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 26**

|  |  |
| --- | --- |
| Выполнил студент гр.9308: | Яловега Н.В. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2020 г.

**Оглавление**

[​ Введение 3](#__RefHeading___Toc280_3584487243)

[​ 1. Задание 3](#__RefHeading___Toc306_2067246640)

[2. Формализация задания 3](#__RefHeading___Toc282_3584487243)

[​ 3. Результаты эксперимента 4](#__RefHeading___Toc320_2067246640)

[​ Вывод 7](#__RefHeading___Toc326_2067246640)

[​ Список используемых источников 8](#__RefHeading___Toc328_2067246640)

[​ Приложение 1 (Исходный текст программы) 9](#__RefHeading___Toc330_2067246640)

## Введение

* 1. 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

Для выполнения задания было решено предусмотреть следущие особые случаи:

1. - некорректные параметры при формировании фигуры;
2. - непопадание точки на экран;
3. - нехватка места на экране для размещения фигуры

Был разработан класс error\_figure, являющийся производным от классов rotatable и reflectable. Если объект повреждается вследствие неверных действий со стороны пользователя или программы, которые приводят к исключительным ситуациям, объект удаляется и заменяется на error\_figure. Особенность его в том, что он реализован так, что не вызывает исключений (все еще может вызвать исключение вследствие недостатка памяти). Это помогает пользователю понять, где и что пошло не так.

Для фиксации особых случаев был создан класс out\_of\_screen.

Конструкторы классов фигур содержат блоки контроля, если объект создается вне экрана, то создается объект типа out\_of\_screen и вызывается исключение, которое передается выше в main. Любой объект в функции main проходит этап проверки благодаря блокам контроля, и если при создании объекта что-то пошло не так, то этот объект удаляется, и фигура подменяется запасной фигурой — знаком ошибки error\_figure. Если над этой фигурой предполагалось выполнять какие-либо изменения, то будет выводится сообщение о том, что сделать это над данной фигурой невозможно.

В программе имеется следующая цепочка вызовов функций:

main( ) → screen\_refresh( ) → rectangle ∷ draw ( ) → put\_line(a, b) → put\_point(x, y) → on\_screen(x, y).

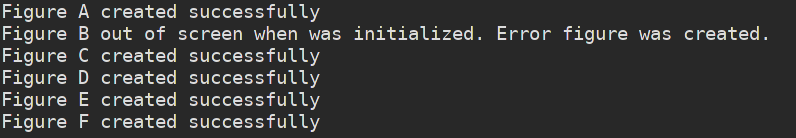
Выход точки за пределы буферного массива SCREEN (экрана) выявляется функцией on\_screen( ). Блок контроля вокруг вызова этой функции (или вызывающей ее put\_point) не имеет смысла: на этом уровне ничего, кроме выдачи сообщения об ошибке, сделать нельзя, а такое сообщение можно выдать и непосредственно, не прибегая к механизму throw— catсh. В то же время блок контроля внутри функции rectangle ∷ draw() позволит локализовать ошибку при выводе прямоугольника — и попробовать изменить его размер. Если же изменение размера не помогло исправить ситуацию, то ошибка передается выше по иерархии в функцию screen\_refresh( ). На данном уровне мы можем вывести знак ошибки error\_figure в правом нижнем углу экрана, который будет означать то, что c картинки были убраны фигуры, которые вызвали ошибку.

## 3. Результаты эксперимента

Для проверки обработки исплючительных ситуаций возьмем фигуру, параметры которой заданы некорректно, и две фигуры, которые не поместились на экран при изменениях.

В качестве фигуры с неверными параметрами будет выступать шляпа. Она создается второй, поэтому ее имя должно быть B.

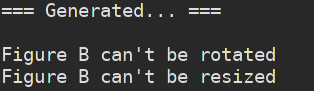
На рисунке 1 видем сообщение о том, что фигура B была за экраном при ее инициализации, поэтому вместо нее была использована специальная фигура ошибки. Это видно на рисунке 2.

Рисунок 1: Ошибка при инициализации

Фигура ошибки рисуется с использованием знаков «!», чтоб было наглядно видно ее отличие от остальных фигур.

