Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа
по курсу «ООП»

Тема:
Наследование. Полиморфизм.

Студент:	Петрин С.А.
Группа:	М80-207Б-18
Преподаватель:	Чернышов Л.Н.
Вариант:	17
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

point.h:

```
#ifndef POINT_H
#define POINT_H 1

#include <iostream>

class Point {
	double x;
	double y;
public:
	Point();
	Point(double a, double b);
	Point(const Point &other);
	double X() const;
	double Y() const;
	Point operator+(const Point &a) const;
	Point operator-(const Point &a) const;
	Point operator*(double a) const;
	Point operator/(double a) const;
	friend std::ostream &operator<<(std::ostream &out, const Point &a);
	friend std::istream &operator>>(std::istream &in, Point &a);
};

#endif // POINT_H
```

point.cpp:

```
#include "point.h"
#include <cmath>

Point::Point() : x{0}, y{0} {}

Point::Point(double a, double b) : x{a}, y{b} {}

Point::Point(const Point &other) : x{other.x}, y{other.y} {}

double Point::X() const {
	return x;
}

double Point::Y() const {
	return y;
}

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

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

```
}

Point Point::operator*(double a) const {
	return {x * a, y * a};
}

Point Point::operator/(double a) const {
	return {x / a, y / a};
}

std::ostream &operator<<(std::ostream &out, const Point &a) {
	out << "(" << a.x << "; " << a.y << ")";

	return out;
}

std::istream &operator>>(std::istream &in, Point &a) {
	in >> a.x >> a.y;

	return in;
}
```

figure.h:

```
#ifndef FIGURE_H
#define FIGURE_H 1

#include "point.h"
#include <iostream>

class Figure {
public:
	virtual Point Center() const = 0;
	virtual double Area() const = 0;
	virtual std::ostream &Print(std::ostream &out) const = 0;
	virtual std::istream &Scan(std::istream &in) = 0;
	virtual ~Figure() = default;
};

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

#endif //FIGURE_H
```

figure.cpp:

```
#include "figure.h"

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

	return out;
```

```
}

std::istream &operator>>(std::istream &in, Figure &fig) {
	fig.Scan(in);

	return in;
}
```

triangle.h:

```
#ifndef TRIANGLE_H
#define TRIANGLE_H 1

#include "point.h"
#include "figure.h"
#include "vector.h"

class Triangle : public Figure {
	Point A, B, C;
public:
	Triangle();
	Triangle(Point a, Point b, Point c);
	Point Center() const override;
	double Area() const override;
	std::ostream &Print(std::ostream &out) const override;
	std::istream &Scan(std::istream &in) override;
};

#endif //TRIANGLE_H
```

triangle.cpp:

```
#include "triangle.h"
#include "figure.h"
#include "point.h"
#include "vector.h"
#include <cmath>

Triangle::Triangle() : A{Point{}}, B{Point{}}, C{Point{}} {}

Triangle::Triangle(Point a, Point b, Point c) : A{a}, B{b}, C{c} {
	double AB = Length(A, B), BC = Length(B, C), AC = Length(A, C);
	if (AB >= BC + AC || BC >= AB + AC || AC >= AB + BC) {
		throw std::logic_error("Any side of the triangle must be less than the sum of the other two
sides");
	}
}

Point Triangle::Center() const {
	Point mid_of_base{ (A + C) / 2 };
```

```
    return { (B + mid_of_base * 2) / 3 };
}

double Triangle::Area() const {
    double AB = Length(A, B), BC = Length(B, C), AC = Length(A, C);
    double perim = AB + BC + AC;

    return sqrt((perim / 2) * (perim / 2 - AB) * (perim / 2 - BC) * (perim / 2 - AC));
}

std::ostream &Triangle::Print(std::ostream &out) const {
    out << "Triangle: p1 = " << A << ", p2 = " << B << ", p3 = " << C;

    return out;
}

std::istream &Triangle::Scan(std::istream &in) {
    in >> A >> B >> C;
    (*this) = Triangle(A, B, C);

    return in;
}
```

square.h:

