

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:

Колинко П.Г.

Санкт-Петербург, 2020 г.

Оглавление

Введение.....	3
1. Задание.....	3
2. Формализация задания.....	3
3. Результаты эксперимента.....	5
Вывод.....	8
Список используемых источников.....	9
Приложение 1 (Исходный текст программы).....	10

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

Для выполнения задания было решено предусмотреть следующие особые случаи:

- некорректные параметры при формировании фигуры;
- непопадание точки на экран;
- 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 некорректных параметров при формировании фигуры был создан класс `bad_init`

Конструкторы классов фигур содержат блоки контроля, если объект создается вне экрана, то создается объект типа `bad_init` 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 выводиться сообщение о том, что сделать это над данной фигурой невозможно.

Для фиксации непопадания точки на экран и нехватки места на экране для размещения фигуры был создан класс `out_of_screen`

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— catch`. 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 удалить испорченную фигуру и вывести знак ошибки `error_figure` в правом нижнем углу экрана, который будет означать то, что картинка была подвержена изменению вследствие удаления фигуры или фигур.

Для фиксации неизвестных ошибок в конце каждого блока контроля стоят `catch(...)`.

3. Результаты эксперимента

Для проверки обработки исключительных 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 В.

```
Figure A created successfully
B figure out of screen when was initialized. Error figure was created.
Figure C created successfully
Figure D created successfully
Figure E created successfully
Figure F created successfully
```

Рисунок 1: Ошибка при инициализации

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



Рисунок 2: Фигура ошибки

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

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

```
=== Generated... ===  
B figure can't be rotated  
B figure can't be resized
```

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 Е.

Получим следующие сообщения об ошибке:

```
C figure out of screen when was transformed.  
C figure can't be fixed. It was deleted.  
E figure out of screen when was transformed.
```

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. На рисунке 5 видим, что появился специальный знак ошибки в правом нижнем углу экрана, который символизирует о том, что при составлении фигуры 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.

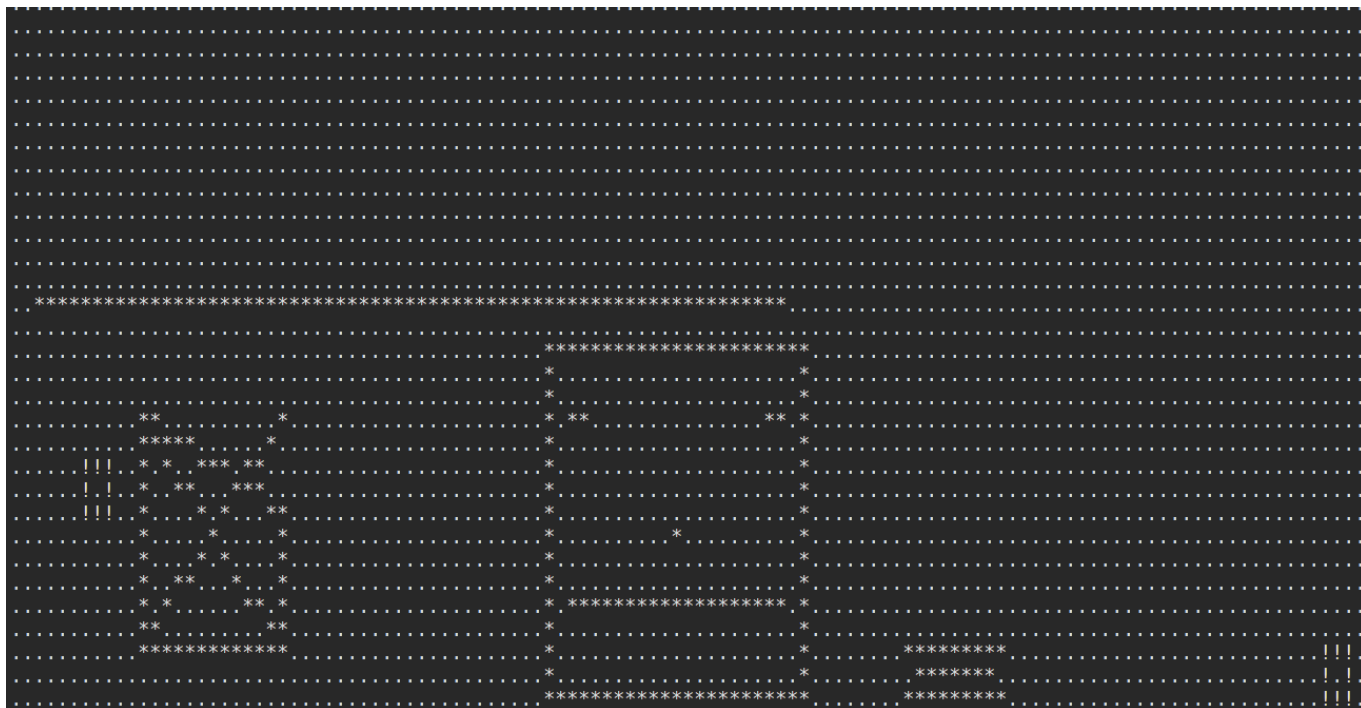


Рисунок 5: Фигуры после преобразований

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. Колинко 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

```
struct bad_init : public std::exception
/*
    Класс ошибки создания фигуры
*/
{
    char id;
```

```

    point center;

    bad_init(int id, point center) noexcept : id(id), center(center){}

    const char* what() const noexcept {return " figure out of screen when was initialized.
Error figure was created.\0"; }

};

struct out_of_screen : public std::exception

/*
    Класс ошибки перемещения фигуры
*/

{

    const char* what() const noexcept {return " figure out of screen when was transformed.\
0"; }

};

```

shape.h

```

#include <list>

#include <iostream>

#define UNXERR catch(...) {cout << "\nUNEXPECTED ERROR\n"; cin.get(); throw;}

using namespace std;

char screen[YMAX][XMAX];

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

bool create_error = false;

bool is_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();  
}
```

```
void put_line(int x0, int y0, int x1, int y1)  
/* Алгоритм Брезенхэма для прямой:
```

рисование отрезка прямой от (x_0, y_0) до (x_1, y_1) .

