**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ**

**ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ**

**Индивидуальное задание**

**«Геометрические фигуры»**

**Вариант 5,10**

**по предмету «Алгоритмизация и программирование»**

**Выполнил: студент гр. 5130904/30002 Севостьянова А.В.**

**Руководитель: Череповский Д.К.**

**Санкт-Петербург**

**2023 г.**

Оглавление

[1. Общая постановка задачи 3](#_Toc154505889)

[A1: Геометрические фигуры 3](#_Toc154505890)

[A2: Масштабирование фигур 3](#_Toc154505891)

[A3: Составные фигуры 4](#_Toc154505892)

[2. Детальные требования и тест план 4](#_Toc154505893)

[Детальные требования: 4](#_Toc154505894)

[Таблица с детальными требованиями и тест планом 4](#_Toc154505895)

[3. Программа 7](#_Toc154505896)

[Main.cpp 7](#_Toc154505897)

[Classes.h 7](#_Toc154505898)

[Classe.cpp 10](#_Toc154505899)

[CompositeShape.h 15](#_Toc154505900)

[CompositeShape.cpp 15](#_Toc154505901)

[Приложение A 17](#_Toc154505902)

[Приложение B 18](#_Toc154505903)

[Вывод 21](#_Toc154505904)

1. **Общая постановка задачи**

## *A1: Геометрические фигуры*

1. Создать класс Point, представляющий точку на плоскости с координатами x\_ и y\_

2. Создать класс FrameRectangle, представляющий прямоугольник шириной width\_ и высотой height\_ с центром в точке pos\_. Стороны прямоугольника всегда параллельны осям координат.

3. Создать абстрактный класс Shape, включающий следующие методы:

* getArea – вычисление площади
* getFrameRectangle – получение ограничивающего прямоугольника для фигуры (объект класса FrameRectangle)
* move – перемещение центра фигуры, в конкретную точку и в виде смещений по осям абсцисс и ординат
* scale - изотропное масштабирование фигуры относительно её центра с указанным коэффициентом (только в случае реализации А2 и А3)
* clone - создание копии фигуры в динамической памяти и получение указателя на эту копию
* getName – получение названия фигуры (RECTANGLE, CIRCLE и т.д.), метод использовать при выводе

4. Реализовать два класса, SQUARE и CONCAVE, как классы, производные от класса Shape.

5. Реализовать функцию вывода информации о фигуре. Вывод должен содержать в одну строчку: название фигуры, её площадь, координаты левого нижнего и правого верхнего углов ограничивающего прямоугольника. Значения округлять до одного знака после запятой и разделять ровно одним пробелом

6. Написать тесты, демонстрирующие работу созданных классов. Демонстрация должна включать полиморфное применение классов.

## *A2: Масштабирование фигур*

1. Добавить в классы метод scale, осуществляющий изотропное масштабирование фигур относительно их центра с указанным коэффициентом.

2. Написать тесты, проверяющие:

* неизменность ширины и высоты, а также площади фигуры при перемещениях;
* квадратичное изменение площади фигуры при масштабировании;
* наличие и обработку некорректных параметров в методах и конструкторах.

## *A3: Составные фигуры*

1. Создать класс CompositeShape, описывающий составную фигуру, состоящую из простых фигур. Необходимо самостоятельно реализовать хранение множества фигур на базе динамического массива. Максимальное количество простых фигур должно задаваться в конструкторе CompositeShape.

2. Реализовать масштабирование и перемещение для CompositeShape. Масштабирование и перемещение выполнить относительно центра составной фигуры, за который принимается центр ограничивающего прямоугольника. В реализации можно использовать только методы move и scale без расширения интерфейса Shape.

3.Написать набор тестов, проверяющий корректную работу созданного класса. Для тестирования можно реализовать дополнительные методы.

1. **Детальные требования и тест план**

## *Детальные требования:*

1. Координаты всех точек должны принимать дробное значений.

