class kế thừa.

Các class kế thừa class Shape gồm có **Circle** (Hình tròn), **Rectangle** (Hình chữ nhật), **Square** (Hình vuông), **Oval** (Hình oval), **Line** (Đoạn thẳng) và **Triangle** (Hình tam giác).

```
class Circle: public Shape // Lop ke thua: Circle
{
    protected:
        float radius; // Ban kinh hinh tron
        const float pi = 3.14;

    public:
        void set() // Ham nhap ban kinh hinh tron
        {
            cout << endl << "Hay nhap cac thong tin cua hinh tron!"
<< endl;
            Shape::set();
            cout << "Hay nhap ban kinh hinh tron: " << endl;
            cin >> radius;
        }

        float getArea() // Ham tinh dien tich hinh tron
        {
            return pi * radius * radius;
        }
```

```
        void show() // Ham in thong tin hinh tron
        {
            cout << endl << "Thong tin hinh tron da nhap la: " << en
dl;
            Shape::show();
            cout << "Ban kinh hinh tron la " << radius << endl;
            cout << "Dien tich hinh tron la " << getArea() << endl <
< endl;
        }

        // Cac ham lay thong tin cua Shape tu File
        virtual void setColor (string c)
        {
            color = c;
        }

        virtual void setX (float hd)
        {
            x = hd;
        }

        virtual void setY (float td)
        {
            y = td;
        }

        virtual void setThickness (float tn)
        {
            thickness = tn;
        }

        virtual void setSize1(double size) // Circle chi co 1 kich t
huoc
        {
            radius = size;
        }

        void writeFile() // Ham ghi file hinh tron
        {
            fstream file_output ("D:\\MAIANH_K64\\text", ios::app);
            file_output << "shape{cirlce" << "," << radius << ",";

            file_output.close();
            Shape::writeFile();
        }
};
```

Ngoài các thuộc tính và phương thức kế thừa từ class **Shape**, mỗi class con có các thuộc tính và phương thức riêng biệt phụ thuộc từng hình. Nói riêng về class **Cirlce**.

4.1.4.1 Thuộc tính

radius là bán kính hình tròn, khai báo dữ liệu kiểu số thực **float**.

pi là số π, khai báo hằng kiểu số thực **float**, bằng **3.14**.

4.1.4.2 Phương thức

Các phương thức của **Circle** gồm **set(), getArea(), show(), setColor(), setX(), setY(), setThickness()** và **setSize1()**.

Hàm **set()** nhập các thông tin riêng của từng hình (bán kính đối với hình tròn) qua thuộc tính riêng của class. Ngoài ra còn có hàm **Shape**::**set()** để nhập thông tin hình ở Shape.

Hàm **getArea()** tính diện tích hình (Với hình tròn, công thức tính diện tích là **pi * radius * radius**).

Hàm **show()** in thông tin hình ra màn hình, ngoài các thuộc tính riêng của hình (bán kính và diện tích đối với hình tròn) thì còn bao gồm các thuộc tính chung của hình ở **Shape** qua hàm **Shape**::**show()**.

Ngoài ra còn có các hàm lấy thông tin từ file như **setColor(), setX(), setY(), setThickness()** và **setSize1()** (hình tròn có 1 kích thước nên chỉ có 1 hàm **setSize1()**).

4.1.5 Class Factory

4.1.5.1 Thuộc tính

Class **Factory** có thuộc tính **Shape_Type** khai báo kiểu **enum**, có tác dụng liệt kê các hình.

```
class ShapeFactory // class Factory
{
    public:
        enum Shape_Type // Khai bao cac shape kieu liet ke
        {
            circle = 1,
            rectangle = 2,
            square = 3,
            oval = 4,
            line = 5,
            triangle = 6
        };
```

4.1.5.2 Phương thức

Class **Factory** bao gồm hàm tĩnh **(static) getShape()** khai báo kiểu con trỏ **Shape***, với tham số là **shape_type** kiểu **int**. Với mỗi **shape_type**, hàm sẽ trả về con trỏ lưu trữ hình tương ứng. Nếu không thỏa mãn, default: trả về con trỏ **NULL (nullptr)**.

4.1.6 Class Graph

4.1.6.1 Thuộc tính

```
class Graph
{
    public:
        vector <Shape*> Shape_List; // vector luu danh sach cac Shape
        vector <Edge*> Edge_List, Edge1_List; // vector luu danh sach
 cac Edge va vector luu danh sach cac Edge xuat phat tu mot hinh
```

Thuộc tính của class **Graph** bao gồm **Shape_List, Edge_List** và **Edge1_List**.

Shape_List là mảng các hình, khai báo kiểu **vector <Shape*>**, truy nhập bằng con trỏ.

Edge_List là mảng các cạnh, khai báo kiểu **vector <Edge*>**, truy nhập bằng con trỏ.

Edge1_List là mảng các cạnh xuất phát từ một hình, khải báo kiểu **vector <Edge*>**, truy nhập bằng con trỏ.

4.1.6.2 *Phương thức*

```
void Menu() // Menu cac lua chon
        {
            int choice; // Lua chon
            while (1)
            {
                cout << endl << endl;
                cout << "                    MENU                    "
<< endl;
                cout << "NHAP SO TUONG UNG VOI CONG VIEC BAN MUON TH
UC HIEN!" << endl;
                cout << "1. Nhap thong tin hinh tu ban phim." << end
l;
                cout << "2. Nhap thong tin hinh va canh tu File." <<
 endl;
                cout << "3. In ra man hinh thong tin cac hinh vua nh
ap." << endl;
                cout << "4. Xoa mot hinh." << endl;
                cout << "5. Sap xep cac hinh theo thu tu tang dan ve
 dien tich." << endl;
                cout << "6. Tim kiem hinh voi toa do tam nhap tu ban
 phim." << endl;
                cout << "7. Them mot canh noi tam tu ban phim." << e
ndl;
                cout << "8. In ra tat ca cac canh noi tam da co." <<
 endl;
                cout << "9. In ra tat ca cac canh noi tam cua mot hi
nh." << endl;
                cout << "10. Ghi file" << endl;
                cout << "11. Thoat khoi chuong trinh!" << endl;
```

Các phương thức ở class **Graph** thể hiện tóm tắt qua hàm **Menu()** và chính **Menu()** cũng là một phương thức ở class này. Bao gồm các hàm: **addShape(), readFile(), showShape(), deleteShape(), arrangeShape(), findShape(), addEdge(), showEdge()** và **showEdge_per_Shape()**.

Với hàm **Menu()**, vòng lặp **while (1)** lặp liên tục cho tới khi người dùng nhập 10 cho **choice** và thoát chương trình. Ngoài ra, với **choice** không thuộc từ 1 tới 10, hàm hiện thông báo mời nhập lại các số thỏa mãn.

