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. 7304 _____ Есиков О.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2019

Цель работы.

Изучить механизм наследования в языке программирования c++, научиться проектировать иерархию классов.

Задача.

Необходимо спроектировать 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 методы для перемещения в указанные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, установки и получения цвета, а также оператор вывода в поток.

Необходимо также обеспечить однозначную идентификацию каждого объекта.

Решение должно содержать:

-условие задания;

-UML диаграмму разработанных классов;

-текстовое обоснование проектных решений;

-реализацию классов на языке C++.

Условие задания.

Вариант 5: прямоугольник, эллипс, сектор эллипса.

Обоснование проектных решений.

1) Для удобства работы с координатами был реализован класс Point, который содержит координаты *X* и *Y*, а также перегруженные операции сложения, вычитания и присваивания.

2) Для реализации цвета был разработан класс Color, который содержит информацию о цвете в формате rgb и содержит перегруженную операцию вывода этой информации.

3) Для реализации фигур был разработан абстрактный класс Shape, который содержит необходимые для всех фигур поля – centre (центр фигуры), angle (количество углов), color (цвет фигуры), общие методы получения и задания цвета, статическое поле для однозначной идентификации объекта, а также виртуальные методы для масштабирования, перемещения, поворота на заданный угол и вывода информации, которые в каждом дочернем классе реализованы по-своему.

4) Для реализации прямоугольника был реализован дочерний класс от Shape, который содержит поля LowerLeftCorner и UpperRightCorner, которые содержат координаты левого нижнего угла и верхнего правого углов прямоугольника – минимальная информация, которая необходима для однозначного определения прямоугольника. В классе также реализованы методы для перемещения, масштабирования, поворота и вывода информации о прямоугольнике.

5) Для реализации эллипса был реализован дочерний класс от Shape, который содержит поля LeftFocus, RightFocus и length, которые содержат координаты левого фокуса, правого фокуса и сумму расстояний от двух фокусов до точки на эллипсе. В классе также реализованы методы для перемещения, масштабирования, поворота и вывода информации об эллипсе.

6) Для реализации сектора эллипса был реализован дочерний класс от Ellipse, который дополнительно содержит 2 поля – AngleWithX (угол θ_1 на рисунке 1), AngleSector (угол θ_2 на рисунке 1), которые содержат значения углов, отсчитываемого от оси X до начала сектора и до конца сектора соответственно. Также был реализован метод для вывода информации.

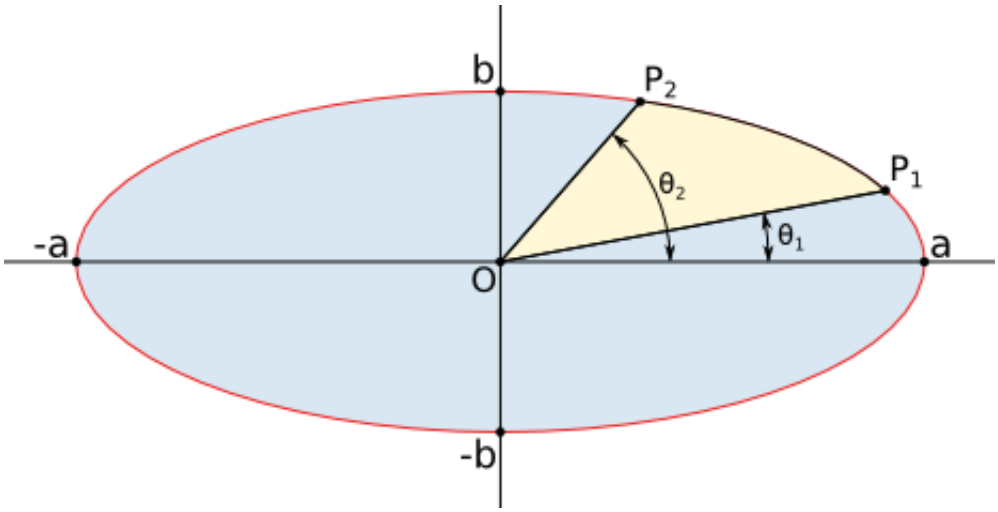


Рисунок 1

UML-диаграмма.

