

**МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И.УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ**

**ЛАБОРАТОРНАЯ РАБОТА
по дисциплине «Конструирование ПО»
Тема: Программирование контейнерных классов**

Студент гр. 6304

Преподаватель

Корытов П.В.

Преподаватель П.П.

Санкт-Петербург

2019

Содержание

1	Постановка задачи	2
1.1	Цель работы	2
1.2	Формулировка задания	2
2	Ход работы	4
2.1	Создание классов фигур	4
2.2	Проверка созданных классов	6
	Приложение А. Код shape.h	9
	Приложение Б. Код shape.cpp	10
	Приложение В. Код point.h	11
	Приложение Г. Код point.cpp	12
	Приложение Д. Код pentagram.h	14
	Приложение Е. Код pentagram.cpp	15
	Приложение Ж. Код atanSegment.h	16
	Приложение З. Код atanSegment.cpp	17
	Приложение И. Код text.h	18
	Приложение К. Код text.cpp	19
	Приложение Л. Код pentagramText.h	21
	Приложение М. Код pentagramText.cpp	22
	Приложение Н. Код test.cpp	23
	Приложение О. Код demo.cpp	27
	Приложение П. Код colors.h	29
	Приложение Р. Код myCli.h	30
	Приложение С. Код myCli.cpp	31

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Цель работы

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2. Формулировка задания

1. Разработка программ

(a) Настройка среды. Выполнение индивидуального задания

- Индивидуальное задание: Написать классы для создания графических объектов. Классы должны иметь общий абстрактный базовый класс Shape с чистыми виртуальными функциями.
- Необходимо использовать множественное наследование. В классах должны быть предусмотрены виртуальные функции для вывода информации об объектах в поток, а Shape должен иметь дружественный перегруженный оператор «.
- Исходный текст должен быть разделен на три файла *.h, *.cpp и *.cpp с тестовой программой.

(b) Работа в режиме отладки.

- Запустить программу и просмотреть ее работу по шагам (Build -> Start Debug -> Go)
- Просмотреть иерархию классов и найти примеры множественного наследования.
- Расставить точки прерывания программы (Break Points) и протестировать её работу.

- Для выяснения текущих значений переменных, использовать механизм “Watch variable”.

2. Применение стандартной библиотеки STL

(a) Составить консольные приложения, демонстрирующие основные операции с контейнерами и итераторами STL.

- Заполняя 3 контейнера строками из `<cstring>` или другими элементами, продемонстрировать отличия
 - последовательностей (`vector`, `list`, `deque`);
 - адаптеров последовательностей (`stack`, `queue`, `priority_queue`);
 - ассоциативных контейнеров на базе `map`.
- На примере заполнения одного контейнера-последовательности из предыдущего задания целыми числами, протестировать интерфейсы контейнера и итератора.
- Аналогично протестировать ассоциативный контейнер, заполняя его указателями на разные графические объекты. Протестировать алгоритмы-методы и алгоритмы-классы на множестве графических элементов.

(b) Реализовать новый шаблон контейнера и шаблон итератора для него по индивидуальному заданию.

- Предусмотреть обработку исключительных ситуаций.
- Протестировать контейнер, заполнив его графическими объектами.
- В отчете формально описать реализуемую структуру данных и абстракцию итерации, перечислить все отношения между классами, описать интерфейсы классов и особенности реализации.

2. ХОД РАБОТЫ

Использованное ПО

1. **JetBrains CLion** — IDE для C/C++
2. **Valgrind** — проверка утечек памяти
3. **Google Test** — юнит-тесты для C++
4. **X_YLA_TE_X, neovim** — сборка и написание отчёта

2.1. Создание классов фигур

1. Создан класс Point (файлы point.h, point.cpp). Он содержит основные математические операции по работе с точками, например конвертацию координат.
2. Создан общий абстрактный класс Shape (shape.h, shape.cpp). Он содержит базовые методы, необходимые всем фигурам, несколько виртуальных и чистых виртуальных методов.

Оператор вывода в поток не виртуальный, т.к. дружественные методы не наследуются. Вместо этого в операторе вызывается чистый виртуальный метод print, который является protected и может наследоваться.

3. Написаны классы Pentagon, AtanSegment, Text, PentagonText. Класс Pentagon наследуется от Pentagon и Text. UML-диаграмма классов представлена на рисунке 1.