4.2 Cấu trúc dữ liệu

Mục này trình bày chi tiết về các cấu trúc dữ liệu được sử dụng trong chương trình.

Cấu trúc dữ liệu được sử dụng trong chương trình là mảng con trỏ kiểu vector.

4.2.1 Giới thiệu chung

vector được hiện thực là một mảng liên tục, nó có thể truy xuất ngẫu nhiên. Không những tối ưu về tốc độ truy xuất như mảng, thiết kế của **vector** còn cho phép thêm và gỡ bỏ 1 phần tử 1 cách linh động bằng toán tử **new** và **delete**.

4.2.2 Sử dụng

- Để sử dụng **vector** cần khai báo thư viện **<vector>** với cú pháp: **#include <vector>**
- Đây là một lớp được định nghĩa sẵn trong STL với các hàm constructor đã được overload và destructor cho phép dọn dẹp vùng nhớ cùng với iterator cho phép truy cập và quản lý phần tử trong nó.
- Cú pháp khai báo vector: **vector<Kiểu dữ liệu> tên_vector;**
- Trong chương trình có sử dụng vector qua các khai báo:
  + **vector <Edge*> Edge1_List;** Mảng các cạnh nối tâm xuất phát từ một hình.
  + **vector <Shape*> Shape_List;** Mảng các hình
  + **vector <Edge*> Edge_List;** Mảng chứa tất cả các cạnh nối tâm

4.2.3 Các hàm sử dụng với vector

4.2.3.1 vector::push_back()

- Chức năng: thêm vào một phần tử cho **vector**.
- Trong chương trình có sử dụng hàm **push_back()** qua:
  + **Shape_List.push_back(ShapeFactory::getShape(shape));** Tạo một hình mới cho **Shape_List** qua Factory Pattern.
  + **Edge_List.push_back(new Edge);** Tạo một cạnh mới cho **Edge_List** đồng thời cấp phát bộ nhớ bằng toán tử **new**.

4.2.3.2 vector::begin()

- Chức năng: Trả về con trỏ trỏ tới phần tử đầu tiên của **vector**.
- Trong chương trình có sử dụng hàm **begin()** qua:

+ **Shape_List.erase(Shape_List.begin() + j - 1);** Xóa hình thứ j do **Shape_List.begin()** trỏ về phần tử thứ nhất.

4.2.3.3 vector::size()

- Chức năng: Trả về số lượng phần tử của **vector**.
- Trong chương trình có sử dụng hàm **size()** qua:
  + **If (Edge_List.size() == 0) cout << "Danh sách không có cạnh nối tâm nào!";** Nếu danh sách chưa các cạnh nối tâm **Edge_List** có số phần tử là **0** thì in ra thông báo.

4.2.3.4 vector::erase()

- Chức năng: Xóa một phần tử của **vector** khi biết vị trí của nó.
- Trong chương trình có sử dụng hàm **erase()** qua:
  + **Shape_List.erase(Shape_List.begin() + j - 1);** Xóa hình thứ j do **Shape_List.begin()** trỏ về phần tử thứ nhất.

4.2.3.5 Một số hàm khác

- **vector::pop_back():** Xóa phần tử cuối cùng của **vector**.
- **vector::swap():** Hoán đổi nội dung của 2 **vector**.
- **vector::empty():** Kiểm tra vector có rỗng hay không, trả về **true** hoặc **false**.
- **vector::clear():** Xóa tất cả phần tử của **vector**.

5. Triển khai thực hiện (implementation)

5.1 Triển khai các hàm

Mục này trình bày chi tiết về cấu trúc các hàm sử dụng trong chương trình.

5.1.1 *Hàm tạo một hình ở lớp FactoryShape*

```
static Shape* getShape (int shape_type)
        {
            if (shape_type == circle)
            {
                return new Circle;
            }
            else if (shape_type == rectangle)
            {
                return new Rectangle;
            }
            else if (shape_type == square)
            {
                return new Square;
            }
            else if (shape_type == oval)
            {
                return new Oval;
            }
            else if (shape_type == line)
            {
                return new Line;
            }
            else if (shape_type == triangle)
            {
                return new Triangle;
            }
            return NULL;
        }
```

Hàm **getShape()** thuộc lớp **FactoryShape**, kiểu **Shape***. Mang tham số **shape_type** kiểu **int**, truy nhập bằng con trỏ. Hàm mang kiểu **static**, là hàm tĩnh, dùng để khai báo thành viên dữ liệu dùng chung cho mọi thể hiện của lớp, là một bản duy nhất tồn tại trong suốt quá trình chạy của chương trình. Hàm được xây dựng dựa vào **Factory Method Design Pattern**.

Sử dụng cấu trúc **if else**, với mỗi **shape_type** tương ứng với kết quả **return** một hình mới và được cấp phát bộ nhớ qua toán tử **new**.

5.1.2 *Thêm một hình mới nhập từ bàn phím*

Hàm **addShape()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>**, truy nhập bằng con trỏ.

```
void addShape(vector <Shape*>& Shape_List) // Ham tao them mot hinh
moi
        {
            int shape; // So tuong ung voi moi hinh
            cout << endl << "Nhap so tuong ung voi hinh ban muon the
m!" << endl; // Tao menu cac hinh
            cout << "1. Hinh tron." << endl;
            cout << "2. Hinh chu nhat." << endl;
            cout << "3. Hinh vuong." << endl;
            cout << "4. Hinh oval." << endl;
            cout << "5. Doan thang." << endl;
            cout << "6. Hinh tam giac." << endl << endl;
            cin >> shape;

            Shape_List.push_back(ShapeFactory::getShape(shape)); //
Goi ham them mot hinh vao vector Shape_List
            Shape_List.back()-
>set(); // Goi ham nhap thong tin hinh vua moi duoc them vao
            for (int i = 1; i <= Shape_List.size(); i++)
            {
                Shape_List.back()->index = i;
            }
        }
```

Tại đây, sử dụng biến **shape** kiểu **int** nhận thông tin lựa chọn hình của người dùng sau menu, chương trình gọi hàm **Shape_List.push_back()** thêm một hình vào **vector Shape_List** thông qua gián tiếp bởi lớp **ShapeFactory**, đồng thời gọi hàm nhập thông tin cho hình mới thêm vào (**Shape_List.back()**) bằng hàm **set()** ở class **Shape**.

5.1.3 In ra tất cả các hình đã nhập

```
void showShape(vector <Shape*>& Shape_List) // Ham in ra tat ca cac
hinh da nhap
        {
            int i;
            if (Shape_List.size() == 0) cout << endl << "Danh sach k
hong co hinh ve nao!" << endl << endl;
            else
            {
                for (i=0; i<Shape_List.size(); i++)
                {
                    cout << endl << "Hinh thu " << i+1 << ":" << end
l;
                    Shape_List[i]->show();
                }
            }
        }
```

Hàm **showShape()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>**, truy nhập bằng con trỏ.

