

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра ВТ**

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Поддержка обработки исключительных ситуаций**

Студентка гр. 2308

\_\_\_\_\_

Рымарь М.И.

Преподаватель

\_\_\_\_\_

Манирагена В.

Санкт-Петербург

2024 год

### **Цель работы.**

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

### **Задание.**

Реализовать генерацию выявления всех возможных ошибок в добавляемой на рисунок фигуре и продемонстрировать перехват не менее двух типов ошибок разного уровня сложности.

### **Выполнение работы.**

В программе были созданы и перехвачены два исключения: первое находится в `line::move()`, перехвачено в `main`, второе создано в `triangle_with_cross::draw()`, перехвачено в `shape_refresh()`. В первом методе с помощью `throw` бросается объект типа исключения, а во втором – целое число, чтобы показать вариативность обработки исключений: в таком случае в блоке `catch` необходимо указать тип `int`.

В коде используется два типа исключений: `std::invalid_argument` и `std::out_of_range`.

*`std::invalid_argument`:*

Это стандартное исключение, которое возникает при недопустимом аргументе функции.

В коде используется для обработки ошибок, связанных с недопустимыми значениями размеров фигур.

Например, при попытке изменить размер фигуры `example2` до значения, которое невозможно выполнить, исключение `std::invalid_argument` будет сгенерировано.

В блоке `try-catch` обработчик перехватывает это исключение и выводит сообщение "Error 1: Resize is declined".

*`std::out_of_range`:*

Это стандартное исключение, которое возникает, когда выполняется обращение к элементу вне диапазона контейнера.

В коде используется для обработки ошибок, связанных с попыткой перемещения фигуры за пределы экрана.

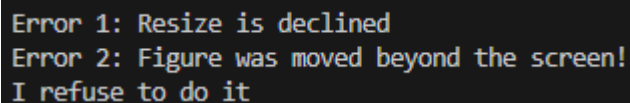
Например, при попытке переместить фигуру `brim` вне допустимых координат экрана, исключение `std::out_of_range` будет сгенерировано.

В блоке `try-catch` обработчик перехватывает это исключение и выводит сообщение об ошибке.

Исходный код программы с выделенными изменениями представлен в Приложении А.

### **Тестирование.**

Работа программы при рисовании фигуры, размеры которой не удовлетворяют параметрам экрана представлена и работа программы при выводе полей шляпы за экраном показана на рисунке 1.



```
Error 1: Resize is declined
Error 2: Figure was moved beyond the screen!
I refuse to do it
```

### **Контрольные вопросы.**

1. Что такое исключительная ситуация при выполнении программы?

Исключительная ситуация — это непредвиденное событие, которое может возникнуть во время выполнения программы и нарушить ее нормальный ход.

2. Как можно выявить исключительную ситуацию?

Исключительную ситуацию можно выявить с помощью механизма обработки исключений, например, через блок `try-catch`.

3. Что можно предпринять при выявлении исключительной ситуации?

При выявлении исключительной ситуации можно предпринять различные действия, такие как вывод сообщения об ошибке, запись информации в лог или попытка восстановления состояния программы.

4. Можно ли передать в обработчик особых ситуаций какую-либо информацию о произошедшем событии?

Да, можно передать в обработчик информацию о произошедшем событии, например, через объект исключения.

5. Можно ли обработать неизвестную особую ситуацию?

Неизвестную особую ситуацию можно обработать с помощью обработчика по умолчанию (`catch(...)`).

6. Можно ли сделать обработчик ситуации пустым?

Обработчик ситуации можно сделать пустым, но это не рекомендуется, так как важно как-то отреагировать на исключение.

7. Что можно предпринять, если для корректной обработки ситуации в данном месте программы у обработчика недостаточно данных?

Если у обработчика недостаточно данных для корректной обработки ситуации, можно передавать дополнительные параметры или использовать глобальные переменные.

8. Если требуется несколько обработчиков особых ситуаций, в каком порядке их следует размещать в программе?

Обработчики особых ситуаций следует размещать в порядке от более конкретных к более общим, чтобы избежать конфликтов при обработке.

9. Можно ли перехватывать одним обработчиком несколько различных особых ситуаций?

Да, один обработчик может перехватывать несколько различных особых ситуаций.