2. Длина стороны квадрата (положительное число), максимальное количество элементов в составной фигуре, коэффициент изотропного масштабирования должны быть целыми положительными числами.

3. Координаты невырожденного треугольника должны быть заданы корректно, т. е. три первые точки (A,B,C) должны образовывать треугольник, 4 точка (D) должна находится внутри этого треугольника.

4. Количество элементов в составной фигуре не должно превышать максимального значения.

5. Выбор продолжения/окончания добавления элементов в составную фигуру должен быть задан корректно (‘Y’ или ‘N’)

6. Выбор добавляемой фигуры должен быть задан корректно (‘1’ или ‘2’)

## *Таблица с детальными требованиями и тест планом*

|  |  |  |  |
| --- | --- | --- | --- |
| Требования | Детальные требования | Данные | Ожидаемый результат |
| 1. Координаты точек – дробные числа. | Если координаты точек не являются числом или принимают не дробное значение, то сообщение: «Некорректный ввод», аварийное завершение программы. | K 4.0 | Сообщение: «Некорректный ввод»  Аварийное завершение программы |
| 2. Координаты точек заданы корректно. | Если координаты точек принимают дробное значение, то продолжение работы программы. | 2.5 0 | Продолжение работы программы |
| 3. Длина стороны квадрата (положительное число), максимальное количество элементов в составной фигуре, коэффициент изотропного масштабирования – целые положительные числа. | Если переменные принимают нечисловое значение, дробное значение, отрицательное значение или равны нулю, то сообщение: «Некорректный ввод», аварийное завершение работы программы. | К  5.5  -7  0 | Сообщение: «Некорректный ввод»  Аварийное завершение программы |
| 4. Длина стороны квадрата, максимальное количество элементов в составной фигуре, коэффициент изотропного масштабирования заданы корректно. | Если переменные принимают целое, положительное значение, то продолжение работы программы | 2 | Продолжение работы программы |
| 5. Координаты невырожденного треугольника заданы некорректно | Если три первые точки не образуют треугольник или четвертая точка не лежит заданного треугольника, сообщение: «Фигура задана некорректно», аварийное завершение работы программы | 0 0, 0 0, 3 2, 1 1  0 0, 3 0, 0 4, 99 1 | Сообщение: «Фигура задана некорректной»  Аварийное завершение программы |
| 6.Координаты невырожденного треугольника заданы некорректно | Если координаты треугольника заданы корректно, продолжение работы программы | 0 0, 3 0, 0 4, 1 1 | Продолжение работы программы |
| 7.Количество элементов составной фигуры превышает максимально допустимое. | Количество элементов составной фигуры превышает максимально допустимое, то сообщение: «Достигнут максимальный размер массива. Элемент не может быть добавлен», аварийное завершение программы. | 7 (>4) | Сообщение: «Достигнут максимальный размер массива. Элемент не может быть добавлен»  Аварийное завершение программы |
| 8. Количество элементов составной фигуры не превышает максимально допустимое. | Количество элементов составной фигуры не превышает максимально допустимое, продолжение работы программы. | 3 (<4) | Продолжение работы программы |
| 9.Выбор продолжения/окончания добавления элементов в составную фигуру должен быть ‘Y’ или ‘N’ | Если выбор не принимает значения ‘Y’ или ‘N’, сообщение: «Некорректный ввод», аварийное завершение работы программы. | R | Сообщение: «Некорректный ввод»  Аварийное завершение программы |
| 10. Выбор продолжения/окончания добавления элементов в составную фигуру должен быть ‘Y’ или ‘N’ | Если выбор корректный, Y- добавление нового элемента  N – завершение добавления новых элементов | Y / N | Y - добавление нового элемента  N – завершение добавления новых элементов |
| 11. Выбор добавляемой фигуры должен принимать значение ‘1’ или ‘2’ | Если выбор добавляемой фигуры не принимает значение ‘1’ или ‘2’, сообщение: «Некорректный ввод», аварийное завершение работы программы. | 3 | Сообщение: «Некорректный ввод»  Аварийное завершение программы |
| 12. Выбор добавляемой фигуры должен принимать значение ‘1’ или ‘2’ | Если выбор добавляемой фигуры принимает значение ‘1’ или ‘2’,то  1 – добавление Square  2 – добавление Concave | 1/2 | 1 – добавление Square  2 – добавление Concave |