На рисунке 2 представлена UML-диаграмма по написанной программе. Линия с белой стрелкой на конце – стандартное обозначение обобщения. Показывает, какие классы от каких наследуются. Линия с чёрным ромбом – стандартное обозначение композиции. Показывает, какой класс является частью другого. Здесь существует строгая зависимость по времени жизни объектов, если экземпляр, который содержит другой класс будет уничтожен, то и зависимый также будет уничтожен.

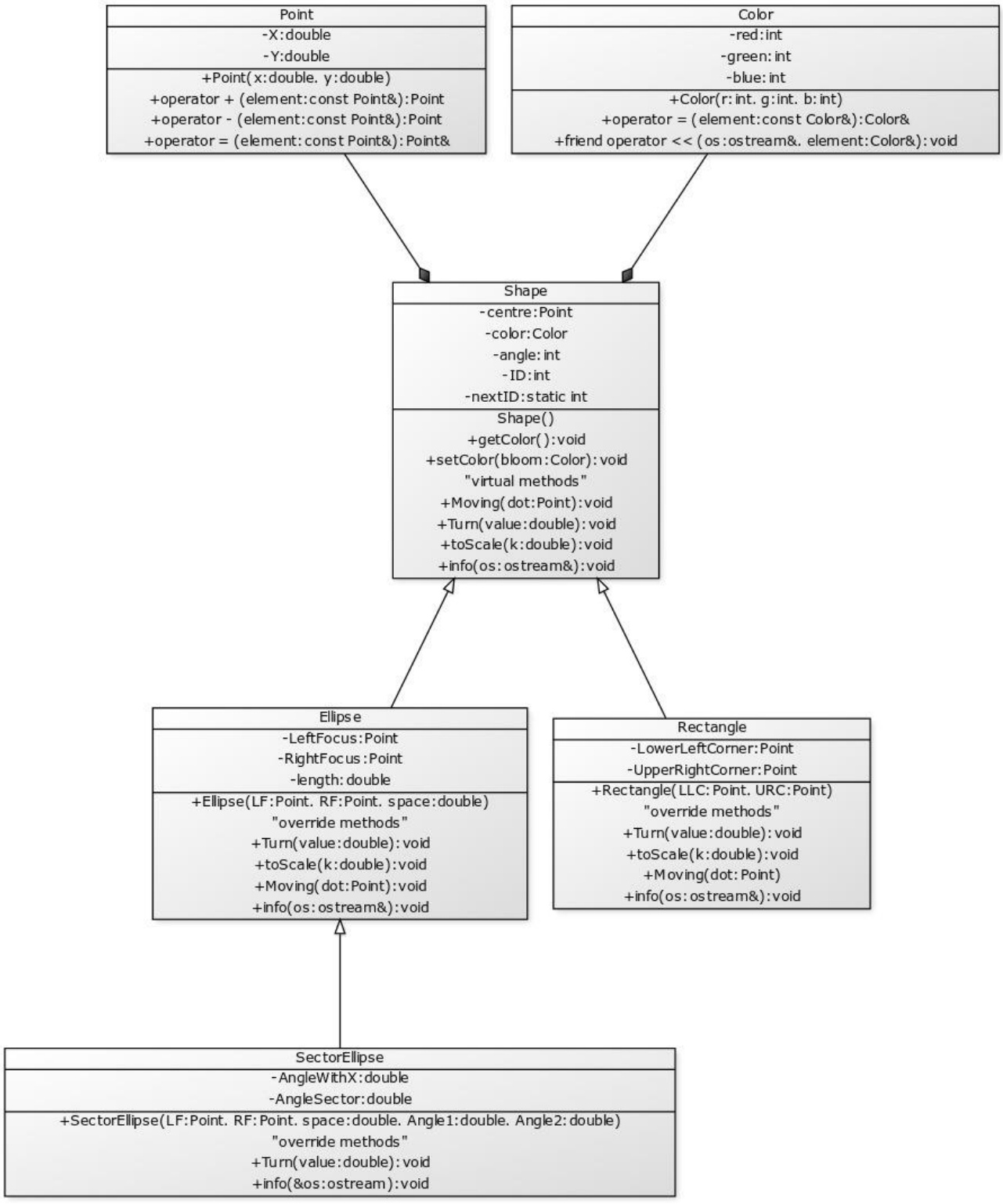
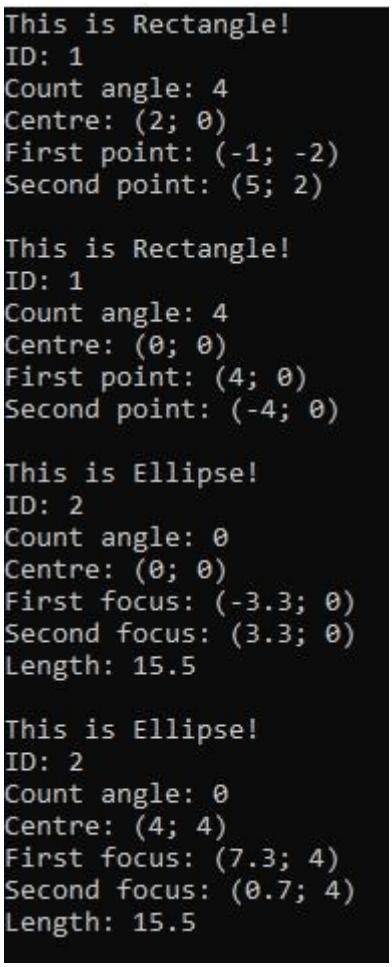
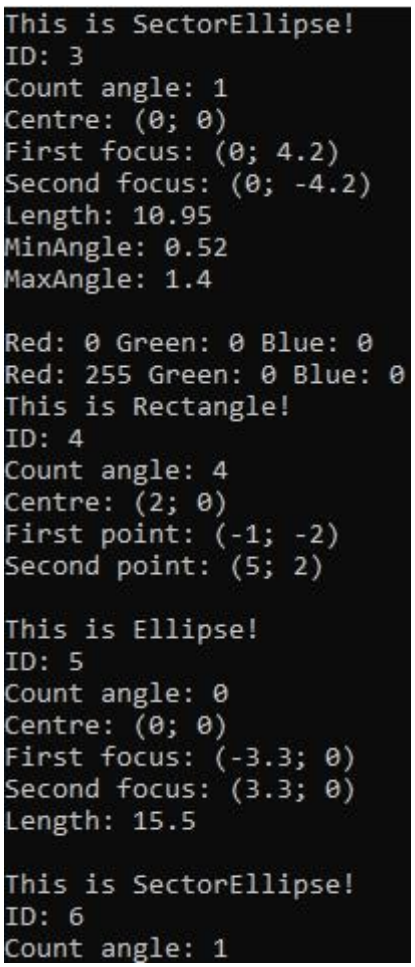
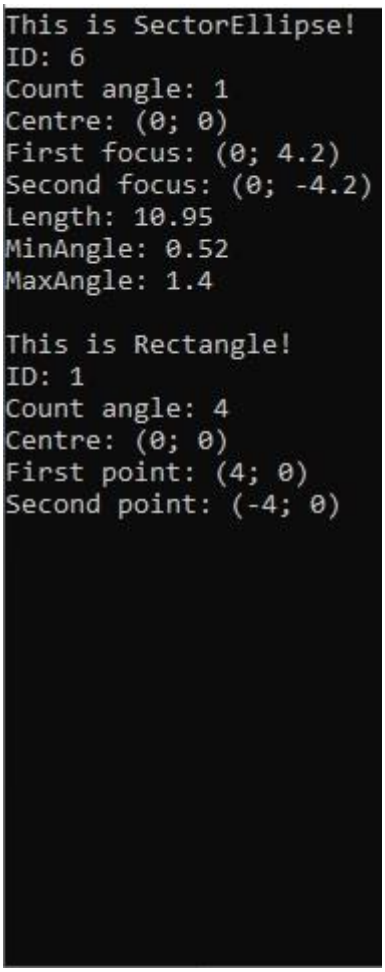

