**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»

**Вариант №9**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6307 |  | Лазарев С.О.  Медведев Е.Р. |
| Преподаватель |  | Колинько П.Г. |

Санкт-Петербург

2018

СОДЕРЖАНИЕ

[ЗАДАНИЕ 3](#_Toc506283915)

[ДОБАВЛЕННЫЕ КЛАССЫ 4](#_Toc506283916)

[ПЕРЕОПРЕДЕЛЁННЫЕ ФУНКЦИИ-ЧЛЕНЫ 6](#_Toc506283917)

[КОНТРОЛЬНЫЙ ПРИМЕР РАБОТЫ ПРОГРАММЫ 8](#_Toc506283918)

[ВЫВОДЫ 11](#_Toc506283919)

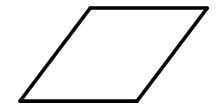
[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 12](#_Toc506283920)

[ПРИЛОЖЕНИЕ: ТЕКСТ ПРОГРАММЫ 13](#_Toc506283921)

# ЗАДАНИЕ

***Постановка задачи:*** доработка учебной программы «Библиотека фигур», а именно: добавление в коллекцию фигуры Параллелограмм, реализовать для данной фигуры поворот на влево и вправо, отражение относительно горизонтальной и вертикальной осей.

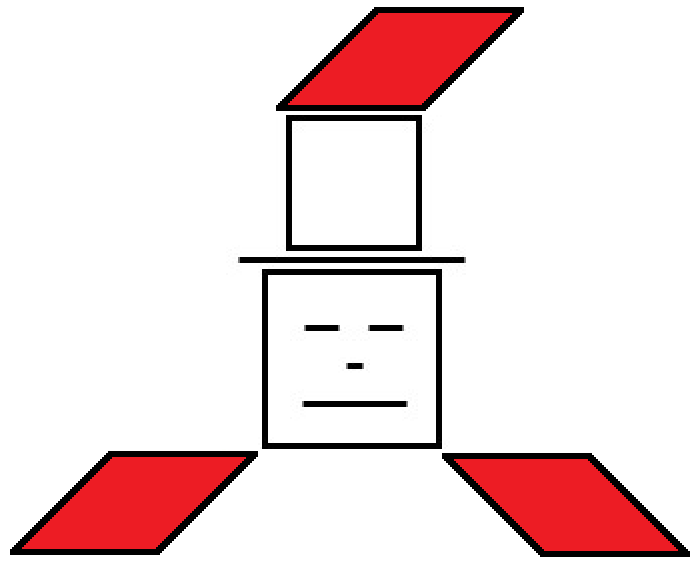
***Фигура:***

**

***Позиции для дополнения:***2, 3 – бакенбарды, 14 – шишак*.*

***Ожидаемый результат:***

*(примечание: красные фигуры – добавленные по заданию)*



# ДОБАВЛЕННЫЕ КЛАССЫ

Добавлен класс « parallelogram»:

class parallelogram : public reflectable, public rotatable

{

/\* nw ------ n ------ ne

/ /

/ /

w c e

/ /

/ /

sw ------ s ------ se \*/

protected:

point sw, ne;

int sw\_se; // длина отрезка (sw, se)

public:

parallelogram(point a, point b, int base) : sw(a), ne(b), sw\_se(base) {}

point north() const { return point(ne.x - sw\_se / 2, ne.y); } //Точки для привязки

point south() const { return point(sw.x + sw\_se / 2, sw.y); }

point east() const

{

point se = seast();

if (se.x == ne.x) // проверка на квадрат

{

return point(se.x, se.y + (ne.y - se.y) / 2);

}

// уравнение прямой

auto y = straightLineEquation(se, ne);

if (se.x < ne.x)

{

return point(se.x + (ne.x - se.x) / 2, y(static\_cast<double>(se.x) + (ne.x - se.x) / 2));

}

else

{

return point(se.x - (se.x - ne.x) / 2, y(static\_cast<double>(se.x) - (se.x - ne.x) / 2));

}

}

point west() const

{

point e = east();

return point(e.x - sw\_se, e.y);