Để kiểm tra danh sách hình vẽ có trống hay không, sử dụng lệnh **Shape_List.size()** trả về số lượng phần tử đang được sử dụng trong mảng, nếu bằng **0**, hiển thị thông báo ra màn hình **"Danh sách không có hình vẽ nào!".** Ngược lại, khi tồn tại ít nhất 1 hình trong danh sách, biến **i** mang kiểu **int** duyệt từ **0** tới cuối mảng, hiển thị lần lượt các hình thông qua lệnh **show()** đã được định nghĩa ở class **Shape**.

5.1.4 Thêm một cạnh mới nhập từ bàn phím

```
void addEdge(vector <Shape*>& Shape_List, vector <Edge*>& Edge_List)
 // Ham them mot canh
        {
            if (Shape_List.size() < 2) cout << "Danh sach khong du h
inh ve de them canh!" << endl;
            else
            {
                int u,v;
                cout << endl << "Nhap thu tu 2 hinh ban muon noi tam
:" << endl;
                cin >> u;
                cin >> v;

                Edge_List.push_back(new Edge); // Goi toi ham them 1
 canh noi tam

                // Goi toi ham them thong tin hinh can noi tam cho c
anh vua them
                Edge_List.back()->getShape1(Shape_List[u-1]);
                Edge_List.back()->getShape2(Shape_List[v-1]);

                // Goi toi ham them thong tin canh vua nhap vao thon
g tin 2 hinh tuong ung
                Shape_List[u-1]-
>Edge1_List.push_back(Edge_List.back());
                Shape_List[v-1]-
>Edge1_List.push_back(Edge_List.back());
                Edge_List.back()->set();
            }
        }
```

Hàm **addEdge()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>** và tham số **Edge_List** kiểu dữ liệu **vector <Edge*>**, đều truy nhập bằng con trỏ.

Để thêm được một cạnh nối tâm 2 hình, danh sách các hình phải có ít nhất **2** hình trở lên. Tương tự, sử dụng **Shape_List.size()** trả về số lượng phần tử đang

được sử dụng trong mảng, nếu kết quả trả về ít hơn **2** hình hiện thông báo **"Danh sách không đủ hình vẽ để thêm cạnh!"**.

Ngược lại, nếu danh sách thỏa mãn số lượng hình để thêm một cạnh nối tâm, sử dụng 2 biến **u** và **v** kiểu **int** để nhập thứ tự 2 hình muốn nối tâm. Tại đây, gọi hàm **Edge_List.push_back(new Edge)** để thêm 1 cạnh nối tâm vào danh sách **Edge_List**, đồng thời cấp phát bộ nhớ cho nó qua toán tử **new**. Ở đây phải sử dụng các hàm thích hợp để set thông tin cần thiết cho cạnh được nối tâm và hình được nối tâm.

5.1.5 Xóa một hình trong danh sách đã nhập

```
void deleteShape(vector <Shape*>& Shape_List, vector <Edge*>& Edge1_
List) // Ham xoa mot hinh va xoa canh noi tam xuat phat tu no
        {
            int j;
            if (Shape_List.size() == 0) cout << "Danh sach khong co
hinh ve nao de xoa!" << endl << endl;
            else
            {
                cout << "Hay nhap thu tu hinh muon xoa!" << endl;
                cin >> j;
                if ((j<1) || (j>=Shape_List.size()))
                    cout << "Khong ton tai thu tu hinh ve nay!" << e
ndl << endl; // Thu tu hinh bat dau tu 1
                else
                {
                    Shape_List.erase(Shape_List.begin()+j-
1); // Xoa hinh thu j
                    Shape_List[j-1]-
>Edge1_List.clear(); // Xoa cac duong noi tam xuat phat tu hinh vua
xoa
                    cout << "Danh sach cac hinh sau khi xoa la: " <<
 endl;
                    showShape(Shape_List);
                }
            }
        }
```

Hàm **deleteShape()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>**, truy nhập bằng con trỏ.

Để kiểm tra danh sách hình vẽ có trống hay không, sử dụng lệnh **Shape_List.size()** trả về số lượng phần tử đang được sử dụng trong mảng, nếu bằng **0**, hiển thị thông báo ra màn hình **"Danh sách không có hình vẽ nào!"**.

Nếu danh sách có ít nhất 1 phần tử, yêu cầu nhập thứ tự hình muốn xóa qua biến j kiểu int. Kiểm tra nếu thứ tự nhỏ hơn 1 hoặc lớn hơn số lượng phần tử đang được sử dụng trong mảng qua lệnh Shape_List.size(), hiển thị thông báo "Không tồn tại hình vẽ này".

Trường hợp cuối cùng, khi thỏa mãn về số lượng phần tử có trong danh sách và thứ tự của hình cần xóa, sử dụng lệnh Shape_List.erase() với tham số Shape_List.begin() + j – 1 để xóa hình thứ j (Do Shape_List.begin() trỏ về hình đầu tiên). Cuối cùng hiện danh sách các hình sau xóa để kiểm tra kết quả việc xóa hình bằng lệnh showShape() đã định nghĩa ở trên.

5.1.6 Sắp xếp các hình theo diện tích tăng dần

```
void arrangeShape(vector <Shape*>& Shape_List) // Ham sap xep cac hi
nh theo dien tich tang dan
        {
            Shape* temp; // Bien tam thoi de so sanh 2 Shape
            int p,q;
            for (p=0; p<Shape_List.size()-1; p++)
            {
                for (q=p+1; q<Shape_List.size(); q++)
                {
                    if (Shape_List[p]->getArea() > Shape_List[q]-
>getArea())
                    {
                        temp = Shape_List[p];
                        Shape_List[p] = Shape_List[q];
                        Shape_List[q] = temp; // Swap vi tri 2 Shape
                    }
                }
            }
            cout << "Danh sach cac hinh sau khi sap xep la:" << endl
;
            showShape(Shape_List);
        }
```

Hàm arrangeShape() thuộc lớp Graph, kiểu void. Mang tham số Shape_List kiểu dữ liệu vector <Shape*>, truy nhập bằng con trỏ.

Tại đây sử dụng biến temp kiểu Shape và 2 biến p, q để duyệt qua các phần tử trong Shape_List. Lần lượt duyệt p từ phần tử đầu tiên, duyệt q từ phần tử thứ 2 và so sánh diện tích của chúng qua lệnh getArea() định nghĩa ở class Shape, đổi chỗ vị trí nếu không thỏa mãn điều kiện với temp. Kết thúc vòng lặp, ta được danh

sách các hình sắp xếp theo diện tích tăng dần. Sử dụng lệnh **showShape()** đã định nghĩa ở trên để kiểm tra lại kết quả.

