

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

**ОТЧЁТ
по лабораторной работе №1
по дисциплине «Конструирование ПО»
Тема: Программирование контейнерных классов**

Студент гр. 6304

Преподаватель

Корытов П.В.

Спицын А.В.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

1	Постановка задачи	2
1.1	Цель работы	2
1.2	Формулировка задания	2
1.3	Индивидуальное задание	3
2	Ход работы	4
2.1	Создание классов фигур	4
2.2	Проверка созданных классов	6
2.3	Применение стандартной библиотеки STL	9
2.4	Реализация нового шаблона контейнера	9
3	Выводы	13
	Список литературы	14
	Приложения	15

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Цель работы

Изучение создания полиморфной иерархии классов и разрешения ромбовидного наследования на языке C++.

Изучение контейнеров стандартной библиотеки STL C++. Создание собственного контейнера с внешним итератором.

Исследования возможностей отладки и профилировки C++.

1.2. Формулировка задания

1. Разработка программ.

1.1. 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.

1.2. 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”.

1.3. Исследование программы при помощи Profiler.

- Изучить возможности оптимизации программы в интегрированной среде, в отчете перечислить и объяснить параметры (опции), влияющие на оптимизацию.

- Построить несколько вариантов, отличающихся способом оптимизации, проанализировать время работы и объем памяти полученных вариантов. С помощью Profiler определить наиболее долго выполнявшиеся функции.
- Изменить текст `main` так, чтобы выполнялись все участки программы.

2. Применение стандартной библиотеки STL.

2.1. 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.

2.2. 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.

1.3. Индивидуальное задание

- Фигуры — прямоугольник, кусок арктангенса, текст, текст в прямоугольнике.
- Контейнер — хэш-таблица на базе списка.

2. ХОД РАБОТЫ

Использованное ПО:

1. **JetBrains CLion** — IDE для C/C++ [4];
2. **Valgrind** — проверка утечек памяти;
3. **Google Test** — юнит-тесты для C++;
4. **Jest** — профилировщик для C++;
5. \LaTeX , **neovim** — сборка и написание отчёта [5]
6. **BibLaTeX** — система управления библиографией

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. [8]

3. Написаны классы **Pentagram**, **AtanSegment**, **Text**, **PentagramText**. Класс **PentagramText** наследуется от **Pentagram** и **Text**. UML-диаграмма классов представлена на рисунке 1.

Исходные коды представлены в приложениях в соответствующих файлах.

В процессе использованы возможности C++11/C++14/C++17, такие как `[[nodiscard]]`, лямбда-выражения, выводение типов (`auto`), `shared_ptr` и т.п. [7] [3] [6]

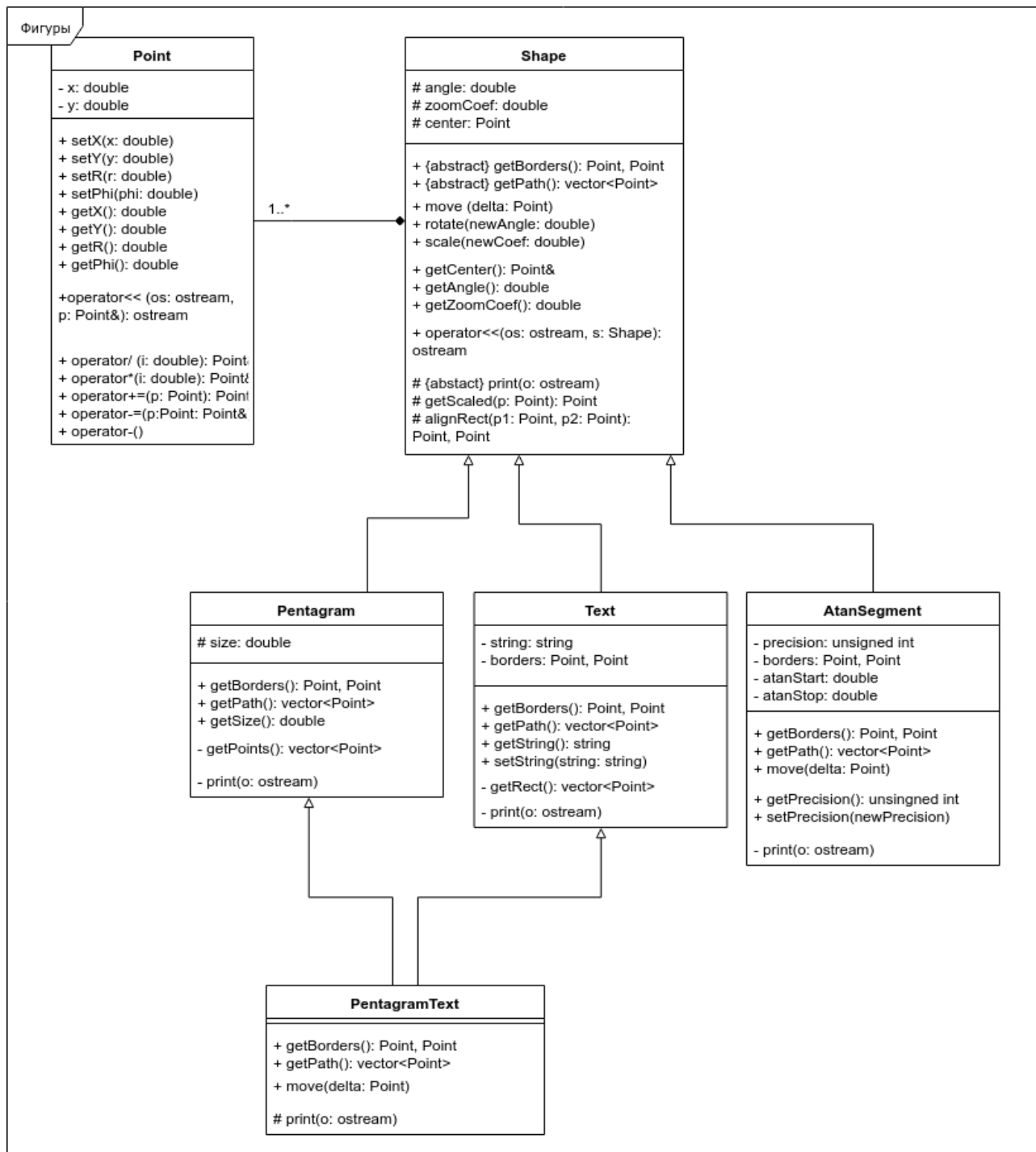
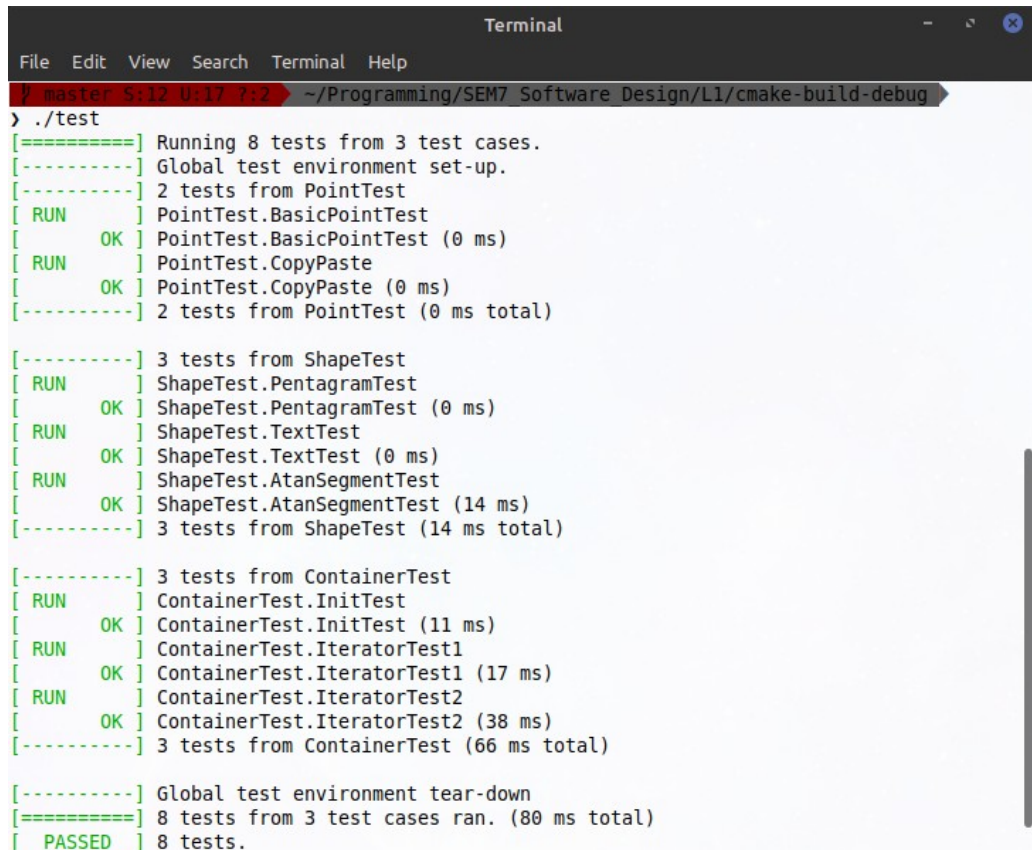


