**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **ПОДДЕРЖКА ОБРАБОТКИ ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 0308 |  | Бобыльков Т.В. |
|  |  | Радабольский В. C. |
| Преподаватель |  | Манирагена В. |

Санкт-Петербург

2022

**Цель работы**

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

**Решение задания**

Так как основная проблема программы для рисования геометрических фигур — это выход за границы экрана, например, при создании, перемещении или изменении размера фигуры. Именно этот тип ошибок и решено было обработать.

В нашей программе 3 типа исключений:

1. CantMove – исключении при попытке передвинуть фигуру за границу экрана. Обработка – сообщение об ошибке, фигура остается в исходном положении.
2. OutOfScreen – исключение, выбрасываемое при выходе точки за границы экрана. Обработка – сообщение о выходе точки за границы экрана.
3. CreationError – исключение в конструкторе, возникающее при попытке создания фигуры вне экрана. Обработка – сообщение об ошибке и замена ошибочной фигуры на эталонную, которая точно находится в границах экрана

**Контрольный тест**

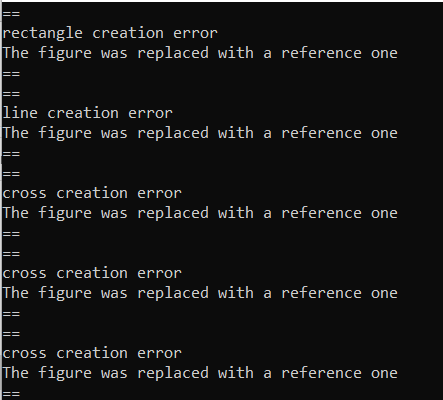


Рисунок 1.Сообщения об ошибке при попытке создания фигуры вне экрана

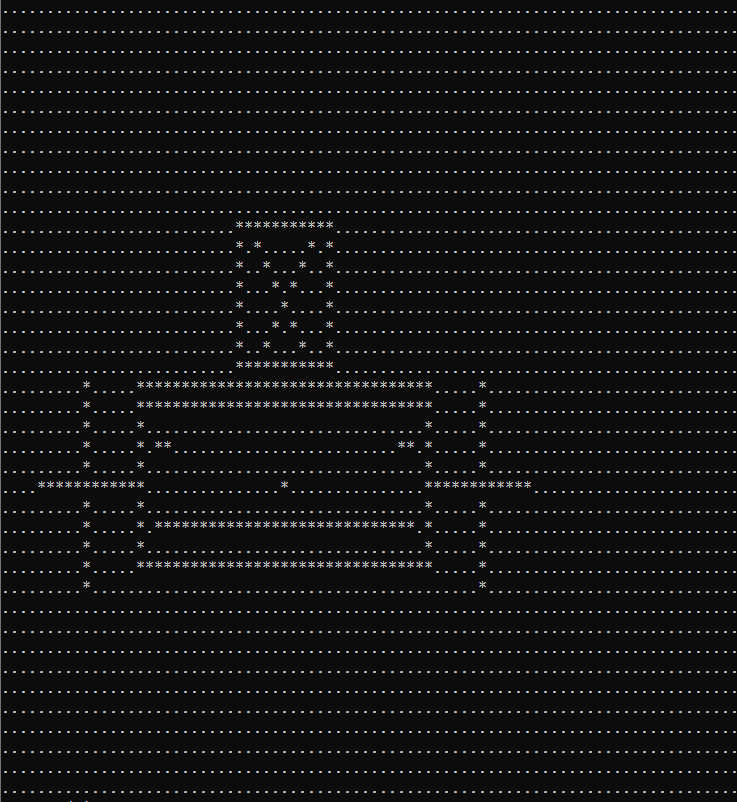


Рисунок 2.CreationError (все фигуры заменены на эталонные)

****

Рисунок 3.Попытка передвинуть фигуру за границы экрана

**Выводы.**

В ходе выполнения работы мы научились обрабатывать исключительные ситуации, возникающие при выполнении программы, что является неотъемлемой частью пользовательской программы. Теперь программа полностью ориентирована на пользователя и в случае ошибки предупредит его о ней и по возможности исправит.

**Список использованных источников.**

П. Г. Колинько, Алгоритмы и структуры данных. Лекция от 21.02.2022.

