**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 1**

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

Тема: **«РАБОТА С ИЕРАРХИЕЙ ОБЪЕКТОВ:  
НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1308 |  | Макаров М.В. |
| Студент гр. 1308 |  | Томилов Д. |  | Макаров М.В. |
| Преподаватель |  | Манирагена В. |  | Морозов С.М. |

Санкт-Петербург

2022

**Цель работы.**

Получить опыт работы с классами в языке C++, научиться писать более структурированный код за счёт использования наследования классов.

**Задание (вариант 3)**

Доработать модуль *shape.cpp*, добавив в коллекцию еще одну фигуру, номер которой указан в табл. 1.2 в строке с вашим вариантом. Для этой фигуры нужно оп­ределить подходящее место в иерархии классов и написать недостающие функ­ции-члены.

Фигура: треугольник с косым крестом

Положения: 1,13, 15.



**Описание решения задачи**

Сначала было принято решение реализовать 3 класса: первый - косой крест, второй - равносторонний треугольник и третий – косой крест с равносторонним треугольником.

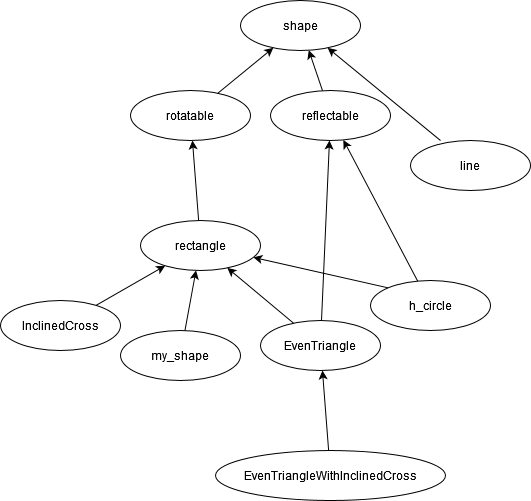
Косой крест был сделан на базе класса прямоугольник. Пришлось переопределить только одну функцию – draw(), поскольку косой крест это, по сути, прямоугольник, у которого рисуются диагонали и не рисуются стороны. Все остальные функции уже были определены.

Класс равносторонний треугольник наследует сразу от двух классов: от класса прямоугольник и от класса reflectable, поскольку нужна и возможность поворота, которая уже есть у прямоугольника, и возможность отражения, свойственная классу reflectable. При создании объекта класса треугольник вдобавок к точкам sw и ne создаются точки, отмечающие положение вершин треугольника, далее используемые для отрисовки линий. Точки sw и ne, также как в прямоугольнике, используются для определения границ фигуры. В классе равносторонний треугольник были переопределены функции draw(), flip\_horisontally(), flip\_vertically(), rotate\_left(), rotate\_right(), resize(int) , move().