5.1.7 Tìm kiếm một hình theo tọa độ

```
void findShape(vector <Shape*>& Shape_List) // Ham tim kiem hinh the
o toa do
        {
            float x, y;
            int m;
            cout << "Hay nhap toa do hinh can tim!" << endl;
            cout << "Hay nhap hoanh do hinh can tim!" << endl;
            cin >> x;
            cout << "Hay nhap tung do hinh can tim!" << endl;
            cin >> y;

            for (m=0; m<Shape_List.size(); m++)
            {
                if ((x == Shape_List[m]-
>getX()) && (y == Shape_List[m]->getY()))
                {
                    cout << "Toa do da nhap ung voi hinh: " << endl;
                    Shape_List[m]->show();
                    break;
                }
                else if (m == Shape_List.size()-1)
                cout << "Khong ton tai hinh ve nay!" << endl << endl
;
            }
        }
```

Hàm **findShape()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>**, truy nhập bằng con trỏ.

Sử dụng biến **x**, **y** kiểu **float** để nhập giá trị tọa độ hình cần tìm, biến **m** để duyệt lần lượt từng phần tử trong danh sách **Shape_List** từ **0** tới **Shape_List.size() – 1**, so sánh lần lượt **x** và **y** với hàm lấy tọa độ **getX** và **getY** đã được định nghĩa ở lớp **Shape**, nếu thỏa mãn cả 2 điều kiện (qua toán tử &&), thông báo có hình tương ứng với tọa độ đã nhập. Tại **m** thỏa mãn, xuất thông tin hình có chỉ số là **m** qua lệnh **Shape_List[m]->show()** đã được định nghĩa ở lớp **Shape**. Ngay lập tức, dừng vòng lặp bằng lệnh **break.**

Nếu vòng lặp tiếp tục chạy tới phần tử cuối cùng của danh sách **Shape_List** mà chưa **break**, lúc này **m** = **Shape_List.size() – 1**, vậy tức là đã duyệt hết phần tử và nhận được kết quả không có phần tử nào thỏa mãn, vì vậy **cout** thông báo **“Không tồn tại hình vẽ này!”.**

5.1.8 *In ra tất cả các cạnh đã nhập*

```
void showEdge(vector <Edge*>& Edge_List) // Ham in tat ca cac canh d
a them
        {
            int m;
            if (Edge_List.size() == 0)
                cout << endl << "Danh sach khong co canh noi tam nao
!" << endl << endl;
            else
            {
                for (m=0; m<Edge_List.size(); m++)
                {
                    cout << "Canh thu " << m+1 << ":" << endl;
                    Edge_List[m]->show();
                }
            }
        }
```

Hàm **showEdge()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Edge_List** kiểu dữ liệu **vector <Edge*>**, truy nhập bằng con trỏ.

Tương tự, để kiểm tra danh sách cạnh nối tâm có trống hay không, sử dụng lệnh **Edge_List.size()** trả về số lượng phần tử đang được sử dụng trong mảng, nếu bằng **0**, hiển thị thông báo ra màn hình **"Danh sách không có cạnh nối tâm nào!".**

Ngược lại, nếu danh sách có ít nhất **1** phần tử, sử dụng biến **m** kiểu **int** duyệt các phần tử trong danh sách từ **0** tới phần tử **Edge_List.size() – 1**, in ra thông tin mỗi cạnh lần lượt bằng lệnh **show()** đã được định nghĩa là lớp **Edge**.

5.1.9 *In ra tất cả các cạnh xuất phát từ một hình*

Hàm **showEdge_per_Shape()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Edge_List** kiểu dữ liệu **vector <Edge*>** và **Shape_List** kiểu dữ liệu **vector <Shape*>**, đều truy nhập bằng con trỏ.

Tương tự, để kiểm tra danh sách cạnh nối tâm với hình này có trống hay không, sử dụng lệnh **Edge1_List.size()** tại vị trí phần tử thứ **n** (kiểu **int**, được nhập từ bàn phím) trong danh sách các hình vẽ (**Shape_List[n]**) trả về số lượng phần tử đang được sử dụng trong mảng, nếu bằng **0**, hiển thị thông báo ra màn hình **"Không có cạnh nào xuất phát từ hình này!".**

```
void showEdge_per_Shape(vector <Edge*>& Edge1_List, vector <Shape*>&
 Shape_List) // Ham xuat thong tin cac canh noi tam 1 hinh xac dinh
        {
            int n,o;
            cout << endl << "Nhap thu tu hinh ban muon xem thong tin
 canh:" << endl;
            cin >> n;
            if (Shape_List[n]->Edge1_List.size() == 0)
                cout << endl << "Khong co canh nao xuat phat tu hinh
 nay!" << endl;
            else
            {
                cout << endl << "Thong tin cac canh xuat phat tu hin
h ban vua nhap la:" << endl;
                Shape_List[n]-
>showEdge1Shape(); // Goi toi ham in ra canh noi tam cua hinh thu n
trong Shape_List
            }
        }
```

Ngược lại, xuất thông tin các cạnh xuất phát từ hình này bởi lệnh **showEdge1Shape()** đối với phần tử **Shape_List[n]** đã được định nghĩa ở class **Shape**.

5.1.10 Đọc file

```
void readFile(vector <Shape*> &Shape_List, vector <Edge*>& Edge_List
, vector <Edge*>& Edge1_List)  // Ham doc File
        {
            ifstream file_input ("E:\\MaiAnh\\STUDY\\TEST\\.vscode\\
file.txt");
            string getinput;
            if (file_input.fail()) // Check xem co lien ket duoc kh
            {
                cout << "Khong mo duoc file nay!" << endl;
                return;
            }
            {
                while (!file_input.eof())
                {
                    getline (file_input, getinput, '{'); // Doc file
 tu dau toi '{', gan vao getinput.

                    switch (getinput[0]) // Xet ki tu dau tien cua g
etinput
                    {
                        case 's': // Shape
                        {
                            getline(file_input, getinput, ','); // D
oc tiep file cho toi ','

                            switch (getinput[0]) // Xet ki tu thu 0
cua getinput
```

Hàm **readFile()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Edge_List** kiểu dữ liệu **vector <Edge*>, Shape_List** kiểu dữ liệu **vector <Shape*>** và **Edge1_List** kiểu dữ liệu **vector <Edge*>**, đều truy nhập bằng con trỏ.