Методическое пособие «Пользовательские Контейнеры» П.Г. Колинько.

**Приложение.**

Исходный код программы для решения задачи на языке C++:

**main.h**

#include **<iostream>**#include **"screen.h"**#include **"shape.h"  
  
  
  
class** cross : **public** shape {  
**protected**:  
 point c\_center;  
 **int** size;  
**public**:  
 cross(): shape(), c\_center(0), size(0) {};  
 cross(point c, **int** s): c\_center(c), size(s) {  
 **if** (on\_screen(c)){  
 c\_center = c;  
 size = s;  
 }  
 **else throw** CreationError(**"cross creation error"**);  
 }  
 point north() **const** {**return** point(c\_center.x, c\_center.y + size);}; *//точки для привязки* point south() **const** {**return** point(c\_center.x, c\_center.y - size);};  
 point west() **const** {**return** point(c\_center.x - size, c\_center.y);};  
 point east() **const** {**return** point(c\_center.x + size, c\_center.y);};  
  
 point neast() **const** {**return** point(c\_center.x + size, c\_center.y + size);};  
 point seast() **const** {**return** point(c\_center.x + size, c\_center.y - size);};  
 point nwest() **const** {**return** point(c\_center.x - size, c\_center.y + size);};  
 point swest() **const** {**return** point(c\_center.x - size, c\_center.y - size);};  
  
 point center() **const** {**return** c\_center;};  
  
  
 **void** move(**int** a, **int** b); *// перемещение* **void** resize(**double** d); *// Изменение размера* **void** draw(); *// переопределим рисование*};  
  
**class** andrew\_cross : **public** cross {  
  
**public**:  
 andrew\_cross() : cross(){}  
 andrew\_cross(point c, **int** s): cross(c, s) {};  
 **void** draw();  
};  
**void** andrew\_cross::draw() {  
 put\_line(neast(), swest());  
 put\_line(nwest(), seast());  
}  
  
  
  
**void** cross ::draw() {  
 put\_line(north(), south());  
 put\_line(east(), west());  
}  
  
**void** cross::resize(**double** d) {  
 size \*= d;  
}  
  
**void** cross::move(**int** a, **int** b) {  
 c\_center.x += a;  
 c\_center.y += b;  
 **if** (!on\_screen(\***this**)) {  
 c\_center.x -= a;  
 c\_center.y -= b;  
 **throw** CantMove(**"After move cross will be out of screen!"**);  
 }  
}  
  
  
*// в центр***void** into(shape &p, **const** shape &q) {  
 point p\_c = p.center();  
 point q\_c = q.center();  
  
 p.move(q\_c.x - p\_c.x, q\_c.y - p\_c.y);  
}  
*// слева***void** left(shape &p, **const** shape &q) {  
 point e = q.west();  
 point w = p.east();  
 p.move(e.x - w.x - 1, e.y - w.y);  
}  
*// справа***void** right(shape &p, **const** shape &q) {  
 point w = q.east();  
 point e = p.west();  
 p.move(w.x - e.x + 1, w.y - e.y);  
}  
  
**class** myshape : **public** rectangle {  
 **int** w, h;  
 line l\_eye; *// Левый глаз* line r\_eye; *// Правый глаз* line mouth; *// рот***public**:  
 myshape(): rectangle(){}  
 myshape(point, point);  
 **void** draw() ;  
 **void** move(**int**, **int**);  
 **void** resize(**double**) {}  
};  
  
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)  
{}  
  
**void** myshape :: draw() {  
 rectangle :: draw(); *// Контур лица* point nose = center();  
  
 put\_point(nose);  
}  
  
**void** myshape :: move(**int** a, **int** b) {  
 **try** {  
 rectangle::move(a, b);  
 l\_eye.move(a, b);  
 r\_eye.move(a, b);  
 mouth.move(a, b);  
 } **catch** (CantMove) { *//Если неможем подвинуть прямоугольник - не можем подвинуть и фигуру* **throw** CantMove(**"Cant move rectangle => CantMove myshape!"**);  
 }  
}  
  
**int** main() {  
  
 setlocale(**LC\_ALL**, **"Rus"**);  
  
 screen\_init();  
  