```
#ifndef SQUARE_H
#define SQUARE_H 1

#include "point.h"
#include "figure.h"
#include "vector.h"

class Square : public Figure {
    Point A, B, C, D;
public:
    Square();
    Square(Point a, Point b, Point c, Point d);
    Point Center() const override;
    double Area() const override;
    std::ostream &Print(std::ostream &out) const override;
    std::istream &Scan(std::istream &in) override;
};

#endif //SQUARE_H
```

square.cpp:

```
#include "square.h"
#include "vector.h"

Square::Square() : A{Point{}}, B{Point{}}, C{Point{}}, D{Point{}} {}
```

```
Square::Square(Point a, Point b, Point c, Point d) : A{a}, B{b}, C{c}, D{d} {
	Vector AB{ A, B }, BC{ B, C }, CD { C, D }, DA { D, A };
	if (!is_parallel(DA, BC)) {
		std::swap(A, B);
		AB = { A, B };
		BC = { B, C };
		CD = { C, D };
		DA = { D, A };
	}
	if (!is_parallel(AB, CD)) {
		std::swap(B, C);
		AB = { A, B };
		BC = { B, C };
		CD = { C, D };
		DA = { D, A };
	}
	if (AB * BC || BC * CD || CD * DA || DA * AB) {
		throw std::logic_error("The sides of the square should be perpendicular");
	}
	if (Length(AB) != Length(BC) || Length(BC) != Length(CD) || Length(CD) != Length(DA) ||
Length(DA) != Length(AB)) {
		throw std::logic_error("The sides of the square should be equal");
	}
	if (!Length(AB) || !Length(BC) || !Length(CD) || !Length(DA)) {
		throw std::logic_error("The sides of the square must be greater than zero");
	}
}

Point Square::Center() const {
	return Point{ (B + D) / 2 };
}

double Square::Area() const {
	return Length(A, B) * Length(A, B);
}

std::ostream &Square::Print(std::ostream &out) const {
	out << "Square: p1 = " << A << ", p2 = " << B << ", p3 = " << C << ", p4 = " << D;

	return out;
}

std::istream &Square::Scan(std::istream &in) {
	in >> A >> B >> C >> D;
	(*this) = Square(A, B, C, D);

	return in;
}
```

rectangle.h:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H 1
#include "point.h"
#include "figure.h"
#include "vector.h"

class Rectangle : public Figure {
    Point A, B, C, D;
public:
    Rectangle();
    Rectangle(Point a, Point b, Point c, Point d);
    Point Center() const override;
    double Area() const override;
    std::ostream &Print(std::ostream &out) const override;
    std::istream &Scan(std::istream &in) override;
}
#endif //RECTANGLE_H
```

rectangle.cpp:

```
#include "rectangle.h"
#include "vector.h"

Rectangle::Rectangle() : A{Point{}}, B{Point{}}, C{Point{}}, D{Point{}} {}

Rectangle::Rectangle(Point a, Point b, Point c, Point d) : A{a}, B{b}, C{c}, D{d} {
    Vector AB{ A, B }, BC{ B, C }, CD { C, D }, DA { D, A };
    if (!is_parallel(DA, BC)) {
        std::swap(A, B);
        AB = { A, B };
        BC = { B, C };
        CD = { C, D };
        DA = { D, A };
    }
    if (!is_parallel(AB, CD)) {
        std::swap(B, C);
        AB = { A, B };
        BC = { B, C };
        CD = { C, D };
        DA = { D, A };
    }
    if (AB * BC || BC * CD || CD * DA || DA * AB) {
        throw std::logic_error("The sides of the rectangle should be perpendicular");
    }
    if (!Length(AB) || !Length(BC) || !Length(CD) || !Length(DA)) {
        throw std::logic_error("The sides of the rectangle must be greater than zero");
    }
}

Point Rectangle::Center() const {
    return Point{ (B + D) / 2 };
}
```

```
double Rectangle::Area() const {
	return Length(D, A) * Length(B, A);
}

std::ostream &Rectangle::Print(std::ostream &out) const {
	out << "Rectangle: p1 = " << A << ", p2 = " << B << ", p3 = " << C << ", p4 = " << D;

	return out;
}

std::istream &Rectangle::Scan(std::istream &in) {
	in >> A >> B >> C >> D;
	(*this) = Rectangle(A, B, C, D);

	return in;
}
```