}

point neast() const { return ne; }

point seast() const { return point(sw.x + sw\_se, sw.y); }

point nwest() const { return point(ne.x - sw\_se, ne.y); }

point swest() const { return sw; }

void draw() //Рисование

{

put\_line(ne, seast());

put\_line(sw, seast());

put\_line(sw, nwest());

put\_line(ne, nwest());

}

void move(int a, int b) //Перемещение

{

sw.x += a;

ne.x += a;

sw.y += b;

ne.y += b;

}

void resize(int d) //Изменение размера

{

sw\_se += sw\_se \* (d - 1);

point se = seast();

auto y = straightLineEquation(se, ne);

if (se.x < ne.x)

{

ne = point(se.x + (ne.x - se.x) \* d, y(static\_cast<double>(se.x) + (static\_cast<double>(ne.x) - se.x) \* d));

}

else

{

ne = point(se.x - (se.x - ne.x) \* d, y(static\_cast<double>(se.x) - (static\_cast<double>(se.x) - ne.x) \* d));

}

}

void flip\_horisontally() // Отразить горизонтально

{

int parallelogramHeight = ne.y - sw.y;

point new\_ne = seast();

new\_ne.y += parallelogramHeight;

point new\_sw = nwest();

new\_sw.y -= parallelogramHeight;

ne = new\_ne;

sw = new\_sw;

}

void flip\_vertically() // Отразить вертикально

{

// y не изменяется

point new\_ne(0, ne.y);

point se = seast();

if (ne.x > se.x)

{

new\_ne.x = ne.x - 2 \* (ne.x - se.x);

}

else

{

new\_ne.x = se.x + (se.x - ne.x);

}

ne = new\_ne;

}

void rotate\_left() //Повернуть влево

{

double new\_sw\_se = distanceBetweenPoints(seast(), ne);

point sw\_save = sw;

move(-sw.x, -sw.y);

point nw = nwest(), se = seast();

double angle = angleBetweenVectors(point(-1, 0), nwest());

point new\_sw(cos(angle) \* nw.x - sin(angle) \* nw.y,

sin(angle) \* nw.x + cos(angle) \* nw.y);

point new\_ne(cos(angle) \* se.x - sin(angle) \* se.y,

sin(angle) \* se.x + cos(angle) \* se.y);

sw\_se = new\_sw\_se;

sw = new\_sw;

ne = new\_ne;

move(sw\_save.x, sw\_save.y);

}

void rotate\_right() //Повернуть вправо

{

double new\_sw\_se = distanceBetweenPoints(seast(), ne);

point se\_save = seast();

move(-se\_save.x, -se\_save.y);

point nw = nwest(), se = seast();

double angle = angleBetweenVectors(point(1, 0), ne);

point new\_sw = se;

point new\_ne(cos(-angle) \* nw.x - sin(-angle) \* nw.y,

sin(-angle) \* nw.x + cos(-angle) \* nw.y);

sw\_se = new\_sw\_se;

sw = new\_sw;

ne = new\_ne;

move(se\_save.x, se\_save.y);

}

};