 *// ==1. Объявление набор фигур ==  
  
// line brim(point(4,1), 17); // козырек  
// myshape face(point(25,1), point(25+12,1+8));  
// cross left\_cross(point(65, 6), 5);  
// cross right\_cross(point(107, 6), 5);  
// andrew\_cross hat\_cross(point(45, 6), 3);* rectangle hat;  
 line brim;  
 myshape face;  
 cross left\_cross;  
 cross right\_cross;  
 andrew\_cross hat\_cross;  
  
 **try**{  
 rectangle test(point(1000,3), point(5+14,3+5));  
 hat = test;  
 }  
 **catch**(CreationError &ex){  
 cout << **"=="** << endl << ex.what() << endl;  
 cout << **"The figure was replaced with a reference one"** << endl <<**"=="**<< endl;  
 rectangle n\_fig(point(5,3), point(5+14,3+5));  
 hat = n\_fig;  
 }  
  
 **try**{  
 line l\_test(point(4000,1), 17);  
 brim = l\_test;  
 }  
 **catch**(CreationError &ex){  
 cout << **"=="** << endl << ex.what() << endl;  
 cout << **"The figure was replaced with a reference one"** << endl <<**"=="**<< endl;  
 line l\_fig(point(4,1), 17);  
 brim = l\_fig;  
 }  
  
 **try**{  
 myshape l\_test(point(5,1), point(25+12,1+8));  
 face = l\_test;  
 }  
 **catch**(CreationError &ex){  
 cout << **"=="** << endl << ex.what() << endl;  
 cout << **"The figure was replaced with a reference one"** << endl <<**"=="**<< endl;  
 myshape l\_test(point(25, 1), point(25 + 12, 1 + 8));  
 face = l\_test;  
 }  
  
 **try**{  
 cross l\_test(point(650, 6), 5);  
 left\_cross = l\_test;  
 }  
 **catch**(CreationError &ex){  
 cout << **"=="** << endl << ex.what() << endl;  
 cout << **"The figure was replaced with a reference one"** << endl <<**"=="**<< endl;  
 cross l\_test(point(65, 6), 5);  
 left\_cross = l\_test;  
 }  
  
 **try**{  
 cross l\_test(point(1070, 6), 5);  
 right\_cross = l\_test;  
 }  
 **catch**(CreationError &ex){  
 cout << **"=="** << endl << ex.what() << endl;  
 cout << **"The figure was replaced with a reference one"** << endl <<**"=="**<< endl;  
 cross l\_test(point(107, 6), 5);  
 right\_cross = l\_test;  
 }  
  
 **try**{  
 andrew\_cross l\_test(point(450, 6), 3);  
 hat\_cross = l\_test;  
 }  
 **catch**(CreationError &ex){  
 cout << **"=="** << endl << ex.what() << endl;  
 cout << **"The figure was replaced with a reference one"** << endl <<**"=="** << endl;  
 andrew\_cross l\_test(point(45, 6), 3);  
 hat\_cross = l\_test;  
 }  
  
  
 *//== 2.Подготовка к сборке ==* hat.rotate\_right( );  
  
 brim.resize(3.0);  
 face.resize(2.0);  
  
*// //== 3.Сборка изображения ==* face.move(10, 10); *// Лицо - в исходное положение* up(brim, face);  
 up(hat, brim);  
 into(hat\_cross, hat);  
 left(left\_cross, face);  
 right(right\_cross, face);  
 shape\_refresh( );  
 std::cout << **"=== Ready! ===\n"**;  
  
*// == 4. Проверка исключений ==  
// {  
// std::cout << "OutOfScreen Exception Check:" << std::endl;  
// std::cout << "Make new point in [1000][1000]" << std::endl;  
// put\_point(1000, 1000);  
// shape\_refresh();  
// }* std::cin.get();  
  
 std::cout << **"CantMove Exception Check: "** << std::endl;  
 std::cout << **"Move face by 100 in x & y axis"** << std::endl;  
  
 **try** {  
 face.move(100, 100);  
 } **catch** (CantMove &ex) {  
 std::cout << ex.what() << std::endl;  
 }  
 shape\_refresh();  
 std::cin.get();  
  
 **return** 0;  
}

**shape.h**

#include **<list>**#include **<iostream>**#include **"screen.h"**#include **<string>***//  
// Created by bobyl on 08.02.2022.  
//*#ifndef **INC\_1\_SHAPE\_H  
  
class** CantMove : **public** std:: exception {  
 std:: string s;  
**public**:  
 CantMove(std::string s\_er);  
 **const char** \* what() **const noexcept override**;  
};  
  