# 3. Программа

## *Main.cpp*

#include"Classes.h"

#include"CompositeShape.h"

int main()

{

Shape\* clone =nullptr;

try

{

setlocale(LC\_ALL, "ru");

cout << "Тестирование класса Square\n\n";

Shape\* figure;

Square square;

getInfo(square);

figure = &square;

cout << "Объект класса Shape:\n";

getInfo(\*figure);

cout << "Координаты нового цетра\n";

Point point;

cin >> point;

square.move(point);

getInfo(square);

cout << "Ширина ограничивающего прямоугольника: " << square.getFrameRectangle().getWidth()

<< "\nВысота ограничивающего прямоугольника: " << square.getFrameRectangle().getHeight();

cout << "\nОбъект класса Shape:\n";

getInfo(\*figure);

cout << "Ширина ограничивающего прямоугольника: " << figure->getFrameRectangle().getWidth()

<< "\nВысота ограничивающего прямоугольника: " << figure->getFrameRectangle().getHeight();

std::string k;

cout << "\nКоэффициент изотропного масштабировани (>0) = ";

cin >> k;

square.scale(inputToDouble(k));

getInfo(square);

cout << "Объект класса Shape:\n";

getInfo(\*figure);

clone = square.clone();

cout << "Создание копии объекта Square\n";

getInfo(\*clone);

delete clone;

clone = nullptr;

cout << "------------------------------------------------------\n\n";

cout << "Тестирование класса Concave\n\n";

Concave concave;

getInfo(concave);

figure = &concave;

cout << "Объект класса Shape:\n";

getInfo(\*figure);

cout << "Координаты нового центра\n";

cin >> point;

concave.move(point);

getInfo(concave);

cout << "Ширина ограничивающего прямоугольника: " << concave.getFrameRectangle().getWidth()

<< "\nВысота ограничивающего прямоугольника: " << concave.getFrameRectangle().getHeight();

cout << "\nОбъект класса Shape:\n";

getInfo(\*figure);

cout << "Ширина ограничивающего прямоугольника: " << figure->getFrameRectangle().getWidth()

<< "\nВысота ограничивающего прямоугольника: " << figure->getFrameRectangle().getHeight();

cout << "\nКоэффициент изотропного масштабировани (>0) = ";

cin >> k;

concave.scale(inputToDouble(k));

getInfo(concave);

cout << "Объект класса Shape:\n";

getInfo(\*figure);

clone = concave.clone();

cout << "Создание копии объекта Square\n";

getInfo(\*clone);

delete clone;

clone = nullptr;

cout << "------------------------------------------------------\n\n";

cout << "Тестирование класса CompositeShape\n";

CompositeShape figures;

cout << "Добавление элементов\nY - продолжение ввода\nN - завершение ввода\n";

char flag = 'Y';

while (flag != 'N' && flag == 'Y')

{

char choice;

cout << "Какую фигуру добавим?\n1-Square\n2-Concave\nВыбор: ";

cin >> choice;

switch (choice)

{

case '1': figures.add(\*new Square());

break;

case '2': figures.add(\*new Concave());

break;

default: throw std::invalid\_argument(ERROR\_OF\_INPUT);

}

cout << "Продолжим добавлять фигуры? ";

cin >> flag;

if (flag != 'Y' && flag != 'N') throw std::runtime\_error(ERROR\_OF\_INPUT);

