МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВЦИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЬЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3 **по курсу**

объектно-ориентированное программирование I семестр, 2019/20 уч. год

Студент *Артемьев Дмитрий Иванович, группа М8О-206Б-18*

Преподаватель *Журавлёв Андрей Андреевич*

Условие

Задание: Вариант 1: Треугольник, Квадрат, Прямоугольник. Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Все классы должны содержать набор общих методов:

1. Вычисление геометрического центра фигуры;
2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;
3. Вычисление площади фигуры;

Создать программу, которая позволяет:

1. Вводить из стандартного ввода std::cin фигуры, согласно варианту задания.
2. Сохранять созданные фигуры в динамический массив std::vector<Figure*>
3. Вызывать для всего массива общие функции (1-3 см. выше). Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь.
4. Необходимо уметь вычислять общую площадь фигур в массиве.
5. Удалять из массива фигуру по индексу

Описание программы

Исходный код лежит в 11 файлах:

1. src/main.cxx: основная программа, взаимодействие с пользователем посредством комманд из меню
2. include/figure.hxx: описание абстрактного класса фигур
3. include/point.hxx: описание класса точки с классом my_double (для перегрузки операторов сравнения)
4. include/triangle.hxx: описание класса треугольника, наследующегося от figures
5. include/rectangle.hxx: описание класса прямоугольника, наследующегося от figures
6. include/square.hxx: описание класса квадрата, наследующегося от rectangle
7. include/figure.cxx: реализация абстрактного класса фигур
8. include/point.cxx: реализация класса точки с классом my_double (для перегрузки операторов сравнения)

9. include/triangle.cxx: реализация класса треугольника, наследующегося от figures
10. include/rectangle.cxx: реализация класса прямоугольника, наследующегося от figures
11. include/square.cxx: реализация класса квадрата, наследующегося от rectangle

Дневник отладки

Трудный путь постижения системы сборки build2.

Недочёты

Не реализована возможность автоматического запуска тестов с помощью buil2 из-за сложности этой системы.

Выводы

Научился использовать наследование в C++, использовать систему сборки build2.

Исходный код

figure.hxx

```
#pragma once

#include <iostream>

#include <include/point.hxx>

struct Figure {
    virtual Point figureCenter() const = 0;
    virtual double figureArea() const = 0;
    virtual void printPoints(std::ostream& ) const = 0;

    virtual ~Figure() = default;
};

std::ostream& operator<< (std::ostream& out, Figure const& fig);
std::istream& operator>> (std::istream& in, Figure*& fig);
```

figure.cxx

```
#include <vector>
#include <algorithm>

#include <include/figure.hxx>
#include <include/triangle.hxx>
#include <include/rectangle.hxx>
#include <include/square.hxx>

std::ostream& operator<< (std::ostream& out, Figure const& fig) {
    fig.printPoints(out);
    return out;
}

void orderPoints(std::vector<Point>& pts);
Figure* whatFigure(std::vector<Point>& pts);

std::istream& operator>> (std::istream& in, Figure*& fig) {
    std::vector<Point> pts;

    while (true) {
        char c = in.get();
        while (c == ' ' || c == '\t' || c == '\n')
            c = in.get();
        in.unget();

        if (((c < '0') || (c > '9')) && c != '-') {
            break;
        }
        else {
            Point pt;
            in >> pt;
            pts.push_back(pt);
        }
    }

    fig = whatFigure(pts);

    return in;
}

Figure* whatFigure(std::vector<Point>& pts) {
```

```
    Figure* ptr = nullptr;
    if (pts.size() == 3) {
        ptr = new Triangle(pts);
    }
    else if (pts.size() == 4) {
        if (dist(pts[0], pts[2]) == dist(pts[1], pts[3]) &&
            dist(pts[0], pts[1]) == dist(pts[2], pts[3]) &&
            dist(pts[1], pts[2]) == dist(pts[0], pts[3])) {
            if (dist(pts[0], pts[1]) == dist(pts[1], pts[2])) {
                ptr = new Square(pts);
            }
            else {
                ptr = new Rectangle(pts);
            }
        }
        else {
            //std::cout << "Wrong tetragon!" << std::endl;
        }
    }
    else {
        //std::cout << "Wrong number of points!" << std::endl;
    }

    return ptr;
}
```

