МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе№2

по дисциплине «Объектно-ориентированное программирование»

Тема:«Наследование»

Студент гр.7382 Гиззатов А.С.

Преподаватель ЖангировТ.Р.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, принцип их работы, способ организации в памяти, раннее и позднее связывания в языке C++. В соответствии с индивидуальным заданием разработать систему классов для представления геометрических фигур.

Задание.

Необходимоспроектировать систему классов для моделирования геометрических фигур (в соответствии с полученным индивидуальным заданием). Задание предполагает использование виртуальных функций в иерархии наследования, проектирование и использование абстрактного базового класса. Разработанные классы должны быть наследниками абстрактного класса Shape,содержащего методыдля перемещения в указанныекоординаты,поворота на заданныйугол,масштабирования на заданныйкоэффициент,у с т а н о в к и и п о л у ч е н и я ц в е т а , а т а к ж е оператор выводав поток.

Необходимо т а к ж е о б е с п е ч и т ь однозначнуюидентификациюкаждогообъекта.

Решение должно содержать:

- Условиезадания;
- UML диаграмму разработанныхклассов;
- Текстовоеобоснование проектныхрешений;
- Реализацию классов на языкеC++.

Индивидуальное задание.

Вариант 3 – реализовать систему классов для фигур:

1. Треугольник;
2. Трапеция;

3. Равносторонний треугольник.

UML диаграмма разработанных классов.

UML диаграмма разработанных классов представлена в приложении А.

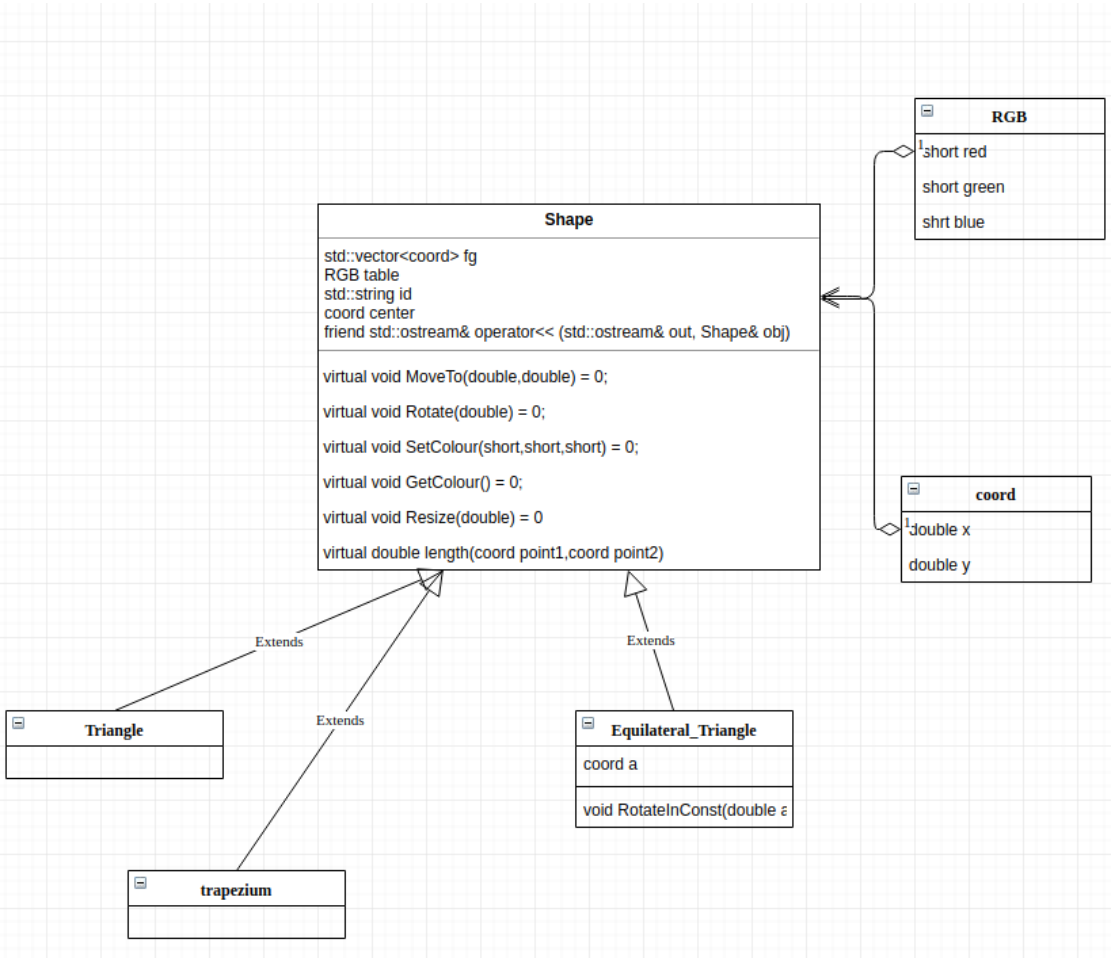
Реализация классов на языке C++.