Рисунок 1 – UML-диаграмма классов

2.2. Проверка созданных классов

1. Для сборки использована система CMake [2].
2. 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

3. 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.

4. Для отладки использована обертка над gdb, встроенная в CLion и Valgrind для контроля над утечками памяти. [4] Приведены скриншоты, сделанные в ходе отладки:

- Рисунок 4 — остановка на breakpoint'е
- Рисунок 5 — оценка выражения
- Рисунок 6 — локализация утечки памяти

```
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

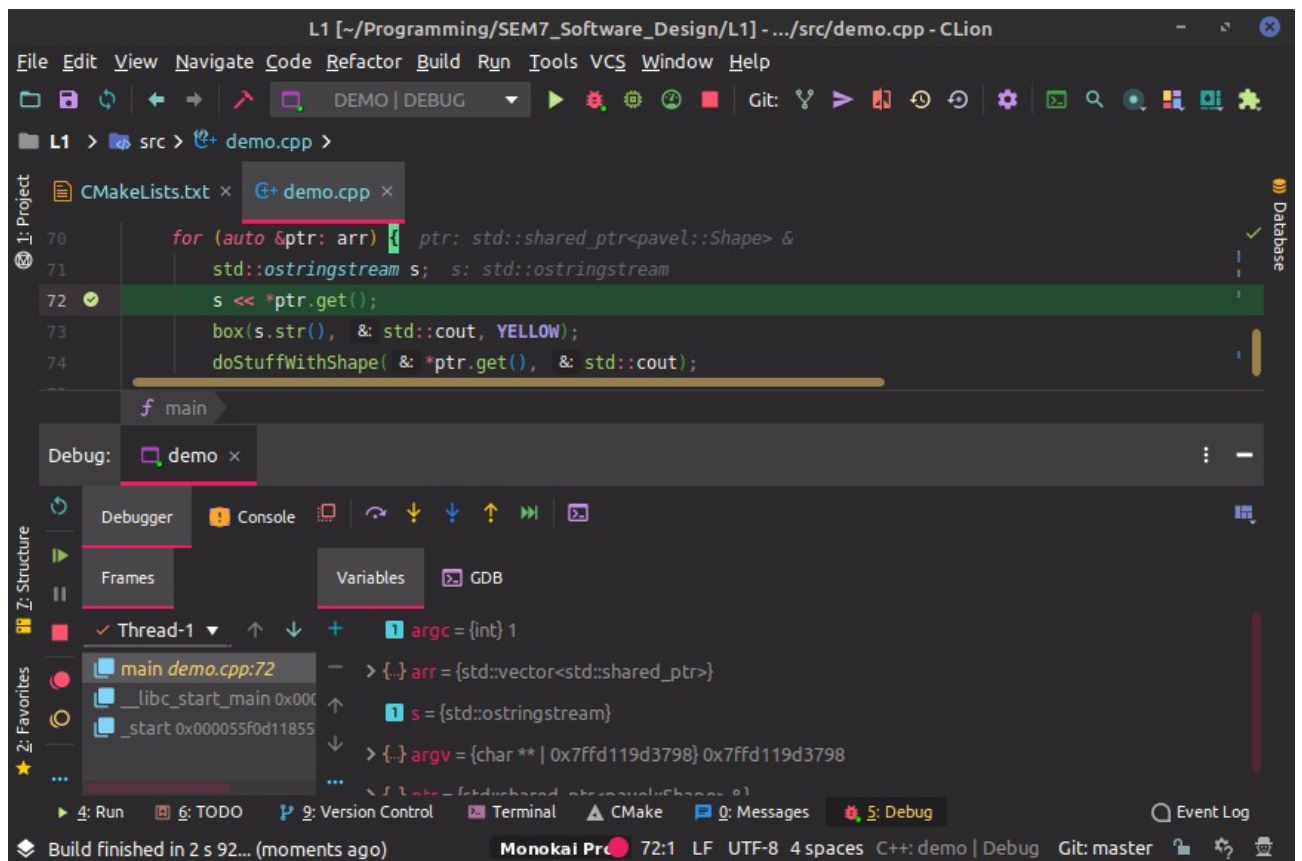


Рисунок 4 – Остановка на breakpoint'е

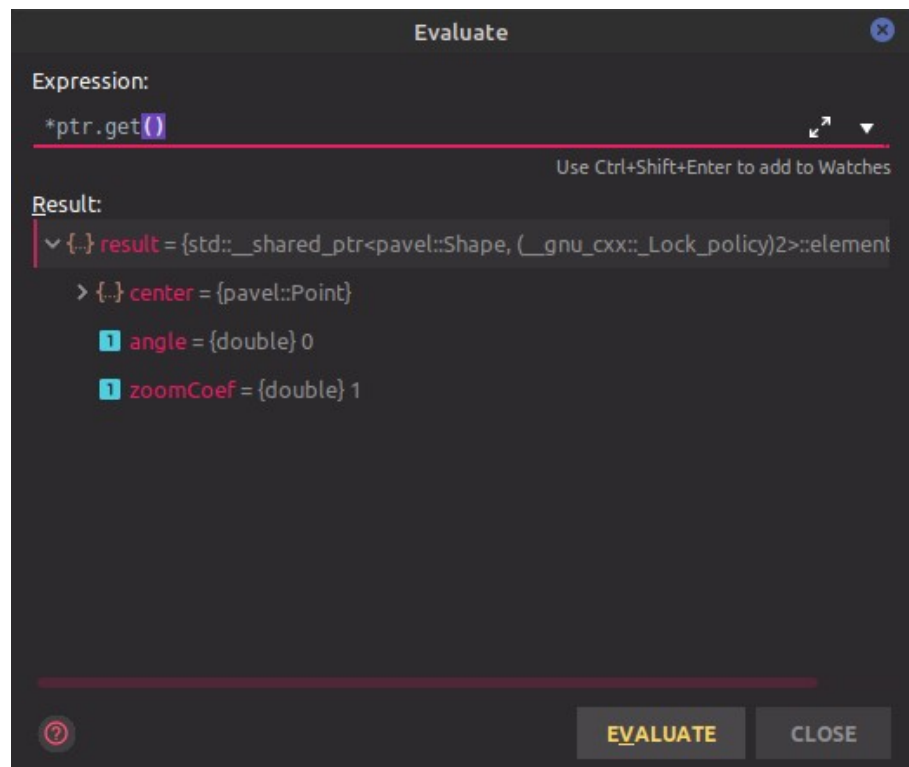


Рисунок 5 – Оценка значения выражения

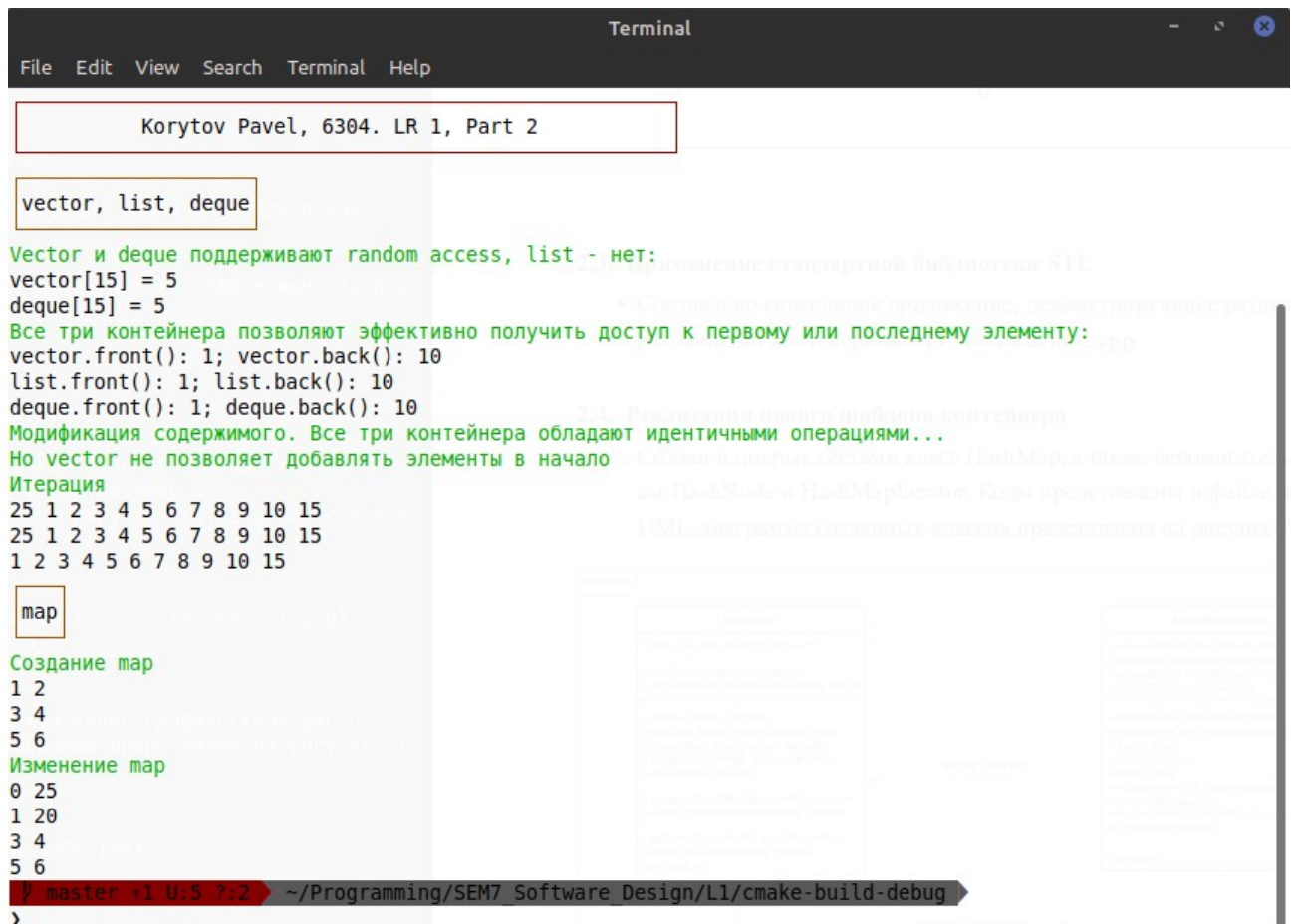


Рисунок 6 – Valgrind обнаружил утечку памяти

2.3. Применение стандартной библиотеки STL

- Составлено консольное приложение, демонстрирующее различия между различными контейнерами STL. Код в demo2.cpp.

Установлено, что интерфейсы контейнеров очень похожи, эффективность различных операций в различается. [9] [1]



