**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «ПОДДЕРЖКА ОБРАБОТКИ ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ»**

Студенты гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Поляков А.И.

Половникова А.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Колинько П.Г.

Санкт-Петербург

2025

**Оглавление**

[**Цель работы.** 3](#_Toc193995504)

[***Задание*** 3](#_Toc193995505)

[**Демонстрация работы исключений** 3](#_Toc193995506)

[**Обоснование набор и вид классов для фиксации особых ситуаций** 5](#_Toc193995507)

[**Вывод** 8](#_Toc193995508)

[**Список использованных источников** 8](#_Toc193995509)

[**Приложение.** 8](#_Toc193995510)

[*Исходные тексты программ* 8](#_Toc193995511)

[*файл “Parallelogram.h”* 15](#_Toc193995512)

[*файл “Parallelogram.cpp”* 15](#_Toc193995513)

# **Цель работы.**

Поддержка и обработка исключительных ситуаций в языке программирования C++.

***Задание***

Переработать программу работы с библиотекой фигур, дополнив её механизмом контроля исключительных ситуаций. Например, возможно выявление следующих ошибок:

— непопадание точки на экран;

— некорректные параметры при формировании фигуры;

— нехватка места на экране для размещения фигуры в одной из позиций (исходной, повёрнутой, отражённой, перемещённой);

— повторный поворот/отражение уже повёрнутой/отражённой фигуры и др.;

— присоединение несимметричной фигуры к картинке неправильной стороной.

Нужно реализовать генерацию выявление всех возможных ошибок в добавляемой на рисунок фигуре и продемонстрировать перехват не менее двух типов ошибок разного уровня сложности.

Если исключение генерируется в конструкторе фигуры, следует обеспечить исключение фигуры из списка для рисования или подмену её запасной фигурой — знаком ошибки.

Перехват исключения в той же функции, в которой оно возбуждено, не применяется. В этом случае, когда ошибку можно обработать в точке обнаружения, механизм исключений избыточен.

Организовать перехват исключений следует таким образом, чтобы искажения итоговой картинки были минимальны. Протестировать исключительные ситуации, результаты эксперимента поместить в отчёт.

# **Демонстрация работы исключений**

Специально создадим исключительную ситуацию в создании объекта параллелограмма (выход точки за пределы экрана) (Рис. 1) и попробуем развернуть фигуру горизонтально 2 раза (Рис. 2)



Рис. 1



Рис. 2

Рис. 1 – Исключение выхода точки за пределы экрана

Рис. 2 – Исключение повторного разворота

При возникновении ситуации с выходом точки за пределы экрана фигура не создается

При возникновении ситуации с повторным поворотом фигура не будет развернута еще раз

# **Обоснование набор и вид классов для фиксации особых ситуаций**

Обоснование набора и вида классов

Программа активно использует механизмы обработки исключений для контроля различных ситуаций, которые могут возникнуть во время выполнения. Основные классы исключений включают:

* ParallelogramException — предназначен для обработки ошибок, связанных с созданием объектов класса Parallelogram, таких как некорректные координаты или нарушение геометрических условий параллелограмма.
* std::out\_of\_range — стандартное исключение STL, используемое для контроля выхода за допустимые границы при работе с контейнерами или координатами.
* std::bad\_alloc — стандартное исключение C++, которое сигнализирует о нехватке памяти при выделении динамических объектов.
* std::runtime\_error — стандартное исключение для общего контроля ошибок, возникающих во время выполнения программы.
* std::exception — базовый класс для обработки исключений общего типа.
* Блок catch(...) — используется для перехвата неожиданных исключений, не относящихся к перечисленным выше.

Использование нескольких уровней обработки исключений позволяет сделать код более надёжным и минимизировать вероятность аварийного завершения программы.

