**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 |  | Нукусси Ф.М |
| Преподаватель |  | Колинько П.Г. |

Санкт-Петербург

2025

**Цель работы**

Изучение способов обработки исключительных ситуаций.

**Задание**

Переработать программу работы с библиотекой фигур, дополнив её механизмом контроля исключительных ситуаций.

Реализовать генерацию и перехват не менее двух типов ошибок разного уровня сложности.

**Набор и вид классов для фиксации особых ситуаций**

1. Попадание фигуры за пределы экрана

class PointOffScreen : public std::runtime\_error {

private:

std::string fullMessage;

public:

PointOffScreen(const std::string& figure, const std::string& message)

: std::runtime\_error(message.c\_str()),

fullMessage("PointOffScreen: " + figure + " - " + message) {}

const char\* what() const noexcept override {

return fullMessage.c\_str();

}

};

1. Недопустимые параметры

class InvalidFigureParameters : public std::runtime\_error {

private:

std::string fullMessage;

public:

InvalidFigureParameters(const std::string& figure, const std::string& message)

: std::runtime\_error(message.c\_str()),

fullMessage("InvalidFigureParameters: " + figure + " - " + message) {}

const char\* what() const noexcept override {

return fullMessage.c\_str();

}

};

Классы ошибок наследуют от класса runtime\_error и имеют конструктор, принимающий строку и фигуру

**Место расположения операторов throw и блоков контроля**

Операторы throw расположены в функциях draw для class cross :

void draw() override {

cross createCross(const std::string& figureName, const point& c, int s) {

try {

return cross(figureName, c, s);

} catch (const PointOffScreen& e) {

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

std::cerr << "Usage of default parameters!" << std::endl;

return cross("DefaultCross", point(8, 3), 3);

} catch (const InvalidFigureParameters& e) {

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

std::cerr << "Usage of default parameters!" << std::endl;

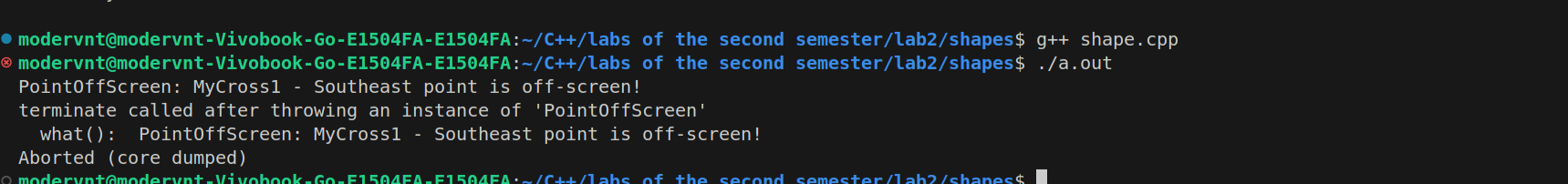
return cross("DefaultCross", point(8, 3), 3);

}

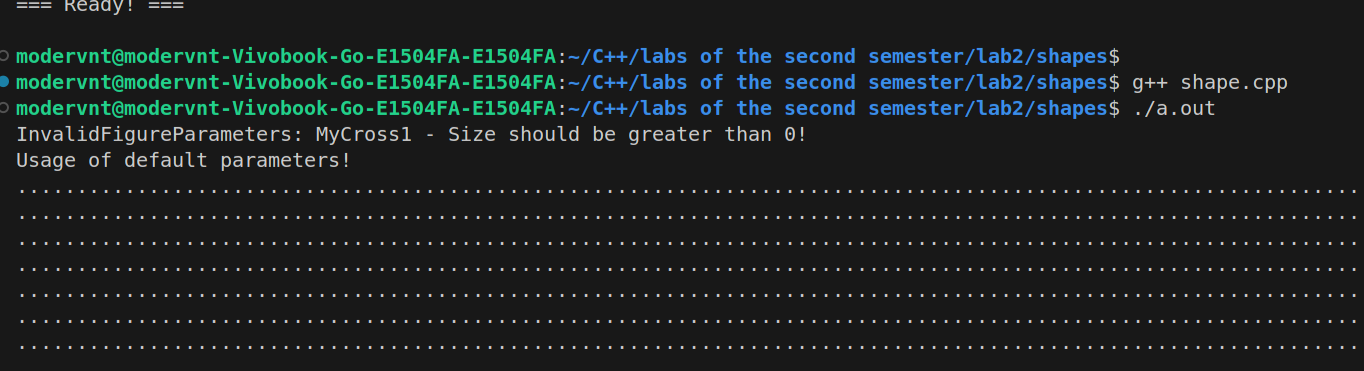
}

**Результаты тестирования**

1. Выход фигуры за пределы экрана

****

1. Неправильные параматры:



**Код программы**

**// Пополнение и использование библиотеки фигур**

**//связь с ОС (пример для Visual C++2017)**

**#include <locale.h>**

**#include <iostream>**

**#include "screen.h"**

**#include "shape.h"**

**// ПРИМЕР ДОБАВКИ: дополнительный фрагмент – полуокружность**

**class h\_circle: public rectangle, public reflectable {**

**public:**

**h\_circle(point a, int rd)**

**: rectangle(point(a.x-rd, a.y), point(a.x+rd,a.y+rd\*0.7+1)) { }**

**void draw();**

**void flip\_horisontally() { }; // Отразить горизонтально (пустая функция)**

**void rotate\_right() {} // Повернуть вправо**

**void rotate\_left() {} // Повернуть влево**

**};**

**void h\_circle :: draw() //Алгоритм Брезенхэма для окружностей**

**{ // (выдаются два сектора, указываемые значением reflected::vert)**

**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) { // Цикл рисования**

**if(vert) { put\_point(x0 + x, y0 - y\*0.7); put\_point(x0 - x, y0 - y\*0.7); }**

**else { put\_point(x0 + x, y0 + y\*0.7); 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;**

**}**

**// rectangle::draw();**

**}**

**// ПРИМЕР ДОБАВКИ: дополнительная функция присоединения…**

**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); }**

**// Cборная пользовательская фигура – физиономия**

**class myshape : public rectangle { // Моя фигура ЯВЛЯЕТСЯ**

**int w, h; // прямоугольником**

**line l\_eye; // левый глаз – моя фигура СОДЕРЖИТ линию**

**line r\_eye; // правый глаз**

**line mouth; // рот**

**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), //Инициализация базового класса**

**w(neast( ).x - swest( ).x + 1), // Инициализация данных**

**h(neast( ).y - swest( ).y + 1), // – строго в порядке объявления!**

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

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

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

**{ }**

**void myshape :: draw( )**

**{**

**rectangle :: draw( ); //Контур лица (глаза и нос рисуются сами!)**

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

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

**/\*int a = (swest( ).x + neast( ).x) / 1.2;**

**int b = (swest( ).y + neast( ).y) / 1.2;\*/**

**put\_point(point(a, b)); // Нос – существует только на рисунке!**

**}**

**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);**

**}**

**cross createCross(const std::string& figureName, const point& c, int s) {**

**try {**

**return cross(figureName, c, s);**

**} catch (const PointOffScreen& e) {**

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

**std::cerr << "Usage of default parameters!" << std::endl;**

**return cross("DefaultCross", point(8, 3), 3);**

**} catch (const InvalidFigureParameters& e) {**

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

**std::cerr << "Usage of default parameters!" << std::endl;**

**return cross("DefaultCross", point(8, 3), 3);**

**}**

**}**

**int main( )**

**{ setlocale(LC\_ALL, "en\_US.UTF-8");**

**screen\_init( );**

**//== 1. Объявление набора фигур ==**

**// Création des croix avec gestion centralisée des exceptions et noms spécifiques**

**cross myCross1 = createCross("MyCross1", point(8, 3), 0);**

**cross myCross2 = createCross("MyCross2", point(34, 3), 3);**

**cross myCross3 = createCross("MyCross3", point(24, 9), 3);**

**cross myCross4 = createCross("MyCross4", point(12, 6), 3);**

**rectangle hat(point(0, 0), point(14, 5));**

**//cross my\_cross(point(21,5),3);**

**line brim(point(20,9),17);**

**myshape face(point(15,10), point(27,18));**

**h\_circle beard(point(40,10), 5);**

**shape\_refresh( );**

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

**std::cin.get(); //Смотреть исходный набор**

**//== 2. Подготовка к сборке ==**

**hat.rotate\_right( );**

**brim.resize(2.0);**

**//face.resize(1.2);**

**face.resize(1.5);**

**beard.flip\_vertically( );**

**beard.resize(1.5);**

**shape\_refresh( );**

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

**std::cin.get(); //Смотреть результат поворотов/отражений**

**//== 3. Сборка изображения ==**

**face.move(0, -8); // Лицо – в исходное положение (если нужно!)**

**up(brim, face);**

**up(hat, brim);**

**down(beard, face);**

**left(myCross1, beard);**

**right(myCross2, beard);**

**push\_vert(myCross2, 10);**

**push\_vert(myCross1, 10);**

**push\_side(myCross1, -2);**

**push\_side(myCross2, 2);**

**push\_side(myCross3, -6);**

**push\_vert(myCross3, -3);**

**push\_side(myCross4, 13);**

**(myCross1, beard);**

**shape\_refresh( );**

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

**std::cin.get(); //Смотреть результат**

**screen\_destroy( );**

**return 0;**

**}**

**Выводы**

В ходе выполнения этого задания мы разобрались с механизмом обработки исключений. Подключив программу к системе перехвата ошибок, мы воспользовались встроенным механизмом для передачи сообщений.

**Список использованных источников**

1. П.Г. Колинько – «Пользовательские контейнеры» учебно-метод. пособие, 2025 г.

**Приложение**

В файле shape.h.