Tại hàm này ta khai báo **file_input** kiểu **ifstream** để liên kết với file cần đọc, ở đây là file **file.txt**, bên cạnh biến **getinput** kiểu **string** để nhận thông tin từ file. Để kiểm tra có liên kết được với file hay không (có đúng địa chỉ file không, file có tồn tại không,…), sử dụng hàm **fail()**. Nếu **file_input.fail()** trả về giá trị **true**, hàm đưa ra thông báo lỗi **"Không mở được file này!".**

Ngược lại, khi đã liên kết được file, sử dụng vòng lặp **while** (**!file_input.eof()**) (mang ý nghĩa chạy cho tới cuối file) để tiến hành các bước đọc file. Lúc này, sử dụng hàm **getline()** để lấy nội dung gán vào **getinput**, kiểm tra kí tự đầu tiên (**getinput**[0]). Lúc này sẽ có 2 trường hợp xảy ra: với kí tự đầu là **'s'**, đây là thông tin cho các hình vẽ; với kí tự đầu là **'e'**, đây là thông tin cho các cạnh nối tâm.

Xét trường hợp đầu cho các hình vẽ. Tiếp tục sử dụng hàm **getline()** để lấy nội dung gán vào **getinput**, kiểm tra kí tự đầu tiên (**getinput**[0]). Lúc này, sẽ có các trường hợp sau: kí tự đầu là **'c'**, tương ứng với hình tròn (**Circle**), kí tự đầu là **'r'**, tương ứng với hình chữ nhật (**Rectangle**),… tương tự với các trường hợp còn lại.

```
case 'c': // Cirlce
{
    Shape_List.push_back(ShapeFactory::getShape(1)); // them Circle
    //Doc ki tu tiep theo cho toi ',', gan vao getinput
    //Gan getinput vao toa do x, y
    //Stof: Chuyen kieu du lieu tu string -> float

    getline (file_input, getinput, ',');
    Shape_List.back()->setX(stof(getinput));
    getline (file_input, getinput, ',');
    Shape_List.back()->setY(stof(getinput));

    getline (file_input, getinput, ',');
    Shape_List.back()-> setSize1 (stof(getinput));

    getline (file_input, getinput, ',');
    Shape_List.back()-> setColor(getinput);
    getline (file_input, getinput, '}'); //
    Shape_List.back()-> setThickness(stof(getinput))

    getline(file_input,getinput); // Gan phan con lai cua dong vao i
nput de ket thuc dong.
    break;
}
```

Xét riêng trường hợp hình tròn. Với kí tự đầu là ‘c’, chương trình sẽ gọi hàm **push_back()** để thêm một hình tròn mới vào **Shape_List** đồng thời cấp bộ nhớ cho nó qua lớp **FactoryShape**. Tiếp tục đọc chuỗi lần lượt bằng **getline()** và gán vào các hàm đã định nghĩa ở class **Shape**, từ đó gọi hàm để gửi thông tin đến phần tử mới được thêm vào **Shape_List** (**Shape_List.back()**). Hàm **stof()** cho phép chuyển kiểu dữ liệu từ **string** sang **float**.

```
case 'e': // Edge
{
	Edge_List.push_back(new Edge); // Them 1 canh noi tam

	getline (file_input, getinput, ','); // Doc ki tu tiep theo ch
o toi ',', gan vao getinput
	Edge_List.back()->getShape1(Shape_List[stof(getinput)-
1]); // Goi toi ham them thong tin hinh getinput cho canh vua them
	Shape_List[stof(getinput)-1]-> Edge1_List.push_back
(Edge_List.back());// Goi toi ham them thong tin canh vua nhap vao t
hong tin hinh tuong ung

	getline (file_input, getinput, ','); // Doc ki tu tiep theo ch
o toi ',', gan vao getinput
	Edge_List.back()->getShape2(Shape_List[stof(getinput)-
1]); // Goi toi ham them thong tin hinh getinput cho canh vua them

	Shape_List[stof(getinput)-1]-> Edge1_List.push_back
(Edge_List.back()); // Goi toi ham them thong tin canh vua nhap vao
thong tin hinh tuong ung

	getline (file_input, getinput, ','); // Doc ki tu tiep theo ch
o toi ',', gan vao getinput
	Edge_List.back()-> setColor(getinput);
// Gan input vao mau sac Edge

	getline (file_input, getinput, '}'); // Doc ki tu tiep theo ch
o toi ',', gan vao getinput
	Edge_List.back()-> setThickness(stof(getinput));
// Gan input vao do day net ve Edge

	getline (file_input, getinput); // Get phan con lai cua dong
}
```

Với trường hợp cho các cạnh nối tâm. Chương trình sẽ gọi hàm **push_back**(**new Edge**) để thêm một cạnh nối tâm mới vào **Edge_List** đồng thời cấp phát bộ nhớ cho nó. Tương tự các hàm trên, hàm **getline()** lấy thông tin lần lượt và gán cho các

hàm đã định nghĩa sẵn ở lớp **Edge**, gửi thông tin tới phần tử mới được thêm vào **Edge_List** (**Edge_List.back()**)

5.1.11 Ghi file

```
void writeFile(vector <Shape*> &Shape_List, vector <Edge*>& Edge_List
) // Ham ghi file
        {
            fstream file_output("E:\\MaiAnh\\STUDY\\TEST\\.vscode\\f
ile.txt",ios::app);
            for (int i = 0; i < Shape_List.size(); i++) // Ghi file
Shape
            {
                Shape_List.at(i)->writeFile();
            }
            for (int i = 0; i < Edge_List.size() ; i++) // Ghi file
Edge
            {
                Edge_List.at(i)->writeFile();
            }
            file_output.close();
        }
```

Hàm **writeFile()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Edge_List** kiểu dữ liệu **vector <Edge*>, Shape_List** kiểu dữ liệu **vector <Shape*>** đều truy nhập bằng con trỏ.

Tại hàm này ta khai báo **file_output** kiểu **ifstream** để liên kết với file cần đọc, ở đây là file **file.txt**. Sử dụng các vòng lặp của biến **i** kiểu **int** chạy từ **0** tới **Shape_List.size()** hoặc **Edge_List.size()** để lần lượt ghi các thông tin đã nhập ra file **file.txt** bằng lệnh **writeFile()** đã được định nghĩa ở class **Shape** và class **Edge**.

Tại đây có sử dụng hàm **vector.at()** để trả về một tham chiếu đến phần tử vị trí bất kì trong vector. Hàm trái ngược với toán từ thành viên [], không kiểm tra giới hạn, nó tự động kiểm tra **n** có nằm trong giới hạn các phần tử hợp lệ trong vector, đưa ra ngoại lệ **out_of_range** nếu nó không phải.

5.2 Một số đoạn mã quan trọng

Mục này trình bày giải thích một số đoạn mã quan trọng trong chương trình.

5.2.1 Hàm in ra tất cả các cạnh nối tâm xuất phát từ một hình bất kì

```
void Shape::showEdge1Shape() // Ham in ra tat ca cac canh noi tam
 1 hinh bat ki
    {
        int i;
        for (i=0; i<Edge1_List.size(); i++)
        {
            Edge1_List[i]->show();
        }
    }
```

Hàm **showEdge1Shape()** thuộc lớp **Shape**, kiểu **void**.