***Место расположения операторов throw и блоков try-catch***

В программе исключения генерируются (операторы throw) и обрабатываются (блоки try-catch) в следующих местах:

**Генерация исключений (throw)**

Операторы throw предположительно находятся в конструкторах классов (например, Parallelogram), а также в методах, изменяющих состояние объектов:

* **Конструкторы фигур**:
  + Проверяет корректность переданных точек.
  + В случае ошибки выбрасывает ParallelogramException или std::out\_of\_range.
* **Методы работы с фигурами** (move, resize, flip\_vertically, flip\_horisontally):
  + Возможные исключения связаны с недопустимыми трансформациями объекта.
  + Например, если после изменения размеров фигура выходит за допустимые границы, может быть выброшено std::out\_of\_range.
* Места базовых действий (draw, move):
  + Исключения связаны с выходом за пределы или отрицательными значениями
* **Выделение памяти (new)**:

При нехватке памяти генерируется исключение std::bad\_alloc.

**Обработка исключений (try-catch)**

Блоки try-catch расположены так, чтобы обеспечить максимальную изоляцию потенциальных точек сбоя:

1. **Локальная обработка в отдельных секциях кода**:

Исключения, возникающие при создании объектов Parallelogram, обрабатываются сразу после new, что позволяет программе продолжить работу даже при неудачном создании некоторых объектов.

1. **Контрольная обработка на более высоком уровне**:

Если ошибка произошла при выполнении операций с фигурами (например, flip\_vertically, resize), она фиксируется в локальных try-catch, позволяя программе продолжить выполнение.

1. **Глобальная обработка в main**:

Общий try-catch перехватывает неожиданные исключения и предотвращает аварийное завершение.

***Безопасность кода***

1. **Изоляция ошибок**:

Каждый потенциально проблемный фрагмент кода заключён в отдельный try-catch, что позволяет локализовать ошибки и предотвратить их распространение.

1. **Обеспечение устойчивости программы**:
   * Даже если один объект не создался, программа продолжает работать с другими.
   * Если один метод (например, resize) выбрасывает исключение, другие методы продолжают выполняться.

# **Вывод**

В данной лабораторной работе мы получили практические навыки работы с исключениями, а также навыки в разборе стороннего кода.

# **Список использованных источников**

Колинько П. Г. Пользовательские структуры данных: Методические указания по дисциплине “Алгоритмы и структуры данных, часть 1”. - СПб.: СПбГЭТУ “ЛЭТИ”, 2023. - 64 c. (вып.2309).

# **Приложение.**

## *Исходные тексты программ*

#include <iostream>

#include "Rectangle.h"

#include "Line.h"

#include "Shape.h"

#include "Screen.h"

#include "ShapeControl.h"

#include "Parallelogram.h"

class h\_circle : public Rectangle, public Reflectable

{

protected:

Screen& screen;

public:

h\_circle(Point a, int rd) : Rectangle(Point(a.x - rd, a.y), Point(a.x + rd, a.y + rd \* 0.7 + 1)), screen(Screen::getInstance()) {}

void draw();

void flip\_horisontally() {};

void rotate\_right() {}

void rotate\_left() {}

};

void h\_circle::draw()

{

int x0 = (sw.x + ne.x) / 2, y0 = vert ? ne.y : sw.y;

int radius = (ne.x - sw.x) / 2;

int x = 0, y = radius, delta = 2 - 2 \* radius, error = 0;

while (y >= 0)

{

if (vert)

{

screen.put\_point(x0 + x, y0 - y \* 0.7);

screen.put\_point(x0 - x, y0 - y \* 0.7);

}

else

{

screen.put\_point(x0 + x, y0 + y \* 0.7);

screen.put\_point(x0 - x, y0 + y \* 0.7);

}

error = 2 \* (delta + y) - 1;

if (delta < 0 && error <= 0)

{

++x;

delta += 2 \* x + 1;

continue;

}

error = 2 \* (delta - x) - 1;

if (delta > 0 && error > 0)

{

--y;

delta += 1 - 2 \* y;

continue;

}

++x;

delta += 2 \* (x - y);

--y;

}

}

void down(Shape& p, const Shape& q)

{

Point n = q.south();

Point s = p.north();

p.move(n.x - s.x, n.y - s.y - 1);

}

class myshape : public Rectangle

{

int w, h;

Line l\_eye;

Line r\_eye;