Реализация классов представлена в приложении Б.

Выводы.

В ходе выполнения лабораторной работы была спроектирована система классов для работы с геометрическими фигурами в соответствии с индивидуальным заданием. В иерархии наследования были использованы виртуальные функции, базовый класс при этом является виртуальным. Были реализованы методы перемещения фигуры в заданные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, была реализована однозначная идентификация объекта.

ПРИЛОЖЕНИЕ А



UML ДИАГРАММА КЛАССОВ

ПРИЛОЖЕНИЕ Б

РЕАЛИЗАЦИЯ КЛАССОВ НА ЯЗЫКЕ C++

```
#include <iostream>
#include <vector>
#include <cmath>
#define PI 3.14159265
struct RGB{
	short red;
	short green;
	short blue;
};

struct coord{
	double x;
	double y;
};

class Shape{
public:
	virtual double length(coord point1,coord point2){
		return sqrt(pow(point1.x-point2.x,2)+pow(point1.y-point2.y,2));
	}
	virtual void MoveTo(double,double) = 0;
	virtual void Rotate(double) = 0;
	virtual void SetColour(short,short,short) = 0;
	virtual void GetColour() = 0;
	virtual void Resize(double) = 0;
	friend std::ostream& operator<< (std::ostream& out, Shape& obj);
protected:
	std::vector<coord> fg;
	RGB table = {0,0,0};
	std::string id;
	coord center;
};
std::ostream& operator<< (std::ostream& out, Shape& obj){
		out << obj.id<< std::endl;
		for(int i = 0; i < obj.fg.size(); i++){
			out << "x "<< i+1 << " - " << obj.fg[i].x << "|" << "y " << i+1 << " - " << obj.fg[i].y
<< std::endl;

		}
		out << "center is - " << obj.center.x << " | " << obj.center.y  << std::endl;
		return out;
	}


class Triangle : public Shape{
public:
	Triangle(std::vector<coord> source){
		fg = source;
		center.x = 0;
		center.y = 0;
		for(int i=0; i< fg.size(); i++){
			center.x += fg[i].x;
			center.y += fg[i].y;
		}
		id = "Triangle";
		center.x = center.x/3;
		center.y = center.y/3;
```

```
            std::cout << length(fg[0],fg[1]) << std::endl;
          std::cout << length(fg[0],fg[2]) << std::endl;
          std::cout << length(fg[1],fg[2]) << std::endl;
      }
      void SetColour(short red,short green,short blue){
            table.red = red;
            table.green = green;
            table.blue = blue;
      }
      void GetColour(){
            std::cout << "red - " << table.red << "| green - " << table.green << "| blue - "
<<table.blue << std::endl;
      }
      void MoveTo(double x,double y){
            double length_x,length_y;
            length_x = x - center.x;
            length_y = y - center.y;
            for(int i = 0;i < fg.size();i++){
                  fg[i].x = fg[i].x + length_x;
                  fg[i].y = fg[i].y + length_y;
            }
            center.x = x;
          center.y = y;
      }
      void Resize(double k){
            coord vec;
            for(int i =0 ;i < fg.size();i++){
                  vec.x = fg[i].x - center.x;
                  vec.y = fg[i].y - center.y;
                  vec.x *=k;
                  vec.y *=k;
                  fg[i].x = vec.x + center.x;
                  fg[i].y = vec.y + center.y;
            }
            std::cout << length(fg[0],fg[1]) << std::endl;
          std::cout << length(fg[0],fg[2]) << std::endl;
          std::cout << length(fg[1],fg[2]) << std::endl;
     }

      void Rotate(double angle){
          for(int i=0;i<fg.size();i++){
              fg[i].x = center.x + (fg[i].x - center.x)*cos(angle*M_PI/180)-(fg[i].y -
center.y)*sin(angle*M_PI/180);
              fg[i].y = center.y + (fg[i].y - center.y)*cos(angle*M_PI/180)+(fg[i].x -
center.x)*sin(angle*M_PI/180);
          }
     }
};

class trapezium : public Shape{
      public:
      trapezium(std::vector <coord> source){
            fg = source;
            id = "Trapezium";
            center.x = 0;
            center.y = 0;
            for(int i = 0;i< fg.size();i++){
                  center.x += fg[i].x;
                  center.y += fg[i].y;
            }
            center.x /= 4;
```

```
            center.y /= 4;
            std::cout << " center - " << center.x << "|" << center.y << std::endl;
        }
        void SetColour(short red,short green,short blue){
            table.red = red;
            table.green = green;
            table.blue = blue;
        }
        void Rotate(double angle){
            for(int i=0;i<fg.size();i++){
                fg[i].x = center.x + (fg[i].x - center.x)*cos(angle*M_PI/180)-(fg[i].y -
center.y)*sin(angle*M_PI/180);
                fg[i].y = center.y + (fg[i].y - center.y)*cos(angle*M_PI/180)-(fg[i].x -
center.x)*sin(angle*M_PI/180);
            }
        }
        void MoveTo(double x,double y){
            double length_x,length_y;
            length_x = x - center.x;
            length_y = y - center.y;
            for(int i = 0;i < fg.size();i++){
                fg[i].x = fg[i].x + length_x;
                fg[i].y = fg[i].y + length_y;
            }
               center.x = x;
            center.y = y;
        }
        void GetColour(){
            std::cout << "red - " << table.red << " | green - " << table.green << " | blue - "
<<table.blue << std::endl;
        }

         void Resize(double k){