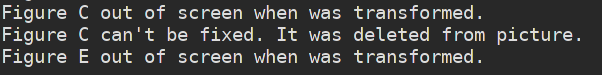
Рисунок 2: Фигура ошибки

В программе предполагалось, что мы будем вращать фигуру и изменять ее размер. При попытке сделать это с фигурой ошибки выводятся сообщения, которые показаны на рисунке 3

Рисунок 3: Сообщения о невозможности изменить фигуру

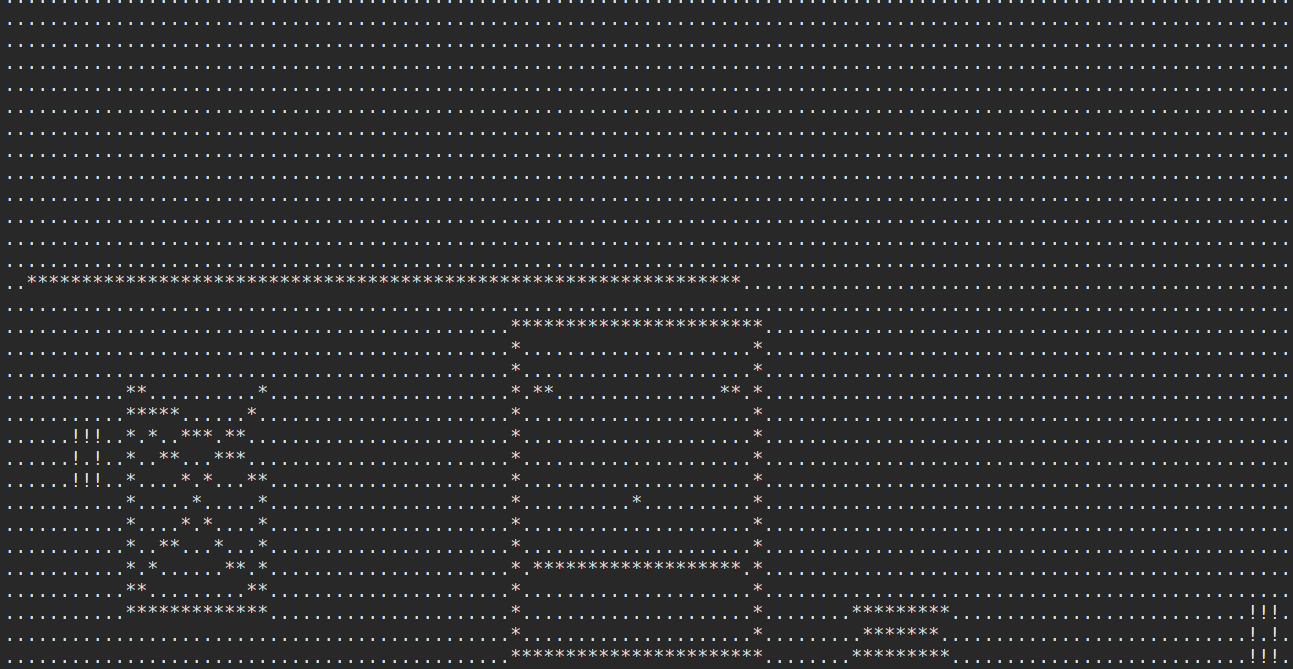
Теперь попробуем специально испортить фигуры. Для этого будем использовать правый рог и шишак, которые имеют имена С и E.

Получим следущие сообщения об ошибке:

Рисунок 4: Сообщения о выходе фигур за границы экрана

Из рисунка 4 видим, что фигура С вышла за экран при ее изменении. Следующее сообщение говорит то, что фигуру не удалось починить так, чтоб она помещалась на экран и она была удалена c рисунка. На рисунке 5 видим, что появился специальный знак ошибки в правом нижнем углу экрана, который символизирует о том, что при составлении картинки были удалены некоторые фигуры.

Так же, из рисунка 5 было видно, что фигура D была уменьшена в размере для того, чтоб она поместилась на экран.

Рисунок 5: Фигуры после преобразований

Итоговую картинку можно увидеть на рисунке 6.

Рисунок 6: Итоговая картинка

## Вывод

При выполнении данной работы были получены практические навыки по работе с наследованием классов, по работе с исключительными ситуациями. Были изучены понятия производного класса, полиморфизма, виртуальных классов, механизмы работы с классами, механизм исключительных ситуаций.