vector.h:

```
#ifndef VECTOR_H
#define VECTOR_H 1

#include "point.h"

class Vector {
public:
	double x, y;
	Vector(double x_cord, double y_cord);
	Vector(Point &p1, Point &p2);
	double operator*(const Vector &a) const;
	Vector &operator=(const Vector &a);
};
	double Length(const Point &a, const Point &b);
	double Length(const Vector &a);
	bool is_parallel(const Vector &a, const Vector &b);

#endif// VECTOR_H
```

vector.cpp:

```
#include "vector.h"
#include "point.h"
#include <cmath>

Vector::Vector(double x_cord, double y_cord) : x{x_cord}, y{y_cord} {}

Vector::Vector(Point &p1, Point &p2) : x{p2.X() - p1.X()}, y{p2.Y() - p1.Y()} {}

double Vector::operator*(const Vector &a) const {
	return (x * a.x) + (y * a.y);
}
```

```
Vector &Vector::operator=(const Vector &a) {
	x = a.x;
	y = a.y;

	return *this;
}

double Length(const Point &a, const Point &b) {
	return sqrt(pow((b.X() - a.X()), 2) + pow((b.Y() - a.Y()), 2));
}

double Length(const Vector &a) {
	return sqrt(pow(a.x, 2) + pow(a.y, 2));
}

bool is_parallel(const Vector &a, const Vector &b) {
	return (a.x * b.y) - (a.y * b.x) == 0;
}
```

main.cpp:

```
#include <iostream>
#include <vector>
#include <string>
#include <exception>
#include <cmath>
#include "point.h"
#include "vector.h"
#include "figure.h"
#include "triangle.h"
#include "square.h"
#include "rectangle.h"

int main() {
	std::vector<Figure *> figures;
	std::string command;
	std::cout << "Operations: Add/ Print/ Area/ Center/ Total_area/ Print_all/ Delete/ Delete_all/
Quit" << std::endl;
	std::cout << "_____" << std::endl;
	while (std::cin >> command) {
		if (command == "Add") {
			std::string fig_type;
			std::cout << "Figures: Triangle/ Square/ Rectangle" << std::endl;
			std::cin >> fig_type;
			Figure *new_fig;
			if (fig_type == "Triangle") {
				new_fig = new Triangle;
			}
			else if (fig_type == "Square") {
				new_fig = new Square;
			}
```

```
        else if (fig_type == "Rectangle") {
            new_fig = new Rectangle;
        }
        else {
            std::cout << "Invalid figure type" << std::endl;
            std::cin.clear();
            std::cin.ignore(30000, '\n');
            std::cout << "_____" << std::endl;
            continue;
        }
        try {
            std::cout << "Points: ";
            std::cin >> (*new_fig);
            std::cout << "_____" << std::endl;
        }
        catch (std::exception &e) {
            std::cout << e.what() << std::endl;
            std::cout << "_____" << std::endl;
            delete new_fig;
            continue;
        }
        figures.push_back(new_fig);
    }
    else if (command == "Print") {
        unsigned int index;
        std::cout << "Index: ";
        std::cin >> index;
        if (index < 0 || index >= figures.size()) {
            std::cout << "No object at that index" << std::endl;
            std::cout << "_____" << std::endl;
            continue;
        }
        std::cout << "Figure at index " << index << ": " << *figures[index] << std::endl;
        std::cout << "_____" << std::endl;
    }
    else if (command == "Area") {
        unsigned int index;
        std::cout << "Index: ";
        std::cin >> index;
        if (index < 0 || index >= figures.size()) {
            std::cout << "No object at that index" << std::endl;
            std::cout << "_____" << std::endl;
            continue;
        }
        std::cout << *figures[index] << ". Area: " << figures[index]->Area() << std::endl;
        std::cout << "_____" << std::endl;
    }
    else if (command == "Center") {
        unsigned int index;
        std::cout << "Index: ";
        std::cin >> index;
        if (index < 0 || index >= figures.size()) {
```

```
                std::cout << "No object at that index" << std::endl;
                std::cout << "_____" << std::endl;
                continue;
            }
            std::cout << *figures[index] << ". Center: " << figures[index]->Center() << std::endl;
            std::cout << "_____" << std::endl;
        }
        else if (command == "Total_area") {
            double area = 0;
            for (Figure *ptr : figures) {
                area += ptr->Area();
            }
            std::cout << "Total area: " << area << std::endl;
            std::cout << "_____" << std::endl;
        }
        else if (command == "Print_all") {
            for (unsigned int i = 0; i < figures.size(); i++) {
                std::cout << i << ". " << *figures[i] << std::endl;
            }
            std::cout << "_____" << std::endl;
        }
        else if (command == "Delete") {
            unsigned int index;
            std::cout << "Index: ";
            std::cin >> index;
            if (index < 0 || index >= figures.size()) {
                std::cout << "No object at that index" << std::endl;
                std::cout << "_____" << std::endl;
                continue;
            }
            delete figures[index];
            figures.erase(figures.begin() + index);
            std::cout << "_____" << std::endl;
        }
        else if (command == "Delete_all") {
            for (std::vector<Figure *>::iterator i = figures.begin(); i != figures.end(); i++) {
                delete *i;
            }
            figures.clear();
            std::cout << "_____" << std::endl;
        }
        else if (command == "Quit") {
            for (unsigned int i = 0; i < figures.size(); i++) {
                delete figures[i];
            }
            std::cout << "_____" << std::endl;
            return 0;
        }
        else {
            std::cout << "Wrong. Operations: Add/ Print/ Area/ Center/ Total_area/ Print_all/ Delete/ Delete_all/ Quit" << std::endl;
            std::cout << "_____" << std::endl;
```

```
            }
        }
        return 0;
}
```