Point
-X:double
-Y:double
+Point(x:double. y:double)
+operator + (element:const Point&):Point
+operator - (element:const Point&):Point
+operator = (element:const Point&):Point&
Color
-red:int
-green:int
-blue:int
+Color(r:int. g:int. b:int)
+operator = (element:const Color&):Color&
+friend operator << (os:ostream&. element:Color&):void
Shape
-centre:Point
-color:Color
-angle:int
-ID:int
-nextID:static int
Shape()
+getColor():void
+setColor(bloom:Color):void
"virtual methods"
+Moving(dot:Point):void
+Turn(value:double):void
+toScale(k:double):void
+info(os:ostream&):void
Ellipse
-LeftFocus:Point
-RightFocus:Point
-length:double
+Ellipse(LF:Point. RF:Point. space:double)
"override methods"
+Turn(value:double):void
+toScale(k:double):void
+Moving(dot:Point):void
+info(os:ostream&):void
Rectangle
-LowerLeftCorner:Point
-UpperRightCorner:Point
+Rectangle(LLC:Point. URC:Point)
"override methods"
+Turn(value:double):void
+toScale(k:double):void
+Moving(dot:Point)
+info(os:ostream&):void
SectorEllipse
-AngleWithX:double
-AngleSector:double
+SectorEllipse(LF:Point. RF:Point. space:double. Angle1:double. Angle2:double)
"override methods"
+Turn(value:double):void
+info(&os:ostream):void

