Программа представляет из себя текстовое поле размерами XMAX * YMAX, на котором с помощью набора из двух символов («.» - белый, «*» - черный) чертятся фигуры. Все фигуры наследуются от класса shape и как минимум имеют конструктор, методы draw, move и resize (названия сами за себя говорят). Все созданные фигуры добавляются в статический список static list<shape*> shapes.

Программа не отслеживает исключительные ситуации (непопадание (полное или частичное) фигуры или точки на экран, некорректные аргументы в конструкторе или методе).

Доработать программу так, чтобы:

1. Все классы фигур имели деструктор (для класса shapes надо использовать виртуальный) который бы также удалял данную фигуру из списка static list<shape*> shapes.
2. Задать список классов для ошибок при непопадании точки или фигуы на экран, при невозможности перемещения фигуры (когда фигура после перемещения полностью или частично не попадает на экран), также при увеличении (когда фигура после увеличения полностью или частично не попадает на экран). Некоторые фигуры могут поворачиваться и/или отражаться, и для этого тоже надо учесть класс ошибок. Все эти классы должны наследоваться от класса exception из стандартной библиотеки exception.
3. Каждый класс фигур надо доработать следующим образом:
   3.1. При некорректных данных конструктора, если фигура полностью не попадает на экран, её надо удалить и выдать краткое сообщение об ошибке, содержащее информацию о фигуре. Если попадает на экран неполностью, фигуру надо обрезать и выдать краткое сообщение об ошибке, содержащее информацию об ошибке, фигуре и методе её исправления.
   3.2. Если при перемещении, увеличении, повороте или отражении фигура не попадает на экран, то это действие надо отменить и выдать сообщение об ошибке, содержащее информацию об ошибке, фигуре и методе её исправления.
   3.3. Все сообщения должны выводится в консоль, аварийного прекращения программы быть не должно.

Пример обработки исключительной ситуации в конструкторе:

```
struct constructor_exc : std::exception {
       constructor_exc(const std::string& s) : std::exception(s.c_str()) { }
};

struct shape {
    static list<shape*> shapes;
    shape() { shapes.push_back(this); }
    // прочие методы
};

class line : public shape {
protected:
    point w, e;
public:
    line(point a, point b) : w(a), e(b) {
        try {
            if (/*Исключительная ситуация*/)
                throw constructor_exc("Описание исключительной ситуации");
        }
        catch (miss_point &ex) {
            cout << ex.what();
            // Обработка исключительной ситуации.
        }
    }

};
```

Исходный код:

Файл screen.h

```
#ifndef SCREEN_H_INCLUDED
#define SCREEN_H_INCLUDED
//=== Файл screen.h - поддержка работы с экраном
const int XMAX = 120; // Размер экрана – ширина (длина строки)
const int YMAX = 40;  //               - высота (к-во строк)
struct point {       // Точка на экране
    int x, y;
    point(int a = 0, int b = 0) : x(a), y(b) {  }
};//=== Файл screen.h - поддержка работы с экраном
// 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); }//=== Файл screen.h - поддержка работы
с экраном
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);
}
extern void screen_init();           // Создание экрана
extern void screen_destroy();     // Удаление экрана
extern void screen_refresh();     // Обновление
extern void screen_clear();// Очистка
//————————————————————————————

#endif // SCREEN_H_INCLUDED
```

Файл shape.h

```
//======== Файл shape.h -- библиотека фигур =========
#include <list>
#include <iostream>
using namespace std;
//==1. Поддержка экрана в форме матрицы символов ==
char screen[YMAX][XMAX];
enum color { black = '*', white = '.' };
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 на экран
{
    return 0 <= a && a < XMAX && 0 <= b && b < YMAX;
}

void put_point(int a, int b)
{
    if (on_screen(a, b)) screen[b][a] = black;
}

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

```
*/
{
    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;
    }
}

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 list<shape*> shapes;// Список фигур (один на все фигуры!)
    shape() { shapes.push_back(this); } //Фигура присоединяется к списку
    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
};

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

void shape_refresh() // Перерисовка всех фигур на экране
{
    screen_clear();
    for (auto p : shape::shapes) p->draw();
    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
                // к зеркальному отражению
public:
```

```
    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) : w(a), e(b) { };
    line(point a, int L) : w(point(a.x + L - 1, a.y)), e(a) {  };
    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() { put_line(w, e); }
    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);
    }
};

// Прямоугольник
class rectangle : public rotatable {
protected:
    point sw, ne;
public:
    rectangle(point, point);
    /* nw ------ n ------ ne
        |                  |
        |                  |
       w        c          e
        |                  |
        |                  |
       sw ------ s ------ se */
    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(sw.x, (sw.y + ne.y) / 2); }
    point west() const { return point(ne.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;
        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)
{
    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;
    }
}
void rectangle::draw()
{
    put_line(nwest(), ne);
    put_line(ne, seast());
    put_line(seast(), sw);
    put_line(sw, nwest());
}

void up(shape& p, const shape& q) // поместить p над q
{      //Функция присоединения сверху: СВОБОДНАЯ, не член класса!
    point n = q.north();
    point s = p.south();
    p.move(n.x - s.x, n.y - s.y + 1);
}
```

r_triangle.h

