Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Отчёт по заданию № 2**

**на тему: “Поддержка обработки исключительных ситуаций”**

**по дисциплине**

**“Алгоритмы и структуры данных”**

**Вариант 44**

|  |  |
| --- | --- |
| Выполнили студенты гр.9308: | Дементьев Д.П., Ручкин Д.А. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2021 г.

Оглавление

[Введение 3](#_Toc12106)

[1. Задание 3](#_Toc20172)

[2. Формализация задания 3](#_Toc12132)

[3. Контрольные примеры 4](#_Toc27699)

[Вывод 7](#_Toc15280)

[Список используемых источников 8](#_Toc14442)

[Приложение 1 (Исходный текст программы) 9](#_Toc8251)

## Введение

Целью практической работы является получение навыков обработки исключительных ситуаций.

## 1. Задание

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

## 2. Формализация задания

## При выполнении задания будут рассмотрены следующие исключительные ситуации:

1. Непопадание точки на экран
2. Некорректные параметры при формировании фигуры
3. Недостаток места для размещения фигуры на экране.

Для выполнения поставленных задач был разработан класс error\_model, являющийся производным от классов rotatable и reflectable.. Если при различных действиях с какой-либо фигурой возникает проблема отрисовки её на экране, то эта фигура удаляется и заменяется фигурой знака ошибки типа model\_error (крест размером 3х3), эту фигуру можно перемещать по экрану, но изменять нельзя.

Также был разработан класс ошибки попадания точки за пределы экрана OffScreen. В конструкторах фигур учитывается непопадание части этой фигуры на экран, если это так, то создается объект OffScreеn и в конструкторе вызывается исключение. При создании фигуры необходимо использовать блоки try-catch, если в конструкторе фигуры будет вызвано исключение, то объект будет удалён и заменён объектом типа error\_model.

## 3. Контрольные примеры

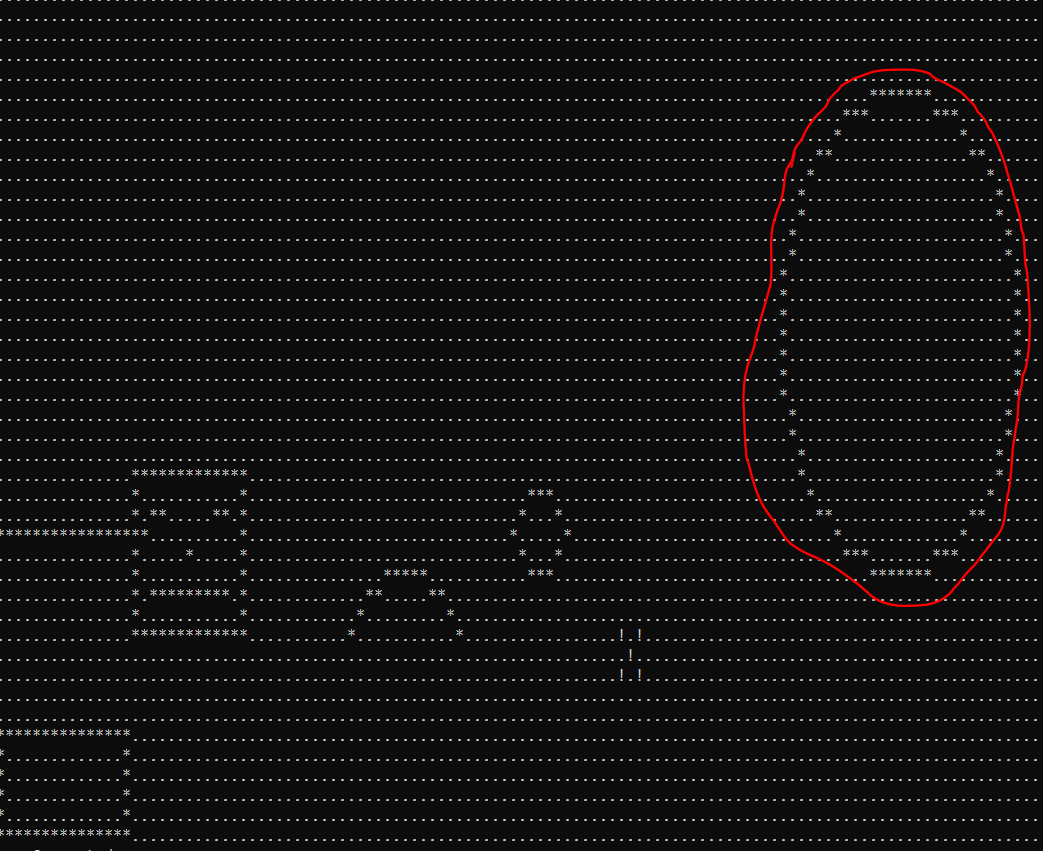
В качестве контрольных примеров возьмем следующие ситуации:

1. Фигура создаётся с некорректными параметрами (залезая за границы экрана)
2. Фигура, которая при увеличении размера выходит за границы экрана
3. Фигура при перемещении на новую позицию выходит за границы экрана



## *Рис.1. 1 контрольный пример*

Фигуре были заданы параметры, при которых часть её оказывалась за границами экрана. Выводится сообщение о том, что не удалось корректно создать фигуру №4 и она была заменена знаком ошибки.



## *Рис.2.1. 2 контрольный пример*

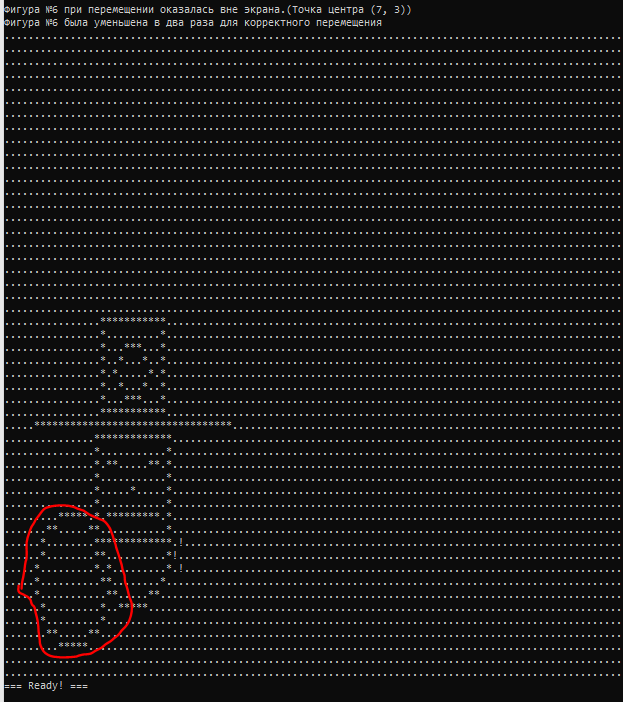


## *Рис.2.2. 2 контрольный пример (следующий шаг)*

Производится попытка увеличить радиус фигуры вдвое. Выводится сообщение о том, что не удалось удачно изменить размер фигуры № 6 и она была заменена знаком ошибки.



## *Рис.3.1. 3 контрольный пример*



## *Рис.3.2. 3 контрольный пример (следующий шаг)*

При попытки перемещения фигуры на позицию, где ей не будет хватать места для размещения на экране, будет произведена попытка уменьшить её в 2 раза, если же не будет достаточно места и для уменьшенной фигуры, то она останется на том же месте с тем же размером. Также выводятся соответствующие сообщения.

## Вывод

При выполнении этой лабораторной работы были получены практические навыки по работе с исключительными ситуациями. Также были задействованы навыки работы с наследованием классов, полиморфизмом, виртуальными функциями.

## Список используемых источников

1. Колинько П.Г. Пользовательские контейнеры / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.
2. Обработка исключительных ситуаций.<https://prog-cpp.ru/cpp-exception/>
3. Урок №163. Виртуальные функции и Полиморфизм. URL: <https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/>

## Приложение 1 (Исходный текст программы)

Файл **shape.cpp:**

//========== Файл shape.cpp (прикладная программа) ==========

// Пополнение и использование библиотеки фигур

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include "screen.h"