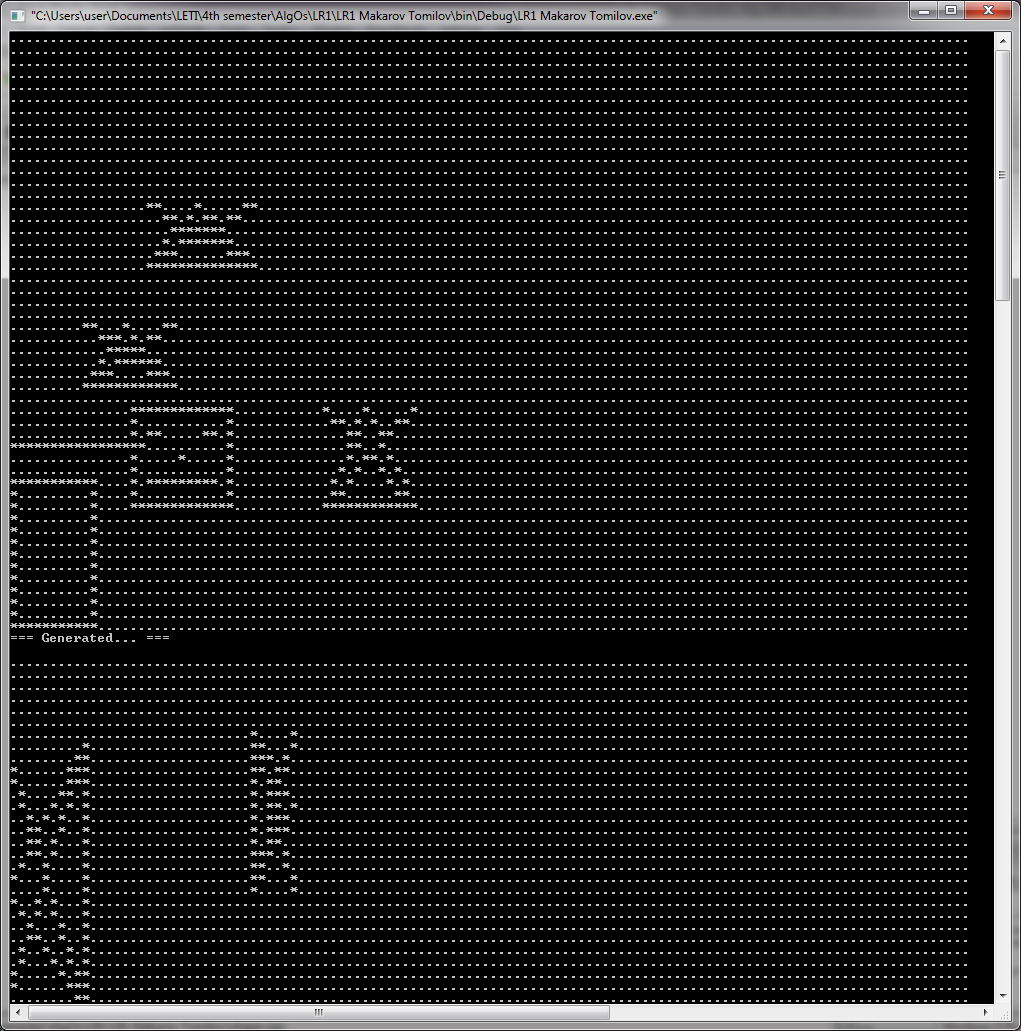
Класс косой крест с равносторонним треугольником сначала планировалось делать на основе сразу двух классов – косой крест и равносторонний треугольник. Однако, поскольку в классе равносторонний треугольник переопределено слишком много функций класса прямоугольник, которые также используются в классе косой крест, было решено отказаться от этой идеи, ведь их все пришлось бы заново переопределять в классе косой крест с равносторонним треугольником. Поэтому этот класс теперь наследует только от класса равносторонний треугольник. Отрисовка креста проходит также как в классе косой крест, ведь у треугольника также как и у прямоугольника есть опорные точки sw и ne. В классе косой крест с равносторонним треугольником была переопределена только функция draw().

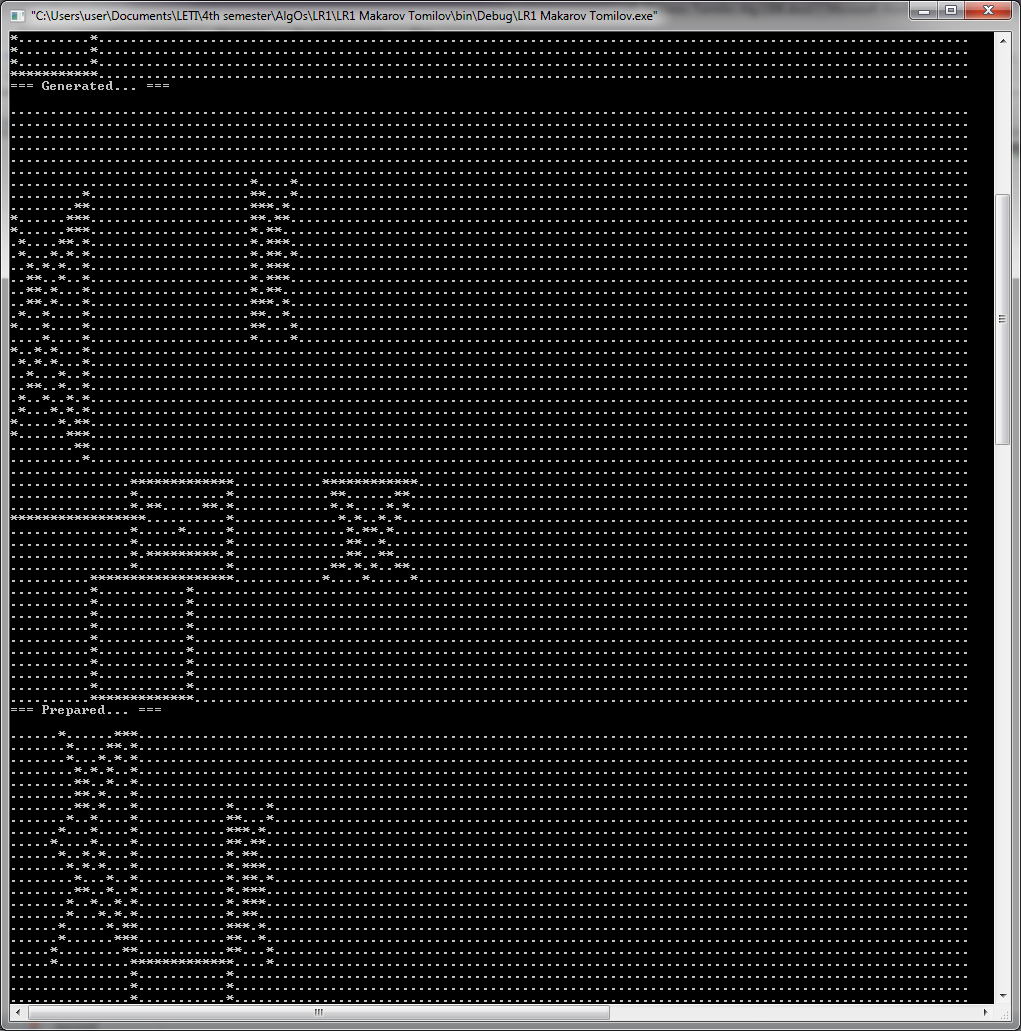
Также стоит добавить, что у класса прямоугольник были переписаны функции поворота, поскольку в исходной программе они были реализованы неправильно и изменяли пропорции фигуры при применении.