CmakeLists.txt:

```
cmake_minimum_required (VERSION 3.2)

project(oop_exercise_03)

add_executable(oop_exercise_03 main.cpp point.cpp figure.cpp triangle.cpp square.cpp rectangle.cpp vector.cpp)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall")

set_target_properties(oop_exercise_03 PROPERTIES CXX_STANDART 14 CXX_STANDART_REQUIRED ON)
```

2. Ссылка на репозиторий на GitHub.

https://github.com/SergeiPetrin/OOP/tree/master/oop_exercise_03

3. Набор testcases.

test_00.test:

```
Add
Square
0 0 0 5 5 5 5 0
Add
Triangle
0 0 1 1 2 0
Add
Rectangle
6 7 6 15 20 15 20 7
Area
0
Area
1
Area
2
Print_all
Center
0
Center
1
Center
2
Total_area
Delete
```

2
Total_area
Delete_all
Print_all
Quit

test_01.test:
Add
Triangle
-5 -5 -4 -4 -3 -5
Add
Square
0 0 1 1 2 1 2 0
Add
Triangle
0 0 1 1 2 0
Add
Rectangle
6 7 6 15 20 15 20 7
Area
0
Area
1
Area
2
Area
3
Print_all
Center
0
Center
1
Center
2
Center
3
Total_area
Delete
0
Total_area
Delete_all
Print_all
Quit

4. Результаты выполнения тестов.

test_00.result:

Operations: Add/ Print/ Area/ Center/ Total_area/ Print_all/ Delete/ Delete_all/ Quit
_____
Figures: Triangle/ Square/ Rectangle
Points: _____
Figures: Triangle/ Square/ Rectangle
Points: _____
Figures: Triangle/ Square/ Rectangle
Points: _____
Index: Square: p1 = (0; 0), p2 = (0; 5), p3 = (5; 5), p4 = (5; 0). Area: 25
_____
Index: Triangle: p1 = (0; 0), p2 = (1; 1), p3 = (2; 0). Area: 1
_____
Index: Rectangle: p1 = (6; 7), p2 = (6; 15), p3 = (20; 15), p4 = (20; 7). Area: 112
_____
0. Square: p1 = (0; 0), p2 = (0; 5), p3 = (5; 5), p4 = (5; 0)
1. Triangle: p1 = (0; 0), p2 = (1; 1), p3 = (2; 0)
2. Rectangle: p1 = (6; 7), p2 = (6; 15), p3 = (20; 15), p4 = (20; 7)
_____
Index: Square: p1 = (0; 0), p2 = (0; 5), p3 = (5; 5), p4 = (5; 0). Center: (2.5; 2.5)
_____
Index: Triangle: p1 = (0; 0), p2 = (1; 1), p3 = (2; 0). Center: (1; 0.333333)
_____
Index: Rectangle: p1 = (6; 7), p2 = (6; 15), p3 = (20; 15), p4 = (20; 7). Center: (13; 11)
_____
Total area: 138
_____
Index: _____
Total area: 26
_____
_____
_____
_____