Tuy hàm thuộc lớp **Shape** nhưng cần có định nghĩa của cả class **Shape** và class **Edge** vì hàm tác động lên 1 hình nhưng lại cần sử dụng các thuộc tính và phương thức của cạnh. Vì vậy, tại class **Shape**, chưa thể định nghĩa ngay hàm được và chỉ có thể khai báo hàm như sau: **void showEdge1Shape();**

Để in danh sách tất cả các cạnh nối tâm 1 hình bất kì, hay là in ra các phần tử của **Edge1_List**, sử dụng biến **i** kiểu **int** để duyệt lần lượt các phần tử từ đầu tới cuối danh sách. In thông tin của cạnh bằng toán tử -> và hàm **show()** đã được định nghĩa ở class **Edge**.

5.2.2 Hàm thêm một hình mới

```
void addShape(vector <Shape*>& Shape_List) // Ham tao them mot hinh
        {
            int shape; // So tuong ung voi moi hinh
            cout << endl << "Nhap so tuong ung voi hinh ban muon the
m!" << endl; // Tao menu cac hinh
            cout << "1. Hinh tron." << endl;
            cout << "2. Hinh chu nhat." << endl;
            cout << "3. Hinh vuong." << endl;
            cout << "4. Hinh oval." << endl;
            cout << "5. Doan thang." << endl;
            cout << "6. Hinh tam giac." << endl << endl;
            cin >> shape;

            Shape_List.push_back(ShapeFactory::getShape(shape));
            Shape_List.back()->set();
// Goi ham nhap thong tin hinh vua moi duoc them vao
            for (int i = 1; i <= Shape_List.size(); i++)
            {
                Shape_List.back()->index = i;
            }
        }
```

Hàm **addShape()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>,** truy nhập bằng con trỏ.

Hàm là ứng dụng của Factory pattern trong việc tạo thêm 1 đối tượng mới một cách linh hoạt hơn, thay vì phải **push_back()** lần lượt từng đối tượng với lệnh **switch case** hay **if else**.

Sử dụng tham số **shape** kiểu **int** để nhận lựa chọn người dùng, chương trình gọi hàm **getShape()** đã được định nghĩa ở **ShapeFactory**, hàm tương đương với **new Loại_Shape (Circle, Rectangle,…)**, để **push_back()** hình mới thỏa mãn với yêu cầu người dùng. Từ đó gọi lại hàm **set()** đã được định nghĩa ở class **Shape** để nhập thông tin cho hình vừa nhập được là **Shape_List.back().**

5.2.3 Hàm thêm một cạnh nối tâm

```
void addEdge(vector <Shape*>& Shape_List, vector <Edge*>& Edge_List) //
 Ham them mot canh
        {
            if (Shape_List.size() < 2) cout << "Danh sach khong du hinh
 ve de them canh!" << endl;
            else
            {
                int u,v;
                cout << endl << "Nhap thu tu 2 hinh ban muon noi tam:"
<< endl;
                cin >> u;
                cin >> v;

                Edge_List.push_back(new Edge); // Goi toi ham them

                Edge_List.back()->getShape1(Shape_List[u-1]);
                Edge_List.back()->getShape2(Shape_List[v-1]);

                Shape_List[u-1]-> Edge1_List.push_back
(Edge_List.back());
                Shape_List[v-1]-
>Edge1_List.push_back(Edge_List.back());
                Edge_List.back()->set();
            }
        }
```

Hàm **addEdge()** thuộc lớp **Graph**, kiểu **void**. Mang tham số **Shape_List** kiểu dữ liệu **vector <Shape*>**, **Edge_List** kiểu dữ liệu **vector <Edge*>**, đều truy nhập bằng con trỏ.

Bỏ qua bước xét điều kiện danh sách Shape_List đủ hình vẽ để thêm cạnh cũng như thứ tự hình không thỏa mãn đã trình bày ở trên. Với trường hợp thỏa mãn, chương trình gọi tới hàm thêm một cạnh nối tâm push_back(new Edge) vào danh sách Edge_List. Ngay sau đó, cần phải xác định hình cần nối tâm cho cạnh nối tâm vừa thêm vào danh sách (Edge_List.back()) bằng lệnh getShape1() và getShape2() đã được định nghĩa ở class Edge.

Cuối cùng, cần phải nhập thông tin cạnh này vào danh sách các cạnh nối tâm xuất phát từ mỗi hình. Tương tự, ta push_back() phần tử vừa mới thêm vào danh sách (Edge_List.back()) vào danh sách Edge1_List, tương ứng với mỗi hình Shape_List[u-1] và Shape_List[v-1].

6. Kết quả thực nghiệm

6.1 Môi trường kiểm tra thuật toán

Mục này mô tả môi trường được sử dụng để biên dịch, chạy thuật toán. Cấu hình máy tính dùng để chạy. Hệ điều hành. Trình biên dịch chương trình.

6.1.1 Trình biên dịch chương trình

Chương trình sử dụng trình biên dịch là VS Code (Visual Studio Code).

- Version: 1.49.0 (user setup)
- Commit: e790b931385d72cf5669fcefc51cdf65990efa5d
- Electron: 9.2.1
- Chrome: 83.0.4103.122
- Node.js: 12.14.1
- V8: 8.3.110.13 – electron.0
- OS: Window_NT x64 10.0.18363

6.1.2 Cấu hình máy tính

6.1.2.1 Bộ xử lí

- Hãng CPU: Intel
- Công nghệ CPU: Core I5

- Loại CPU: 7200U
- Tốc độ CPU: 2.50 Ghz
- Tốc độ tối đa: 3.18 Ghz
- Bộ nhớ đệm: 3M Cache

6.1.2.2 RAM

- Dung lượng RAM: 4 GB
- Loại RAM: DDR3L
- Tốc độ BUS RAM: 1600 Mhz
- Số RAM onboard: 1 RAM 4 GB
- Số khe RAM còn lại: 1
- Hỗ trợ RAM tối đa: 12 GB

6.1.2.3 Đồ họa

- Chipset đồ họa: NVIDIA Geforce GT 920M
- Loại DAC: Intergrated RAMDAC
- Bộ nhớ đồ họa: 2 GB
- Thiết kế card: Card rời

6.1.2.4 Lưu trữ

- Loại ổ đĩa: HDD
- Ổ cứng: 500GB

6.1.2.5 Thông tin pin

- Loại pin: Lithium-ion
- Pin: 3 Cells
- Pin không thể tháo rời

6.1.3 Hệ điều hành

- OS: Window 10
- Loại: Window 10 Pro
- Phiên bản: 1909

6.2 Bộ dữ liệu đầu vào

Mục này trình bày cụ thể về bộ dữ liệu đầu vào (các thiết kế ví dụ) được sử dụng để chạy thử chương trình.