Исходные коды представлены в приложениях в соответствующих файлах

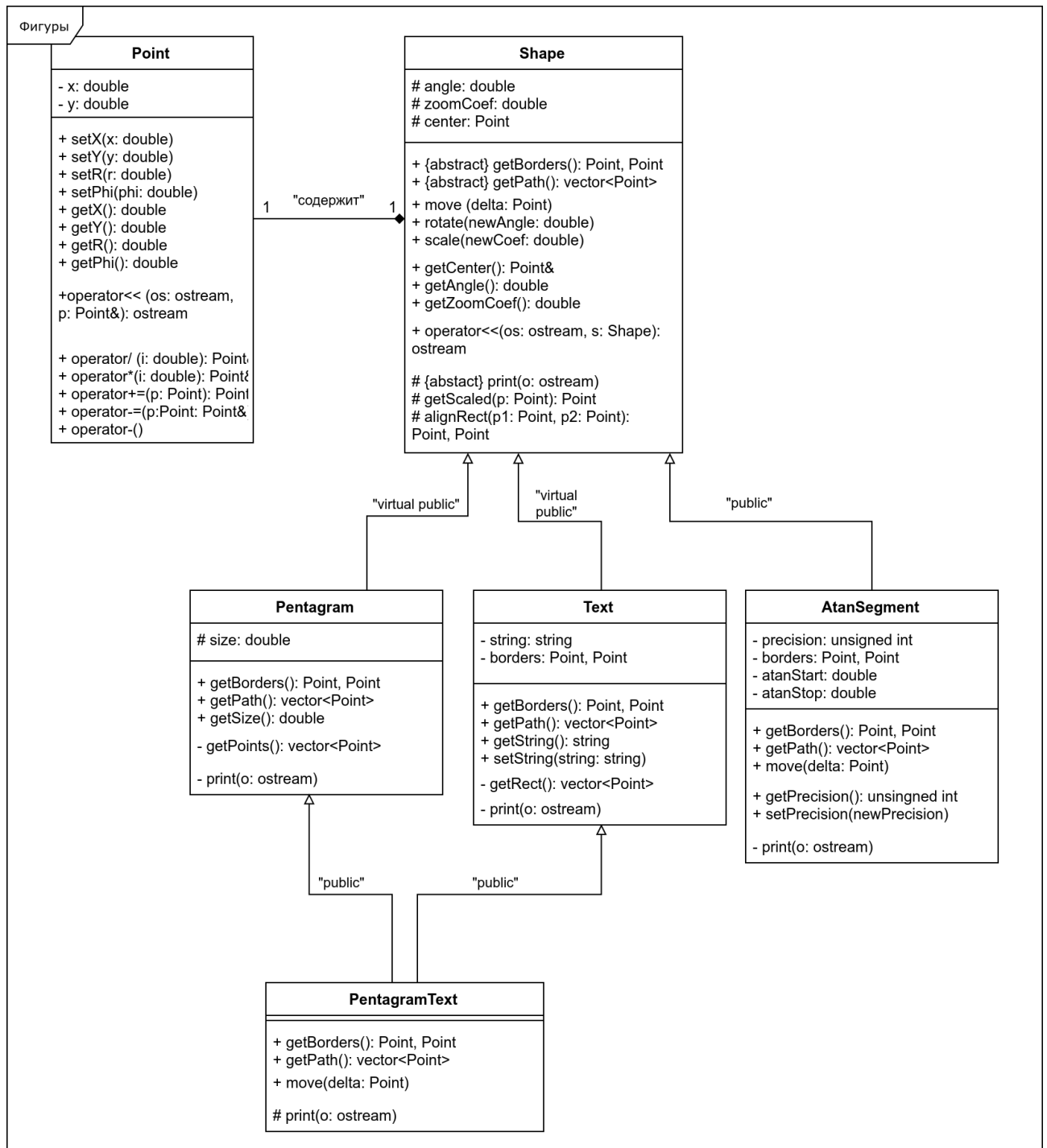
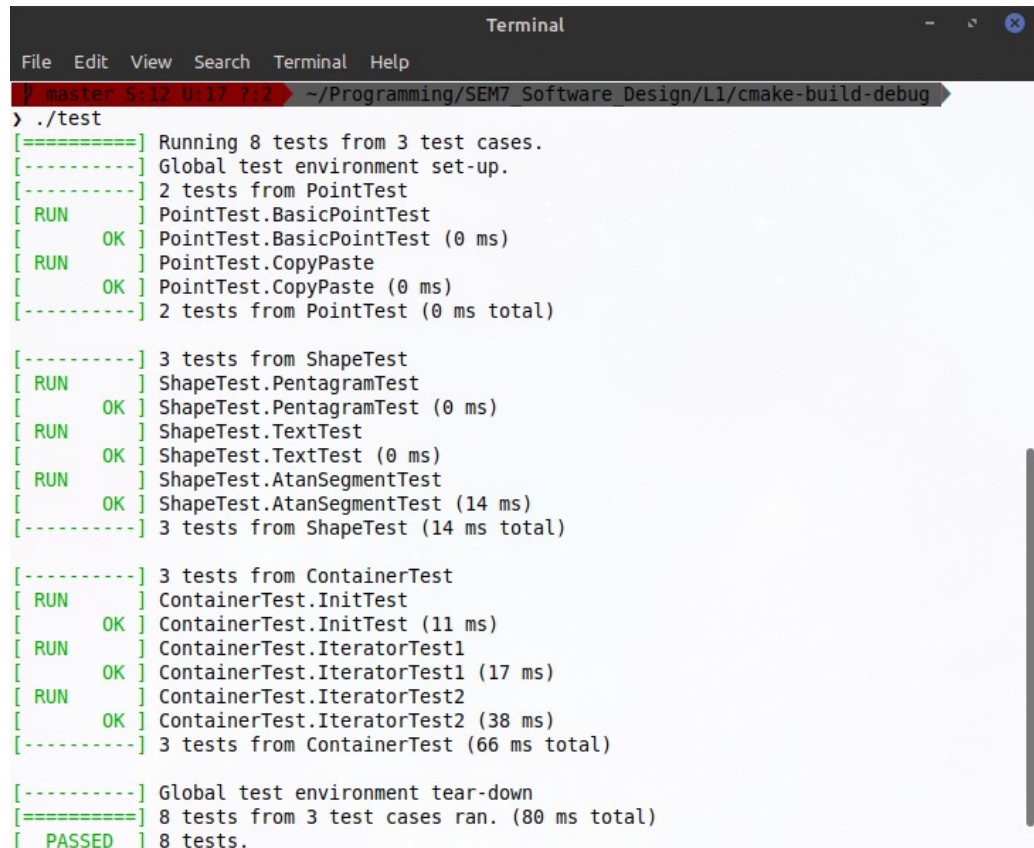


Рисунок 1. UML-диаграмма классов

2.2. Проверка созданных классов

1. Для тестирования созданных классов подключен фреймворк GoogleTest, написан ряд юнит-тестов. Исходный код тестов находится в test.cpp. Процесс запуска юнит-тестов представлен на 2



```
Terminal
File Edit View Search Terminal Help
~/Programming/SEM7 Software Design/L1/cmake-build-debug
> ./test
[=====] Running 8 tests from 3 test cases.
[-----] Global test environment set-up.
[-----] 2 tests from PointTest
[ RUN      ] PointTest.BasicPointTest
[ OK       ] PointTest.BasicPointTest (0 ms)
[ RUN      ] PointTest.CopyPaste
[ OK       ] PointTest.CopyPaste (0 ms)
[-----] 2 tests from PointTest (0 ms total)

[-----] 3 tests from ShapeTest
[ RUN      ] ShapeTest.PentagramTest
[ OK       ] ShapeTest.PentagramTest (0 ms)
[ RUN      ] ShapeTest.TextTest
[ OK       ] ShapeTest.TextTest (0 ms)
[ RUN      ] ShapeTest.AtanSegmentTest
[ OK       ] ShapeTest.AtanSegmentTest (14 ms)
[-----] 3 tests from ShapeTest (14 ms total)

[-----] 3 tests from ContainerTest
[ RUN      ] ContainerTest.InitTest
[ OK       ] ContainerTest.InitTest (11 ms)
[ RUN      ] ContainerTest.IteratorTest1
[ OK       ] ContainerTest.IteratorTest1 (17 ms)
[ RUN      ] ContainerTest.IteratorTest2
[ OK       ] ContainerTest.IteratorTest2 (38 ms)
[-----] 3 tests from ContainerTest (66 ms total)

[-----] Global test environment tear-down
[=====] 8 tests from 3 test cases ran. (80 ms total)
[ PASSED  ] 8 tests.
```