## Список используемых источников

1. Колинько П.Г. Пользовательские контейнеры / Методические указания по дисциплине «Алгоритмы и структуры данных» - Санкт-Петербург: СПбГЭТУ «ЛЭТИ», 2020.

## Приложение 1 (Исходный текст программы)

**screen.h**

/\*

Поддержка работы с экраном

\*/

const int XMAX = 120;

const int YMAX = 50;

struct point

{

int x, y;

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

};

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

----------------------------------------------------------------------------------------------------------------

**errors.h**

class out\_of\_screen : public std::exception

/\*

Класс ошибки непопадания фигуры на экран

\*/

{

std::string what\_str;

public:

out\_of\_screen(const std::string &what\_str) noexcept : what\_str(what\_str) {}

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

};

----------------------------------------------------------------------------------------------------------------

**shape.h**

#include <list>

#include <iostream>

using namespace std;

char screen[YMAX][XMAX];

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

bool create\_error = false;

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(create\_error)

screen[abs(b) % YMAX][abs(a) % XMAX] = err;

else if (on\_screen(a, b))

screen[b][a] = black;

else

// бросаем исключение в put\_line

throw out\_of\_screen(" out of screen when was transformed.");

}

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(out\_of\_screen)

{

throw; // ничего не можем сделать, передаем исключение в draw

}

}

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

cout << x;

cout << '\n';

}

}

struct shape

{

char id; // имя фигуры

static int count; // счетчик фигур

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

// Фигура присоединяется к списку

shape() {shapes.push\_back(this); id = 'A' + count; ++count;}

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); --count;}

};

int shape::count = 0; // установка счетчика фигур в 0

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

class rotatable : virtual public shape

//Фигуры, пригодные к повороту

{

public:

virtual void rotate\_left() = 0;

virtual void rotate\_right() = 0;

};

class reflectable : virtual public shape

//Фигуры пригодные к зеркальному отражению

{

public:

virtual void flip\_horisontally() = 0;

virtual void flip\_vertically() = 0;

};

/\*

Специальная фигура - знак ошибки

\*/

class error\_figure : public rotatable, public reflectable

{

point s;

public:

error\_figure(): s(point(3, 3)) {}

error\_figure(point a): s(a) {}

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 move(int, int);

void draw();

/\*

При использовании методов изменения фигуры выводим сообщение о том,

что данное действие над испорченной фигурой совершить невозможно

\*/

void resize(int){cout << "Figure " << id << " сan't be resized\n";}

void rotate\_left(){cout << "Figure " << id << " сan't be rotated\n";}

void rotate\_right(){cout << "Figure " << id << " сan't be rotated\n";}

void flip\_vertically(){cout << "Figure " << id << " сan't be flipped\n";}

void flip\_horisontally(){cout << "Figure " << id << " сan't be flipped\n";}

};

void error\_figure::move(int dx, int dy)

{

s.x += dx;

s.y += dy;

}

void error\_figure::draw()

{

create\_error = true;

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

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

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

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

create\_error = false;

}

void shape\_refresh()

{

bool is\_error = false;

screen\_clear();

for(auto p: shape::shapes)

{

try

{

p->draw();

}

catch(out\_of\_screen)

{

// если изменения параметров фигур в draw не помогли решить проблему,

// то не рисуем эту фигуру и выводим знак ошибки

cout << "Figure " << p->id << " can't be fixed. It was deleted from picture. \n";

is\_error = true;

}

}

if (is\_error)

{

error\_figure s = error\_figure(point(XMAX - 3, 2));

s.draw();

}

screen\_refresh();

}

// Линия

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.

\*/

{

line(const line&);

line(const line&&);

line& operator=(const line&);

line& operator=(line&&);

protected:

point w, e;

public:

line(point, point);

line(point, int);

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 e; }

point west() const { return w; }

point neast() const { return e; }

point seast() const { return e; }

point nwest() const { return w; }

point swest() const { return w; }

void move(int, int);

void draw();

void resize(int);

};

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

{

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

throw out\_of\_screen("Figure " + string(1, id) + " out of screen when was initialized. Error figure was created.");

};

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 out\_of\_screen("Figure " + string(1, id) + " out of screen when was initialized. Error figure was created.");

};

void line::draw()