}

getInfo(figures);

cout << "Координаты нового центра\n";

cin >> point;

figures.move(point);

getInfo(figures);

cout << "\nКоэффициент изотропного масштабировани (>0) = ";

cin >> k;

figures.scale(inputToDouble(k));

getInfo(figures);

}

catch (std::invalid\_argument& ex)

{

std::cerr << ex.what() << std::endl;

if (clone != nullptr) delete clone;

return EXIT\_FAILURE;

}

catch (std::runtime\_error& ex)

{

std::cerr << ex.what() << std::endl;

if (clone != nullptr) delete clone;

return EXIT\_FAILURE;

}

return 0;

}

## *Classes.h*

#pragma once

#include<iostream>

#include <iomanip>

#include<regex>

using std::cin;

using std::cout;

const std::string ERROR\_OF\_INPUT = "Некорректный ввод\n";

const std::string ERROR\_OF\_FIGURE = "Фигура задана некорректно\n";

class Point

{

friend class CompositeShape;

friend class Concave;

friend class Square;

friend class FrameRectangle;

friend std::ostream& operator << (std::ostream& stream, const Point& point);

friend std::istream& operator >> (std::istream& stream, Point& point);

public:

Point();

Point(double x, double y);

Point(const Point& point);

~Point();

double getDistance(const Point& point)const;

Point& operator = (const Point \*point);

Point& operator + (const double value)const;

Point& operator + (const Point &point);

Point& operator - (const double value)const;

Point& operator - (const Point& point);

Point& operator \* (const double value);

private:

double x\_, y\_;

};

class FrameRectangle

{

public:

FrameRectangle(double width, double height, Point pos);

~FrameRectangle();

Point getLeftDown() const;

Point getRightTop() const;

Point getMiddle() const;

double getHeight() const;

double getWidth() const;

private:

double width\_, height\_;

Point pos\_;

};

class Shape

{

public:

virtual double getArea() const = 0;

virtual FrameRectangle getFrameRectangle()const = 0;

virtual void move(const double x, const double y) = 0;

virtual void move(const Point &point) = 0;

virtual Shape\* clone() = 0;

virtual std::string getName()const = 0;

virtual Point getMiddle()const = 0;

virtual void scale(double k) = 0;

};

class Square : public Shape

{

public:

Square();

Square(const Square &sq);

~Square();

double getArea()const override;

FrameRectangle getFrameRectangle()const override;

void move(const double x, const double y) override;

void move(const Point& point) override;

Shape\* clone() override;

std::string getName() const override;

Point getMiddle() const override;

Square& operator = (const Square &square);

void scale(double k) override;

private:

Point A\_, middle\_;

double length\_;

};

class Concave : public Shape

{

public:

Concave();

Concave(const Concave& cn);

~Concave();

double getArea()const override;

FrameRectangle getFrameRectangle()const override;

void move(const Point& point) override;

void move(const double x, const double y);

Shape\* clone() override;

Concave& operator = (const Concave& triangle);

std::string getName()const override;

Point getMiddle() const override;

void scale(double k) override;

private:

Point A\_, B\_, C\_, D\_;

};

double inputToDouble(std::string& value);

bool isTriangle(const Point& A, const Point& B, const Point& C);

float getSquare(const Point& A, const Point& B, const Point& C);

void getInfo(const Shape& figure);

## *Classe.cpp*

#include "Classes.h"

double inputToDouble(std::string& value)

{

std::regex regular("-?\\d+,?\\d\*");

std::regex replace("\\.");

value = std::regex\_replace(value, replace, ",");

if (std::regex\_match(value, regular)) return std::stod(value);

else throw std::invalid\_argument(ERROR\_OF\_INPUT);

}

// класс Point

Point::Point() { x\_ = 0; y\_ = 0; }

Point::Point(double x, double y) { x\_ = x; y\_ =y; }

Point::Point(const Point& point) { x\_ = point.x\_; y\_ = point.y\_; }

Point::~Point() {}

double Point::getDistance(const Point& point)const

{

return sqrt(pow(x\_-point.x\_,2) + pow(y\_-point.y\_,2));