# ПЕРЕОПРЕДЕЛЁННЫЕ ФУНКЦИИ-ЧЛЕНЫ

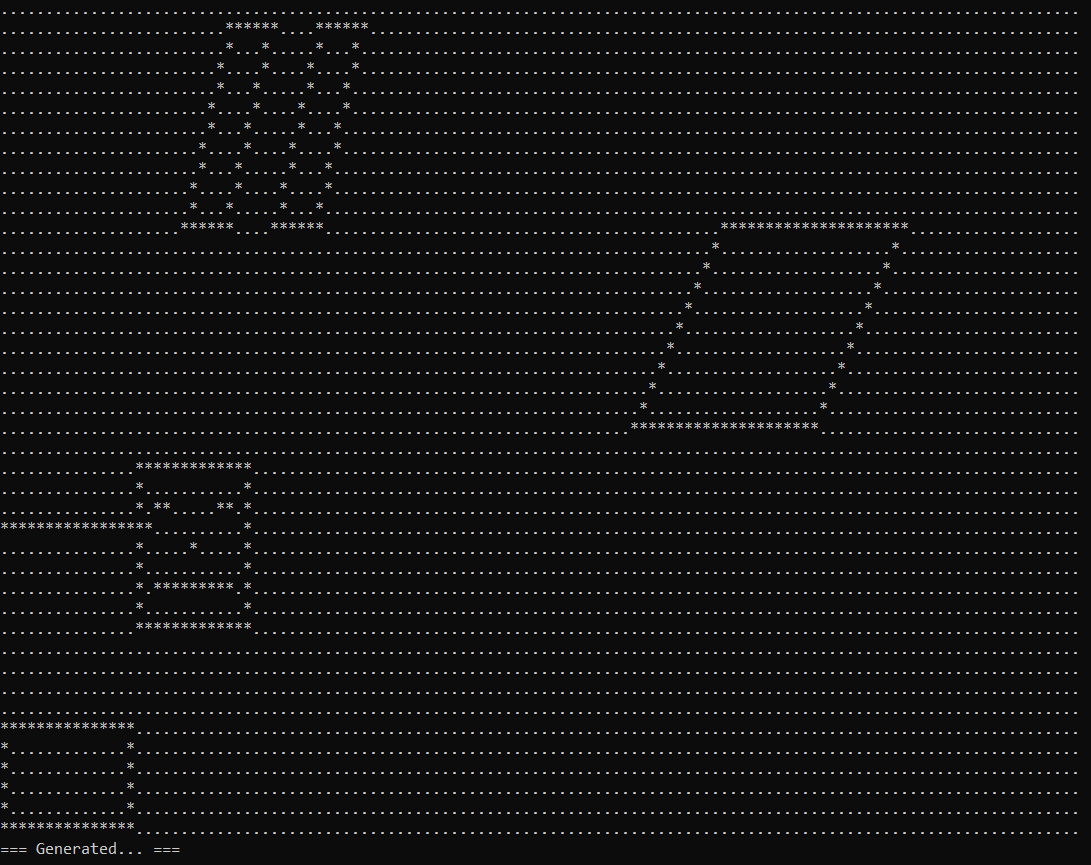
***Переопределены все функции-член, так как у параллелограмма особая форма, не похожая на квадрат, и он может быть отражен относительно осей. Все функции показаны в предыдущем разделе.***

НЕДОСТУПНЫЕ ФУНКЦИИ-ЧЛЕНЫ

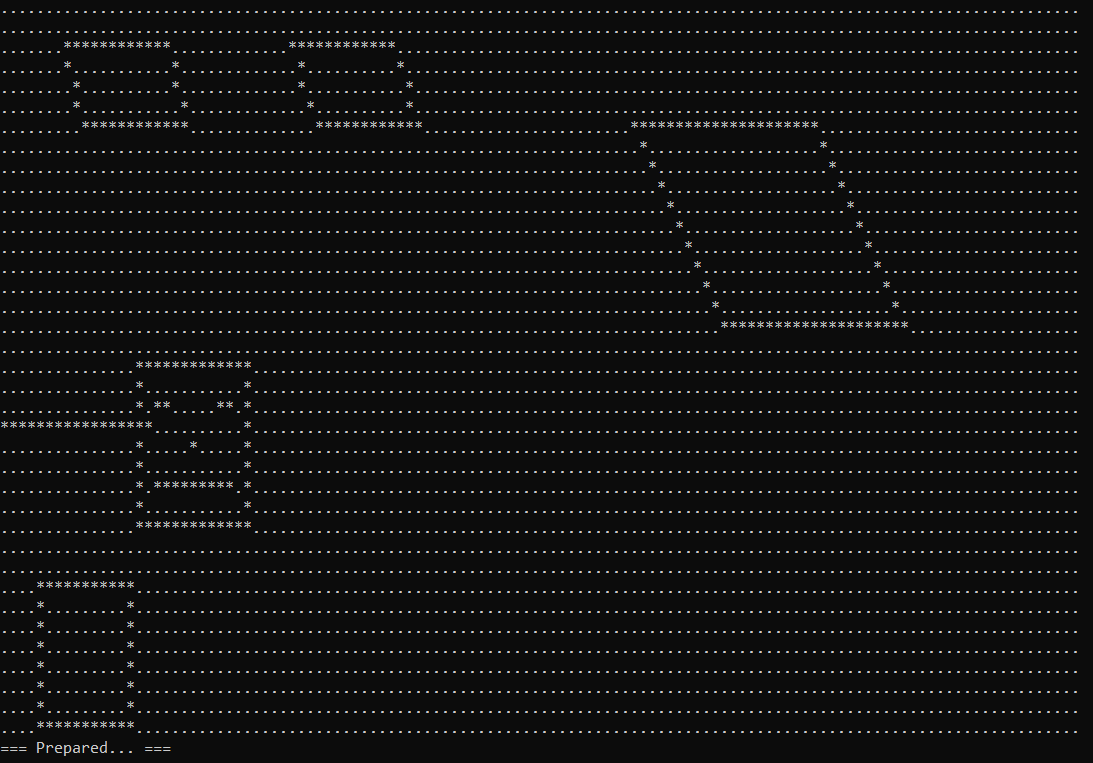
В данной работе ни одна функция-член не была реализована недоступной, так как при выполнении используются все описанные функции-члены.