```
Terminal
File Edit View Search Terminal Help

Korytov Pavel, 6304. LR 1, Part 2

vector, list, deque

Vector и deque поддерживают random access, list - нет:
vector[15] = 5
deque[15] = 5
Все три контейнера позволяют эффективно получить доступ к первому или последнему элементу:
vector.front(): 1; vector.back(): 10
list.front(): 1; list.back(): 10
deque.front(): 1; deque.back(): 10
Модификация содержимого. Все три контейнера обладают идентичными операциями...
Но vector не позволяет добавлять элементы в начало
Итерация
25 1 2 3 4 5 6 7 8 9 10 15
25 1 2 3 4 5 6 7 8 9 10 15
1 2 3 4 5 6 7 8 9 10 15

map

Создание map
1 2
3 4
5 6
Изменение map
0 25
1 20
3 4
5 6

~/Programming/SEM7 Software Design/L1/cmake-build-debug
>
```

Рисунок 7 – Результаты запуска демонстрационной программы

Некоторые различия между vector, list и deque указаны в таблице 1.

2.4. Реализация нового шаблона контейнера

1. Создан и покрыт тестами класс HashMap, а также вспомогательные классы HashNode и HashMapIterator. Коды представлены в файле hashMap.h. UML-диаграмма созданных классов представлена на рисунке 8
- Создаваемый контейнер — хэш-таблица на основе списка. На списке основано разрешение коллизий; если у элементов совпадает хэш, они хранятся в односвязном списке.

При конструировании хэш-таблицы, если задается значение ключа, непри-

Таблица 1. Различия между контейнерами

Метод	<i>std::vector</i>	<i>std::list</i>	<i>std::deque</i>	<i>std::map</i>	<i>std::queue</i>	<i>std::stack</i>	<i>std::priority_queue</i>
Доступ по индексу	+	—	+	+	—	—	—
Индекс любого типа	—	—	—	+	—	—	—
Доступ с начала	+	+	+	—	+	—	—
Доступ с конца	+	+	+	—	+	+	+/-
Добавление элементов в конец	+	+	+	—	+	+	+/-
Добавление элементов в начало	—	+	+	—	—	—	—
Удаление элементов с начала	—	+	+	—	+	—	—
Удаление элементов с конца	+	+	+	—	—	+	+/-
Изменение элементов по индексу	+	—	+	+	—	—	—

водимое `static_cast` с `unsigned int`, нужно задать хэш-функцию и функцию сравнения ключей. При это хэш-функция должна возвращать `unsigned int` не больше, чем `SIZE` в файле с `hashMap`.

2. Создано приложение, заполняющее созданный контейнер графическими фигурами. Код в `demo3.cpp`. Запуск демонстрационной программы представлен на рисунке 9
3. Изучение профилировщика