Рисунок 2

Результат работы.







Выводы.

В ходе выполнения данной лабораторной работы был изучен механизм наследования в языке программирования c++, была спроектирована, реализована и обоснована система классов геометрических фигур – прямоугольника, эллипса и сектора эллипса.

Приложение.

Исходный код.

Файл main.cpp.

```
#include "rectangle.h"
#include "ellipse.h"
#include "sectorellipse.h"

int main()
{
    Rectangle test1({-1.0, -2.0}, {5.0, 2.0});
    cout << test1;
    test1.Turn(M_PI/2);
    test1.toScale(2);
    test1.Moving({0.0, 0.0});
    cout << test1;

    Ellipse test2({-3.3, 0.0}, {3.3, 0.0}, 15.5);
    cout << test2;
    test2.Turn(M_PI/2);
    test2.Turn(M_PI/2);
    test2.toScale(3.0);
    test2.toScale(1.0/3.0);
    test2.Moving({4.0, 4.0});
    cout << test2;

    SectorEllipse test3({0.0, 4.2}, {0.0, -4.2}, 10.95, 0.52, 1.4);
    cout << test3;
    test3.getColor();
    test3.setColor({255, 0, 0});
    test3.getColor();


    Rectangle a({-1.0, -2.0}, {5.0, 2.0});
    cout << a;
    Ellipse b({-3.3, 0.0}, {3.3, 0.0}, 15.5);
    cout << b;
    SectorEllipse c({0.0, 4.2}, {0.0, -4.2}, 10.95, 0.52, 1.4);
    cout << c;

    cout << test1;

    return 0;
}
```

Файл color.h.

```
#ifndef COLOR_H
#define COLOR_H

#include <iostream>

using namespace std;

class Color
{
private:
    int red;
```

```
    int green;
    int blue;

public:
    Color(int r, int g, int b) : red(r), green(g), blue(b){}
    ~Color(){}

    Color& operator = (const Color& element);
    friend void operator << (ostream& os, const Color& element);
};

#endif // COLOR_H
```

Файл color.cpp.

```
#include "color.h"

Color& Color::operator = (const Color& element)
{
    red = element.red;
    green = element.green;
    blue = element.blue;
    return *this;
}

void operator << (ostream& os, const Color& element)
{
    os << "Red: " << element.red << " Green: " << element.green << " Blue: " <<
element.blue << std::endl;
}
```

Файл point.h.

```
#ifndef POINT_H
#define POINT_H

class Point
{
public:
    double X;
    double Y;

    Point() : X(0), Y(0){}
    Point(double x, double y) : X(x), Y(y){}
    ~Point(){}

    Point operator + (const Point&);
    Point operator - (const Point&);
    Point& operator = (const Point&);
};

#endif // POINT_H
```

Файл point.cpp.

```
#include "point.h"

Point Point::operator + (const Point& element)
{
```

```
    Point result;
    result.X = X + element.X;
    result.Y = Y + element.Y;
    return result;
}

Point Point::operator - (const Point& element)
{
    Point result;
    result.X = X - element.X;
    result.Y = Y - element.Y;
    return result;
}

Point& Point::operator = (const Point& element)
{
    X = element.X;
    Y = element.Y;
    return *this;
}
```

Файл shape.h.

```
#ifndef SHAPE_H
#define SHAPE_H

#include <cmath>
#include "point.h"
#include "color.h"

class Shape
{
protected:
    Point centre;
    Color color;
    int angle;
    int ID;
    static int nextID;
    friend ostream& operator << (ostream& os, Shape& element)
    {
        element.info(os);
        return os;
    }

public:
    Shape() : color(0, 0, 0){
        ID = ++nextID;
    }
    ~Shape(){}

    virtual void Moving(Point dot) = 0;     //перемещение в указанные координаты
    virtual void Turn(double value) = 0;    //поворот на указанный угол против
часовой стрелки
    virtual void toScale(double k) = 0;     //масштабирование на заданный
коэффициент
    virtual void info(ostream& os) = 0;     //вывод информации

    void getColor()
    {
        cout << color;
    }
```

```
    void setColor(Color bloom)
    {
        color = bloom;
    }
};

#endif // SHAPE_H
```

Файл shape.cpp.

```
#include "shape.h"

int Shape::nextID = 0;
```

Файл rectangle.h.