Dữ liệu đầu vào bao gồm:

6.2.1 File text (File text E:\\MaiAnh\\STUDY\\TEST\\.vscode\\file.txt) có nội dung:

- shape{rect,15,15,13.1,5.3,blue,2}
- shape{circle,10,10,5.2,red,1)
- shape{square,9,9,12.6,white,3}
- shape{oval,12,12,14.1,7.2,yellow,2.5}
- shape{tri,20,20,10,14,17,orange,7}
- edge{1,2,green,8}
- edge{3,2,gray,1}
- edge{4,5,black,4}

6.2.2 Các dữ liệu nhập từ bàn phím

*6.2.2.1 Class **Shape***

- **color** kiểu **string**
- **x, y, thickness** kiểu **float**

*6.2.2.2 Class kế thừa class **Shape***

- **radius** (class **Circle**) kiểu **float**
- **legth, width** (class **Rectangle**) kiểu **float**
- **width** (class **Square**) kiểu **float**
- **major, minor** (class **Oval**) kiểu **float**
- **length** (class **Line**) kiểu **float**
- **edge1, edge1, edge3** (class **Triangle**) kiểu **float**

*6.2.2.3 Class **Edge***

- **color** kiểu **string**
- **thickness** kiểu **float**

Ngoài ra đầu vào còn một số biến đầu vào kiểu **int** mang thông tin lựa chọn của người dùng (lựa chọn ở menu chung, lựa chọn ở menu các hình,…)

6.3 Kết quả

Mục này trình bày các kết quả thu được khi chạy thử chương trình (sự đúng sai, thời gian thực hiện, số lượng bộ nhớ cần sử dụng).

```
                    MENU
NHAP SO TUONG UNG VOI CONG VIEC BAN MUON THUC HIEN!
1. Nhap thong tin hinh tu ban phim.
2. Nhap thong tin hinh va canh tu File.
3. In ra man hinh thong tin cac hinh vua nhap.
4. Xoa mot hinh.
5. Sap xep cac hinh theo thu tu tang dan ve dien tich.
6. Tim kiem hinh voi toa do tam nhap tu ban phim.
7. Them mot canh noi tam tu ban phim.
8. In ra tat ca cac canh noi tam da co.
9. In ra tat ca cac canh noi tam cua mot hinh.
10. Ghi file
11. Thoat khoi chuong trinh!
```

Hình 6.1. Menu các công việc cần thực hiện

Khi chạy chương trình, một menu như hình sẽ hiện ra. Nhập các số từ 1 – 10 để thực hiện các công việc tương ứng. Nếu không chương trình thông báo không hợp lệ và yêu cầu nhập lại.

Khi nhấn 1, chương trình hiện ra như sau:

```
Nhap thong tin hinh tu ban phim.

Nhap so tuong ung voi hinh ban muon them!
1. Hinh tron.
2. Hinh chu nhat.
3. Hinh vuong.
4. Hinh oval.
5. Doan thang.
6. Hinh tam giac.
```

Hình 6.2: Menu các hình

Nhập các số từ 1 – 6 để chọn hình mong muốn nhập. Nhấn 5 để tạo mới một đoạn thẳng. Lúc đó, chương trình sẽ hiện ra lần lượt các thuộc tính của đoạn thẳng yêu cầu nhập từ bàn phím như sau:

```
Nhap thong tin cua doan thang!
Hay nhap mau sac cua hinh ve:
Green
Hay nhap vi tri cua hinh ve:
Hay nhap hoanh do cua hinh ve:
2
Hay nhap tung do cua hinh ve:
2
Hay nhap do rong net cua hinh ve:
1
Hay nhap do dai doan thang:
9
```

Hình 6.3: Yêu cầu nhập thuộc tính đoạn thẳng

Menu lại hiện ra. Nhấn 2 để đọc file có sẵn. Màn hình báo đã nhập thông tin như sau:

```
2
Nhap thong tin hinh va canh tu File.
```

Hình 6.4: Xác nhận đã nhập thông tin từ file

Menu xuất hiện. Nhấn 3 để in ra màn hình tất cả hình đã nhập từ bàn phím và từ file.

```
Hinh thu 1:

Thong tin doan thang vua nhap la:
Vi tri cua hinh ve la (2,2)
Mau sac cua hinh ve la Green
Do rong net cua hinh ve la 1
Do dai doan thang la 9
Dien tich doan thang la 0


Hinh thu 2:

Thong tin hinh chu nhat vua nhap la:
Vi tri cua hinh ve la (15,15)
Mau sac cua hinh ve la blue
Do rong net cua hinh ve la 2
Chieu dai va chieu rong hinh chu nhat la 13.1 va 5.3
Dien tich hinh chu nhat la 69.43


Hinh thu 3:

Thong tin hinh tron da nhap la:
Vi tri cua hinh ve la (10,10)
Mau sac cua hinh ve la red
Do rong net cua hinh ve la 1
Ban kinh hinh tron la 5.2
Dien tich hinh tron la 84.9056


Hinh thu 4:

Thong tin hinh Oval vua nhap la:
Vi tri cua hinh ve la (12,12)
Mau sac cua hinh ve la yellow
Do rong net cua hinh ve la 2.5
Do dai truc dai va truc ngan hinh Oval la 14.1va 7.2
Dien tich hinh Oval la 79.6932


Hinh thu 5:
```

Hình 6.5: Kết quả xuất các hình đã nhập

Kết quả cho tất cả 6 hình gồm đoạn thẳng đã nhập ở đầu chương trình và 5 hình nhập từ file ở mục 2.

Làm tương tự với các lựa chọn còn lại, ta có các kết quả:

- Bấm phím 4 để xóa một hình.

```
Hay nhap thu tu hinh muon xoa!
1
Danh sach cac hinh sau khi xoa la:

Hinh thu 1:

Thong tin hinh chu nhat vua nhap la:
Vi tri cua hinh ve la (15,15)
Mau sac cua hinh ve la blue
Do rong net cua hinh ve la 2
Chieu dai va chieu rong hinh chu nhat la 13.1 va 5.3
Dien tich hinh chu nhat la 69.43


Hinh thu 2:

Thong tin hinh tron da nhap la:
Vi tri cua hinh ve la (10,10)
Mau sac cua hinh ve la red
Do rong net cua hinh ve la 1
Ban kinh hinh tron la 5.2
Dien tich hinh tron la 84.9056


Hinh thu 3:

Thong tin hinh Oval vua nhap la:
Vi tri cua hinh ve la (12,12)
Mau sac cua hinh ve la yellow
Do rong net cua hinh ve la 2.5
Do dai truc dai va truc ngan hinh Oval la 14.1va 7.2
Dien tich hinh Oval la 79.6932


Hinh thu 4:

Thong tin hinh tam giac vua nhap la:
Vi tri cua hinh ve la (20,20)
Mau sac cua hinh ve la orange
Do rong net cua hinh ve la 7
Do dai 3 canh tam giac la 10, 14va 17
```