}

Point &Point::operator=(const Point \*point)

{

x\_ = point->x\_;

y\_ = point->y\_;

return \*this;

}

Point &Point::operator+(const double value) const

{

Point point;

point.x\_ = x\_ + value;

point.y\_ = y\_ + value;

return point;

}

Point& Point::operator+(const Point &point)

{

x\_ += point.x\_;

y\_ += point.y\_;

return \*this;

}

Point& Point::operator-(const double value) const

{

Point point;

point.x\_ = x\_ - value;

point.y\_=y\_ - value;

return point;

}

Point& Point::operator-(const Point& point)

{

x\_ -= point.x\_;

y\_ -= point.y\_;

return \*this;

}

Point& Point:: operator \* (const double value)

{

x\_ \*= value;

y\_ \*= value;

return \*this;

}

std::ostream& operator<<(std::ostream& stream, const Point& point)

{

stream << point.x\_ << " " << point.y\_ << " ";

return stream;

}

std::istream& operator>>(std::istream& stream, Point& point)

{

std::string value;

cout << "Введите координаты точек\nx = ";

stream >> value;

point.x\_ = inputToDouble(value);

cout << "y = ";

stream >> value;

point.y\_ = inputToDouble(value);

return stream;

}

//класс FrameRectangle

FrameRectangle::~FrameRectangle() {}

FrameRectangle::FrameRectangle(double width, double height, Point pos)

{

width\_ = width;

height\_ = height;

pos\_ = pos;

}

Point FrameRectangle::getLeftDown() const

{

Point leftDown (pos\_.x\_ - width\_ /2, pos\_.y\_ - height\_/2);

return leftDown;

}

Point FrameRectangle::getRightTop() const

{

Point rightTop(pos\_.x\_ + width\_ / 2, pos\_.y\_ + height\_ / 2);

return rightTop;

}

Point FrameRectangle::getMiddle() const{ return pos\_;}

double FrameRectangle::getHeight() const {return height\_;}

double FrameRectangle::getWidth() const { return width\_;}

// класс Square

Square::Square()

{

std::string value;

cout << "Сторона квадрата = ";

cin >> value;

if (inputToDouble(value) > 0) length\_ = inputToDouble(value);

else throw std::invalid\_argument(ERROR\_OF\_INPUT);

cin >> A\_;

middle\_ = A\_ + 0.5 \* length\_;

}

Square::Square(const Square& sq)

{

length\_ = sq.length\_;

A\_ = sq.A\_;

middle\_ = sq.middle\_;

}

Square::~Square() {};

Square& Square::operator=(const Square& square)

{

A\_ = square.A\_;

length\_ = square.length\_;

middle\_ = square.middle\_;

return \*this;

}

void Square::scale(double k)

{

if (k <= 0) throw std::invalid\_argument(ERROR\_OF\_INPUT);

else

{

length\_ = length\_ \* k;

A\_ = middle\_ - length\_ \* k/2;

}

}

double Square::getArea() const{ return length\_\*length\_;}

//ограничивающий прямогульник квадрата совпадает с самим квадратом

FrameRectangle Square::getFrameRectangle()const

{

FrameRectangle rectangle(length\_,length\_, middle\_);

return rectangle;

}

void Square::move(const double x, const double y)

{

middle\_.x\_ = middle\_.x\_ + x;

middle\_.y\_ = middle\_.y\_ + y;

A\_.x\_ = A\_.x\_ +x;

A\_.y\_ = A\_.y\_ + y;

}

void Square::move( const Point &point)

{

middle\_ = point;

A\_ = middle\_ - length\_/2;

}

Shape\* Square::clone() { return new Square(\*this);}

std::string Square::getName() const{return "SQUARE";}

Point Square::getMiddle() const { return middle\_;}

bool isTriangle(const Point& A, const Point& B, const Point& C)