Использована обёртка над профилировщиком `Perf`, встроенная в `CLion`. [4]
На рисунке 10 — дерево вызовов, на 11 — вызываемые методы.

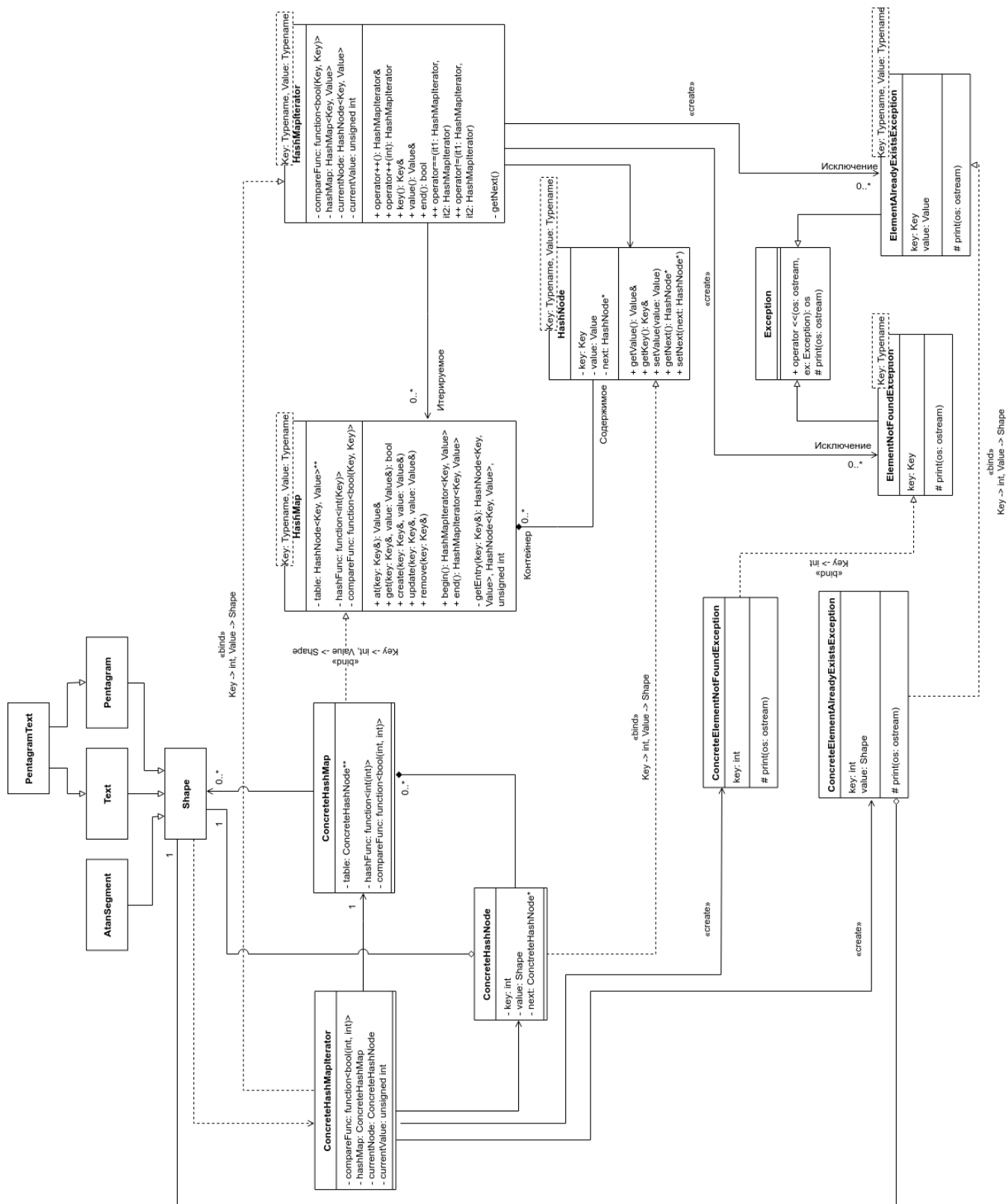


Рисунок 8 – UML-диаграмма классов контейнера

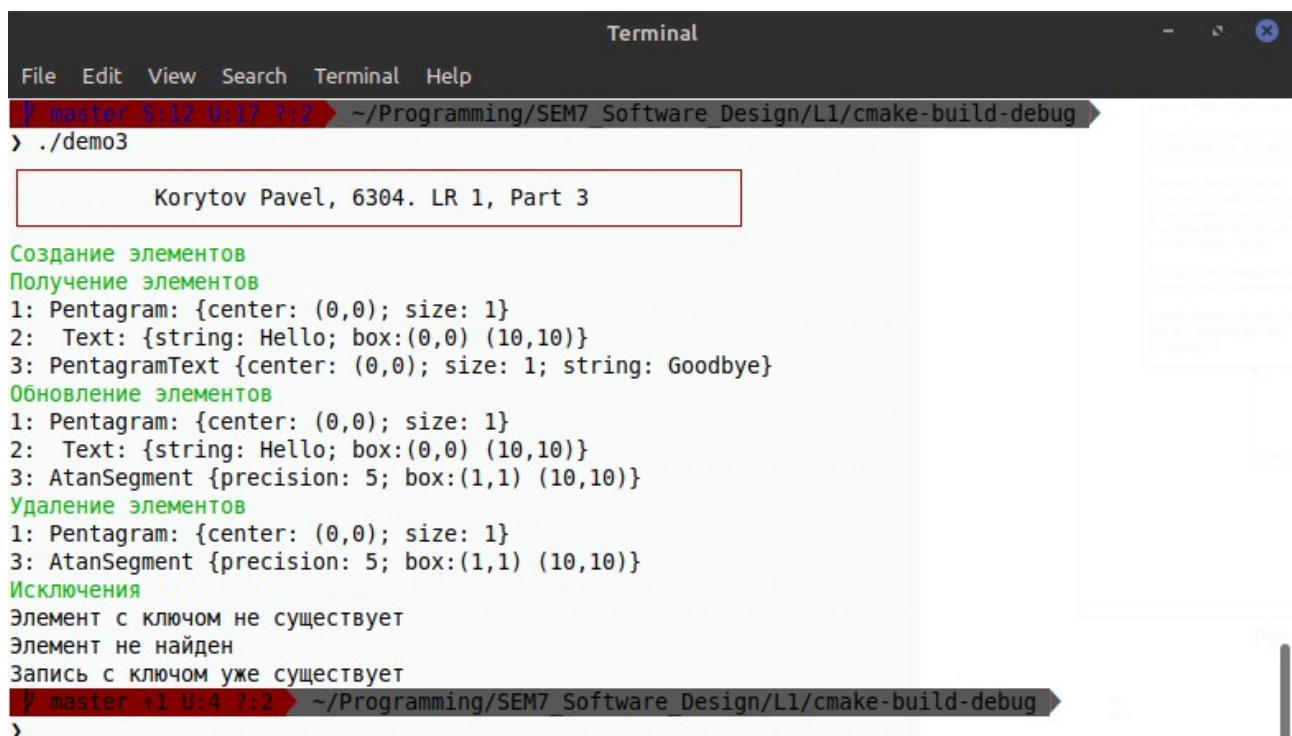


Рисунок 9 – Запуск демонстрации HashMap

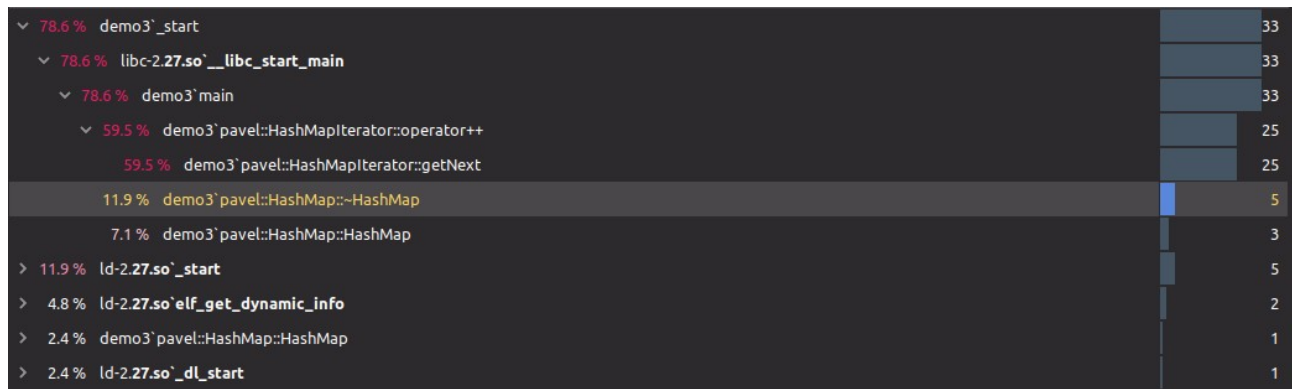


Рисунок 10 – Дерево вызовов

Method	Samples
demo3`_start	33
libc-2.27.so`__libc_start_main	33
demo3`main	33
demo3`pavel::HashMapIterator::operator++	25
demo3`pavel::HashMapIterator::getNext	25
demo3`pavel::HashMap::~HashMap	5

Рисунок 11 – Список вызываемых методов

3. ВЫВОДЫ

Протестированы и изучены классы контейнеров стандартной библиотеки STL.

Написаны классы графических объектов, поддерживающие полиморфизм. Изучено виртуальное наследование для разрешения проблем, связанных с ромбовидным наследованием.

Написан класс-контейнер — хэш-таблица на базе списка.

Исследованы возможности тестирования, отладки и профилировки программ на C++.

СПИСОК ЛИТЕРАТУРЫ

1. C++ Reference [Электронный ресурс]. — URL: <https://en.cppreference.com/w/> (дата обр. 16.10.2019).
2. CMake Refence Documentation [Электронный ресурс]. — URL: <https://cmake.org/cmake/help/v3.15/> (дата обр. 16.10.2019).
3. GitHub. AnthonyCalandra/modern-cpp-features [Электронный ресурс]. — URL: <https://github.com/AnthonyCalandra/modern-cpp-features> (дата обр. 16.10.2019).
4. JetBrains. Clion Learning Center [Электронный ресурс]. — URL: <https://www.jetbrains.com/clion/learning-center/> (дата обр. 16.10.2019).
5. LaTeX - Wikibooks [Электронный ресурс]. — URL: <https://en.wikibooks.org/wiki/LaTeX> (дата обр. 16.10.2019).
6. *Stroustrup B.* The C++ Programming Language [Текст] : пер. с англ. — Бостон, Массачусетс, США : Addison-Wesley, 2014. — 1363 с.
7. Wikibooks. C++ Programming [Электронный ресурс]. — URL: https://en.wikibooks.org/wiki/C++_Programming (дата обр. 16.10.2019).
8. Wikibooks. More C++ Idioms [Электронный ресурс]. — URL: https://en.wikibooks.org/wiki/More_C++_Idioms (дата обр. 16.10.2019).
9. *Шлее М.* Профессиональное программирование на C++ [Текст]. — СПб. : БЖХ-Петербург, 2018. — 1073 с.

ПРИЛОЖЕНИЯ

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  #include "exception.h"
9
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 }
```