Line mouth;

Screen& screen;

public:

myshape(Point, Point);

void draw();

void move(int, int);

void resize(double r) { Rectangle::resize(r); Rectangle::move(w \* (1 - r) \* 0.5, h \* (1 - r) \* 0.5); }

void rotate\_left() {}

void rotate\_right() {}

};

myshape::myshape(Point a, Point b) : Rectangle(a, b),

w(neast().x - swest().x + 1),

h(neast().y - swest().y + 1),

l\_eye(Point(swest().x + 2, swest().y + h \* 3 / 4), 2),

r\_eye(Point(swest().x + w - 4, swest().y + h \* 3 / 4), 2),

mouth(Point(swest().x + 2, swest().y + h / 4), w - 4),

screen(Screen::getInstance())

{}

void myshape::draw()

{

Rectangle::draw();

int a = (swest().x + neast().x) / 2;

int b = (swest().y + neast().y) / 2;

screen.put\_point(Point(a, b));

}

void myshape::move(int a, int b)

{

Rectangle::move(a, b);

l\_eye.move(a, b);

r\_eye.move(a, b);

mouth.move(a, b);

}

void up(Shape& p, const Shape& q);

void upM(Shape& p, const Shape& q, int y\_offset = 1, bool center = true)

{

Point n = q.north();

Point s = p.south();

int dy = n.y - s.y - y\_offset + 9;

int dx = 0;

if (center)

dx = (n.x - s.x);

p.move(dx, dy);

}

void rightS(Shape& p, const Shape& q)

{

Point bottomRight = q.seast();

Point pWest = p.west();

int dx = bottomRight.x - pWest.x;

int dy = bottomRight.y - pWest.y;

p.move(dx, dy - 3);

}

void leftS(Shape& p, const Shape& q)

{

Point bottomRight = q.swest();

Point pWest = p.east();

int dx = bottomRight.x - pWest.x;

int dy = bottomRight.y - pWest.y;

p.move(dx, dy - 3);

}

int main()

{

Parallelogram\* p = nullptr;

Parallelogram\* b1 = nullptr;

Parallelogram\* b2 = nullptr;

Screen& screen = Screen::getInstance();

ShapeControl shapeControl;

try

{

try

{

p = new Parallelogram(Point(-5, 5), Point(15, 5), Point(20, 15), Point(10, 15));

}

catch (const ParallelogramException& e)

{

std::cerr << "Creation error p: " << e.what() << std::endl;

}

catch (const std::out\_of\_range& e)

{

std::cerr << "Out Of Range error p: " << e.what() << std::endl;

}

try

{

b1 = new Parallelogram(Point(5, 5), Point(15, 5), Point(20, 15), Point(10, 15));

}

catch (const ParallelogramException& e)

{

std::cerr << "Creation error b1: " << e.what() << std::endl;

}

catch (const std::out\_of\_range& e)

{

std::cerr << "Out Of Range error p: " << e.what() << std::endl;

}

try

{

b2 = new Parallelogram(Point(5, 5), Point(15, 5), Point(20, 15), Point(10, 15));

}

catch (const ParallelogramException& e)

{

std::cerr << "Creation error b2: " << e.what() << std::endl;

}

catch (const std::out\_of\_range& e)

{

std::cerr << "Out Of Range error p: " << e.what() << std::endl;

}

if (p) p->move(10, 20);

if (b1) b1->move(40, 20);

if (b2) b2->move(25, 30);

Rectangle hat(Point(0, 0), Point(14, 5));

Line brim(Point(20, 9), 17);

myshape face(Point(15, 10), Point(27, 18));

h\_circle beard(Point(40, 10), 5);

shapeControl.shape\_refresh();

std::cout << "=== Generated... ===\n";

std::cin.get();

try

{

if (p) p->flip\_vertically();

}

catch (const std::exception& e)

{

std::cerr << "Error flip\_vertically(): " << e.what() << std::endl;

}

try

{

if (b1) b1->flip\_horisontally();

if (b1) b1->flip\_horisontally();

}

catch (const std::exception& e)