Hình 6.6: Kết quả lựa chọn xóa một hình

- Bấm phím 5 để sắp xếp các hình theo thứ tự tăng dần về diện tích.

```
Hinh thu 1:

Thong tin hinh tam giac vua nhap la:
Vi tri cua hinh ve la (20,20)
Mau sac cua hinh ve la orange
Do rong net cua hinh ve la 7
Do dai 3 canh tam giac la 10, 14va 17
Dien tich hinh tam giac la 15.4556


Hinh thu 2:

Thong tin hinh chu nhat vua nhap la:
Vi tri cua hinh ve la (15,15)
Mau sac cua hinh ve la blue
Do rong net cua hinh ve la 2
Chieu dai va chieu rong hinh chu nhat la 13.1 va 5.3
Dien tich hinh chu nhat la 69.43


Hinh thu 3:

Thong tin hinh Oval vua nhap la:
Vi tri cua hinh ve la (12,12)
Mau sac cua hinh ve la yellow
Do rong net cua hinh ve la 2.5
Do dai truc dai va truc ngan hinh Oval la 14.1va 7.2
Dien tich hinh Oval la 79.6932


Hinh thu 4:

Thong tin hinh tron da nhap la:
Vi tri cua hinh ve la (10,10)
Mau sac cua hinh ve la red
Do rong net cua hinh ve la 1
Ban kinh hinh tron la 5.2
Dien tich hinh tron la 84.9056
```

Hình 6.7: Sắp xếp các hình theo diện tích

- Bấm phím 6 để tìm kiếm hình với tọa độ nhập từ bàn phím.

```
Hay nhap toa do hinh can tim!
Hay nhap hoanh do hinh can tim!
12
Hay nhap tung do hinh can tim!
12
Toa do da nhap ung voi hinh:

Thong tin hinh Oval vua nhap la:
Vi tri cua hinh ve la (12,12)
Mau sac cua hinh ve la yellow
Do rong net cua hinh ve la 2.5
Do dai truc dai va truc ngan hinh Oval la 14.1va 7.2
Dien tich hinh Oval la 79.6932
```

Hình 6.8: Tìm kiếm hình với tọa độ nhập từ bàn phím

- Bấm phím 7 để thêm một cạnh nối tâm từ bàn phím.

```
Nhap thu tu 2 hinh ban muon noi tam:
1
2

Hay nhap mau sac cua canh noi tam:
Red
Hay nhap do day net ve cua canh noi tam:

2
```

Hình 6.9: Thêm một cạnh nối tâm từ bàn phím

- Bấm phím 8 để in ra tất cả các cạnh nối tâm đã có.

```
In ra tat ca cac canh noi tam da nhap.
Canh thu 1:

Doan thang noi tam 2 hinh:

Thong tin doan thang vua nhap la:
Vi tri cua hinh ve la (2,2)
Mau sac cua hinh ve la green
Do rong net cua hinh ve la 1
Do dai doan thang la 4
Dien tich doan thang la 0


Thong tin hinh chu nhat vua nhap la:
Vi tri cua hinh ve la (15,15)
Mau sac cua hinh ve la blue
Do rong net cua hinh ve la 2
Chieu dai va chieu rong hinh chu nhat la 13.1 va 5.3
Dien tich hinh chu nhat la 69.43

Mau sac cua canh noi tam la green
Do day net ve cua canh noi tam la 8
Canh thu 2:

Doan thang noi tam 2 hinh:

Thong tin hinh tron da nhap la:
Vi tri cua hinh ve la (10,10)
Mau sac cua hinh ve la red
Do rong net cua hinh ve la 1
Ban kinh hinh tron la 5.2
Dien tich hinh tron la 84.9056


Thong tin hinh chu nhat vua nhap la:
Vi tri cua hinh ve la (15,15)
Mau sac cua hinh ve la blue
Do rong net cua hinh ve la 2
Chieu dai va chieu rong hinh chu nhat la 13.1 va 5.3
Dien tich hinh chu nhat la 69.43

Mau sac cua canh noi tam la gray
Do day net ve cua canh noi tam la 1
Canh thu 3:
```

Hình 6.10: In ra tất cả các cạnh nối tâm đã có

- Bấm phím 9 để in ra tất cả các cạnh nối tâm xuất phát từ một hình.

```
Nhap thu tu hinh ban muon xem thong tin canh:
2

Thong tin cac canh xuat phat tu hinh ban vua nhap la:

Doan thang noi tam 2 hinh:

Thong tin hinh Oval vua nhap la:
Vi tri cua hinh ve la (12,12)
Mau sac cua hinh ve la yellow
Do rong net cua hinh ve la 2.5
Do dai truc dai va truc ngan hinh Oval la 14.1va 7.2
Dien tich hinh Oval la 79.6932


Thong tin hinh tam giac vua nhap la:
Vi tri cua hinh ve la (20,20)
Mau sac cua hinh ve la orange
Do rong net cua hinh ve la 7
Do dai 3 canh tam giac la 10, 14va 17
Dien tich hinh tam giac la 15.4556

Mau sac cua canh noi tam la black
Do day net ve cua canh noi tam la 4
```

Hình 6.11: In ra tất cả các cạnh xuất phát từ một hình

- Bấm phím 10 để ghi file.

```
10
Ghi file.
```

Hình 6.12. Xác nhận đã ghi file.

7. Kết luận

Chương trình Shape – Edge – Graph đã thể hiện rõ các tính chất của lập trình hướng đối tượng.

Qua khóa học, em đã bước đầu làm quen với với lập trình C++ nói chung cũng như lập trình hướng đối tượng nói riêng. Ngoài việc cài đặt và sử dụng thành thạo Visual Studio Code, em đã học thêm nhiều kiến thức bổ ích về OOP cũng như cách

trình bày code sao cho rõ ràng, rành mạch. Cuối cùng là cách làm một bài báo cáo đúng chuẩn, hệ thống lại toàn bộ kiến thức đã học.

Em cũng cảm ơn thầy cô đã tạo điều kiện cho em tham gia khóa học, cũng như 2 anh chị mentor đã dẫn dắt và giúp đỡ em trong suốt khóa học vừa qua.

8. Tài liệu tham khảo

Mục này bao gồm các tài liệu tham khảo theo đúng chuẩn trích dẫn bài báo khoa học.

https://www.geeksforgeeks.org/

https://www.cplusplus.com/