{

try

{

put\_line(w, e);

}

catch(out\_of\_screen &err)

{

// сообщаем об ошибке

cout << "Figure " << id << err.what() << "\n";

// пробуем немного изменить фигуру

resize(-2);

// пробуем отрисовать фигуру еще раз

try

{

put\_line(w, e);

}

catch(out\_of\_screen)

{

resize(2);

throw; // если изменение параметров не помогло, передаем ошибку выше

}

}

}

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

{

w.x += dx; w.y += dy; e.x += dx; e.y += dy;

}

void line::resize(int r)

{

if (r > 1)

{

e.x += (e.x - w.x) \* (r - 1);

e.y += (e.y - w.y) \* (r - 1);

}

else if (r < 0)

{

e.x = w.x + (e.x - w.x) / (- r);

e.y = w.y + (e.y - w.y) / (- r);

}

}

// Прямоугольник

class rectangle: public rotatable

/\*

nw-----n-----ne

| |

| |

w c e

| |

| |

sw-----s-----se

\*/

{

rectangle(const rectangle&);

rectangle(const rectangle&&);

rectangle& operator=(const rectangle &);

rectangle& operator=(rectangle &&);

protected:

point sw, ne;

public:

rectangle(point, point);

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();

void rotate\_left();

void move(int, int);

void resize(int);

void draw();

};

rectangle::rectangle(point a, point b)

{

if (a.x <= b.x)

if (a.y <= b.y)

{

sw = a;

ne = b;

}

else

{

sw = point(a.x, b.y);

ne = point(b.x, a.y);

}

else

if (a.y <= b.y)

{

sw = point(b.x, a.y);

ne = point(a.x, b.y);

}

else

{

sw = b;

ne = a;

}

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

throw out\_of\_screen("Figure " + string(1, id) + " out of screen when was initialized. Error figure was created.");

}

void rectangle::draw()

{

try

{

point nw(sw.x, ne.y);

point se(ne.x, sw.y);

put\_line(nw, ne);

put\_line(ne, se);

put\_line(se, sw);

put\_line(sw, nw);

}

catch(out\_of\_screen &err)

{

cout << "Figure " << id << err.what() << "\n";

resize(-2);

try

{

point nw(sw.x, ne.y);

point se(ne.x, sw.y);

put\_line(nw, ne);

put\_line(ne, se);

put\_line(se, sw);

put\_line(sw, nw);

}

catch(out\_of\_screen)

{

resize(2);

throw; // если изменение параметров не помогло, передаем ошибку выше

}

}

}

void rectangle::resize(int r)

{

if (r > 1)

{

ne.x += (ne.x - sw.x) \* (r - 1);

ne.y += (ne.y - sw.y) \* (r - 1);

}

else if (r < 0)

{

ne.x = sw.x + (ne.x - sw.x) / (- r);

ne.y = sw.y + (ne.y - sw.y) / (- r);

}

}

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

{

sw.x += dx;

sw.y += dy;

ne.x += dx;

ne.y += dy;

}

void rectangle::rotate\_right()

{

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

sw.x = ne.x - h \* 2;

ne.y = sw.y + w / 2;

}

void rectangle::rotate\_left()

{

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

ne.x = sw.x + h \* 2;

ne.y = sw.y + w / 2;

}

/\*

Реализация трапеции с косым крестом без наследования от отдельных фигур

\*/

class crossed\_trapezoid : public rotatable, public reflectable

{

crossed\_trapezoid(const crossed\_trapezoid&);

crossed\_trapezoid(const crossed\_trapezoid&&);

crossed\_trapezoid& operator=(const crossed\_trapezoid &);

crossed\_trapezoid& operator=(crossed\_trapezoid &&);

protected:

point a, b, c, d;

public:

crossed\_trapezoid(point, int, point, int);

point north() const { return point((swest().x + seast().x) / 2, nwest().y); }

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

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

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

point neast() const { return point(max(a.x, max(b.x, max(c.x, d.x))), max(a.y, max(b.y, max(c.y, d.y)))); }

point seast() const { return point(max(a.x, max(b.x, max(c.x, d.x))), min(a.y, min(b.y, min(c.y, d.y)))); }

point nwest() const { return point(min(a.x, min(b.x, min(c.x, d.x))), max(a.y, max(b.y, max(c.y, d.y))));}

point swest() const { return point(min(a.x, min(b.x, min(c.x, d.x))), min(a.y, min(b.y, min(c.y, d.y)))); }

void rotate\_left();

void rotate\_right();

void flip\_horisontally();

void flip\_vertically();

void move(int, int);

void resize(int);

void draw();

};

crossed\_trapezoid :: crossed\_trapezoid (point a\_, int lena, point b\_, int lenb)

/\*

Конструктор принимает точку sw и длину основания,

точку nw и длину основания. Этот набор данных может

задавать трапецию любого вида.