{

double AB = A.getDistance(B);

double BC = B.getDistance(C);

double AC = A.getDistance(C);

return (AB + BC > AC) && (BC + AC > AB) && (AC + AB > BC);

}

float getSquare(const Point& A, const Point& B, const Point& C)

{

if (isTriangle(A, B, C))

{

double AB = A.getDistance(B);

double BC = B.getDistance(C);

double AC = A.getDistance(C);

double p = 0.5 \* (AB + BC + AC);

return sqrt(p \* (p - AB) \* (p - BC) \* (p - AC));

}

else throw std::runtime\_error(ERROR\_OF\_FIGURE);

}

// класс Concave

Concave::Concave()

{

cin >> A\_ >> B\_ >> C\_ >> D\_;

if (round(getSquare(A\_, B\_, C\_)\*10)/10 != round((getSquare(A\_, B\_, D\_) + getSquare(B\_, C\_, D\_) + getSquare(A\_, C\_, D\_))\*10)/10)

throw std::runtime\_error(ERROR\_OF\_FIGURE);

}

Concave::Concave(const Concave& cn)

{

A\_ = cn.A\_;

B\_ = cn.B\_;

C\_ = cn.C\_;

D\_ = cn.D\_;

}

Concave::~Concave(){}

double Concave::getArea() const

{

return getSquare(A\_,B\_,D\_) + getSquare(B\_, C\_, D\_);

}

FrameRectangle Concave::getFrameRectangle()const

{

double x\_min = std::min(A\_.x\_,std::min( B\_.x\_, C\_.x\_));

double x\_max = std::max(A\_.x\_, std::max(B\_.x\_, C\_.x\_));

double y\_min = std::min(A\_.y\_, std::min(B\_.y\_, C\_.y\_));

double y\_max = std::max(A\_.y\_, std::max(B\_.y\_, C\_.y\_));

Point rectangleMiddle((x\_max+x\_min)/2, (y\_max + y\_min)/2);

FrameRectangle rectangle(x\_max - x\_min, y\_max - y\_min, rectangleMiddle);

return rectangle;

}

void Concave::move(const Point& point)

{

// перемещение верших

Point move(point.x\_ - D\_.x\_, point.y\_ - D\_.y\_);

A\_ + move;

B\_ + move;

C\_ + move;

D\_ = point; //перемещение центра

}

void Concave::move(const double x, const double y)

{

Point movement(x, y);

D\_ + movement;

A\_ + movement;

B\_ + movement;

C\_ + movement;

}

Shape\* Concave::clone(){ return new Concave(\*this); }

Concave& Concave::operator=(const Concave& triangle)

{

A\_ = triangle.A\_;

B\_ = triangle.B\_;

C\_ = triangle.C\_;

D\_ = triangle.D\_;

return \*this;

}

std::string Concave::getName() const {return "CONCAVE"; }

Point Concave::getMiddle() const{ return D\_;}

void Concave::scale(double k)

{

if (k <= 0) throw std::invalid\_argument(ERROR\_OF\_INPUT);

else

{

A\_ = (A\_ - D\_) \* k + D\_;

B\_ = (B\_ - D\_) \* k + D\_;

C\_ = (C\_ - D\_) \* k + D\_;

}

}

void getInfo(const Shape &figure)

{

cout << std::setprecision(1) << std::fixed;

cout<<figure.getName()<<" ";

cout << figure.getArea() << " ";

cout << figure.getFrameRectangle().getLeftDown();

cout<< figure.getFrameRectangle().getRightTop()<<"\n";

}

## *CompositeShape.h*

#pragma once

#include"Classes.h"

#include<vector>

const std::string ERROR\_OF\_SIZE = "Достигнут максимальный размер массива.Элемент не может быть добавлен\n";

class CompositeShape

{

private:

std::vector <Shape\*> arr;

int maxCount;

public:

CompositeShape();

~CompositeShape();

void scale(const double k);

void move(const Point &point);

void add( Shape& figure);