**const char** \* CantMove::what() **const noexcept** {  
 **return** s.c\_str();  
}  
  
CantMove::CantMove(std::string s\_er) : s(s\_er) {}  
  
**class** OutOfScreen : **public** std::exception {  
 std::string s;  
**public**:  
 OutOfScreen(std::string s\_er, **int** a, **int** b);  
 **const char** \* what() **const noexcept override**;  
};  
  
OutOfScreen::OutOfScreen(std::string s\_er, **int** a, **int** b) {  
 s\_er += **" Bad point: ["** + std::to\_string(a) + **"]["** + std::to\_string(b) + **"]"**;  
 s = s\_er;  
}  
  
**const char** \* OutOfScreen::what() **const noexcept** {  
 **return** s.c\_str();  
}  
  
**class** CreationError: **public** std::exception{  
 std::string s;  
**public**:  
 **explicit** CreationError(std::string s\_er);  
 **const char** \* what() **const noexcept override**;  
};  
  
CreationError::CreationError(std::string s\_er) : s(std::move(s\_er)) {}  
  
**const char** \* CreationError::what() **const noexcept** {  
 **return** s.c\_str();  
}  
  
**using namespace** std;  
  
**char** screen[YMAX][XMAX];  
**enum** color {***black*** = **'\*'**, ***white*** = **'.'**};  
*// === Поддержка экрана в форме матрицы символов ===***void** screen\_init() { *// все в белый* **for** (**auto** & y : screen)  
 **for** (**auto** & x: y) x = ***white***;  
}  
  
**void** screen\_destroy() {  
 **for** (**auto** & y: screen)  
 **for** (**auto** & x : y) x = ***black***;  
}  
  
**bool** on\_screen(**int** a, **int** b) {  
 **bool** in\_x = (0 <= a && a < XMAX);  
 **bool** in\_y = (0 <= b && b < YMAX);  
  
 **return** in\_x && in\_y;  
}  
  
**bool** on\_screen(point p) {  
 **return** on\_screen(p.x, p.y);  
}  
  
  
**void** put\_point(**int** a, **int** b) {  
 **if**(on\_screen(a,b)) {  
 screen[b][a] = ***black***;  
 } **else throw** OutOfScreen(**"Point out of canvas."**, a, b);  
}  
  
**void** put\_point(point p) {put\_point(p.x, p.y);}  
  
**void** put\_line(**int** x0, **int** y0, **int** x1, **int** y1) {  
 **int** dx = 1; *// шаг* **int** a = x1 - x0; *// ширина* **if** (a < 0) {  
 dx = -1, a = -a; *//Отрицательный шаг по горизонтали* }  
  
 **int** dy = 1; *// шаг* **int** b = y1 - y0; *// высота* **if** (b < 0) {  
 dy = -1, b = -b; *//Отрицательный шаг по вертикали* }  
  
 **int** double\_a = 2\*a;  
 **int** double\_b = 2\*b;  
  
 **int** x\_crit = -b + double\_a;  
 **int** eps = 0;  
  
 **for** (;;){ *// Рисование линии* put\_point(x0, y0);  
 **if** (x0 == x1 && y0 == y1) **break**;  
 **if** (eps <= x\_crit) x0 += dx, eps += double\_b;  
 **if** (eps >= a || a < b) y0 += dy, eps -= double\_a;  
 }  
}  
  
**void** put\_line(point a, point b) {  
 put\_line(a.x, a.y, b.x, b.y);  
}  
  
**void** screen\_clear() {  
 screen\_init();  
}  
  
**void** screen\_refresh() {  
 **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';* }  
}  
  
*// === Библиотека фигур ===***struct** shape { *// виртуальный базовый класс "Фигура"* **static** std::list<shape\*> shapes; *// Список всех фигур* shape() { *// конструктор* shapes.push\_back(**this**); *// Фигура присоединяется к списку* }  
 **virtual** point north() **const** = 0; *//точки для привязки* **virtual** point south() **const** = 0;  
 **virtual** point west() **const** = 0;  
 **virtual** point east() **const** = 0;  
  