# КОНТРОЛЬНЫЙ ПРИМЕР РАБОТЫ ПРОГРАММЫ

***Этап 1:***

****

***Этап 2:***

******

***Этап 3:***



**Результат совпал с ожидаемым! Не считая наклона параллелограммов, это можно исправить отражая необходимые параллелограммы.**

# ВЫВОДЫ

В ходе выполнения данной работы освоены принципы полиморфизма и наследования классов. Применение данных принципов даёт большую выгоду при разработке более широких классов, на основе имеющихся, позволяя не описывать заново часть кода.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы и структуры данных. – Лекция от 6.02.2018.
2. Алгоритмы и структуры данных. – Методические указания к лабораторным работам, практическим занятиям и курсовому проектированию, часть 2, глава 1 «Работа с иерархией объектов: наследование и полиморфизм».

# ПРИЛОЖЕНИЕ: ТЕКСТ ПРОГРАММЫ

**Класс parallelogram:**

class parallelogram : public reflectable, public rotatable

{

/\* nw ------ n ------ ne

/ /

/ /

w c e

/ /

/ /

sw ------ s ------ se \*/

protected:

point sw, ne;

int sw\_se; // длина отрезка (sw, se)

public:

parallelogram(point a, point b, int base) : sw(a), ne(b), sw\_se(base) {}

point north() const { return point(ne.x - sw\_se / 2, ne.y); } //Точки для привязки

point south() const { return point(sw.x + sw\_se / 2, sw.y); }

point east() const

{

point se = seast();

if (se.x == ne.x) // проверка на квадрат

{

return point(se.x, se.y + (ne.y - se.y) / 2);

}

// уравнение прямой

auto y = straightLineEquation(se, ne);

if (se.x < ne.x)

{

return point(se.x + (ne.x - se.x) / 2, y(static\_cast<double>(se.x) + (ne.x - se.x) / 2));

}

else

{

return point(se.x - (se.x - ne.x) / 2, y(static\_cast<double>(se.x) - (se.x - ne.x) / 2));

}

}

point west() const

{

point e = east();

return point(e.x - sw\_se, e.y);

}

point neast() const { return ne; }

point seast() const { return point(sw.x + sw\_se, sw.y); }

point nwest() const { return point(ne.x - sw\_se, ne.y); }

point swest() const { return sw; }

void draw() //Рисование

{

put\_line(ne, seast());

put\_line(sw, seast());

put\_line(sw, nwest());

put\_line(ne, nwest());

}

void move(int a, int b) //Перемещение

{

sw.x += a;

ne.x += a;

sw.y += b;

ne.y += b;

}

void resize(int d) //Изменение размера

{

sw\_se += sw\_se \* (d - 1);

point se = seast();

auto y = straightLineEquation(se, ne);

if (se.x < ne.x)