{

std::cerr << "Error flip\_horisontally(): " << e.what() << std::endl;

}

try

{

hat.rotate\_right();

}

catch (const std::exception& e)

{

std::cerr << "Error rotate\_right(): " << e.what() << std::endl;

}

try

{

brim.resize(2.0);

}

catch (const std::exception& e)

{

std::cerr << "Error resize(): " << e.what() << std::endl;

}

try

{

face.resize(1.2);

}

catch (const std::exception& e)

{

std::cerr << "Error resize(1.2): " << e.what() << std::endl;

}

try

{

if (b1) b1->resize(0.7);

}

catch (const std::exception& e)

{

std::cerr << "Error resize(0.7): " << e.what() << std::endl;

}

try

{

if (b2) b2->resize(0.7);

}

catch (const std::exception& e)

{

std::cerr << "Error resize(0.7): " << e.what() << std::endl;

}

try

{

beard.flip\_vertically();

}

catch (const std::exception& e)

{

std::cerr << "Error flip\_vertically() (beard): " << e.what() << std::endl;

}

try

{

beard.resize(1.2);

}

catch (const std::exception& e)

{

std::cerr << "Error resize(1.2) (beard): " << e.what() << std::endl;

}

shapeControl.shape\_refresh();

std::cout << "=== Prepared... ===\n";

std::cin.get();

//== 3. Финальное расположение ==

try

{

up(brim, face);

}

catch (const std::exception& e)

{

std::cerr << "Error up(brim, face): " << e.what() << std::endl;

}

try

{

up(hat, brim);

}

catch (const std::exception& e)

{

std::cerr << "Error up(hat, brim): " << e.what() << std::endl;

}

try

{

if (p) upM(\*p, hat, false);

}

catch (const std::exception& e)

{

std::cerr << "Error upM(p, hat): " << e.what() << std::endl;

}

try

{

down(beard, face);

}

catch (const std::exception& e)

{

std::cerr << "Error down(beard, face): " << e.what() << std::endl;

}

try

{

if (b2) rightS(\*b2, face);

}

catch (const std::exception& e)

{

std::cerr << "Error rightS(b2, face): " << e.what() << std::endl;

}

try

{

if (b1) leftS(\*b1, face);

}

catch (const std::exception& e)

{

std::cerr << "Error leftS(b1, face): " << e.what() << std::endl;

}

shapeControl.shape\_refresh();

std::cout << "=== Ready! ===\n";

std::cin.get();

}

catch (const std::bad\_alloc& e)

{

std::cerr << "Memory isolation error: " << e.what() << std::endl;

}

catch (const std::runtime\_error& e)

{

std::cerr << "Runtime error: " << e.what() << std::endl;

}

catch (const std::exception& e)

{

std::cerr << "General exception: " << e.what() << std::endl;

}

catch (...)

{

std::cerr << "Unknown exception!" << std::endl;

}

screen.screen\_destroy();

delete p;

delete b1;

delete b2;

return 0;

}

void up(Shape& p, const Shape& q)

{

Point n = q.north();

Point s = p.south();

p.move(n.x - s.x, n.y - s.y + 1);

## }*файл “Parallelogram.h”*

#pragma once

#include "Shape.h"

#include "Screen.h"

class Parallelogram : public Rotatable, public Reflectable

{

private:

Point p1, p2, p3, p4;

Screen& screen;

public:

Parallelogram(Point a, Point b, Point c, Point d);

Point north() const override;

Point south() const override;

Point east() const override;

Point west() const override;

Point neast() const override;

Point seast() const override;

Point nwest() const override;

Point swest() const override;

void draw() override;

void move(int dx, int dy) override;

void resize(double factor) override;

void rotate\_left();

void rotate\_right();

void flip\_horisontally();

void flip\_vertically();

};

## *файл “Parallelogram.cpp”*

#include "Parallelogram.h"

Parallelogram::Parallelogram(Point a, Point b, Point c, Point d) : p1(a), p2(b), p3(c), p4(d), screen(Screen::getInstance())