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

ПРИЛОЖЕНИЕ В

Код point.h

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

ПРИЛОЖЕНИЕ Г

Код point.cpp

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

```

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

```

ПРИЛОЖЕНИЕ Д

Код pentagram.h

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

ПРИЛОЖЕНИЕ Е

Код pentagram.cpp

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

ПРИЛОЖЕНИЕ Ж

Код atanSegment.h

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

ПРИЛОЖЕНИЕ 3

Код atanSegment.cpp

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


ПРИЛОЖЕНИЕ И

Код text.h

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

ПРИЛОЖЕНИЕ К

Код text.cpp

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

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

ПРИЛОЖЕНИЕ Л

Код pentagramText.h

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

ПРИЛОЖЕНИЕ М

Код pentagramText.cpp

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

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     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     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(Shape& figure) {
56     auto [p2, p1] = figure.getBorders();
```

```

57     for (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(Shape& figure) {
67     ASSERT_NO_FATAL_FAILURE(checkBorders(figure));
68
69     figure.move(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 = Point{2, 3};
82     auto fig = Pentagon(center, 10);
83     ASSERT_NO_FATAL_FAILURE(checkBorders(fig));
84
85     auto [ap1, ap2] = fig.getBorders();
86     fig.move(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 + Point(2, 5));
102 }
103
104 TEST(ShapeTest, TextTest){
105     auto text = Text("Hello",
106                     Point(0, 0),
107                     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 = AtanSegment(Point{0, 0}, Point{10, 10}, 0);
120     ASSERT_NO_FATAL_FAILURE(checkBorders(seg));
121     ASSERT_NO_FATAL_FAILURE(standartMoveSequence(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 = 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 = HashMap<int, double>();
151     m.create(1, 2);
152     m.create(2, 3);
153
154     auto iter = 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 = 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(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(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 = 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<Shape>> arr {
65         std::make_shared<Pentagram>(Pentagram(Point {0, 0}, 10)),
66         std::make_shared<AtanSegment>(AtanSegment(Point{1, 1}, Point{10, 10}, 5)),
67         std::make_shared<Text>(Text("Hello_world", Point {-5, -5}, Point {5, 5})),
68         std::make_shared<PentagramText>(PentagramText("Text_with_pentagram",
69             Point{1, 2}, 10))
70     };
71     for (auto &ptr: arr) {
72         std::ostringstream s;
73         s << *ptr.get();
74         box(s.str(), std::cout, YELLOW);
75         doStuffWithShape(*ptr.get(), std::cout);
76     }
77 }

```

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 BOLD CYAN "\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 }
```

ПРИЛОЖЕНИЕ Т

Код demo2.cpp

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <list>
5  #include <deque>
6  #include <map>
7
8  #include "myCli.h"
9  #include "colors.h"
10
11 using std::cout, std::endl;
12
13 int main(int argc, char *argv[])
14 {
15     box("Korytov_Pavel, 6304_LR1_Part2", std::cout, RED);
16     std::initializer_list<int> init = {
17         1, 2, 3, 4, 5, 6, 7, 8, 9, 10
18     };
19     box("vector, list, deque", std::cout, YELLOW);
20     auto vector = std::vector<int>(init);
21     auto list = std::list<int>(init);
22     auto deque = std::deque<int>(init);
23
24     cout << GREEN << "Vector и deque поддерживают random access, list — нет:" << RESET
25         << endl;
26     cout << "vector[15] = " << vector[4] << endl;
27     cout << "deque[15] = " << deque[4] << endl;
28
29     cout << GREEN << "Все три контейнера позволяют эффективно получить доступ к первому,
30         или последнему элементу:" << RESET << endl;
31     cout << "vector.front(): " << vector.front() << "; vector.back(): " <<
32         vector.back() << endl;
33     cout << "list.front(): " << list.front() << "; list.back(): " << list.back() <<
34         endl;
35     cout << "deque.front(): " << deque.front() << "; deque.back(): " << deque.back() <<
36         endl;
37
38     cout << GREEN << "Модификация содержимого. Все три контейнера обладают идентичными
39         операциями..." << RESET << endl;
40     deque.push_back(15);
41     deque.push_front(25);
42     list.push_back(15);
43     list.push_front(25);
44     cout << GREEN << "Но vector не позволяет добавлять элементы в начало" << RESET <<
45         endl;
46     vector.push_back(15);
47
48     cout << GREEN << "Итерация" << RESET << endl;
49     for (auto & it : deque) {
50         cout << it << " ";
51     }
52     cout << endl;
53
54     for (auto & it : list) {
55         cout << it << " ";
56     }
57 }
```