Рисунок 2. Запуск Google Tests

2. Для демонстрации написан ряд преобразований над фигурами. Код преобразований в файле demo.cpp. Ход преобразований представлен на рисунке 3
- Преобразования над всеми фигурами выполнены корректно.
3. Для отладки использована обертка над gdb, встроенная в CLion и Valgrind для контроля над утечками памяти. Приведены скриншоты, сделанные в ходе отладки:
 - Рисунок 4 — остановка на breakpoint'е
 - Рисунок 5 — оценка выражения

```
Terminal
File Edit View Search Terminal Help
~/Programming/SEM7 Software Design/L1/cmake-build-debug
> ./demo

Korytov Pavel, 6304. LR 1

Pentagram: {center: (0,0); size: 10}

Starting transformation sequence
Path[ (10,0) (-8.09,5.88) (3.09,-9.51) (3.09,9.51) (-8.09,-5.88) (10,0) ]
Moving to (10,20)
Path[ (20,20) (1.91,25.9) (13.1,10.5) (13.1,29.5) (1.91,14.1) (20,20) ]
Rotating at 45 degrees
Path[ (17.1,27.1) (0.123,18.4) (18.9,15.5) (5.46,28.9) (8.44,10.1) (17.1,27.1) ]
Scaling 12.5 times
Path[ (98.4,108) (-113,0.446) (121,-36.7) (-46.7,131) (-9.55,-103) (98.4,108) ]
Scaling back
Path[ (17.1,27.1) (0.123,18.4) (18.9,15.5) (5.46,28.9) (8.44,10.1) (17.1,27.1) ]
Rotating back
Path[ (20,20) (1.91,25.9) (13.1,10.5) (13.1,29.5) (1.91,14.1) (20,20) ]
Moving back
Path[ (10,0) (-8.09,5.88) (3.09,-9.51) (3.09,9.51) (-8.09,-5.88) (10,0) ]
Transformation sequence complete

AtanSegment {precision: 5; box:(1,1) (10,10)}

Starting transformation sequence
Path[ (1,3.25) (2.8,3.95) (4.6,4.93) (6.4,6.07) (8.2,7.05) (10,7.75) ]
Moving to (10,20)
Path[ (11,23.2) (12.8,24) (14.6,24.9) (16.4,26.1) (18.2,27) (20,27.8) ]
Rotating at 45 degrees
Path[ (13.9,20.7) (14.7,22.5) (15.3,24.5) (15.7,26.5) (16.3,28.5) (17.1,30.3) ]
```