```
#ifndef RTRIANG
#define RTRIANG

class r_triangle : public rectangle, public reflectable {
private:
       bool flip_h, flip_v;
public:
       r_triangle(point a, point b) : rectangle(a, b), flip_h(false), flip_v(false) { }
       void draw() {
              if (flip_h == flip_v) {
                     put_line(nwest(), seast());
              }
              else {
                     put_line(ne, sw);
              }
              if (flip_h) {
                     put_line(ne, nwest());
              }
              else {
                     put_line(sw, seast());
              }
              if (flip_v) {
                     put_line(seast(), ne);
              }
              else {
                     put_line(sw, nwest());
              }
       }
       void flip_horisontally() {
              flip_h = !flip_h;
       }
       void flip_vertically() {
              flip_v = !flip_v;
       }
       void rotate_right() {
```

```
		rectangle::rotate_right();
		if (flip_h == flip_v) {
			flip_v = !flip_h;
		}
		else {
			flip_h = flip_v;
		}
	}
	void rotate_left() {
		rectangle::rotate_left();
		if (flip_h == flip_v) {
			flip_h = !flip_v;
		}
		else {
			flip_v = flip_h;
		}
	}
};

#endif
```

Файл main.cpp

```
// Прикладная программа === chape.cpp ===
// Пополнение и использование библиотеки фигур
//#include "pch.h"  //связь с ОС (включить для Visual C++2017)
#include "screen.h"
#include "shape.h"
#include "r_triangle.h"

/*
// Пример: дополнительный фрагмент - полуокружность
class h_circle : public rectangle, public reflectable {
	bool reflected;
public:
	h_circle(point a, point b, bool r = true) : rectangle(a, b), reflected(r) { }
	void draw();
	void flip_horisontally() { };     // Отразить горизонтально
	void flip_vertically() { reflected = !reflected; };   // Отразить вертикально
};


void h_circle::draw()
{
	int x0 = (sw.x + ne.x) / 2;
	int y0 = reflected ? sw.y : ne.y;
	int radius = (ne.x - sw.x) / 2;
	int x = 0;
	int y = radius;
	int delta = 2 - 2 * radius;
	int error = 0;
	while (y >= 0) {
		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;
	}
}
*/
```

```
// Пример: дополнительная функция присоединения - снизу
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);
}
// Дополнительная "сборная" фигура
class myshape : public rectangle {
	//Моя фигура ЯВЛЯЕТСЯ прямоугольником
	int w, h;
	line l_eye; // левый глаз – моя фигура СОДЕРЖИТ линию
	line r_eye; // правый глаз
	line mouth; // рот

	r_triangle tr10, tr11; // added

public:
	myshape(point, point);
	void draw();
	void move(int, int);
	void resize(int) {}
};
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),
	tr10(l_eye.swest(), point(l_eye.neast().x, l_eye.neast().y + 1)), // added
	tr11(r_eye.swest(), point(r_eye.neast().x, r_eye.neast().y + 1)) // added
{
	tr10.flip_horisontally(); //added
	tr11.flip_horisontally(); //added
	down(tr10, l_eye); // added
	down(tr11, r_eye); // added
}

void myshape::draw()
{
	rectangle::draw();
	int a = (swest().x + neast().x) / 2;
	int b = (swest().y + neast().y) / 2;
	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);

	tr10.move(a, b); // added
	tr11.move(a, b); // added
}

// user's correction
void center(shape& p, const shape& q)
{
	point n((q.neast().x - q.swest().x) / 2 + q.swest().x, (q.neast().y -
q.swest().y) / 2 + q.swest().y);
	point s((p.neast().x - p.swest().x) / 2 + p.swest().x, (p.neast().y -
p.swest().y) / 2 + p.swest().y);
	p.move(n.x - s.x, n.y - s.y);
```

```
}
// end

int main()
{
	setlocale(LC_ALL, "Rus");
	screen_init();
	//== 1.Объявление набора фигур ==
	rectangle hat(point(0, 0), point(14, 5));
	line brim(point(0, 15), 17);
	myshape face(point(15, 10), point(27, 18));
	//h_circle beard(point(40, 10), point(50, 20));

	r_triangle tr1(face.neast(), face.swest()), tr12(hat.swest(), hat.neast()); //
added

	shape_refresh();
	std::cout << "=== Generated... ===\n";
	std::cin.get(); //Смотреть исходный набор
//== 2.Ориентация ==
	hat.rotate_right();
	brim.resize(2);
	face.resize(2);
	//beard.flip_vertically();

	tr1.flip_horisontally(); // added
	tr12.rotate_left(); // added

	shape_refresh();
	std::cout << "=== Prepared... ===\n";
	std::cin.get(); //Смотреть ориентацию
//== 3.Сборка изображения ==
	face.move(0, -10); //Стартовая позиция исходной фигуры.

	face.move(0, tr1.neast().y - tr1.swest().y + 5);

	up(brim, face);
	up(hat, brim);
	//down(beard, face);

	down(tr1, face); // added
	center(tr12, hat); // added

	shape_refresh();
	std::cout << "=== Ready! ===\n";
	std::cin.get(); //Смотреть результат
	screen_destroy();
	return 0;
}
```