FrameRectangle& getFrameRectangle()const;

void countOfTypes() const;

};

void getInfo( CompositeShape& figure);

## *CompositeShape.cpp*

#include "CompositeShape.h"

CompositeShape::CompositeShape()

{

std::string value;

cout << "Максимальное количество элементов в составной фигуре = ";

cin >> value;

if (trunc(inputToDouble(value)) == inputToDouble(value) && inputToDouble(value) > 0) maxCount = inputToDouble(value);

else throw std::runtime\_error(ERROR\_OF\_INPUT);

}

CompositeShape::~CompositeShape()

{

for (auto figure : arr) delete figure;

}

void CompositeShape::scale(const double k)

{

double a = getFrameRectangle().getMiddle().x\_;

double b = getFrameRectangle().getMiddle().y\_;

for (auto element : arr)

{

double x = k \* (element->getMiddle().x\_ - a);

double y = k \* (element->getMiddle().y\_ - b );

element->scale(k);

element->move(x, y);

}

}

void CompositeShape::move(const Point &point)

{

double x = point.x\_ - getFrameRectangle().getMiddle().x\_;

double y = point.y\_ - getFrameRectangle().getMiddle().y\_;

for (auto element : arr) element->move(x,y);

}

void CompositeShape::add( Shape& figure)

{

if (arr.size() < maxCount) arr.push\_back(&figure);

else throw std::runtime\_error(ERROR\_OF\_SIZE);

}

FrameRectangle& CompositeShape::getFrameRectangle()const

{

double x\_max , y\_max;

x\_max=y\_max=DBL\_MIN;

double x\_min, y\_min;

x\_min = y\_min = DBL\_MAX;

for (auto element : arr)

{

x\_max = std::max(element->getFrameRectangle().getRightTop().x\_, x\_max);

y\_max = std::max(element->getFrameRectangle().getRightTop().y\_,y\_max);

x\_min = std::min(element->getFrameRectangle().getLeftDown().x\_, x\_min);

y\_min = std::min(element->getFrameRectangle().getLeftDown().y\_, y\_min);

}

Point rectangleMiddle((x\_max + x\_min) / 2, (y\_max + y\_min) / 2);

FrameRectangle rectangle(x\_max - x\_min, y\_max - y\_min, rectangleMiddle);

return rectangle;

}

void CompositeShape::countOfTypes() const

{

int count\_square, count\_concave;

count\_square = count\_concave = 0;

for (auto figure : arr)

{

if (figure->getName() == "SQUARE")count\_square++;

else count\_concave++;

}

cout << "SQUARE = " << count\_square << " CONCAVE = " << count\_concave << "\n";

}

void getInfo( CompositeShape& figure)

{

cout << std::setprecision(1) << std::fixed;

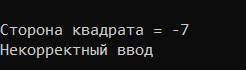
figure.countOfTypes();

cout << figure.getFrameRectangle().getLeftDown();

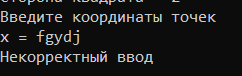
cout << figure.getFrameRectangle().getRightTop() << "\n";

}

# Приложение A



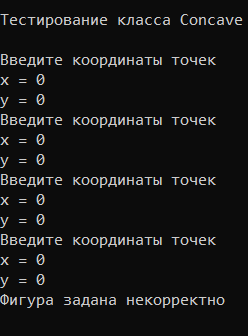
*Сторона квадрата задана некорректно*



*Координаты точки заданы некорректно*



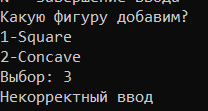
*Коэффициент изотропного масштабирования задан некорректно*