{

if (!screen.isWithinBounds(p1) || !screen.isWithinBounds(p2) || !screen.isWithinBounds(p3) || !screen.isWithinBounds(p4))

throw ParallelogramException("One or more points are out of screen bounds");

if ((p2 - p1).cross(p4 - p1) == 0)

throw ParallelogramException("Points do not form a parallelogram");

}

Point Parallelogram::north() const { return (p1.y < p2.y && p1.y < p3.y && p1.y < p4.y) ? p1 : (p2.y < p3.y && p2.y < p4.y) ? p2 : (p3.y < p4.y) ? p3 : p4; }

Point Parallelogram::south() const { return (p1.y > p2.y && p1.y > p3.y && p1.y > p4.y) ? p1 : (p2.y > p3.y && p2.y > p4.y) ? p2 : (p3.y > p4.y) ? p3 : p4; }

Point Parallelogram::east() const { return (p1.x > p2.x && p1.x > p3.x && p1.x > p4.x) ? p1 : (p2.x > p3.x && p2.x > p4.x) ? p2 : (p3.x > p4.x) ? p3 : p4; }

Point Parallelogram::west() const { return (p1.x < p2.x && p1.x < p3.x && p1.x < p4.x) ? p1 : (p2.x < p3.x && p2.x < p4.x) ? p2 : (p3.x < p4.x) ? p3 : p4; }

Point Parallelogram::neast() const { return (p1.x > p2.x && p1.y < p2.y) ? p1 : (p2.x > p3.x && p2.y < p3.y) ? p2 : p3; }

Point Parallelogram::seast() const { return (p1.x > p2.x && p1.y > p2.y) ? p1 : (p2.x > p3.x && p2.y > p3.y) ? p2 : p3; }

Point Parallelogram::nwest() const { return (p1.x < p2.x && p1.y < p2.y) ? p1 : (p2.x < p3.x && p2.y < p3.y) ? p2 : p3; }

Point Parallelogram::swest() const { return (p1.x < p2.x && p1.y > p2.y) ? p1 : (p2.x < p3.x && p2.y > p3.y) ? p2 : p3; }

void Parallelogram::draw()

{

try

{

screen.put\_line(p1.x, p1.y, p2.x, p2.y);

screen.put\_line(p2.x, p2.y, p3.x, p3.y);

screen.put\_line(p3.x, p3.y, p4.x, p4.y);

screen.put\_line(p4.x, p4.y, p1.x, p1.y);

}

catch (const ParallelogramException& e)

{

std::cerr << "Drawing error: " << e.what() << std::endl;

}

}

void Parallelogram::move(int dx, int dy)

{

Point new\_p1 = p1 + Point(dx, dy);

Point new\_p2 = p2 + Point(dx, dy);

Point new\_p3 = p3 + Point(dx, dy);

Point new\_p4 = p4 + Point(dx, dy);

if (!screen.isWithinBounds(new\_p1) || !screen.isWithinBounds(new\_p2) || !screen.isWithinBounds(new\_p3) || !screen.isWithinBounds(new\_p4))

throw ParallelogramException("Moved parallelogram is out of screen bounds");

p1 = new\_p1;

p2 = new\_p2;

p3 = new\_p3;

p4 = new\_p4;

}

void Parallelogram::resize(double factor) {

if (factor <= 0)

throw ParallelogramException("Resize factor must be positive");

p2 = p1 + (p2 - p1) \* factor;

p3 = p1 + (p3 - p1) \* factor;

p4 = p1 + (p4 - p1) \* factor;

}

void Parallelogram::rotate\_left()

{

if (state == Rotation::Left)

throw ParallelogramException("Parallelogram has already been rotated");

Point center = (p1 + p2 + p3 + p4) / 4;

p1 = center + Point(-(p1.y - center.y), p1.x - center.x);

p2 = center + Point(-(p2.y - center.y), p2.x - center.x);

p3 = center + Point(-(p3.y - center.y), p3.x - center.x);

p4 = center + Point(-(p4.y - center.y), p4.x - center.x);

state = Rotation::Left;

}

void Parallelogram::rotate\_right()