Рисунок 3. Запуск преобразований

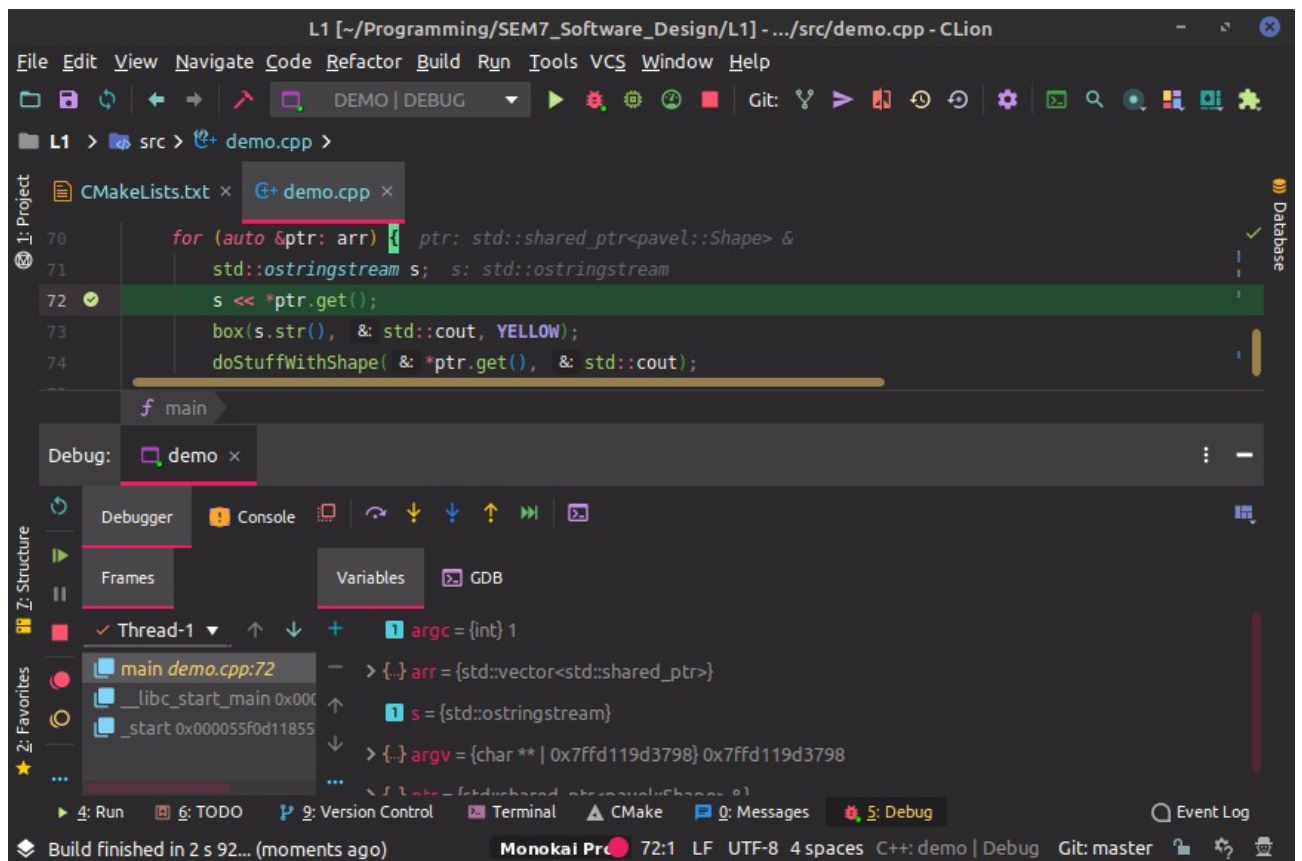


Рисунок 4. Остановка на breakpoint'e

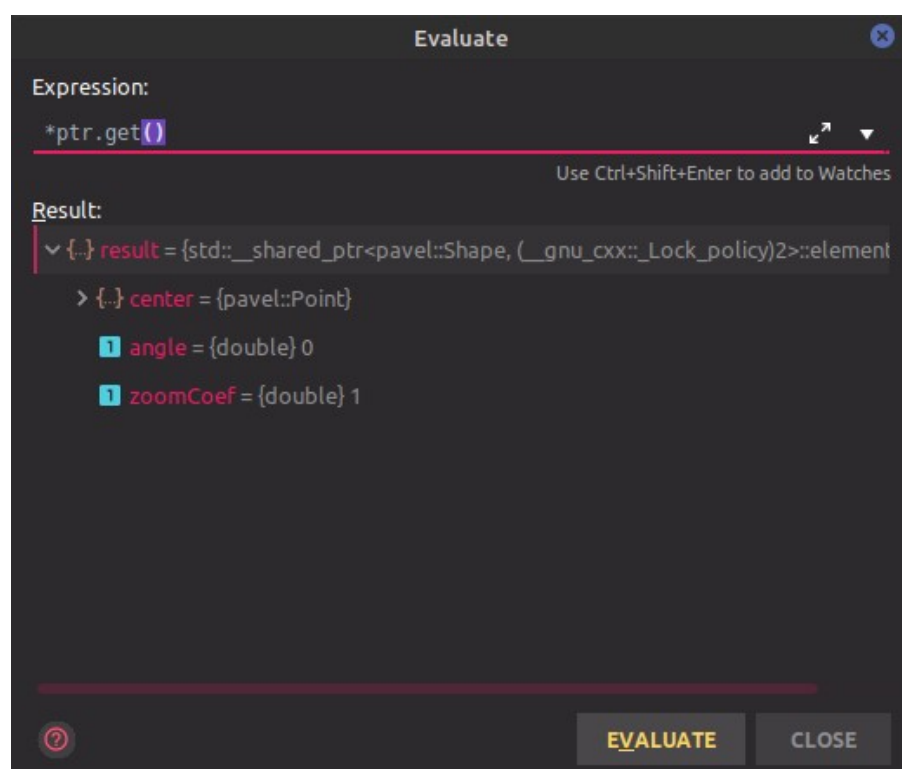


Рисунок 5. Оценка значения выражения

ПРИЛОЖЕНИЕ А

Код shape.h

```
1  #pragma once
2
3  #include <tuple>
4  #include <vector>
5  #include <utility>
6  #include <ostream>
7  #include "point.h"
8
9  namespace pavel {
10     class Shape {
11     public:
12         explicit Shape(Point center = Point {0, 0}): center(std::move(center)) {}
13         virtual ~Shape() = default;
14         [[nodiscard]] virtual std::pair<Point, Point> getBorders() = 0;
15         [[nodiscard]] virtual std::vector<Point> getPath() = 0;
16
17         virtual void move(Point delta);
18         virtual void rotate(double newAngle);
19         virtual void scale(double newCoef);
20
21         [[nodiscard]] const Point &getCenter() const;
22         [[nodiscard]] double getAngle() const;
23         [[nodiscard]] double getZoomCoef() const;
24
25         friend std::ostream &operator<<(std::ostream &os, const Shape &shape);
26
27     protected:
28         virtual void print(std::ostream &o) const = 0;
29
30         Point getScaled(Point p);
31         static std::pair<Point, Point> alignRect(Point p1, Point p2);
32         Point center;
33         double angle = 0;
34         double zoomCoef = 1;
35     };
36
37     inline std::ostream &operator<<(std::ostream &os, const Shape &shape) {
38         shape.print(os);
39         return os;
40     }
41 }
```

ПРИЛОЖЕНИЕ Б

Код shape.cpp

```
1  #include <algorithm>
2  #include <cmath>
3  #define _USE_MATH_DEFINES
4
5  #include "point.h"
6  #include "shape.h"
7
8  namespace Pavel {
9      const Point &Pavel::Shape::getCenter() const {
10         return center;
11     }
12
13     double Shape::getAngle() const {
14         return angle;
15     }
16
17     double Shape::getZoomCoef() const {
18         return zoomCoef;
19     }
20
21     void Shape::move(Point delta) {
22         center += delta;
23     }
24
25     void Shape::rotate(double newAngle) {
26         angle = std::fmod(angle + newAngle, M_PI * 1);
27     }
28
29     void Shape::scale(double newCoef) {
30         zoomCoef *= newCoef;
31     }
32
33     Point Shape::getScaled(Point p) {
34         auto p2 = p - center;
35         p2.setR(p2.getR() * getZoomCoef());
36         p2.setPhi(p2.getPhi() + angle);
37         p2 += center;
38         return p2;
39     }
40
41     std::pair<Point, Point> Shape::alignRect(Point p1, Point p2) {
42         return std::make_pair(
43             Point(
44                 std::min(p1.getX(), p2.getX()),
45                 std::min(p1.getY(), p2.getY())
46             ),
47             Point(
48                 std::max(p1.getX(), p2.getX()),
49                 std::max(p1.getY(), p2.getY())
50             )
51         );
52     }
53 }
54 }
```

ПРИЛОЖЕНИЕ В

Код point.h

```
1  #pragma once
2
3  #include <iostream>
4
5  namespace pavel {
6      class Point {
7          public:
8              /* Constructors and copy */
9              explicit Point(double x = 0, double y = 0): x(x), y(y) {}
10             Point(const Point &p) = default;
11             ~Point() = default;
12
13             friend void swap(Point& a, Point& b);
14             Point& operator=(Point p);
15             Point(Point&& p) noexcept;
16
17             /* Getters, setters */
18             [[nodiscard]] double getX() const { return x; }
19             [[nodiscard]] double getY() const { return y; }
20             void setX(double x_) { x = x_; }
21             void setY(double y_) { y = y_; }
22
23             [[nodiscard]] double getR() const;
24             [[nodiscard]] double getPhi() const;
25             void setR(double newR);
26             void setPhi(double newPhi);
27
28             /* Arithmetic */
29             Point& operator/(double i);
30             Point& operator*(double i);
31             Point& operator+=(Point p);
32             Point& operator—=(Point p);
33             Point& operator—();
34
35             /* Stream */
36             friend std::ostream& operator<< (std::ostream& os, const Point& p);
37             friend std::istream& operator>> (std::istream& is, Point& p);
38
39             private:
40                 double x{};
41                 double y{};
42     };
43
44     bool operator==(const Point& a, const Point& b);
45     Point operator+(const Point& p1, const Point& p2);
46     Point operator—(const Point& p1, const Point& p2);
47
48     Point operator+(const Point& p1, double a);
49     Point operator—(const Point& p1, double a);
50 }
```