test_01.result:
Operations: Add/ Print/ Area/ Center/ Total_area/ Print_all/ Delete/ Delete_all/ Quit
_____
Figures: Triangle/ Square/ Rectangle
Points: _____
Figures: Triangle/ Square/ Rectangle
Points: The sides of the square should be perpendicular
_____
Figures: Triangle/ Square/ Rectangle
Points: _____
Figures: Triangle/ Square/ Rectangle

Points: _____
Index: Triangle: p1 = (-5; -5), p2 = (-4; -4), p3 = (-3; -5). Area: 1

_____
Index: Triangle: p1 = (0; 0), p2 = (1; 1), p3 = (2; 0). Area: 1

_____
Index: Rectangle: p1 = (6; 7), p2 = (6; 15), p3 = (20; 15), p4 = (20; 7). Area: 112

_____
Index: No object at that index

_____
0. Triangle: p1 = (-5; -5), p2 = (-4; -4), p3 = (-3; -5)
1. Triangle: p1 = (0; 0), p2 = (1; 1), p3 = (2; 0)
2. Rectangle: p1 = (6; 7), p2 = (6; 15), p3 = (20; 15), p4 = (20; 7)

_____
Index: Triangle: p1 = (-5; -5), p2 = (-4; -4), p3 = (-3; -5). Center: (-4; -4.66667)

_____
Index: Triangle: p1 = (0; 0), p2 = (1; 1), p3 = (2; 0). Center: (1; 0.333333)

_____
Index: Rectangle: p1 = (6; 7), p2 = (6; 15), p3 = (20; 15), p4 = (20; 7). Center: (13; 11)

_____
Index: No object at that index

_____
Total area: 114

_____
Index: _____
Total area: 113

_____
_____
_____

5. Объяснение результатов работы программы.

Реализован абстрактный класс Figure, имеющий чисто виртуальные функции для вычисления центра, площади, для ввода/вывода из потоков. В конструкторах классов-наследников Triangle, Square, Rectangle предусмотрены проверки на корректность введенных точек. Также точки передаются в любом порядке. Площадь треугольника высчитывается по формуле Герона (s = sqrt(p(p-a)(p-b)(p-c)), где p — полупериметр). Площадь квадрата высчитывается по формуле (a * a). Площадь прямоугольника высчитывается по стандартной формуле (a * b). Центр треугольника вычисляется по формуле (B + mid_of_base * 2) / 3, где B — точка, не лежащая на основании, mid_of_base — середина основания. Центры квадрата и прямоугольника вычисляются по одной формуле: (B + D) / 2, где B и D — точки, лежащие на одной диагонали.

Для удобства использования создано меню:

- Add FIG_TYPE POINTS — создание новой фигуры типа FIG_TYPE по переданным точкам. Фигура добавляется в конец вектора.
- Print INDEX — вывод фигуры по индексу.
- Area INDEX — вывод площади фигуры по индексу.
- Center INDEX — вывод центра фигуры по индексу.
- Total_area — общая площадь фигур в векторе.
- Print_all — вывод всех фигур в векторе.
- Delete — удаление фигуры по индексу.
- Delete_all — удаление всех фигур.
- Quit — выход.

6. Вывод.

Наследование — принцип ООП, позволяющий благодаря ему создавать иерархические классификации. Используя наследование, можно создать общий класс, который определяет характеристики, присущие множеству связанных элементов.

Полиморфизм — процесс, в котором различные реализации функции могут быть доступны посредством одного и того же имени. В C++ полиморфизм поддерживается как во время выполнения, так и в период компиляции программы. Перегрузка операторов и функций — примеры полиморфизма, относящегося ко времени компиляции. Но несмотря на мощность механизма перегрузки операторов и функций, он не в состоянии решить все задачи. Поэтому в C++ также реализован полиморфизм периода выполнения на основе использования производных классов (класс, который наследует базовый класс) и виртуальных функций (функция, которая объявляется в базовом классе с использованием ключевого слова virtual и переопределяется в одном или нескольких производных классах).