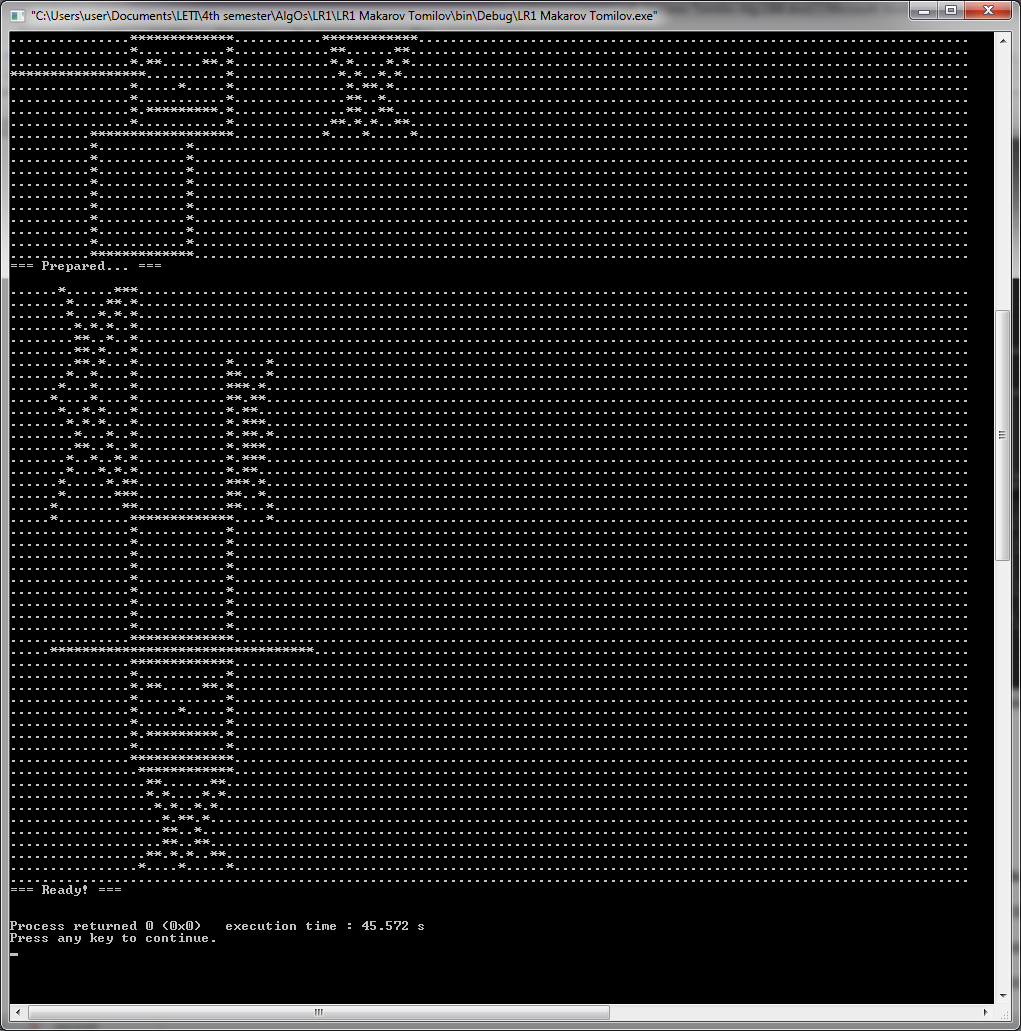
**Диаграмма классов**

****

**Примеры выполнения программы**

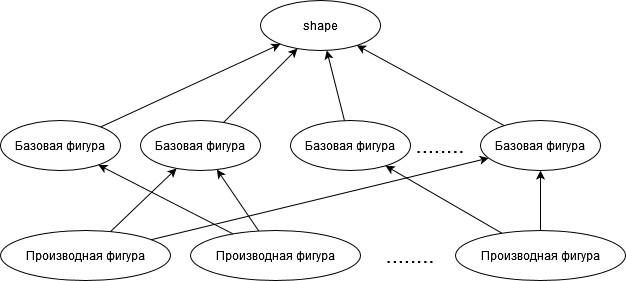






**Выводы.**

В ходке выполнения работы было изучено наследование в C++. Однако, представленная диаграмма классов не является аккуратной и структурированной. Поскольку предполагается, что эта программа уже написана до нас и может только дополняться, но не изменяться, с этим приходится смириться. Однако, если бы её можно было изменять, мы бы предложили другую структуру классов, которая выглядела бы примерно вот так:



В ней класс shape объединял бы свойства класса rotatable и reflectable, то есть позволял бы и отражать и поворачивать любую фигуру, на его основе строились бы базовые фигуры (крест, круг, квадрат и т.д.), а уже из их комбинации получались бы более сложные фигуры. Такой вариант был бы куда более структурирован.

Текст программы (только дополнения и изменения):

shape.h

point get\_point\_rotated\_right(point p, float xCenter,float yCenter){

float x,y,deltaX,deltaY;

deltaY = (((float)p.y)-yCenter);

deltaX = (((float)p.x)-xCenter);

x = p.x;

y = p.y;

x = x-deltaX;

y = y-deltaY;

x = x+deltaY;

y = y - deltaX;

return point(round(x),round(y));

}

point get\_point\_rotated\_left(point p, float xCenter,float yCenter){

float x,y,deltaX,deltaY;

deltaY = (((float)p.y)-yCenter);

deltaX = (((float)p.x)-xCenter);

x = p.x;

y = p.y;

x = x-deltaX;

y = y-deltaY;

x = x-deltaY;

y = y + deltaX;

return point(round(x),round(y));

}

Повороты прямоугольника:

void rotate\_right() // Поворот вправо относительно se

{

float xCenter,yCenter;

point swNew,neNew;

xCenter = seast().x;

yCenter = seast().y;