ПРИЛОЖЕНИЕ Г

Код point.cpp

```
1  #include <algorithm>
2  #include <cmath>
3
4  #include "point.h"
5
6  namespace pavel {
7      void swap(Point &a, Point &b) {
8          std::swap(a.x, b.x);
9          std::swap(a.y, b.y);
10     }
11
12     Point &Point::operator=(Point p) {
13         swap(*this, p);
14         return *this;
15     }
16
17     Point::Point(Point &&p) noexcept {
18         swap(*this, p);
19     }
20
21     Point & Point::operator/(double i)
22     {
23         x = x / i;
24         y = y / i;
25         return *this;
26     }
27
28     Point & Point::operator*(double i)
29     {
30         x = x * i;
31         y = y * i;
32         return *this;
33     }
34
35     std::ostream & operator<<(std::ostream & os, const Point& p)
36     {
37         os << "(" << p.x << "," << p.y << ")";
38         return os;
39     }
40
41     std::istream & operator>> (std::istream & is, Point& p)
42     {
43         is >> p.x >> p.y;
44         return is;
45     }
46
47     Point &Point::operator+=(Point p) {
48         x = x + p.getX();
49         y = y + p.getY();
50         return *this;
51     }
52
53     Point &Point::operator-=(Point p) {
54         x = x - p.getX();
55         y = y - p.getY();
56         return *this;
```

```

57     }
58
59     double Point::getR() const {
60         return std::sqrt(x*x + y*y);
61     }
62
63     double Point::getPhi() const {
64         // if ((x == 0) && (y > 0)) return M_PI/2;
65         // if ((x == 0) && (y < 0)) return 3*M_PI/2;
66         // if ((x > 0) && (y >= 0)) return std::atan(y/x);
67         // if ((x > 0) && (y < 0)) return std::atan(y/x) + 2*M_PI;
68         // if (x < 0) return std::atan(y/x) + M_PI;
69         // if (x > 0) return std::atan2(y, x);
70         // if ((x < 0) && (y >= 0)) return std::atan2(y, x) + M_PI;
71         // if ((x < 0) && (y < 0)) return std::atan2(y, x) - M_PI;
72         // if ((x == 0) && (y > 0)) return M_PI / 2;
73         // if ((x == 0) && (y < 0)) return -M_PI / 2;
74         return std::atan2(y, x);
75     }
76
77     void Point::setR(double newR) {
78         double phi = getPhi();
79         x = newR * std::cos(phi);
80         y = newR * std::sin(phi);
81     }
82
83     void Point::setPhi(double newPhi) {
84         double r = getR();
85         x = r * std::cos(newPhi);
86         y = r * std::sin(newPhi);
87     }
88
89     Point &Point::operator~() {
90         x = -x;
91         y = -y;
92     }
93
94     bool operator==(const Point & a, const Point & b){
95         return ((a.getX() == b.getX()) && (a.getY() == b.getY()));
96     }
97
98     Point operator+(const Point& p1, const Point& p2) {
99         return Point(p1.getX() + p2.getX(), p1.getY() + p2.getY());
100     }
101
102     Point operator-(const Point& p1, const Point& p2) {
103         return Point(p1.getX() - p2.getX(), p1.getY() - p2.getY());
104     }
105
106     Point operator+(const Point& p1, double a) {
107         return Point{p1.getX() + a, p1.getY() + a};
108     }
109     Point operator-(const Point& p1, double a) {
110         return Point {p1.getX() - a, p1.getY() - a};
111     }
112 }

```

ПРИЛОЖЕНИЕ Д

Код pentagram.h

```
1  #pragma once
2
3  #include <ostream>
4  #include "point.h"
5  #include "shape.h"
6
7  namespace pavel {
8      class Pentagon: virtual public Shape {
9      public:
10         explicit Pentagon(Point center = Point {0, 0}, double size = 1);
11
12         std::pair<Point, Point> getBorders() override;
13
14         std::vector<Point> getPath() override;
15
16         [[nodiscard]] double getSize() const;
17
18     protected:
19         double size;
20
21     private:
22         void print(std::ostream &o) const override;
23         [[nodiscard]] std::vector<Point> getPoints();
24     };
25 }
```

ПРИЛОЖЕНИЕ Е

Код pentagram.cpp

```
1  #include <cmath>
2  #include <tuple>
3
4  #include "pentagram.h"
5
6  namespace pavel {
7      Pentagon::Pentagon(Point center, double size): Shape(center), size(size) {}
8
9      std::vector<Point> Pentagon::getPoints() {
10         auto points = std::vector<Point>();
11         for (int i = 0; i < 5; i++) {
12             auto p = Point {size, 0};
13             p.setPhi(i * M_PI * 2 / 5);
14             p += center;
15             p = getScaled(p);
16             points.push_back(p);
17         }
18         return points;
19     }
20
21     std::pair<Point, Point> Pentagon::getBorders() {
22         auto coef = size * zoomCoef;
23         return std::make_pair(
24             center - Point {coef, coef},
25             center + Point {coef, coef}
26         );
27     }
28
29     std::vector<Point> Pentagon::getPath() {
30         auto points = getPoints();
31         return std::vector<Point> {
32             points[0],
33             points[2],
34             points[4],
35             points[1],
36             points[3],
37             points[0]
38         };
39     }
40
41     double Pentagon::getSize() const {
42         return size * zoomCoef;
43     }
44
45     void Pentagon::print(std::ostream &o) const {
46         o << "Pentagram: {center: " << getCenter() << "; size: " << getSize() << "};";
47     }
48
49
50 }
```


ПРИЛОЖЕНИЕ Ж

Код atanSegment.h

```
1  #pragma once
2
3  #include "shape.h"
4
5  namespace pavel {
6      class AtanSegment : public Shape {
7      public:
8          explicit AtanSegment(Point bottomLeft, Point topRight, unsigned int precision =
9              20);
10
11          std::pair<Point, Point> getBorders() override;
12
13          std::vector<Point> getPath() override;
14
15          [[nodiscard]] unsigned int getPrecision() const;
16          void setPrecision(unsigned int newPrecision);
17
18          void move(Point delta) override;
19
20      protected:
21          void print(std::ostream &o) const override;
22
23      private:
24          unsigned int precision;
25          std::pair<Point, Point> borders;
26          double atanStart = -1;
27          double atanStop = 1;
28      };
29 }
```