            coord vec;
            for(int i =0 ;i < fg.size();i++){
                vec.x = fg[i].x - center.x;
                vec.y = fg[i].y - center.y;
                vec.x *=k;
                vec.y *=k;
                fg[i].x = vec.x + center.x;
                fg[i].y = vec.y + center.y;
            }
            std::cout << length(fg[0],fg[1]) << std::endl;
            std::cout << length(fg[0],fg[2]) << std::endl;
            std::cout << length(fg[1],fg[2]) << std::endl;
        }
};
class Equilateral_Triangle : public Shape{
        public:
        Equilateral_Triangle(std::vector<coord> source){
                fg.push_back(source[0]);
                a = source[0];
                center = source.back();
                for(int i = 1;i < 3;i++){
                        RotateInConst(120);
                        a = fg[i];
                }
        }
        void RotateInConst(double angle){
                coord b;
                b.x = center.x + (a.x - center.x)*cos(angle*M_PI/180)-(a.y - center.y)*sin(angle*M_PI/180);
```

```
                b.y = center.y + (a.y - center.y)*cos(angle*M_PI/180)+(a.x - center.x)*sin(angle*M_PI/180);
                fg.push_back(b);

        }
        void SetColour(short red,short green,short blue){
                table.red = red;
                table.green = green;
                table.blue = blue;
        }
        void Rotate(double angle){
                for(int i=0;i<fg.size();i++){
                        fg[i].x = center.x + (fg[i].x - center.x)*cos(angle*M_PI/180)-(fg[i].y -
center.y)*sin(angle*M_PI/180);
                        fg[i].y = center.y + (fg[i].y - center.y)*cos(angle*M_PI/180)-(fg[i].x -
center.x)*sin(angle*M_PI/180);
                }
        }
        void MoveTo(double x,double y){
                double length_x,length_y;
                length_x = x - center.x;
                length_y = y - center.y;
                for(int i = 0;i < fg.size();i++){
                        fg[i].x = fg[i].x + length_x;
                        fg[i].y = fg[i].y + length_y;
                }
                center.x = x;
                center.y = y;
        }
        void GetColour(){
                std::cout << "red - " << table.red << " | green - " << table.green << " | blue - "
<<table.blue << std::endl;
        }

        void Resize(double k){
                coord vec;
                for(int i =0 ;i < fg.size();i++){
                        vec.x = fg[i].x - center.x;
                        vec.y = fg[i].y - center.y;
                        vec.x *=k;
                        vec.y *=k;
                        fg[i].x = vec.x + center.x;
                        fg[i].y = vec.y + center.y;
                }
                std::cout << length(fg[0],fg[1]) << std::endl;
                std::cout << length(fg[0],fg[2]) << std::endl;
                std::cout << length(fg[1],fg[2]) << std::endl;
        }

                private:
        coord a;

};



int main(){
        short n;
        double x,y;
        double width;
        std::cin >> n;
        std::vector<coord> coords;
```

```
        coord point;
        switch (n){
        case 1:{
                        for(int i = 0; i < 3;i++){
                              std::cin >> x >> y;
                                   point.x = x;
                                   point.y = y;
                                   coords.push_back(point);
                }
                          Triangle Trngl(coords);
                          std::cout << Trngl;
                          Trngl.GetColour();
                          Trngl.Resize(2);
                          std::cout << Trngl;
                          break;
                }
                case 2:{
                          for(int i = 0; i < 4;i++){
                                    std::cin >> x >> y;
                            point.x = x;
                             point.y = y;
                             coords.push_back(point);
                          }

                          trapezium Tr(coords);
                          std::cout << Tr;
                          break;
                }
                case 3:{
                          for(int i = 0; i < 2;i++){
                             std::cin >> x >> y;
                                    point.x = x;
                             point.y = y;
                             coords.push_back(point);
                     }
                     Equilateral_Triangle Tr_1(coords);
                          std::cout << Tr_1;
                          Tr_1.SetColour(1,5,10);
                          Tr_1.GetColour();
                          Tr_1.MoveTo(5,5);
                          Tr_1.Resize(2);
                          Tr_1.Rotate(180.0);
                          std::cout << Tr_1;
                     Tr_1.Rotate(180.0);
                          std::cout << Tr_1;
                     break;
               }

                  default:{
                          std::cout << "error" << std::endl;
                          break;
                }
       }

}
```