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "shape.h"

class Rectangle : public Shape
{
protected:
    Point LowerLeftCorner;
    Point UpperRightCorner;

public:
    Rectangle(Point LLC, Point URC);
    ~Rectangle(){}

    void Turn(double value);
    void toScale(double k);
    void Moving(Point dot);
    void info(ostream& os);
};

#endif // RECTANGLE_H
```

Файл rectangle.cpp.

```
#include "rectangle.h"

Rectangle::Rectangle(Point LLC, Point URC) : LowerLeftCorner(LLC),
UpperRightCorner(URC)
{
    centre.X = (LowerLeftCorner.X + UpperRightCorner.X) / 2;
    centre.Y = (LowerLeftCorner.Y + UpperRightCorner.Y) / 2;
    angle = 4;
}

void Rectangle::Turn(double value)
{
    double TempX = LowerLeftCorner.X, TempY = LowerLeftCorner.Y;
    LowerLeftCorner.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
    LowerLeftCorner.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);
```

```
    TempX = UpperRightCorner.X, TempY = UpperRightCorner.Y;
    UpperRightCorner.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
    UpperRightCorner.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);
}

void Rectangle::toScale(double k)
{
    double side1 = UpperRightCorner.X - LowerLeftCorner.X;
    side1 = fabs(side1);
    double side2 = UpperRightCorner.Y - LowerLeftCorner.Y;
    side2 = fabs(side2);

    if(k > 1)
    {
        if(UpperRightCorner.X > LowerLeftCorner.X)
        {
            UpperRightCorner.X += side1/2 * (k-1);
            LowerLeftCorner.X -= side1/2 * (k-1);
            UpperRightCorner.Y += side2/2 * (k-1);
            LowerLeftCorner.Y -= side2/2 * (k-1);
        }
        else
        {
            UpperRightCorner.X -= side1/2 * (k-1);
            LowerLeftCorner.X += side1/2 * (k-1);
            UpperRightCorner.Y -= side2/2 * (k-1);
            LowerLeftCorner.Y += side2/2 * (k-1);
        }
    }
    else
    {
        double delta1 = (side1 - side1*k) / 2;
        double delta2 = (side2 - side2*k) / 2;
        if(UpperRightCorner.X > LowerLeftCorner.X)
        {
            UpperRightCorner.X -= delta1;
            LowerLeftCorner.X += delta1;
            UpperRightCorner.Y -= delta2;
            LowerLeftCorner.Y += delta2;
        }
        else
        {
            UpperRightCorner.X += delta1;
            LowerLeftCorner.X -= delta1;
            UpperRightCorner.Y += delta2;
            LowerLeftCorner.Y -= delta2;
        }
    }
}

void Rectangle::Moving(Point dot)
{
    Point DeltaUp = UpperRightCorner - centre;
    Point DeltaLow = LowerLeftCorner - centre;

    centre = dot;

    UpperRightCorner = centre + DeltaUp;
    LowerLeftCorner = centre + DeltaLow;
}
```

```
void Rectangle::info(ostream& os)
{
    os << "This is Rectangle!" << endl;
    os << "ID: " << ID << endl;
    os << "Count angle: " << angle << endl;
    os << "Centre: (" << centre.X << "; " << centre.Y << ")" << endl;
    os << "First point: (" << LowerLeftCorner.X << "; "  << LowerLeftCorner.Y <<
")" << endl;
    os << "Second point: (" << UpperRightCorner.X << "; "  << UpperRightCorner.Y
<< ")" << endl;
    os << endl;
}
```

Файл ellipse.h.