 **virtual** point neast() **const** = 0;  
 **virtual** point seast() **const** = 0;  
 **virtual** point nwest() **const** = 0;  
 **virtual** point swest() **const** = 0;  
  
 **virtual** point center() **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**);  
 };  
  
  
};  
  
std::list<shape\*> shape::shapes; *//размещение списка фигур в памяти***void** shape\_refresh() { *// перерисовка всех фигур на экране* screen\_clear();  
 **for** (**auto** p\_sh: shape::shapes) {  
 **try** {  
 p\_sh->draw(); *// Динамическое связывание* } **catch** (OutOfScreen &ex) {  
 std::cout << ex.what() << std::endl;  
 }  
 }  
 screen\_refresh();  
}  
  
**class** rotatable : **virtual public** shape { *// Фигуры, пригодные к повороту***public**:  
 **virtual void** rotate\_left() = 0; *// Поворот влево* **virtual void** rotate\_right() = 0; *// Поворот вправо*};  
  
**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() : shape(){}  
 line(point a, point b){  
 **if** (on\_screen(a) && on\_screen(b)){  
 w = a;  
 e = b;  
 } **else throw** CreationError(**"line creation error"**);  
 } *// Произвольная линия по двум точкам* line(point a, **int** L) {  
 w = point(a.x + L - 1, a.y);  
 e = a;  
 **if** (on\_screen(w) && on\_screen(e)){} **else throw** CreationError(**"line creation error"**);  
 }  
 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); }  
  
 point center ( ) **const** {  
 **return** point(north().x, east().y);  
 }  
  
 **void** move(**int** a, **int** b);  
 **void** draw( ) {  
 put\_line(w, e);  
 }  
  
 **void** resize(**double** d) { *// изменение длины линии в d раз* e.x += abs(e.x - w.x) \* d;  
 e.y += abs(e.y - w.y) \* d;  
 }  
  
};  
  
  
  
**bool** on\_screen(shape& my\_shape) { *// Если левый верхний угол и правый нижний на холсте - то вся фигура на холсте* **return** on\_screen(my\_shape.nwest()) && on\_screen(my\_shape.seast());  
}  
  
**void** line::move(**int** a, **int** b) {  
 w.x += a;  
 w.y += b;  
 e.x += a;  
 e.y += b;  
 **if** (!on\_screen(\***this**)) {  
 w.x += a;  
 w.y += b;  
 e.x += a;  
 e.y += b;  
 **throw** CantMove(**"Line will be out of screen!"**);  
 }  
  
}  
  
  
*// Прямоугольник***class** rectangle : **public** rotatable {  
**protected**:  
 point sw, ne;  
**public**:  
 rectangle() : shape(){}  
 rectangle(point a, point b)*//: sw(a), ne(b){}  
// try: sw(a), ne(b)* {  
 **if** (on\_screen(a) && on\_screen(b)) {  
 sw = a;  
 ne = b;  
 } **else throw** CreationError(**"rectangle creation error"**);  
 }  
 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; }  
  
 point center( ) **const** {  
 **return** point(north().x, east().y);  
 }  
  
 **void** rotate\_right() { *//поворот вправо относительно se* **int** w = ne.x - sw.x; *// учитывается масштаб по осям* **int** 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;  
 **int** 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;  
 **if** (!on\_screen(\***this**)) {  
 sw.x -= a; sw.y -= b; ne.x -= a; ne.y -= b;  
 **throw** CantMove(**"Rectangle will be out of screen!"**);  
 }}  
 **void** resize(**double** d)  
 { ne.x += (ne.x - sw.x) \* d; ne.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 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);  
}  
  
  
#define **INC\_1\_SHAPE\_H**#endif *//INC\_1\_SHAPE\_H*

**screen.h**

*//  
// Created by bobyl on 08.02.2022.  
//*#ifndef **INC\_1\_SCREEN\_H**#define **INC\_1\_SCREEN\_H  
  
  
  
const int** XMAX = 120; *// ширина***const int** YMAX = 40; *// высота***class** point {  
**public**:  
 **int** x, y;  
  
 **explicit** point(**int** a = 0, **int** b = 0) : x(a), y(b) {}; *//конструктор точки* **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 экрана***void** screen\_refresh(); *// обновление***void** screen\_clear(); *// очистка*#endif *//INC\_1\_SCREEN\_H*