point.hxx

```
#pragma once

#include <iostream>

struct my_double {
    double v;
    my_double() {};
    my_double(double x) : v(x) {};
};

struct Point {
    my_double x, y;
    Point() {};
    Point(double x, double y) : x(my_double(x)), y(my_double(y)) {};
};

std::istream& operator>> (std::istream& , Point& );
std::ostream& operator<< (std::ostream& , Point const& );
Point operator+ (Point const& , Point const& );
Point operator- (Point const& , Point const& );
Point operator* (Point const& , double const& );
Point operator/ (Point const& , double const& );

my_double operator+ (my_double const& , my_double const& );
my_double operator- (my_double const& , my_double const& );
my_double operator* (my_double const& , my_double const& );
my_double operator/ (my_double const& , my_double const& );
bool operator<  (my_double const& a, my_double const& b);
bool operator>  (my_double const& a, my_double const& b);
bool operator== (my_double const& a, my_double const& b);

bool operator< (Point const& , Point const& );
bool operator> (Point const& , Point const& );
bool operator<= (Point const& , Point const& );
bool operator>= (Point const& , Point const& );
bool operator== (Point const& , Point const& );

my_double dist(const Point& , const Point& );
```

point.cxx

```
#include <iostream>
#include <cmath>
#include <include/point.hxx>

std::istream& operator>> (std::istream& in, Point& p) {
    return in >> p.x.v >> p.y.v;
}

std::ostream& operator<< (std::ostream& out, Point const& p) {
    return out << "{"  << p.x.v << ' ' << p.y.v << "} ";
}

Point operator+ (Point const& a, Point const& b) {
    return Point{a.x.v + b.x.v, a.y.v + b.y.v};
}

Point operator- (Point const& a, Point const& b) {
    return Point{a.x.v - b.x.v, a.y.v - b.y.v};
}

Point operator* (Point const& a, double const& b) {
    return Point{a.x.v * b, a.y.v * b};
}

Point operator/ (Point const& a, double const& b) {
    return Point{a.x.v / b, a.y.v / b};
}



my_double operator+ (my_double const& a, my_double const& b) {
    return my_double{a.v + b.v};
}

my_double operator- (my_double const& a, my_double const& b) {
    return my_double{a.v - b.v};
}

my_double operator* (my_double const& a, my_double const& b) {
    return my_double{a.v * b.v};
}
```

```
my_double operator/ (my_double const& a, my_double const& b) {
    return my_double{a.v / b.v};
}



const double EPS = 1e-9;

bool operator< (my_double const& a, my_double const& b) {
    return b.v - a.v > EPS;
}

bool operator> (my_double const& a, my_double const& b) {
    return a.v - b.v > EPS;
}

bool operator== (my_double const& a, my_double const& b) {
    return !(a.v < b.v) && !(a.v > b.v);
}

bool operator<  (Point const& a, Point const& b) {
    return (a.x.v < b.x.v) || (a.x.v == b.x.v && a.y.v < b.y.v);
}

bool operator>  (Point const& a, Point const& b) {
    return (a.x.v > b.x.v) || (a.x.v == b.x.v && a.y.v > b.y.v);
}

bool operator== (Point const& a, Point const& b) {
    return (a.x.v == b.x.v) && (a.y.v == b.y.v);
}

bool operator<= (Point const& a, Point const& b) {
    return a == b || a < b;
}

bool operator>= (Point const& a, Point const& b) {
    return a == b || a > b;
}

my_double dist(const Point& a, const Point& b) {
```

```
    return my_double(sqrt((a.x.v - b.x.v) * (a.x.v - b.x.v) +
                          (a.y.v - b.y.v) * (a.y.v - b.y.v)));
}
```

triangle.hxx

```
#pragma once

#include <iostream>
#include <vector>

#include <include/figure.hxx>

class Triangle : public Figure{
public:
    Triangle(std::vector<Point> const& );

    Point figureCenter() const override;
    double figureArea() const override;
    void printPoints(std::ostream& ) const override;

    ~Triangle() = default;
protected:
    static const int NUM = 3;
    Point pts[NUM];
};
```