```

50     cout << endl;
51
52     for (auto & it : vector) {
53         cout << it << " ";
54     }
55     cout << endl;
56
57     box("map", std::cout, YELLOW);
58     cout << GREEN << "Создание_map" << RESET << endl;
59     auto map = std::map<int, int> {{1, 2}, {3, 4}, {5, 6}};
60     for (auto & it : map) {
61         std::cout << it.first << " " << it.second << std::endl;
62     }
63
64     cout << GREEN << "Изменение_map" << RESET << endl;
65     map[1] = 20;
66     map[0] = 25;
67
68     for (auto & it : map) {
69         std::cout << it.first << " " << it.second << std::endl;
70     }
71 }

```


ПРИЛОЖЕНИЕ У

Код hashMap.h

```
1  #pragma once
2
3  #include "exception.h"
4  #include <algorithm>
5  #include <iostream>
6  #include <functional>
7  #include <string>
8
9  #define SIZE 2000000
10
11 template<typename Key>
12 unsigned int defaultHash(Key key) {
13     return static_cast<unsigned int>(key) % SIZE;
14 }
15
16 template<typename Key>
17 bool defaultCompare(const Key& v1, const Key& v2) {
18     return v1 == v2;
19 }
20
21 template <typename Key, typename Value>
22 class HashMap;
23
24 template <typename Key, typename Value>
25 class HashNode {
26 public:
27     explicit HashNode(const Key& key, const Value& value): key(key), value(value) {}
28
29     HashNode (HashNode& node): key(node.key), value(node.value) {}
30
31     friend void swap(HashNode& a, HashNode& b){
32         std::swap(a.key, b.key);
33         std::swap(a.value, b.value);
34     }
35
36     HashNode& operator=(const HashNode& other) {
37         swap(*this, other);
38         return *this;
39     }
40
41     HashNode& operator=(HashNode&& other) noexcept {
42         swap(*this, other);
43     }
44
45     [[nodiscard]] const Value& getValue() {
46         return value;
47     }
48
49     void setValue(Value newValue) {
50         value = newValue;
51     }
52
53     [[nodiscard]] const Key& getKey() {
54         return key;
55     }
56 }
```

```

57     [[nodiscard]] HashNode *getNext() const {
58         return next;
59     }
60
61     void setNext(HashNode *newNext) {
62         next = newNext;
63     }
64
65 private:
66     Key key;
67     Value value;
68     HashNode* next = nullptr;
69 };
70
71 template <typename Key, typename Value>
72 class HashMapIterator {
73 public:
74     explicit HashMapIterator(const HashMap<Key, Value>& hashMap,
75                             HashNode<Key, Value>* node = nullptr)
76         :hashMap(hashMap), currentNode(node), compareFunc(hashMap.compareFunc) {
77         if (node == nullptr) {
78             getNext();
79         } else {
80             currentValue = hashMap.hashFunc(node->getKey());
81         }
82     }
83
84     HashMapIterator(HashMapIterator& it)
85         : HashMapIterator(it.hashMap, it.currentNode)
86     {
87         compareFunc = it.compareFunc;
88     }
89
90     HashMapIterator&operator++() {
91         getNext();
92         return *this;
93     }
94
95     HashMapIterator operator++(int) {
96         auto ret = HashMapIterator(*this);
97         getNext();
98         return ret;
99     }
100
101     const Key& key() {
102         return currentNode->getKey();
103     }
104
105     const Value& value() {
106         return currentNode->getValue();
107     }
108
109     bool end(){
110         return currentValue >= SIZE - 1;
111     }
112
113     friend bool operator==(const HashMapIterator<Key, Value>& it1,
114                           const HashMapIterator<Key, Value>& it2) {
115         return it1.compareFunc(it1.currentNode->getKey(), it2.currentNode->getKey());
116     }

```

```

117
118     friend bool operator!=(const HashMapIterator<Key, Value>& it1,
119                           const HashMapIterator<Key, Value>& it2) {
120         return !it1.compareFunc(it1.currentNode->getKey(), it2.currentNode->getKey());
121     }
122 private:
123     void getNext() {
124         if (currentNode != nullptr) {
125             currentNode = currentNode->getNext();
126         }
127         while (currentNode == nullptr && (currentValue < SIZE - 1)) {
128             currentValue++;
129             currentNode = hashMap.table[currentValue];
130         }
131     }
132
133     std::function<int(Key, Key)> compareFunc;
134     const HashMap<Key, Value>& hashMap;
135     HashNode<Key, Value>* currentNode = nullptr;
136     unsigned int currentValue = 0;
137 };
138
139
140 template<typename Key, typename Value>
141 class HashMap {
142     template <typename K, typename V>
143     friend class HashMapIterator;
144 public:
145     explicit HashMap(
146         const std::function<unsigned int(Key)>& hash = defaultHash<Key>,
147         const std::function<int(Key, Key)>& compare = defaultCompare<Key>) {
148         : hashFunc(hash), compareFunc(compare) {
149         table = new HashNode<Key, Value>* [SIZE]();
150     }
151
152     ~HashMap() {
153         for (unsigned int i = 0; i < SIZE; i++) {
154             HashNode<Key, Value> *entry = table[i];
155             while (entry != nullptr) {
156                 HashNode<Key, Value> *prev = entry;
157                 entry = entry->getNext();
158                 delete prev;
159             }
160         }
161         delete[] table;
162     }
163
164     const Value& at(const Key& key) {
165         unsigned int hashValue = hashFunc(key);
166         HashNode<Key, Value>* node = table[hashValue];
167
168         while (node != nullptr) {
169             if (compareFunc(node->getKey(), key)) {
170                 return node->getValue();
171             }
172             node = node->getNext();
173         }
174         throw ElementNotFoundException<Key>(key);
175     }
176

```