#include "shape.h"

class circle : public shape {

protected:

point c;

int radius;

public:

circle(point a, int r);

void draw();

point north() const { return point(c.x,c.y+radius); }

point south() const { return point(c.x,c.y-radius); }

point east() const { return point(c.x+radius,c.y); }

point west() const { return point(c.x-radius,c.y); }

point neast() const { return point(c.x+radius\*0.7, c.y+radius\*0.7); }

point seast() const { return point(c.x+radius\*0.7, c.y-radius\*0.7); }

point nwest() const { return point(c.x-radius\*0.7, c.y+radius\*0.7); }

point swest() const { return point(c.x-radius\*0.7, c.y-radius\*0.7); }

void move(int a , int b); //Перемещение

void resize(int d) { (d>0) ? radius \*= d : radius /= -d; } //Изменение размера

private:

circle(const circle &x) { std::cout << "copy constructor\n"; } // конструктор копирования

circle& operator=(const circle &x) { }// оператор присваивания копированием

circle(circle &&x) { std::cout << "move constructor\n"; } // конструктор перемещения

circle& operator=(circle &&x) { } // оператор присваивания перемещением

};

circle::circle(point a, int r) : c(a), radius(r) {

if((!on\_screen(c.x+r, c.y)) || (!on\_screen(c.x,c.y+r)) || (!on\_screen(c.x-r,c.y)) || (!on\_screen(c.x,c.y-r)))

throw OffScreen("Фигура №" + std::to\_string(id) + " при создании оказалась вне экрана.", a);

}

void circle::move(int a, int b) {

point p(c.x+a, c.y+b);

if (!on\_screen(p.x+radius,p.y) || !on\_screen(p.x,p.y+radius) || !on\_screen(p.x-radius,p.y) || !on\_screen(p.x,p.y-radius))

throw OffScreen("Фигура №" + std::to\_string(id) + " при перемещении оказалась вне экрана.", p);

else { c.x = p.x; c.y = p.y; }

}

void circle::draw() {

char buff[YMAX][XMAX];

// копируем изображение экрана до изменений

for (int i=0; i<YMAX; i++)

for (int j=0; j<XMAX; j++)

buff[i][j] = screen[i][j];

try {

int x = 0, y = radius, delta = 1 - 2 \* radius, error = 0;

while (y >= 0) {

put\_point(c.x + x, c.y + y);

put\_point(c.x + x, c.y - y);

put\_point(c.x - x, c.y + y);

put\_point(c.x - x, c.y - y);

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;

}

}

catch (OffScreen &ex) {

std::cout << "Ошибка вывода фигуры №" << id << " (" << ex.what() << ex.crnds() << ")\n\n";

resize(-2);

try {

// возвращаем изображение экрана в исходное состояние

for (int i=0; i<YMAX; i++)

for (int j=0; j<XMAX; j++)

screen[i][j] = buff[i][j];

int x = 0, y = radius, delta = 1 - 2 \* radius, error = 0;

while (y >= 0) {

put\_point(c.x + x, c.y + y);

put\_point(c.x + x, c.y - y);

put\_point(c.x - x, c.y + y);

put\_point(c.x - x, c.y - y);

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;

}

std::cout << "Фигура №" << id << " была уменьшена в два раза для корректного вывода\n";

}

catch (OffScreen) {

// возвращаем изображение экрана в исходное состояние

for (int i=0; i<YMAX; i++)

for (int j=0; j<XMAX; j++)

screen[i][j] = buff[i][j];

resize(2);

throw OffScreen("Фигура №" + std::to\_string(id) + " при изменении оказалась вне экрана.", c);

}

}

}

// ПРИМЕР ДОБАВКИ: дополнительный фрагмент - полуокружность

class h\_circle: public rectangle, public reflectable {

bool reflected;

public:

h\_circle(point a, point b, bool r);

void draw();

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

void flip\_vertically( ) { reflected = !reflected; }; // Отразить вертикально

};

h\_circle::h\_circle(point a, point b, bool r=true) : rectangle(a, b), reflected(r) {

if((!on\_screen(a.x, a.y)) || (!on\_screen(b.x,b.y)))

throw OffScreen("Фигура №" + std::to\_string(id) + " при создании оказалась вне экрана.", point((east().x-west().x)/2, (north().y-south().y)/2));

}

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

{ //(выдаются два сектора, указываемые значением reflected)

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

}

}

// 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(int) { }

};

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), 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)

{ }

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;

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);