10. Как действуют обработчики в случае, когда никакой особой ситуации не произошло?

Обработчики не выполняются, если никакая особая ситуация не произошла.

11. Как следует размещать блоки контроля, чтобы получить безопасный программный код?

Блоки контроля следует размещать так, чтобы минимизировать возможность возникновения исключительных ситуаций и обеспечить безопасность программы.

12. Можно ли продолжить выполнение программы с точки, в которой выявлена ошибка, после внесения исправлений в данные?

Да, после внесения исправлений в данные и обработки исключения можно продолжить выполнение программы с точки, где произошла ошибка.

### **Выводы.**

В ходе выполнения лабораторной работы был изучен принцип обработки исключений. С его помощью стало проще тестировать программу, находить и исправлять ошибки.

## ПРИЛОЖЕНИЕ А

### ИЗМЕНЁННЫЕ ЧАСТИ ПРОГРАММЫ

```
#include <list>
#include <iostream>
//#include "screen.h"
using namespace std;
//== 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(double) = 0; //Изменение размера
    virtual ~shape( ) { shapes.remove(this); } //Деструктор
};
list<shape*> shape::shapes; // Размещение списка фигур
void shape_refresh( ) // Перерисовка всех фигур на экране
{
    screen_clear( );
    for (auto p : shape :: shapes){
        try{ p->draw( ); }
        catch(int i)
        {
            cout << "Shape can't be displayed on the screen\n";
            shape::shapes.remove(p);
            break;
        }
    }
    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 {
protected:
    point w, e;
public:
    line(point a, point b) : w(a), e(b) { }; //Произвольная линия (по
    двум точкам)
```

```

        line(point a, int L) : w(point(a.x + L - 1, a.y)), e(a) { };
//Горизонтальная линия
        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) {

            if(!on_screen(w.x+a, w.y+b) || !on_screen(e.x+a, e.y+b))

                throw std::invalid_argument("Figure was moved beyond the screen!\n");
            w.x += a; w.y += b; e.x += a; e.y += b;
        }
        void draw( ) { put_line(w, e); }
        void resize(double d) {
            int ex = e.x, ey=e.y;
            e.x = w.x + (ex - w.x) * d;
            e.y = w.y + (ey - w.y) * d;
        }
};
class rectangle : public rotatable { // ==== Прямоугольник ====
/* nw ----- n ----- ne
   |           |
   |           |
   w      c      e
   |           |
   |           |
   sw ----- s ----- se */
protected:
    point sw, ne;
public:
    rectangle(point a, point b) : sw(a), ne(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( ) {
        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(){
        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(double d){
        ne.x = sw.x + (ne.x - sw.x) * d;
        ne.y = sw.y + (ne.y - sw.y) * d;
    }
    void 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 фигуру p над фигурой q
{
    point n = q.north( );
    point s = p.south( );
    p.move(n.x - s.x, n.y - s.y + 1);
}
void down(shape &p, const shape &q) //Поместить фигуру p под фигурой q
{
    point n = q.south( );
    point s = p.north( );
    p.move(n.x - s.x, n.y - s.y - 1);
}
void right(shape& p, const shape& q) // поместить фигуру p справа от
фигуры q
{
    point e = q.east( );
    point w = p.west( );
    p.move(e.x - w.x , e.y - w.y - 3 );
}
void left(shape& p, const shape& q) // поместить фигуру p слева от фигуры
q
{
    point e = q.west( );
    point w = p.east( );
    p.move(e.x - w.x, e.y - w.y - 3);
}

//-----
class triangle_with_cross: public rectangle, public reflectable {
    bool reflected;
    short rotate; //0-ничего 1- повернут вправо 2 - повернут влево на 90
public:

```



```

        triangle_with_cross(point a, point b, bool re=true, short ro = 0) :
rectangle(a, b),

reflected(re),

rotate(ro){ }
    void resize(double d);
    void draw();
    void flip_horizontally() {reflected = !reflected;};
    void flip_vertically() {};
    void rotate_left(){
        rotate = 2;
        rectangle :: rotate_left();
    };
    void rotate_right(){
        rotate = 1;
        rectangle :: rotate_right();
    };
};

void triangle_with_cross ::resize(double d) {
    ne.x = sw.x + (ne.x - sw.x) * d;
    ne.y = sw.y + (ne.y - sw.y) * d;
}

void triangle_with_cross :: draw()
{
    if (!on_screen(sw.x, sw.y) || !on_screen(sw.x, ne.y) ||
!on_screen(ne.x,ne.y) || !on_screen(ne.x, sw.y))
        throw 2311;

    put_line((sw.x+ne.x)/2, sw.y, (sw.x+ne.x)/2, ne.y);
    put_line(sw.x, (sw.y+ne.y)/2, ne.x, (sw.y+ne.y)/2);
    if(rotate == 0) {
        put_line(sw.x, sw.y, sw.x, ne.y);
        put_line(sw.x,sw.y, ne.x, sw.y);
        put_line(sw.x, ne.y, ne.x, sw.y);
    } else if(rotate == 1){
        put_line(sw.x,ne.y,ne.x,ne.y);
        put_line(ne.x,ne.y,ne.x,sw.y);
        put_line(sw.x,ne.y, ne.x,sw.y);
    } else if(rotate == 2){
        put_line(sw.x, sw.y, ne.x, ne.y);
        put_line(sw.x, sw.y, ne.x, sw.y);
        put_line(ne.x, ne.y, ne.x, sw.y);
    } else if(rotate == 3){
        put_line(sw.x, ne.y, ne.x, ne.y);
        put_line(sw.x, ne.y, ne.x, sw.y);
        put_line(ne.x, ne.y, ne.x, sw.y);
    }
}

#include "screen.h"
#include "shape.h"
// Сборная пользовательская фигура - физиономия

```

```

class myshape : public triangle_with_cross {           // Моя фигура ЯВЛЯЕТСЯ
    int w, h;                                         //      треугольником
    line l_eye;   // левый глаз - моя фигура СОДЕРЖИТ линию
    line r_eye;   // правый глаз
    line mouth;   // рот
public:
    myshape(point, point);
    void draw( );
    void move(int, int);
    void resize(double);
};

myshape :: myshape(point a, point b) : triangle_with_cross(a, b),
    //Инициализация базового класса
    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 - 8)
{ }
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);
}
void myshape :: resize(double d) {
    l_eye.resize(d+1);
    r_eye.resize(d+1);
    mouth.resize(d);
    l_eye.move((int)d*2+1, (int)((d-1)* h * 3 / 4));
    r_eye.move((int)d*4+1, (int)((d-1)* h * 3 / 4));
    mouth.move((int)((d-1)*(w-4))+1, (int)((d-1)* h / 4));
    triangle_with_cross :: resize(d);
}

int main( )
{
    setlocale(LC_ALL, "Rus");
    screen_init( );
    //== 1. Объявление набора фигур ==
    rectangle hat(point(22, 28), point(30, 33));
    line brim(point(14,27),25);
    myshape face(point(20,18), point(32,26));
    triangle_with_cross shishak(point(20, 40), point(30, 45));
    triangle_with_cross l_sideburn(point(60, 40), point(70, 45));

```

```

triangle_with_cross r_sideburn(point(60, 20), point(70, 25));
triangle_with_cross example(point(1000, 1000), point(500, 500));
shape_refresh( );
std::cout << "=== Generated... ===\n";
std::cin.get(); //Смотреть исходный набор
//== 2. Подготовка к сборке ==
hat.resize(2);
hat.rotate_right( );
brim.resize(2);
face.resize(2);
shishak.resize(0.8);
l_sideburn.resize(0.5);
r_sideburn.resize(0.5);
shishak.rotate_right();
r_sideburn.rotate_left();
try{ brim.move(10000, 5000); }
catch(std::invalid_argument e)
{
    cout << e.what() << "I refuse to do it\n";
}
shape_refresh( );
std::cout << "=== Prepared... ===\n";
std::cin.get(); //Смотреть результат поворотов/отражений
//== 3. Сборка изображения ==
up(brim, face);
up(hat, brim);
right(l_sideburn, face);
left(r_sideburn, face);
up(shishak, hat);
shape_refresh( );
std::cout << "=== Ready! ===\n";
std::cin.get(); //Смотреть результат
screen_destroy( );
return 0; }

```