```
#ifndef ELLIPSE_H
#define ELLIPSE_H

#include "shape.h"

class Ellipse : public Shape
{
protected:
    Point LeftFocus;
    Point RightFocus;
    double length;  // |MF1| + |MF2| = const = 2a = length!

public:
    Ellipse(Point LF, Point RF, double space);
    ~Ellipse(){}

    void Turn(double value);
    void toScale(double k);
    void Moving(Point dot);
    void info(ostream& os);
};

#endif // ELLIPSE_H
```

Файл ellipse.cpp.

```
#include "ellipse.h"

Ellipse::Ellipse(Point LF, Point RF, double space) : LeftFocus(LF),
RightFocus(RF), length(space)
{
    centre.X = (LF.X + RF.X) / 2;
    centre.Y = (LF.Y + RF.Y) / 2;
    angle = 0;
}

void Ellipse::Turn(double value)
{
    double TempX = LeftFocus.X, TempY = LeftFocus.Y;
    LeftFocus.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
    LeftFocus.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);

    TempX = RightFocus.X, TempY = RightFocus.Y;
```

```
    RightFocus.X = centre.X + (TempX - centre.X)*cos(value) - (TempY -
centre.Y)*sin(value);
    RightFocus.Y = centre.Y + (TempY - centre.Y)*cos(value) + (TempX -
centre.X)*sin(value);
}

void Ellipse::toScale(double k)
{
    if(k > 1)
    {
        if(RightFocus.X > LeftFocus.X)
        {
            RightFocus.X += length/2 * (k-1);
            LeftFocus.X -= length/2 * (k-1);
            length *= k;
        }
        else
        {
            RightFocus.X -= length/2 * (k-1);
            LeftFocus.X += length/2 * (k-1);
            length *= k;
        }
    }
    else
    {
        double delta = (length - length*k)/2;
        if(RightFocus.X > LeftFocus.X)
        {
            RightFocus.X -= delta;
            LeftFocus.X += delta;
            length *= k;
        }
        else
        {
            RightFocus.X += delta;
            LeftFocus.X -= delta;
            length *= k;
        }
    }
}

void Ellipse::Moving(Point dot)
{
    Point Delta1 = RightFocus - centre;
    Point Delta2 = LeftFocus - centre;

    centre = dot;

    RightFocus = centre + Delta1;
    LeftFocus = centre + Delta2;
}

void Ellipse::info(ostream& os)
{
    os << "This is Ellipse!" << endl;
    os << "ID: " << ID << endl;
    os << "Count angle: " << angle << endl;
    os << "Centre: (" << centre.X << "; " << centre.Y << ")" << endl;
    os << "First focus: (" << LeftFocus.X << "; "  << LeftFocus.Y << ")" <<
endl;
    os << "Second focus: (" << RightFocus.X << "; "  << RightFocus.Y << ")" <<
endl;
    os << "Length: " << length << endl;
```

```
    os << endl;
}
```

Файл sectorellipse.h.

```
#ifndef SECTORELLIPSE_H
#define SECTORELLIPSE_H

#include "ellipse.h"

class SectorEllipse : public Ellipse
{
protected:
    double AngleWithX;
    double AngleSector;

public:
    SectorEllipse(Point LF, Point RF, double space, double Angle1, double
Angle2);
    ~SectorEllipse(){}

    void Turn(double value);
    void info(ostream &os);
};

#endif // SECTORELLIPSE_H
```

Файл sectorellipse.cpp.

```
#include "sectorellipse.h"

SectorEllipse::SectorEllipse(Point LF, Point RF, double space, double Angle1,
double Angle2) : Ellipse(LF, RF, space)
{
    AngleWithX = Angle1;
    AngleSector = Angle2;
    angle = 1;
}

void SectorEllipse::Turn(double value)
{
    Ellipse::Turn(value);
    AngleWithX += value;
    AngleSector += value;
}

void SectorEllipse::info(ostream &os)
{
    os << "This is SectorEllipse!" << endl;
    os << "ID: " << ID << endl;
    os << "Count angle: " << angle << endl;
    os << "Centre: (" << centre.X << "; " << centre.Y << ")" << endl;
    os << "First focus: (" << LeftFocus.X << "; "  << LeftFocus.Y << ")" <<
endl;
    os << "Second focus: (" << RightFocus.X << "; "  << RightFocus.Y << ")" <<
endl;
    os << "Length: " << length << endl;
    os << "MinAngle: " << AngleWithX << endl;
    os << "MaxAngle: " << AngleSector << endl;
    os << endl;
}
```