}

// Модель, заменяющая фигуру с ошибкой при создании/изменении (крестик размером 3x3)

class error\_model : public rotatable, public reflectable {

private:

point s;

public:

error\_model(point point\_) : s(point\_) {}

point north() const{ return point(s.x, s.y + 1); }

point south() const{ return point(s.x, s.y - 1); }

point east() const{ return point (s.x + 1, s.y); }

point west() const{ return point(s.x - 1, s.y); }

point neast() const{ return point(s.x + 1, s.y + 1); }

point seast() const{ return point(s.x + 1, s.y - 1); }

point nwest() const{ return point(s.x - 1, s.y + 1); }

point swest() const{ return point(s.x - 1, s.y - 1); }

void draw();

void move(int dx, int dy) { s.x += dx; s.y += dy; };

// изменение ошибочной фигуры недопустимо

void resize(int) { }

void rotate\_left() { }

void rotate\_right() { }

void flip\_vertically() { }

void flip\_horisontally() { }

};

void error\_model::draw() {

error\_sym = true;

put\_line(nwest(), seast());

put\_line(swest(), neast());

error\_sym = false;

}

void shape\_refresh( ) { // Перерисовка всех фигур на экране

screen\_clear( );

for (auto p : shape::shapes) {

try {

p->draw( ); //Динамическое связывание!!!

}

catch(OffScreen &ex) {

std::cout << "Фигура №" << p->id << " не будет выведена (" << ex.what() << ex.crnds() << ")\n\n";

error\_model err = error\_model(ex.get\_point());

err.draw();

}

}

screen\_refresh( );

}

// ПРИМЕР ДОБАВКИ: дополнительная функция присоединения…

void down(shape &p, const shape &q) {

try {

point n = q.south( );

point s = p.north( );

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

}

catch (OffScreen &ex) {

std::cout << ex.what() << "(Точка центра " << ex.crnds() << ")\n";

p.resize(-2);

try {

point n = q.south( );

point s = p.north( );

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

std::cout << "Фигура №" << p.id << " была уменьшена в два раза для корректного перемещения\n";

}

catch (OffScreen) {

p.resize(2);

std::cout << "Фигура №" << p.id << " не будет перемещена\n";

}

}

}

void centering(shape &p, const shape &q) {

int delta\_x, delta\_y;

try {

delta\_x = (q.east().x + q.west().x)/2 - (p.east().x + p.west().x)/2, // (центр фигуры q) - (центр фигуры p), компонента икс

delta\_y = (q.east().y + q.west().y)/2 - (p.east().y + p.west().y)/2; // (центр фигуры q) - (центр фигуры p), компонента игрек

p.move(delta\_x, delta\_y);

}

catch (OffScreen &ex) {

std::cout << ex.what() << "(Точка центра " << ex.crnds() << ")\n";

p.resize(-2);

try {

delta\_x = (q.east().x + q.west().x)/2 - (p.east().x + p.west().x)/2, // (центр фигуры q) - (центр фигуры p), компонента икс

delta\_y = (q.east().y + q.west().y)/2 - (p.east().y + p.west().y)/2; // (центр фигуры q) - (центр фигуры p), компонента игрек

p.move(delta\_x, delta\_y);

std::cout << "Фигура №" << p.id << " была уменьшена в два раза для корректного перемещения\n";

}

catch (OffScreen) {

p.resize(2);

std::cout << "Фигура №" << p.id << " не будет перемещена\n";

}

}

}

void rightDown(shape &p, shape&q) {

try {

point n = q.seast();

point s = p.nwest();

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

}

catch (OffScreen &ex) {

std::cout << ex.what() << "(Точка центра " << ex.crnds() << ")\n";

p.resize(-2);

try {

point n = q.seast();

point s = p.nwest();

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

std::cout << "Фигура №" << p.id << " была уменьшена в два раза для корректного перемещения\n";

}

catch (OffScreen) {

p.resize(2);

std::cout << "Фигура №" << p.id << " не будет перемещена\n";

}

}

}

void leftDown(shape &p, shape &q){

try {

point n = q.swest();

point s = p.neast();

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

}

catch (OffScreen &ex) {

std::cout << ex.what() << "(Точка центра " << ex.crnds() << ")\n";

p.resize(-2);

try {

point n = q.swest();

point s = p.neast();

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

std::cout << "Фигура №" << p.id << " была уменьшена в два раза для корректного перемещения\n";