{

if (state == Rotation::Right)

throw ParallelogramException("Parallelogram has already been rotated");

Point center = (p1 + p2 + p3 + p4) / 4;

p1 = center + Point(p1.y - center.y, -(p1.x - center.x));

p2 = center + Point(p2.y - center.y, -(p2.x - center.x));

p3 = center + Point(p3.y - center.y, -(p3.x - center.x));

p4 = center + Point(p4.y - center.y, -(p4.x - center.x));

state = Rotation::Right;

}

void Parallelogram::flip\_horisontally()

{

if (hor)

throw ParallelogramException("Parallelogram has already been flipped");

Point center = (p1 + p2 + p3 + p4) / 4;

p1.x = 2 \* center.x - p1.x;

p2.x = 2 \* center.x - p2.x;

p3.x = 2 \* center.x - p3.x;

p4.x = 2 \* center.x - p4.x;

hor = true;

}

void Parallelogram::flip\_vertically()

{

if (vert)

throw ParallelogramException("Parallelogram has already been flipped");

Point center = (p1 + p2 + p3 + p4) / 4;

p1.y = 2 \* center.y - p1.y;

p2.y = 2 \* center.y - p2.y;

p3.y = 2 \* center.y - p3.y;

p4.y = 2 \* center.y - p4.y;

vert = true;

}

***“Shape.h”***

#pragma once

#include <iostream>

#include <stdexcept>

#include "Shape.h"

class Screen

{

protected:

static const int XMAX = 120;

static const int YMAX = 50;

char screen[YMAX][XMAX];

enum color { black = '\*', white = '.' };

public:

static Screen& getInstance();

void put\_point(int x, int y);

void put\_point(Point p) { put\_point(p.x, p.y); }

void put\_line(int, int, int, int);

void put\_line(Point a, Point b) { put\_line(a.x, a.y, b.x, b.y); }

void screen\_init();

void screen\_destroy();

void screen\_refresh();

void screen\_clear();

bool on\_screen(int a, int b);

bool isWithinBounds(const Point& p);

};

***“Shape.cpp”***

#include "Screen.h"

void Screen::screen\_init()

{

for (auto y = 0; y < YMAX; ++y)

for (auto& x : screen[y])

x = white;

}

void Screen::screen\_destroy()

{

for (auto y = 0; y < YMAX; ++y)

for (auto& x : screen[y])

x = black;

}

bool Screen::on\_screen(int a, int b) { return 0 <= a && a < XMAX && 0 <= b && b < YMAX; }

bool Screen::isWithinBounds(const Point& p)

{

return on\_screen(p.x, p.y);

}

void Screen::put\_point(int a, int b)

{

if (!isWithinBounds(Point(a, b)))

throw std::out\_of\_range("put\_point: Coordinates out of screen bounds");

screen[b][a] = black;

}

void Screen::put\_line(int x0, int y0, int x1, int y1)

{

if (!on\_screen(x0, y0) || !on\_screen(x1, y1))

throw std::out\_of\_range("put\_line: Line endpoints out of screen bounds");

int dx = 1;

int a = x1 - x0;

if (a < 0)

dx = -1, a = -a;

int dy = 1;

int b = y1 - y0;

if (b < 0)

dy = -1, b = -b;

int two\_a = 2 \* a;

int two\_b = 2 \* b;

int xcrit = -b + two\_a;

int eps = 0;

for (;;)

{

try

{

put\_point(x0, y0);

}

catch (const std::exception& e)

{

std::cerr << "Error: " << e.what() << '\n';

return;

}

if (x0 == x1 && y0 == y1)

break;

if (eps <= xcrit)

x0 += dx, eps += two\_b;

if (eps >= a || a < b)

y0 += dy, eps -= two\_a;

}

}

void Screen::screen\_clear() { screen\_init(); }

void Screen::screen\_refresh()

{

for (int y = YMAX - 1; 0 <= y; --y)

{

for (auto x : screen[y])

std::cout << x;

std::cout << '\n';

}

}

Screen& Screen::getInstance()

{

static Screen instance;

return instance;

}