ПРИЛОЖЕНИЕ 3

Код atanSegment.cpp

```
1  #include <cmath>
2  #include "atanSegment.h"
3
4  namespace pavel {
5      AtanSegment::AtanSegment(Point bottomLeft, Point topRight, unsigned int precision)
6          : precision(precision), borders(std::make_pair(topRight, bottomLeft)) {
7          center = (topRight + bottomLeft) / 2;
8      }
9
10     std::pair<Point, Point> AtanSegment::getBorders() {
11         return alignRect(getScaled(borders.first), getScaled(borders.second));
12     }
13
14     void AtanSegment::move(Point delta) {
15         Shape::move(delta);
16         borders.first += delta;
17         borders.second += delta;
18     }
19
20     std::vector<Point> AtanSegment::getPath() {
21         double x = atanStart;
22         auto path = std::vector<Point>();
23         auto boxDelta = borders.first - center;
24         auto delta = (atanStop - atanStart) / precision;
25         double xSize = boxDelta.getX();
26         double ySize = boxDelta.getY();
27
28         for (unsigned int i = 0; i <= precision; i++) {
29             auto y = std::atan(x) * 2 / M_PI;
30             auto p = Point {x * xSize, y * ySize} + center;
31             p = getScaled(p);
32             path.push_back(p);
33             x += delta;
34         }
35         return path;
36     }
37
38     void AtanSegment::print(std::ostream &o) const {
39         o << "AtanSegment_{precision:}" << precision
40         << ";_box:" << borders.second << "_" << borders.first << "}";
41     }
42
43     unsigned int AtanSegment::getPrecision() const {
44         return precision;
45     }
46
47     void AtanSegment::setPrecision(unsigned int newPrecision) {
48         AtanSegment::precision = newPrecision;
49     }
50
51 }
```

ПРИЛОЖЕНИЕ И

Код text.h

```
1  #pragma once
2
3  #include "shape.h"
4
5  namespace pavel {
6      class Text : virtual public Shape {
7      public:
8          explicit Text(std::string string, Point bottomLeft, Point topRight);
9
10         std::pair<Point, Point> getBorders() override;
11
12         std::vector<Point> getPath() override;
13
14         [[nodiscard]] const std::string &getString() const;
15         void setString(const std::string &newString);
16
17         void move(Point delta) override;
18
19     protected:
20         void print(std::ostream &o) const override;
21
22     private:
23         std::string string;
24         std::pair<Point, Point> borders;
25
26         std::vector<Point> getRect();
27     };
28 }
```

ПРИЛОЖЕНИЕ К

Код text.cpp

```
1  #include <tuple>
2  #include <utility>
3  #include <algorithm>
4  #include "text.h"
5
6  namespace pavel {
7
8      const std::string &Text::getString() const {
9          return string;
10     }
11
12     void Text::setString(const std::string &newString) {
13         string = newString;
14     }
15
16     std::vector<Point> Text::getRect() {
17         auto [p1, p3] = borders;
18         auto points = std::vector(
19             {getScaled(p1),
20              (getScaled(Point{p1.getX(), p3.getY()})),
21              getScaled(p3),
22              getScaled(Point{p1.getX(), p3.getY()})
23             });
24         return points;
25     }
26
27     std::pair<Point, Point> Text::getBorders() {
28         auto points = getRect();
29         auto xS = std::vector<double>();
30         auto yS = std::vector<double>();
31         for (auto & point: points) {
32             xS.push_back(point.getX());
33             yS.push_back(point.getY());
34         }
35         const auto [minX, maxX] = std::minmax_element(xS.begin(), xS.end());
36         const auto [minY, maxY] = std::minmax_element(yS.begin(), yS.end());
37         return std::make_pair(
38             Point {*minX, *minY},
39             Point {*maxX, *maxY}
40         );
41     }
42
43     std::vector<Point> Text::getPath() {
44         return getRect();
45     }
46
47     void Text::move(Point delta) {
48         Shape::move(delta);
49         borders.first += delta;
50         borders.second += delta;
51     }
52
53     void Text::print(std::ostream &o) const {
54
55         o << "└─Text:└─{string:└─" << string<< "└─box:"
56         << borders.second << "└─" << borders.first << "└─";
```

```
57     }
58
59     Text::Text(std::string string, Point bottomLeft, Point topRight)
60         :string(std::move(string)), borders(std::make_pair(topRight, bottomLeft)) {
61         center = (topRight + bottomLeft) / 2;
62     }
63 }
```

ПРИЛОЖЕНИЕ Л

Код pentagramText.h

```
1  #include "text.h"
2  #include "pentagram.h"
3
4  namespace pavel {
5      class PentagonText : public Text, public Pentagon {
6      public:
7          explicit PentagonText(std::string string, Point center = Point {0, 0}, double
              size = 1);
8
9          std::pair<Point, Point> getBorders() override;
10
11         std::vector<Point> getPath() override;
12
13         void move(Point delta) override;
14
15     protected:
16         void print(std::ostream &o) const override;
17     };
18 }
```

ПРИЛОЖЕНИЕ М

Код pentagramText.cpp

```
1  #include "pentagramText.h"
2
3  #include <utility>
4
5  namespace pavel {
6      void PentagramText::print(std::ostream &o) const {
7          o << "PentagramText{center:_ " << Pentagram::center << ";_size:_ " << size << ";_
8              string:_ " << getString() << "}";
9      }
10
11     std::pair<Point, Point> PentagramText::getBorders() {
12         return Pentagram::getBorders();
13     }
14
15     std::vector<Point> PentagramText::getPath() {
16         auto pentagramPath = Pentagram::getPath();
17         auto textPath = Text::getPath();
18         pentagramPath.insert(pentagramPath.end(), textPath.begin(), textPath.end());
19         return pentagramPath;
20     }
21
22     void PentagramText::move(Point delta) {
23         Text::move(delta);
24     }
25
26     PentagramText::PentagramText(std::string string, Point center, double size)
27         : Pentagram(center, size), Text(std::move(string), center + (size / 2), center -
28             (size / 2))
29     {}
30 }
```