neNew = get\_point\_rotated\_right(ne,xCenter,yCenter);

swNew = get\_point\_rotated\_right(sw,xCenter,yCenter);

ne = point(std::max(neNew.x,swNew.x),std::max(neNew.y,swNew.y));

sw = point(std::min(neNew.x,swNew.x),std::min(neNew.y,swNew.y));

}

void rotate\_left() // Поворот влево относительно sw

{

float xCenter,yCenter;

point swNew,neNew;

xCenter = swest().x;

yCenter = swest().y;

neNew = get\_point\_rotated\_left(ne,xCenter,yCenter);

swNew = get\_point\_rotated\_left(sw,xCenter,yCenter);

ne = point(std::max(neNew.x,swNew.x),std::max(neNew.y,swNew.y));

sw = point(std::min(neNew.x,swNew.x),std::min(neNew.y,swNew.y));

}

shape.cpp

void upRight(shape &p, const shape &q){

int deltaX = q.neast().x - p.swest().x;

int deltaY = q.neast().y - p.swest().y;

p.move(deltaX,deltaY);

}

void upLeft(shape &p, const shape &q){

int deltaX = q.nwest().x - p.seast().x;

int deltaY = q.nwest().y - p.seast().y;

p.move(deltaX,deltaY);

}

class InclinedCross : public rectangle

{

public:

InclinedCross(point swest, point neast);

void draw() override;

};

InclinedCross::InclinedCross(point swest, point neast) : rectangle(swest,neast){}

void InclinedCross::draw(){

put\_line(neast().x,neast().y,swest().x,swest().y);

put\_line(seast().x,seast().y,nwest().x,nwest().y);

}

class EvenTriangle : public rectangle, virtual public reflectable

{

protected:

point trBotLeft;

point trBotRight;

point trApex;

public:

void draw() override;

void flip\_horisontally() override;

void flip\_vertically() override;

void rotate\_left() override;

void rotate\_right() override;

void resize(int) override;

void move(int,int) override;

EvenTriangle(point, point);