triangle.cxx

```
#include <include/triangle.hxx>
#include <algorithm>

Triangle::Triangle(std::vector<Point> const& fig) {
    for (int i = 0; i < NUM; i++) {
        pts[i] = fig[i];
    }
}

Point Triangle::figureCenter() const {
    return (pts[0] + pts[1] + pts[2]) / NUM;
}



double Triangle::figureArea() const {
    return std::abs((
                    (pts[1].x - pts[0].x) * (pts[3].y - pts[0].y) -
                    (pts[1].y - pts[0].y) * (pts[3].x - pts[0].x)
                    ).v) / 2;
}

void Triangle::printPoints(std::ostream& out) const {
    out << "Triangle: " << std::endl;
    for (Point const& pt : pts) {
        out << pt << std::endl;
    }
}
```

rectangle.hxx

```
#pragma once

#include <iostream>
#include <vector>

#include <include/figure.hxx>

class Rectangle : public Figure {
public:
    Rectangle(std::vector<Point> const& );

    Point figureCenter() const override;
    double figureArea() const override;
    void printPoints(std::ostream& ) const override;

    ~Rectangle() = default;
protected:
    static const int NUM = 4;
    Point pts[NUM];
};
```

rectangle.cxx

```
#include <include/rectangle.hxx>
#include <algorithm>

Rectangle::Rectangle(std::vector<Point> const& fig) {
    for (int i = 0; i < NUM; i++) {
        pts[i] = fig[i];
    }
}

Point Rectangle::figureCenter() const {
    return (pts[0] + pts[1] + pts[2] + pts[3]) / 4;
}

double Rectangle::figureArea() const {
    return std::abs(((pts[2].x - pts[0].x) * (pts[1].y - pts[0].y)).v);
}

void Rectangle::printPoints(std::ostream& out) const {
    out << "Rectangle: " << std::endl;
    for (Point const & pt : pts) {
        out << pt << std::endl;
    }
}
```

square.hxx

```
#pragma once

#include <include/figure.hxx>
#include <include/rectangle.hxx>

class Square : public Rectangle {
public:
    Square(std::vector<Point> const& fig) : Rectangle(fig) {};
    void printPoints(std::ostream& out) const override;
    ~Square() = default;
};
```

square.cxx

```
#include <include/square.hxx>

void Square::printPoints(std::ostream& out) const {
    out << "Square: " << std::endl;
    for (Point const& pt : pts) {
        out << pt << std::endl;
    }
}
```

main.cxx

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>

#include <include/figure.hxx>
#include <include/triangle.hxx>
#include <include/square.hxx>
#include <include/rectangle.hxx>

int main(){
    std::vector<Figure*> figures;
    std::string query;
    std::cout << std::fixed << std::setprecision(6);

    while (std::cin >> query) {

        if (query == "read") {
            Figure *fig;
            std::cin >> fig;
            if (fig != nullptr)
                figures.push_back(fig);
        }

        else if (query == "delete") {
            int id;
            std::cin >> id;
            if (id < 1 || id > figures.size()){
                std::cout << "There is no such element!" << std::endl;
                continue;
            }
            id--;
            delete figures[id];
            figures.erase(figures.begin() + id);
        }

        else if (query == "area" || query == "areas") {
            std::cout << "Areas: " << std::endl;
            for (Figure* ptr : figures) {
                std::cout << ptr->figureArea() << std::endl;
            }
```

```
    }

    else if (query == "center" || query == "centers"){
        std::cout << "Centers: " << std::endl;
        for (Figure* ptr : figures) {
            std::cout << ptr->figureCenter() << std::endl;
        }
    }

    else if (query == "figure" || query == "figures") {
        std::cout << "Figures: " << std::endl;
        for (Figure* ptr : figures) {
            std::cout << (*ptr) << std::endl;
        }
    }

    else if (query == "sum_area" || query == "sum_areas") {
        double area = 0;
        std::cout << "Summary area: " << std::endl;
        for (Figure* ptr : figures) {
            area += ptr->figureArea();
        }
        std::cout << area << std::endl;
    }

    else if (query == "help"){
        std::cout << "read, delete, areas, centers, figures, sum_area, help, exit" <
    }

    else if (query == "exit"){
        return 0;
    }

    else {
        std::cout << "Wrong command!" << std::endl;
    }

}

for (Figure* ptr : figures) {
    delete ptr;
}
```

```
    return 0;
}
```