```

177     bool get(const Key& key, Value& value) {
178         try {
179             value = at(key);
180             return true;
181         } catch (ElementNotFoundException<Key>& ex) {
182             return false;
183         }
184     }
185
186     void create(const Key& key, const Value& value) {
187         auto [prev, entry, hashValue] = getEntry(key);
188
189         if (entry == nullptr) {
190             entry = new HashNode<Key, Value>(key, value);
191             if (prev == nullptr) {
192                 table[hashValue] = entry;
193             } else {
194                 prev->setNext(entry);
195             }
196         } else {
197             throw ElementAlreadyExistsException<Key, Value>(key, entry->getValue());
198         }
199     }
200
201     void update(const Key& key, const Value& value) {
202         auto [prev, entry, hashValue] = getEntry(key);
203
204         if (entry == nullptr) {
205             entry = new HashNode<Key, Value>(key, value);
206             if (prev == nullptr) {
207                 table[hashValue] = entry;
208             } else {
209                 prev->setNext(entry);
210             }
211         } else {
212             entry->setValue(value);
213         }
214     }
215
216     void remove(const Key& key) {
217         auto [prev, entry, hashValue] = getEntry(key);
218
219         if (entry == nullptr) {
220             throw ElementNotFoundException<Key>(key);
221         } else {
222             if (prev == nullptr) {
223                 table[hashValue] = entry->getNext();
224             } else {
225                 prev->setNext(entry->getNext());
226             }
227             delete entry;
228         }
229     }
230
231     HashMapIterator<Key, Value> begin(){
232         return HashMapIterator<Key, Value>(*this);
233     }
234
235     HashMapIterator<Key, Value> end(){
236         unsigned int lastHash;

```

```

237         for (lastHash = SIZE - 1; table[lastHash] == nullptr && lastHash > 0;
238               lastHash--);
239
240         if (lastHash > 0) {
241             HashNode<Key, Value>* node = table[lastHash];
242             while (node->getNext() != nullptr) {
243                 node = node->getNext();
244             }
245             return HashMapIterator<Key, Value>(*this, node);
246         }
247
248     private:
249         std::tuple<HashNode<Key, Value>*, HashNode<Key, Value>*,
250                 unsigned int> getEntry(const Key& key) {
251             unsigned int hashValue = hashFunc(key);
252
253             HashNode<Key, Value> *entry = table[hashValue];
254             HashNode<Key, Value>* prev = nullptr;
255             while (entry != nullptr && !compareFunc(entry->getKey(), key)) {
256                 prev = entry;
257                 entry = entry->getNext();
258             }
259             return std::make_tuple(prev, entry, hashValue);
260         }
261
262         HashNode<Key, Value> **table;
263         std::function<unsigned int(Key)> hashFunc;
264         std::function<int(Key, Key)> compareFunc;
265     };

```

ПРИЛОЖЕНИЕ Ф

Код demo3.cpp

```
1  #include "shape.h"
2  #include "hashMap.h"
3  #include "pentagram.h"
4  #include "atanSegment.h"
5  #include "text.h"
6  #include "pentagramText.h"
7  #include "point.h"
8  #include "colors.h"
9  #include "myCli.h"
10 #include "exception.h"
11
12 #include <iostream>
13 #include <memory>
14
15 int main(int argc, char *argv[]) {
16     box("Korytov_Pavel, 6304.LR, Part_3", std::cout, RED);
17     std::cout.precision(3);
18     auto map = HashMap<int, std::shared_ptr<Shape>>();
19     std::cout << GREEN << "Создание_элементов" << RESET << std::endl;
20     map.create(1, std::make_shared<Pentagram>(Pentagram()));
21     map.create(2, std::make_shared<Text>(Text("Hello", Point{0, 0}, Point{10, 10})));
22     map.create(3, std::make_shared<PentagramText>(PentagramText("Goodbye")));
23
24     std::cout << GREEN << "Получение_элементов" << RESET << std::endl;
25     for (auto it = map.begin(); !it.end(); ++it) {
26         std::cout << it.key() << ":_ " << *it.value() << std::endl;
27     }
28
29     std::cout << GREEN << "Обновление_элементов" << RESET << std::endl;
30     map.update(3, std::make_shared<AtanSegment>(AtanSegment(Point{1, 1}, Point{10, 10},
31     5)));
32     for (auto it = map.begin(); !it.end(); ++it) {
33         std::cout << it.key() << ":_ " << *it.value() << std::endl;
34     }
35
36     std::cout << GREEN << "Удаление_элементов" << RESET << std::endl;
37     map.remove(2);
38
39     for (auto it = map.begin(); !it.end(); ++it) {
40         std::cout << it.key() << ":_ " << *it.value() << std::endl;
41     }
42
43     std::cout << GREEN << "Исключения" << RESET << std::endl;
44     try{
45         map.remove(2);
46     } catch (Exception& ex) {
47         std::cout << "Поймано_исключение:_ " << ex << std::endl;
48     }
49     try{
50         auto a = map.at(2);
51     } catch (Exception& ex) {
52         std::cout << "Поймано_исключение:_ " << ex << std::endl;
53     }
54     try{
55         map.create(3, std::make_shared<Pentagram>(Pentagram()));
56     } catch (Exception& ex) {
```

```
56         std::cout << "Поймано_исключение:_" << ex << std::endl;
57     }
58
59 }
```

ПРИЛОЖЕНИЕ X

Код exception.h

```
1  # pragma once
2
3  class Exception {
4  public:
5      friend std::ostream &operator<<(std::ostream &os, const Exception &ex);
6      virtual ~Exception() = default;
7
8  protected:
9      virtual void print(std::ostream &o) const = 0;
10 };
11
12 inline std::ostream &operator<<(std::ostream &os, const Exception &ex) {
13     ex.print(os);
14     return os;
15 }
16
17 template <typename Key>
18 class ElementNotFoundException: public Exception {
19 public:
20     explicit ElementNotFoundException(Key key) : key(key) {}
21
22 protected:
23     void print(std::ostream &o) const override {
24         o << "ElementNotFoundException_{Key:}" << key << " ";
25     }
26
27 private:
28     Key key;
29 };
30
31 template <typename Key, typename Value>
32 class ElementAlreadyExistsException: public Exception {
33 public:
34     ElementAlreadyExistsException(Key key, const Value &element) : key(key),
35         value(element) {}
36
37 protected:
38     void print(std::ostream &o) const override {
39         o << "ElementAlreadyExistsException_{Key:}" << key << ", Value:" << value <<
40             " ";
41     }
42
43 private:
44     Key key;
45     Value value;
46 };
```