ПРИЛОЖЕНИЕ Н

Код test.cpp

```
1  #pragma clang diagnostic push
2  #pragma ide diagnostic ignored "cert-msc32-c"
3  #pragma ide diagnostic ignored "cert-err58-cpp"
4
5  #include <ctime>
6  #include <iostream>
7  #include <random>
8  #include "gtest/gtest.h"
9
10 #include "point.h"
11 #include "pentagram.h"
12 #include "text.h"
13 #include "atanSegment.h"
14 #include "pentagramText.h"
15 #include "hashMap.h"
16
17
18 TEST(PointTest, BasicPointTest) {
19     std::mt19937 gen(time(nullptr));
20     std::uniform_real_distribution<> urd(-10, 10);
21     auto x = urd(gen);
22     auto y = urd(gen);
23
24     pavel::Point p1 {x, y};
25     ASSERT_EQ(p1.getX(), x);
26     ASSERT_EQ(p1.getY(), y);
27     ASSERT_GE(p1.getR(), 0);
28     ASSERT_NEAR(p1.getPhi(), 0, M_PI);
29
30     p1.setX(y);
31     p1.setY(x);
32     ASSERT_EQ(p1.getX(), y);
33     ASSERT_EQ(p1.getY(), x);
34 }
35
36 TEST(PointTest, CopyPaste) {
37     pavel::Point p1 {2, 3};
38     auto p2 = p1;
39     auto p3(p2);
40
41     ASSERT_EQ(p1.getX(), p2.getX());
42     ASSERT_EQ(p2.getX(), p3.getX());
43     ASSERT_EQ(p1.getPhi(), p2.getPhi());
44     ASSERT_EQ(p1.getR(), p3.getR());
45
46     ASSERT_TRUE(p1 == p2);
47     auto p4 = p2 + p3;
48     auto p5 = p3 * 2;
49     ASSERT_TRUE(p4 == p5);
50     ASSERT_GT(p4.getR(), p1.getR());
51     ASSERT_GT(p5.getR(), p1.getR());
52     ASSERT_EQ(p3.getPhi(), p5.getPhi());
53 }
54
55 void checkBorders(pavel::Shape& figure) {
56     auto [p2, p1] = figure.getBorders();
```



```

57     for (pavel::Point& point : figure.getPath()) {
58         ASSERT_LE(point.getX(), p1.getX());
59         ASSERT_LE(point.getY(), p1.getY());
60
61         ASSERT_GE(point.getX(), p2.getX());
62         ASSERT_GE(point.getY(), p2.getY());
63     }
64 }
65
66 void standartMoveSequence(pavel::Shape& figure) {
67     ASSERT_NO_FATAL_FAILURE(checkBorders(figure));
68
69     figure.move(pavel::Point(99, -200));
70     ASSERT_NO_FATAL_FAILURE(checkBorders(figure));
71
72     figure.rotate(M_PI / 3);
73     figure.scale(14);
74     ASSERT_NO_FATAL_FAILURE(checkBorders(figure));
75
76     figure.scale(0.0001);
77     ASSERT_NO_FATAL_FAILURE(checkBorders(figure));
78 }
79
80 TEST(ShapeTest, PentagonTest) {
81     auto center = pavel::Point{2, 3};
82     auto fig = pavel::Pentagram(center, 10);
83     ASSERT_NO_FATAL_FAILURE(checkBorders(fig));
84
85     auto [ap1, ap2] = fig.getBorders();
86     fig.move(pavel::Point {2, 5});
87     auto [bp1, bp2] = fig.getBorders();
88
89     ASSERT_LT(ap1.getX(), bp1.getX());
90     ASSERT_LT(ap2.getY(), bp2.getY());
91
92     fig.rotate(M_PI * 0.42);
93     fig.scale(13.4);
94     auto [cp2, cp1] = fig.getBorders();
95     ASSERT_NO_FATAL_FAILURE(checkBorders(fig));
96     ASSERT_EQ(fig.getAngle(), M_PI*0.42);
97
98     ASSERT_LT(cp2.getX(), bp2.getX());
99     ASSERT_GT(cp1.getY(), bp1.getY());
100
101     ASSERT_EQ(fig.getCenter(), center + pavel::Point(2, 5));
102 }
103
104 TEST(ShapeTest, TextTest){
105     auto text = pavel::Text("Hello",
106                             pavel::Point(0, 0),
107                             pavel::Point(10, 10));
108     ASSERT_EQ(text.getString(), "Hello");
109     ASSERT_NO_FATAL_FAILURE(checkBorders(text));
110
111     text.setString("Goodbye");
112     ASSERT_EQ(text.getString(), "Goodbye");
113
114     ASSERT_NO_FATAL_FAILURE(standartMoveSequence(text));
115 }
116
117

```

```

118 TEST(ShapeTest, AtanSegmentTest) {
119     auto seg = Pavel::AtanSegment(Pavel::Point{0, 0}, Pavel::Point{10, 10}, 0);
120     ASSERT_NO_FATAL_FAILURE(checkBorders(seg));
121     ASSERT_NO_FATAL_FAILURE(standardMoveSequence(seg));
122
123     seg.setPrecision(20000);
124     ASSERT_NO_FATAL_FAILURE(checkBorders(seg));
125     ASSERT_EQ(seg.getPrecision(), 20000);
126     seg.setPrecision(2);
127     ASSERT_NO_FATAL_FAILURE(checkBorders(seg));
128 }
129
130
131
132 TEST(ContainerTest, InitTest) {
133     auto m = Pavel::HashMap<int, int>();
134     m.create(15, 20);
135     m.create(20, 30);
136     m.create(30, 40);
137     ASSERT_EQ(m.at(15), 20);
138     ASSERT_EQ(m.at(20), 30);
139     ASSERT_EQ(m.at(30), 40);
140     int val;
141     ASSERT_FALSE(m.get(16, val));
142     ASSERT_TRUE(m.get(15, val));
143     m.update(15, 25);
144     ASSERT_EQ(m.at(15), 25);
145     m.remove(15);
146     ASSERT_FALSE(m.get(15, val));
147 }
148
149 TEST(ContainerTest, IteratorTest1) {
150     auto m = Pavel::HashMap<int, double>();
151     m.create(1, 2);
152     m.create(2, 3);
153
154     auto iter = Pavel::HashMapIterator<int, double>(m);
155     while (!iter.end()) {
156         ASSERT_NE(iter.value(), 0);
157         ASSERT_NE(iter.key(), 0);
158         ++iter;
159     }
160 }
161
162 TEST(ContainerTest, IteratorTest2) {
163     auto m = Pavel::HashMap<int, double>();
164     m.create(1, 2);
165     m.create(2, 3);
166     m.create(10, 4);
167     m.create(20, 40);
168     m.create(25, 45);
169
170     auto it1 = m.begin();
171     auto it2 = m.end();
172     bool cmp = it1 == it2;
173     ASSERT_FALSE(cmp);
174
175     for (auto it = m.begin(); it != m.end(); it++) {
176         ASSERT_NE(it.value(), 0);
177         ASSERT_NE(it.key(), 0);
178     }