Для хранения вычисляются координаты 4 точек.

\*/

{

a = a\_;

b = b\_;

c.x = b.x + lenb; c.y = b.y;

d.x = a.x + lena; d.y = a.y;

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

{

throw out\_of\_screen("Figure " + string(1, id) + " out of screen when was initialized.");

}

}

void crossed\_trapezoid::rotate\_right()

{

int x0 = (neast().x + seast().x + nwest().x + swest().x)/4;

int y0 = (neast().y + seast().y + nwest().y + swest().y)/4;

int x, y;

x = a.x; y = a.y;

a.x = x0 + (y - y0)\*2;

a.y = y0 - (x - x0)/2;

x = b.x; y = b.y;

b.x = x0 + (y - y0)\*2;

b.y = y0 - (x - x0)/2;

x = c.x; y = c.y;

c.x = x0 + (y - y0)\*2;

c.y = y0 - (x - x0)/2;

x = d.x; y = d.y;

d.x = x0 + (y - y0)\*2;

d.y = y0 - (x - x0)/2;

}

void crossed\_trapezoid::rotate\_left()

{

int x0 = (neast().x + seast().x + nwest().x + swest().x)/4;

int y0 = (neast().y + seast().y + nwest().y + swest().y)/4;

int x, y;

x = a.x; y = a.y;

a.x = x0 - (y - y0)\*2;

a.y = y0 + (x - x0)/2;

x = b.x; y = b.y;

b.x = x0 - (y - y0)\*2;

b.y = y0 + (x - x0)/2;

x = c.x; y = c.y;

c.x = x0 - (y - y0)\*2;

c.y = y0 + (x - x0)/2;

x = d.x; y = d.y;

d.x = x0 - (y - y0)\*2;

d.y = y0 + (x - x0)/2;

}

void crossed\_trapezoid :: flip\_horisontally()

{

swap(a.y, b.y);

swap(d.y, c.y);

}

void crossed\_trapezoid :: flip\_vertically()

{

swap(a.x, b.x);

swap(c.x, d.x);

}

void crossed\_trapezoid :: move(int dx, int dy)

{

a.x += dx; a.y += dy;

b.x += dx; b.y += dy;

c.x += dx; c.y += dy;

d.x += dx; d.y += dy;

}

void crossed\_trapezoid :: resize(int r)

{

if (r > 1)

{

b.x += (b.x - a.x) \* (r - 1);

b.y += (b.y - a.y) \* (r - 1);

c.x += (c.x - a.x) \* (r - 1);

c.y += (c.y - a.y) \* (r - 1);

d.x += (d.x - a.x) \* (r - 1);

}

else if (r < 0)

{

b.x = a.x + (b.x - a.x) / (- r);

b.y = a.y + (b.y - a.y) / (- r);

c.x = a.x + (c.x - a.x) / (- r);

c.y = a.y + (c.y - a.y) / (- r);

d.x = a.x + (d.x - a.x) / (- r);

}

}

void crossed\_trapezoid :: draw()

{

try

{

put\_line(a, b);

put\_line(b, c);

put\_line(c, d);

put\_line(a, d);

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

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

}

catch(out\_of\_screen &err)

{

// сообщаем об ошибке

cout << "Figure " << id << err.what() << "\n";

resize(-2);

try

{

put\_line(a, b);

put\_line(b, c);

put\_line(c, d);

put\_line(a, d);

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

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

}

catch(out\_of\_screen)

{

resize(2);

throw; // если изменение параметров не помогло, передаем ошибку выше

}

}

}

class face: public rectangle

{

int w, h;

line l\_eye, r\_eye, mouth;

public:

face(point, point);

void draw();

void move(int, int);

void resize(int) {};