Уравнение прямой: $b(x-x_0) + a(y-y_0) = 0$.

Минимизируется величина $\text{abs}(\text{eps})$, где $\text{eps} = 2*(b(x-x_0)) + a(y-y_0)$.

```
*/  
{  
    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  
    } UNXERR  
}
```

```
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() = default;
}

```

```
};
```

```
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(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 << id << " figure can't be resized\n";}
void rotate_left(){cout << id << " figure can't be rotated\n";}
void rotate_right(){cout << id << " figure can't be rotated\n";}
void flip_vertically(){cout << id << " figure can't be flipped\n";}
void flip_horisontally(){cout << id << " figure can't be flipped\n";}

```

```

~error_figure(){shapes.remove(this); --count;}

```

```

};

```

```

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()
```

```
{  
    screen_clear();  
    for(list <shape*> p = shape::shapes; !p.empty(); p.pop_front())  
    {  
        try  
        {  
            p.front()->draw();  
        }  
        catch(out_of_screen)  
        {  
            // если изменения параметров фигур в draw не помогли решить проблему,  
            // удаляем эту фигуру и выводим знак ошибки  
            cout << p.front()->id << " figure can't be fixed. It was deleted. \n";  
            shape::shapes.remove(p.front());  
            is_error = true;  
        } UNXERR  
    }  
}
```

```

// если есть удаленная фигура, появляется специальный знак
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);
```

{ return point((w.x + e.x) / 2, e.y < w.y ? w.y : e.y); }	point north() const
{ return point((w.x + e.x) / 2, e.y < w.y ? e.y : w.y); }	point south() const
{ return e; }	point east() const
{ return w; }	point west() const
{ return e; }	point neast() const
{ return e; }	point seast() const
const { return w; }	point nwest()
{ return w; }	point swest() const
int);	void move(int,
void resize(int);	void draw();
~line(){shapes.remove(this); --count;}	
};	
line::line (point a, point b) : w(a), e(b)	
{	
if((!on_screen(w.x, w.y)) (!on_screen(e.x,e.y)))	
{	
bad_init i(id, point(10, 10));	
this->~line();	
throw i;	
}	
};	

```

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)))
    {
        bad_init i(id, point(10, 10));
        this->~line();
        throw i;
    }
};

```

```

void line::draw()
{
    try
    {
        put_line(w, e);
    }
    catch(out_of_screen &err)
    {
        // сообщаем об ошибке
        cout << id << err.what() << "\n";
        // пробуем немного изменить фигуру

        resize(-2);

        // пробуем отрисовать фигуру еще раз
        try
        {
            put_line(w, e);
        }
    }
}

```

```

        catch(out_of_screen)
        {
            throw; // если изменение параметров не помогло, передаем ошибку выше
        }
    } UNXERR

}

```

```

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
```

```
  point south() const
```

```
  point east() const
```

```
  point west() const
```

```
  point neast() const
```

```
  point seast() const
```

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

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

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

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

```
{ return ne; }
```

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

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

```
point nwest()
```

```
{ return sw; }
```

```
point swest() const
```

```
rotate_right();
```

```
void
```

```
void rotate_left();
```

```
void move(int,
```

```
int);
```

```
void resize(int);
```

```
void draw();
```

```
~rectangle(){shapes.remove(this); --count;}
```

```
};
```

```
rectangle::rectangle(point a, point b)
```

```
{
```

```
if (a.x <= b.x)
```

```
{
```

```
if (a.y <= b.y)
```

```
sw = a;
```

```
ne = b;
```

```
}
```

```
else
```

```
{
```

```
sw =
```

```
ne =
```

```
}
```

```
else
```

```
point(a.x, b.y);
```

```
point(b.x, a.y);
```

	if (a.y <= b.y)
{	
point(b.x, a.y);	sw =
	ne =
point(a.x, b.y);	
	}
	else
{	
	sw = b;
	ne = a;
	}
if((!on_screen(sw.x, sw.y)) (!on_screen(ne.x, ne.y)))	
{	
bad_init i(id, point(10, 10));	
this->~rectangle();	
throw i;	
}	
}	
void rectangle::draw()	
{	
try	
{	
ne.y);	point nw(sw.x,
sw.y);	point se(ne.x,
	put_line(nw, ne);
	put_line(ne, se);
	put_line(se, sw);


```
put_line(sw, nw);
```

```
}  
catch(out_of_screen &e)  
{  
    cout << id << e.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)  
    {  
        throw; // если изменение параметров не помогло, передаем ошибку выше  
    }  
} UNXERR  
}  
  
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_horizontally();
void flip_vertically();
void move(int, int);
void resize(int);
void draw();

```

```

~crossed_trapezoid() {shapes.remove(this); --count;}

```

```

};

```

```

crossed_trapezoid :: crossed_trapezoid (point a_, int lena, point b_, int lenb)

```

```

/*

```

Конструктор принимает точку sw и длину основания,
точку pw 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)))

```

```

{

```

```

bad_init i(id, point(10, 10));

```

```

this->~crossed_trapezoid();

```

```

throw i;

```

```

}

```

```
}
```

```
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 &e)
    {
        // сообщаем об ошибке
        cout << id << e.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)
        {

```



```

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

```

```

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();  
исходный набор //Смотреть
```

```
// 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;
```

```
}
```