```

```
179 }
180
181 int main(int argc, char *argv[])
182 {
183     ::testing::InitGoogleTest(&argc, argv);
184     return RUN_ALL_TESTS();
185 }
186 #pragma clang diagnostic pop
```

ПРИЛОЖЕНИЕ О

Код demo.cpp

```
1  #include <cmath>
2  #include <memory>
3  #include <iostream>
4  #include <iomanip>
5
6  #include "atanSegment.h"
7  #include "text.h"
8  #include "myCli.h"
9
10 #include "shape.h"
11 #include "point.h"
12 #include "pentagram.h"
13 #include "colors.h"
14 #include "pentagramText.h"
15
16
17 std::string outPath(pavel::Shape& shape) {
18     std::ostringstream os;
19     os.precision(3);
20     os << "[";
21     for (auto &p : shape.getPath()) {
22         os << p << " ";
23     }
24     os << "]";
25     return os.str();
26 }
27
28 void doStuffWithShape(pavel::Shape & shape, std::ostream& os) {
29     os << BLUE << "Starting_transformation_sequence" << RESET << std::endl;
30     os << offset(1) << "Path" << outPath(shape) << std::endl;
31     auto move = pavel::Point {10, 20};
32     os << offset(1) << GREEN << "Moving_to_" << move << RESET << std::endl;
33     shape.move(move);
34     os << offset(1) << "Path" << outPath(shape) << std::endl;
35
36     double angle = 45.0 / 180.0 * M_PI;
37     os << offset(2) << GREEN << "Rotating_at_45_degrees" << RESET << std::endl;
38     shape.rotate(angle);
39     os << offset(2) << "Path" << outPath(shape) << std::endl;
40
41     double scale = 12.5;
42     os << offset(3) << GREEN << "Scaling_" << scale << "_times" << RESET << std::endl;
43     shape.scale(scale);
44     os << offset(3) << "Path" << outPath(shape) << std::endl;
45
46     os << offset(3) << GREEN << "Scaling_back" << RESET << std::endl;
47     shape.scale(1 / scale);
48     os << offset(3) << "Path" << outPath(shape) << std::endl;
49
50     os << offset(2) << GREEN << "Rotating_back" << RESET << std::endl;
51     shape.rotate(-angle);
52     os << offset(2) << "Path" << outPath(shape) << std::endl;
53
54     os << offset(1) << GREEN << "Moving_back" << RESET << std::endl;
55     shape.move(-move);
56     os << offset(1) << "Path" << outPath(shape) << std::endl;
```

```

57     os << BLUE << "Transformation_sequence_complete" << std::endl;
58 }
59
60 int main(int argc, char *argv[])
61 {
62     std::cout.precision(3);
63     box("Korytov_Pavel_6304_LR1_Part1", std::cout, RED);
64     std::vector<std::shared_ptr<pavel::Shape>> arr {
65         std::make_shared<pavel::Pentagram>(pavel::Pentagram(pavel::Point {0, 0},
66             10)),
67         std::make_shared<pavel::AtanSegment>(pavel::AtanSegment(pavel::Point{1, 1},
68             pavel::Point{10, 10}, 5)),
69         std::make_shared<pavel::Text>(pavel::Text("Hello_world", pavel::Point {-5,
70             -5}, pavel::Point {5, 5})),
71         std::make_shared<pavel::PentagramText>(pavel::PentagramText("Text_with_
72             pentagram", pavel::Point{1, 2}, 10))
73     };
74     for (auto &ptr: arr) {
75         std::ostringstream s;
76         s << *ptr.get();
77         box(s.str(), std::cout, YELLOW);
78         doStuffWithShape(*ptr.get(), std::cout);
79     }
80 }

```

ПРИЛОЖЕНИЕ П

Код colors.h

```
1  #pragma once
2
3  #define RESET    "\033[0m"
4  #define BLACK    "\033[30m"      /* Black */
5  #define RED      "\033[31m"      /* Red */
6  #define GREEN    "\033[32m"      /* Green */
7  #define YELLOW   "\033[33m"      /* Yellow */
8  #define BLUE     "\033[34m"      /* Blue */
9  #define MAGENTA  "\033[35m"      /* Magenta */
10 #define CYAN     "\033[36m"      /* Cyan */
11 #define WHITE    "\033[37m"      /* White */
12 #define BOLDBLACK "\033[1m\033[30m" /* Bold Black */
13 #define BOLDRED   "\033[1m\033[31m" /* Bold Red */
14 #define BOLDGREEN "\033[1m\033[32m" /* Bold Green */
15 #define BOLDYELLOW "\033[1m\033[33m" /* Bold Yellow */
16 #define BOLDBLUE  "\033[1m\033[34m" /* Bold Blue */
17 #define BOLDMAGENTA "\033[1m\033[35m" /* Bold Magenta */
18 #define BOLDCYAN  "\033[1m\033[36m" /* Bold Cyan */
19 #define BOLDWHITE "\033[1m\033[37m" /* Bold White */
```

ПРИЛОЖЕНИЕ Р

Код myCli.h

```
1  #pragma once
2  #include <string>
3
4  void box(const std::string& string, std::ostream& os,
5          const std::string& boxColor = "\033[0m",
6          const std::string& textColor = "\033[0m");
7
8  std::string offset(int offset);
```

ПРИЛОЖЕНИЕ С

Код myCli.cpp

```
1  #include "myCli.h"
2  #include "colors.h"
3  #include <iostream>
4  #include <sstream>
5
6  #define OFFSET "    "
7
8  void box(const std::string& string, std::ostream& os,
9          const std::string& boxColor,
10         const std::string& textColor) {
11     os << boxColor << "┌";
12     for (int i = 0; i < string.length(); i++) {
13         os << "-";
14     }
15     os << "┐" << std::endl;
16     os << "| " << textColor << string << boxColor << "| " << std::endl;
17     os << "└";
18     for (int i = 0; i < string.length(); i++) {
19         os << "-";
20     }
21     os << "┘" << std::endl << RESET;
22 }
23
24 std::string offset(int offset) {
25     std::string str;
26     for (int i = 0; i < offset; i++) {
27         str += OFFSET;
28     }
29     return str;
30 }
```