};

void EvenTriangle::draw(){

put\_line(trBotLeft.x,trBotLeft.y,trBotRight.x,trBotRight.y);

put\_line(trApex.x,trApex.y,trBotRight.x,trBotRight.y);

put\_line(trBotLeft.x,trBotLeft.y,trApex.x,trApex.y);

}

void EvenTriangle::flip\_horisontally(){

float xCenter = ((float)(west().x + east().x))/2;

trBotLeft.x = std::round(xCenter - (((float)trBotLeft.x)-xCenter));

trBotRight.x = std::round(xCenter - (((float)trBotRight.x)-xCenter));

trApex.x = std::round(xCenter - (((float)trApex.x)-xCenter));

}

void EvenTriangle::flip\_vertically(){

float yCenter = ((float)(south().y + north().y))/2;

trBotLeft.y = std::round(yCenter - (((float)trBotLeft.y)-yCenter));

trBotRight.y = std::round(yCenter - (((float)trBotRight.y)-yCenter));

trApex.y = std::round(yCenter - (((float)trApex.y)-yCenter));

}

void EvenTriangle::rotate\_left(){

float xCenter = swest().x;

float yCenter = swest().y;

trApex = get\_point\_rotated\_left(trApex,xCenter,yCenter);

trBotLeft = get\_point\_rotated\_left(trBotLeft,xCenter,yCenter);

trBotRight = get\_point\_rotated\_left(trBotRight,xCenter,yCenter);

rectangle::rotate\_left();

}

void EvenTriangle::rotate\_right(){

float xCenter = seast().x;

float yCenter = seast().y;

trApex = get\_point\_rotated\_right(trApex,xCenter,yCenter);

trBotLeft = get\_point\_rotated\_right(trBotLeft,xCenter,yCenter);

trBotRight = get\_point\_rotated\_right(trBotRight,xCenter,yCenter);

rectangle::rotate\_right();

}

void EvenTriangle::resize(int dimension){

rectangle::resize(dimension);

trApex.x += (trApex.x - sw.x) \* (dimension - 1);

trApex.y += (trApex.y - sw.y) \* (dimension - 1);

trBotLeft.x += (trBotLeft.x - sw.x) \* (dimension - 1);

trBotLeft.y += (trBotLeft.y - sw.y) \* (dimension - 1);

trBotRight.x += (trBotRight.x - sw.x) \* (dimension - 1);

trBotRight.y += (trBotRight.y - sw.y) \* (dimension - 1);

}

void EvenTriangle::move(int x,int y){

rectangle::move(x,y);

trApex.x+=x;

trApex.y+=y;

trBotLeft.x+=x;

trBotLeft.y+=y;

trBotRight.x+=x;

trBotRight.y+=y;

}

EvenTriangle::EvenTriangle(point sWest, point nEast): rectangle(sWest,nEast){

trBotLeft = swest();

trBotRight = seast();

trApex = north();

}

class EvenTriangleWithInclinedCross : public EvenTriangle

{

public:

void draw() override;

EvenTriangleWithInclinedCross(point, point);

};

void EvenTriangleWithInclinedCross::draw(){

EvenTriangle::draw();

put\_line(neast().x,neast().y,swest().x,swest().y);

put\_line(seast().x,seast().y,nwest().x,nwest().y);

}

EvenTriangleWithInclinedCross::EvenTriangleWithInclinedCross(point sWest, point nEast): EvenTriangle(sWest,nEast){}

Функция main()

int main( )

{ setlocale(LC\_ALL, "Rus");

screen\_init( );

//== 1.Объявление набора фигур ==

rectangle hat(point(0, 0), point(10, 12));

line brim(point(0,15),17);

myshape face(point(15,10), point(27,18));

EvenTriangleWithInclinedCross beard(point(39,10), point(50,18));

EvenTriangleWithInclinedCross leftHorn(point(39-30,10+10), point(50-30,20+5));

EvenTriangleWithInclinedCross rightHorn(point(39-22,10+20), point(50-20,20+15));

shape\_refresh( );

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

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

//== 2.Подготовка к сборке ==

hat.rotate\_right( );

brim.resize(2);

face.resize(2);

leftHorn.resize(2);

beard.flip\_vertically();

leftHorn.rotate\_left();

rightHorn.rotate\_right();

shape\_refresh( );

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

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

//== 3.Сборка изображения ==

// face.move(0, -10); // Лицо - в исходное положение

up(brim, face);

up(hat, brim);

down(beard, face);

upRight(rightHorn,hat);

upLeft(leftHorn,hat);

shape\_refresh( );

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

std::cin.get(); //Смотреть результат

screen\_destroy( );

return 0;

}