{

ne = point(se.x + (ne.x - se.x) \* d, y(static\_cast<double>(se.x) + (static\_cast<double>(ne.x) - se.x) \* d));

}

else

{

ne = point(se.x - (se.x - ne.x) \* d, y(static\_cast<double>(se.x) - (static\_cast<double>(se.x) - ne.x) \* d));

}

}

void flip\_horisontally() // Отразить горизонтально

{

int parallelogramHeight = ne.y - sw.y;

point new\_ne = seast();

new\_ne.y += parallelogramHeight;

point new\_sw = nwest();

new\_sw.y -= parallelogramHeight;

ne = new\_ne;

sw = new\_sw;

}

void flip\_vertically() // Отразить вертикально

{

// y не изменяется

point new\_ne(0, ne.y);

point se = seast();

if (ne.x > se.x)

{

new\_ne.x = ne.x - 2 \* (ne.x - se.x);

}

else

{

new\_ne.x = se.x + (se.x - ne.x);

}

ne = new\_ne;

}

void rotate\_left() //Повернуть влево

{

double new\_sw\_se = distanceBetweenPoints(seast(), ne);

point sw\_save = sw;

move(-sw.x, -sw.y);

point nw = nwest(), se = seast();

double angle = angleBetweenVectors(point(-1, 0), nwest());

point new\_sw(cos(angle) \* nw.x - sin(angle) \* nw.y,

sin(angle) \* nw.x + cos(angle) \* nw.y);

point new\_ne(cos(angle) \* se.x - sin(angle) \* se.y,

sin(angle) \* se.x + cos(angle) \* se.y);

sw\_se = new\_sw\_se;

sw = new\_sw;

ne = new\_ne;

move(sw\_save.x, sw\_save.y);

}

void rotate\_right() //Повернуть вправо

{

double new\_sw\_se = distanceBetweenPoints(seast(), ne);

point se\_save = seast();

move(-se\_save.x, -se\_save.y);

point nw = nwest(), se = seast();

double angle = angleBetweenVectors(point(1, 0), ne);

point new\_sw = se;

point new\_ne(cos(-angle) \* nw.x - sin(-angle) \* nw.y,

sin(-angle) \* nw.x + cos(-angle) \* nw.y);

sw\_se = new\_sw\_se;

sw = new\_sw;

ne = new\_ne;

move(se\_save.x, se\_save.y);

}

};

**Функции не члены класса:**

// уравнение прямой

auto straightLineEquation(const point& p1, const point& p2)

{

return [=](const double& x) -> double

{

return (static\_cast<double>(p2.y) - p1.y) / (static\_cast<double>(p2.x) - p1.x) \* (x - p1.x) + p1.y;

};

}

double distanceBetweenPoints(const point& p1, const point& p2)

{

return sqrt((static\_cast<double>(p1.x) - p2.x) \* (static\_cast<double>(p1.x) - p2.x) +

(static\_cast<double>(p1.y) - p2.y) \* (p1.y - p2.y));

}

double scalarProduct(const point& v1, const point& v2)

{

return static\_cast<double>(v1.x) \* v2.x + static\_cast<double>(v1.y) \* v2.y;

}

double vectorLength(const point& v)

{

return sqrt(static\_cast<double>(v.x) \* v.x + static\_cast<double>(v.y) \* v.y);

}

double angleBetweenVectors(const point& v1, const point& v2)

{

double ScalarProduct = scalarProduct(v1, v2);

double v1Lenght = vectorLength(v1);

double v2Lenght = vectorLength(v2);

return acos(ScalarProduct / (v1Lenght \* v2Lenght));

}

void lowerLeft(shape& p, const shape& q) // поместить p слева снизу от q

{ //Это ОБЫЧНАЯ функция, не член класса! Динамическое связывание!!

point n = q.swest();

point s = p.neast();

p.move(n.x - s.x, n.y - s.y);

}

void lowerRight(shape& p, const shape& q) // поместить p над q

{ //Это ОБЫЧНАЯ функция, не член класса! Динамическое связывание!!

point n = q.seast();

point s = p.nwest();

p.move(n.x - s.x, n.y - s.y);

}