}

catch (OffScreen) {

p.resize(2);

std::cout << "Фигура №" << p.id << " не будет перемещена\n";

}

}

}

int main( )

{

setlocale(LC\_ALL, "Rus");

screen\_init( );

//== 1.Объявление набора фигур ==

rotatable \*hat;

try {

hat = new rectangle(point(0, 0), point(14, 5));

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

hat = new error\_model(ex.get\_point());

}

shape \*brim;

try {

brim = new line(point(0,15),17);

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

brim = new error\_model(ex.get\_point());

}

shape \*emblem;

try {

emblem = new circle(point(60, 15), 2);

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

emblem = new error\_model(ex.get\_point());

}

shape \*whisker\_left;

try {

whisker\_left = new circle(point(70,9), 10);

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

whisker\_left = new error\_model(ex.get\_point());

}

shape \*whisker\_right;

try {

whisker\_right = new circle(point(70,20), 12);

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

whisker\_right = new error\_model(ex.get\_point());

}

shape \*face;

try {

face = new myshape(point(15,10), point(27,18));

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

face = new error\_model(ex.get\_point());

}

reflectable \*beard;

try {

beard = new h\_circle(point(40,10), point(50,20));

}

catch (OffScreen &ex) {

std::cout << ex.what() << "\n";

beard = new error\_model(ex.get\_point());

}

shape\_refresh();

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

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

//== 2.Подготовка к сборке ==

hat->rotate\_right();

brim->resize(2);

face->resize(2);

beard->flip\_vertically();

whisker\_left->resize(-5);

whisker\_right->resize(-4);

shape\_refresh( );

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

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

//== 3.Сборка изображения ==

up(\*brim, \*face);

up(\*hat, \*brim);

down(\*beard, \*face);

centering(\*emblem, \*hat);

rightDown(\*whisker\_left, \*face);

leftDown(\*whisker\_right, \*face);

shape\_refresh( );

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

std::cin.get(); //Смотреть результат

screen\_destroy( );

delete hat;

delete brim;

delete emblem;

delete whisker\_left;

delete whisker\_right;

delete face;

delete beard;

return 0;

}

Файл **shape.h:**

//======== Файл shape.h — библиотека фигур =========

#include <list>

#include "exceptions.h"

//==1. Поддержка экрана в форме матрицы символов ==

char screen[YMAX] [XMAX];

bool error\_sym = false;

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

void screen\_init( ) {

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

for (auto &x : screen[y]) x = white;

}

void screen\_destroy( ) {

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

for (auto &x : screen[y]) x = black;

}

bool on\_screen(int a, int b) // проверка попадания точки на экран

{ return 0 <= a && a < XMAX && 0 <= b && b < YMAX; }

void put\_point(int a, int b) {

if (error\_sym) { if (on\_screen(a,b)) screen[b][a] = error; }

else if (on\_screen(a,b)) screen[b][a] = black;

else throw OffScreen("Точка лежит вне экрана", point(a,b));

}

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

/\* Алгоритм Брезенхэма для прямой:

рисование отрезка прямой от (x0,y0) до (x1,y1).

Уравнение прямой: b(x-x0) + a(y-y0) = 0.

Минимизируется величина abs(eps), где eps = 2\*(b(x-x0)) + a(y-y0). \*/

{

try {

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 (;;) { //Формирование прямой линии по точкам

put\_point(x0, y0);

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

if (eps <= xcrit) x0 += dx, eps += two\_b;

if (eps >= a || a < b) y0 += dy, eps -= two\_a;

}

}

catch (OffScreen &ex) {

throw; // передаём выше

}

}

void screen\_clear( ) { screen\_init( ); } //Очистка экрана

void screen\_refresh( ) { // Обновление экрана

for (int y = YMAX-1; 0 <= y; --y) { // с верхней строки до нижней

for (auto x : screen[y]) // от левого столбца до правого

std::cout << x;

std::cout << '\n';

}

}

//== 2. Библиотека фигур ==

struct shape { // Виртуальный базовый класс "фигура"

static std::list <shape\*> shapes;// Список фигур (один на все фигуры!)