};

face :: face (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)

{}

void face :: draw()

{

rectangle :: draw();

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

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

put\_point(point(a, b));

}

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

{

rectangle :: move(a, b);

l\_eye.move(a, b);

r\_eye.move(a, b);

mouth.move(a, b);

}

/\*

Функции размещения фигур относительно друг друга

\*/

void down(shape\* p, const shape\* q)

// Поместить p над q

{

p->move(q->south().x - p->north().x, q->south().y - p->north().y - 1);

}

void left\_up(shape\* p, const shape\* q)

// Поместить p слева над q

{

p->move(q->nwest().x - p->swest().x, q->nwest().y - p->swest().y + 1);

}

void right\_up(shape\* p, const shape\* q)

// Поместить p справа над q

{

p->move(q->neast().x - p->seast().x, q->nwest().y - p->swest().y + 1);

}

void right\_down(shape\* p, const shape\* q)

// Поместить p справа под q

{

p->move(q->east().x - p->west().x, q->swest().y - p->nwest().y);

}

void left\_down(shape\* p, const shape\* q)

// Поместить p справа под q

{

p->move(q->west().x - p->east().x, q->swest().y - p->nwest().y);

}

void up(shape\* p, const shape\* q)

{

p->move(q->north().x - p->south().x, q->north().y - p->south().y + 1);

}

----------------------------------------------------------------------------------------------------------------

**main.cpp**

#include <iostream>

#include <string>

#include <exception>

#include "screen.h"

#include "errors.h"

#include "shape.h"

int main()

{

screen\_init();

// объявление набора фигур

shape \*brim;

try

{

brim = new line(point(5, 18), 17);

std::cout << "Figure " << brim->id << " created successfully \n";

}

catch (bad\_init &e)

{

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

brim = new error\_figure(e.center);

} UNXERR

rotatable \*hat;

try

{

hat = new rectangle(point(-10, -10), point(69, 25)); // Изначально испорченная фигура

std::cout << "Figure " << hat->id << " created successfully \n";

}

catch (bad\_init &e)

{

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

hat = new error\_figure(e.center);

} UNXERR

rotatable \*right\_horn;

try

{

right\_horn = new crossed\_trapezoid(point(10, 25), 10, point(10, 28), 6);

std::cout << "Figure " << right\_horn->id << " created successfully \n";

}

catch (bad\_init &e)

{

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

right\_horn = new error\_figure(e.center);

} UNXERR

rotatable \*left\_horn;

try

{

left\_horn = new crossed\_trapezoid(point(10, 5), 10, point(14, 8), 6);

std::cout << "Figure " << left\_horn->id << " created successfully \n";

}

catch (bad\_init &e)

{

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

left\_horn = new error\_figure(e.center);

} UNXERR

reflectable \*shishak;

try

{

shishak = new crossed\_trapezoid(point(80, 5), 16, point(85, 10), 6);

std::cout << "Figure " << shishak->id << " created successfully \n";

}

catch (bad\_init &e)

{

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

shishak = new error\_figure(e.center);

} UNXERR

rotatable \*f;

try

{

f = new face(point(49, 1), point(71, 16));

std::cout << "Figure " << f->id << " created successfully \n";

}

catch(bad\_init &e)

{

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

f = new error\_figure(e.center);

} UNXERR

shape\_refresh();

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

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

// подготовка к сборке

hat->rotate\_right();

hat->resize(2);

brim->resize(4);

// фигура, испорченная трансформированием

right\_horn->move(0, 40);

right\_horn->resize(2);

right\_horn->rotate\_left();

left\_horn->resize(2);

left\_horn->rotate\_right();

// фигура, испорченная трансформированием, но пригодная для изменения

shishak->flip\_horisontally();

shishak->move(0, -7);

shape\_refresh();

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

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

// сборка изображения

up(brim, f);

up(hat, brim);

right\_up(right\_horn, brim);

left\_up(left\_horn, brim);

up(shishak, hat);

shape\_refresh();

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

std::cin.get(); //Смотреть результат

std::cout << "\nScreen contains " << shape::shapes.size() << " elements (nose is not an element!)\n";

screen\_destroy();

delete f;

delete brim;

delete hat;

delete right\_horn;

delete left\_horn;

delete shishak;

return 0;

}