*Координаты невырожденного треугольника заданы некорректно*



*Максимальное количество элементов в составной фигуре задано некорректно*

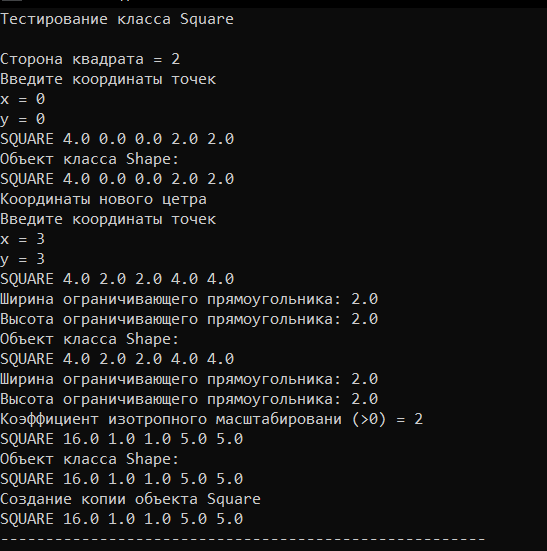


*Выбор добавляемой фигуры задан некорректно*

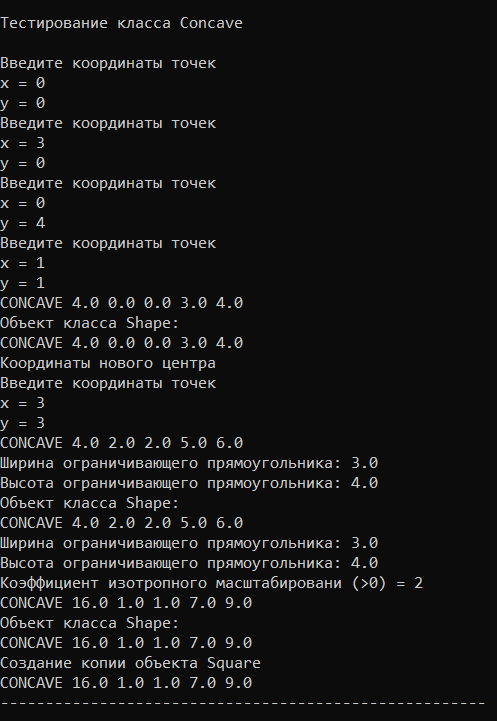


*Выбор о продолжении вводы некорректен*

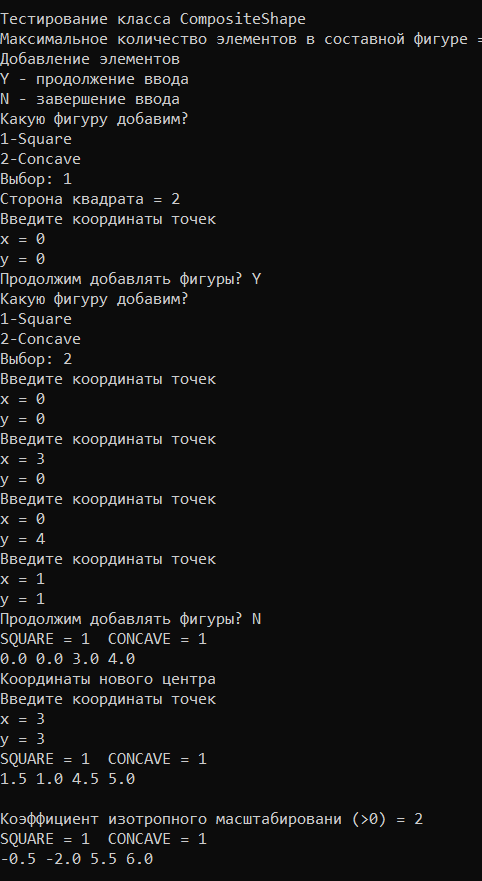
# Приложение B



*Корректная работа программы*



*Корректная работа программы*



*Корректная работа программы*

# Вывод

В ходе работы над программой были закреплены навыки работы с классами, их свойствами и методами, осуществлена перегрузка операторов и написаны дружественные функции. Использована конструкция try … catch() для обработки ошибок.