static int count; // счётчик фигур

int id; // уникальный номер фигуры

shape( ) { shapes.push\_back(this); id = count++; } //Фигура присоединяется к списку и ей присваивается уникальный номер (id)

virtual point north( ) const = 0; //Точки для привязки

virtual point south( ) const = 0;

virtual point east( ) const = 0;

virtual point west( ) const = 0;

virtual point neast( ) const = 0;

virtual point seast( ) const = 0;

virtual point nwest( ) const = 0;

virtual point swest( ) const = 0;

virtual void draw( ) = 0; //Рисование

virtual void move(int, int) = 0; //Перемещение

virtual void resize(int) = 0; //Изменение размера

virtual ~shape() {shapes.remove(this);}

};

std::list <shape\*> shape::shapes; // Размещение списка фигур

int shape::count = 1; // инициализация счётчика фигур

/\*void shape\_refresh( ) { // Перерисовка всех фигур на экране

screen\_clear( );

for (auto p : shape::shapes) {

try {

p->draw( ); //Динамическое связывание!!!

}

catch(OffScreen &ex) {

std::cout << "Ошибка вывода фигуры №" << p->id << " (" << ex.what() << ex.crnds() << ")\n\n";

shape::shapes.remove(p);

shape \*err = new error\_model(ex.get\_point());

shape::shapes.push\_back(err);

}

}

screen\_refresh( );

}\*/

class rotatable : virtual public shape { //Фигуры, пригодные к повороту

public:

virtual void rotate\_left( ) = 0; //Повернуть влево

virtual void rotate\_right( ) = 0; //Повернуть вправо

};

class reflectable : virtual public shape { // Фигуры, пригодные

public: // к зеркальному отражению

virtual void flip\_horisontally( ) = 0; // Отразить горизонтально

virtual void flip\_vertically( ) = 0; // Отразить вертикально

};

class line : public shape {

/\* отрезок прямой ["w", "e"].

north( ) определяет точку "выше центра отрезка и так далеко

на север, как самая его северная точка", и т. п. \*/

protected:

point w, e;

public:

line(point a, point b);

line(point a, int L);

point north( ) const { return point((w.x+e.x)/2, e.y<w.y? w.y : e.y); }

point south( ) const { return point((w.x+e.x)/2, e.y<w.y? e.y : w.y); }

point east( ) const { return point(e.x<w.x? w.x : e.x, (w.y+e.y)/2); }

point west( ) const { return point(e.x<w.x? e.x : w.x, (w.y+e.y)/2); }

point neast( ) const { return point(w.x<e.x? e.x : w.x, e.y<w.y? w.y : e.y); }

point seast( ) const { return point(w.x<e.x? e.x : w.x, e.y<w.y? e.y : w.y); }

point nwest( ) const { return point(w.x<e.x? w.x : e.x, e.y<w.y? w.y : e.y); }

point swest( ) const { return point(w.x<e.x? w.x : e.x, e.y<w.y? e.y : w.y); }

void move(int a, int b) { w.x += a; w.y += b; e.x += a; e.y += b; }

void draw( );

void resize(int d) // Увеличение длины линии в (d) раз

{ e.x += (e.x - w.x) \* (d - 1); e.y += (e.y - w.y) \* (d - 1); }

};

line::line (point a, point b) : w(a), e(b) {

if((!on\_screen(w.x, w.y)) || (!on\_screen(e.x,e.y)))

throw OffScreen("Фигура №" + std::to\_string(id) + " при создании оказалась вне экрана.", point((east().x-west().x)/2, (north().y-south().y)/2));

};

line::line(point a, int L): w(a), e(point(a.x + L - 1, a.y)) {

if((!on\_screen(w.x, w.y)) || (!on\_screen(e.x,e.y)))

throw OffScreen("Фигура №" + std::to\_string(id) + " при создании оказалась вне экрана.", point((east().x-west().x)/2, (north().y-south().y)/2));

};

void line::draw() {

try {

put\_line(w, e);

}

catch(OffScreen &ex) {

std::cout << "Для фигуры №" << id << " " << ex.what() << ex.crnds() << "\n\n";

resize(-2);

try {

put\_line(w, e);

std::cout << "Фигура №" << id << " была уменьшена в два раза для корректного вывода\n";

}

catch(OffScreen) {

resize(2);

throw OffScreen("Фигура №" + std::to\_string(id) + " при изменении оказалась вне экрана.", point((east().x-west().x)/2, (north().y-south().y)/2));

}

}

}

// Прямоугольник

class rectangle : public rotatable {

/\* nw ------ n ------ ne

| |

| |

w c e

| |

| |

sw ------- s ------ se \*/

protected:

point sw, ne;

public:

rectangle(point a, point b);

point north( ) const { return point((sw.x + ne.x) / 2, ne.y); }

point south( ) const { return point((sw.x + ne.x) / 2, sw.y); }

point east( ) const { return point(ne.x, (sw.y + ne.y) / 2); }

point west( ) const { return point(sw.x, (sw.y + ne.y) / 2); }

point neast( ) const { return ne; }

point seast( ) const { return point(ne.x, sw.y); }

point nwest( ) const { return point(sw.x, ne.y); }

point swest( ) const { return sw; }

void rotate\_right() // Поворот вправо относительно se

{ int w = ne.x - sw.x, h = ne.y - sw.y; //(учитывается масштаб по осям)

sw.x = ne.x - h \* 2; ne.y = sw.y + w / 2; }

void rotate\_left() // Поворот влево относительно sw

{ int w = ne.x - sw.x, h = ne.y - sw.y;

ne.x = sw.x + h \* 2; ne.y = sw.y + w / 2; }

void move(int a, int b)

{ sw.x += a; sw.y += b; ne.x += a; ne.y += b; }

void resize(int d)

{ ne.x += (ne.x - sw.x) \* (d - 1); ne.y += (ne.y - sw.y) \* (d - 1); }

void draw( );

};

rectangle::rectangle(point a, point b)

{

sw = a; ne = b;

if((!on\_screen(sw.x, sw.y)) || (!on\_screen(ne.x,ne.y)))

throw OffScreen("Фигура №" + std::to\_string(id) + " при создании оказалась вне экрана.", point((east().x-west().x)/2, (north().y-south().y)/2));

}

void rectangle::draw()

{

try

{

put\_line(nwest( ), ne); put\_line(ne, seast( ));

put\_line(seast( ), sw); put\_line(sw, nwest( ));

}

catch(OffScreen &ex)

{

std::cout << "Для фигуры №" << id << " " << ex.what() << ex.crnds() << "\n\n";

resize(-2);

try

{

put\_line(nwest( ), ne); put\_line(ne, seast( ));

put\_line(seast( ), sw); put\_line(sw, nwest( ));

std::cout << "Фигура №" << id << " была уменьшена в два раза для корректного вывода\n";

}

catch(OffScreen)

{

resize(2);

throw OffScreen("Фигура №" + std::to\_string(id) + " при изменении оказалась вне экрана.", point((east().x-west().x)/2, (north().y-south().y)/2));

// если изменение параметров не помогло, возбуждаем ошибку с другим параметром - точкой центра ошибочной фигуры

}

}

}

void up(shape &p, const shape &q) // поместить p над q

{ //Это ОБЫЧНАЯ функция, не член класса! Динамическое связывание!!

point n = q.north( );

point s = p.south( );

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

}

Файл **exceptions.h:**

#include <exception>

#include <string>

// ошибка: точка вне экрана

struct OffScreen : std::exception {

public:

OffScreen(const std::string& s, point err) : msg\_(s), point\_(err) { } // ошибка при изменении фигуры

virtual char const\* what() const noexcept { return msg\_.c\_str(); }

virtual point get\_point() { return point\_; }

virtual std::string crnds() { return "(" + std::to\_string(point\_.x) + ", " + std::to\_string(point\_.y) + ")"; }

private:

std::string msg\_; // сообщение ошибки

point point\_; // точка, вызвавшая ошибку

};

Файл **screen.h:**

//=== Файл screen.h -- поддержка работы с экраном

const int XMAX = 120; //Размер экрана

const int YMAX = 50;

class point { //Точка на экране

public:

int x, y;

point(int a = 0, int b = 0) : x(a), y(b) { }

};

// Набор утилит для работы с экраном

void put\_point(int a, int b); // Вывод точки (2 варианта)

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

void put\_line(int, int, int, int); // Вывод линии (2 варианта)

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

void screen\_init(); // Создание экрана

void screen\_destroy(); // Удаление

void screen\_refresh(); // Обновление

void screen\_clear(); // Очистка

char\* screen\_copy(); // копия текущего экрана