Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование» III семестр

Задание 3: «Наследование, полиморфизм»

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №26 |
| Студент: | Хитриков Артемий Юрьевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 09.01.2019 |

Москва, 2019

# 1. Задание

(*вариант № 6*):

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать 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: • Вводить из стандартного ввода std::cin фигуры, согласно варианту задания. • Сохранять созданные фигуры в динамический массив std::vector • Вызывать для всего массива общие функции (1-3 см. выше).Т.е. распечатывать для каждой фигуры в массиве геометрический центр, координаты вершин и площадь. • Необходимо уметь вычислять общую площадь фигур в массиве. • Удалять из массива фигуру по индексу;

Согласно 26 варианту, необходимо реализовать следующие фигуры:

Квадрат; прямоугольник; трапеция.

**2. Адрес репозитория на GitHub**

[https://github.com/DeZellt/oop\_exercise\_0](https://github.com/DeZellt/oop_exercise_02)3

# 3. Код программы на С++

Заголовочные файлы:

Point.h

#pragma once

#include <iostream>

#include <cmath>

struct Point {

double x;

double y;

};

std::istream& operator >> (std::istream& str, Point& p);

std::ostream& operator << (std::ostream& str, const Point& p);

Point operator + (Point lhs, Point rhs);

Point operator - (Point lhs, Point rhs);

Point operator / (Point lhs, double a);

Point operator \* (Point lhs, double a);

class Vector {

public:

explicit Vector(double a, double b);

explicit Vector(Point a, Point b);

bool operator == (Vector rhs);

Vector operator - ();

friend double operator \* (Vector lhs, Vector rhs);

double length() const;

double x;

double y;

};

bool is\_parallel(const Vector& lhs, const Vector& rhs);

bool is\_perpendecular(const Vector& lhs, const Vector& rhs);

figure.h

#pragma once

#include <numeric>

#include <iostream>

#include <vector>

#include <cmath>

#include <limits>

#include "Point.h"

class Figure {

public:

virtual Point Center() const = 0;

virtual double Area() const = 0;

virtual void Scan(std::istream& is) = 0;

virtual void Print(std::ostream& os) const = 0;

virtual ~Figure() = default;

};

std::ostream& operator << (std::ostream& os, const Figure& fig);

std::istream& operator >> (std::istream& is, Figure& fig);

Rectangle.h

#pragma once

#include "Figure.h"

class Rectangle : public Figure {

public:

Rectangle() = default;

Rectangle(Point p1, Point p2, Point p3, Point p4);

Point Center() const override;

virtual void Scan(std::istream& is) override;

void Print(std::ostream& os) const override;

double Area() const override;

private:

Point p1\_, p2\_, p3\_, p4\_;

};

#pragma once

#include "Figure.h"

class Square : public Figure {

public:

Square() = default;

Square(Point p1, Point p2, Point p3, Point p4);

Point Center() const override;

void Scan(std::istream& is) override;

void Print(std::ostream& os) const override;

double Area() const override;

private:

Point p1\_, p2\_, p3\_, p4\_;

};

Trapeze.h

#pragma once

#include "Figure.h"

#include <exception>

class Trapeze : public Figure {

public:

Trapeze() = default;

Trapeze(Point p1, Point p2, Point p3, Point p4);

void Print(std::ostream& os) const override;

void Scan(std::istream& is) override;

Point Center() const override;

double Area() const override;

private:

Point p1\_, p2\_, p3\_, p4\_;

};

**cpp-файлы**

#include "Point.h"

Point operator + (Point lhs, Point rhs) {

return {lhs.x + rhs.x, lhs.y + rhs.y};

}

Point operator - (Point lhs, Point rhs) {

return {lhs.x - rhs.x, lhs.y - rhs.y};

}

Point operator / (Point lhs, double a) {

return { lhs.x / a, lhs.y / a};

}

Point operator \* (Point lhs, double a) {

return {lhs.x \* a, lhs.y \* a};

}

bool operator < (Point lhs, Point rhs) {

return (lhs.x \* lhs.x + lhs.y \* lhs.y) < (lhs.x \* lhs.x + lhs.y \* lhs.y);

}

double operator \* (Vector lhs, Vector rhs) {

return lhs.x \* rhs.x + lhs.y \* rhs.y;

}

bool is\_parallel(const Vector& lhs, const Vector& rhs) {

return (lhs.x \* rhs.y - lhs.y \* rhs.y) == 0;

}

bool Vector::operator == (Vector rhs) {

return

std::abs(x - rhs.x) < std::numeric\_limits<double>::epsilon() \* 100

&& std::abs(y - rhs.y) < std::numeric\_limits<double>::epsilon() \* 100;

}

double Vector::length() const {

return sqrt(x\*x + y\*y);

}

Vector::Vector(double a, double b)

: x(a), y(b) {

}

Vector::Vector(Point a, Point b)

: x(b.x - a.x), y(b.y - a.y){

}

Vector Vector::operator - () {

return Vector(-x, -y);

}

bool is\_perpendecular(const Vector& lhs, const Vector& rhs) {

return (lhs \* rhs) == 0;

}

std::ostream& operator << (std::ostream& str, const Point& p) {

return str << p.x << " " << p.y;

}

std::istream& operator >> (std::istream& str, Point& p) {

return str >> p.x >> p.y;

}

Figure.cpp

#include "Figure.h"

std::ostream& operator << (std::ostream& os, const Figure& fig) {

fig.Print(os);

return os;

}

std::istream& operator >> (std::istream& is, Figure& fig) {

fig.Scan(is);

return is;

}

#include "Rectangle.h"

Rectangle::Rectangle(Point p1, Point p2, Point p3, Point p4)

: p1\_(p1), p2\_(p2), p3\_(p3), p4\_(p4){

if (is\_perpendecular(Vector(p1\_, p2\_), Vector(p1\_,p3\_))

&& is\_perpendecular(Vector(p4\_, p2\_), Vector(p4\_,p3\_))

&& is\_perpendecular(Vector(p1\_, p3\_), Vector(p3\_,p4\_))

&& is\_perpendecular(Vector(p1\_, p2\_), Vector(p2\_,p4\_))) {

} else if (is\_perpendecular(Vector(p1\_, p4\_), Vector(p1\_,p3\_))

&& is\_perpendecular(Vector(p2\_, p4\_), Vector(p2\_,p3\_))

&& is\_perpendecular(Vector(p1\_, p3\_), Vector(p3\_,p2\_))

&& is\_perpendecular(Vector(p1\_, p4\_), Vector(p2\_,p4\_))){

std::swap(p2\_,p4\_);

} else if (is\_perpendecular(Vector(p1\_, p2\_), Vector(p1\_,p4\_))

&& is\_perpendecular(Vector(p3\_, p2\_), Vector(p3\_,p4\_))

&& is\_perpendecular(Vector(p1\_, p2\_), Vector(p2\_,p3\_))

&& is\_perpendecular(Vector(p1\_, p4\_), Vector(p4\_,p3\_))) {

std::swap(p3\_,p4\_);

} else {

throw std::logic\_error("Это не прямоугольник, стороны не перпендикулярны");

}

double s1 = Vector(p1\_, p2\_).length();

double s2 = Vector(p3\_, p4\_).length();

double s3 = Vector(p1\_, p3\_).length();

double s4 = Vector(p2\_, p4\_).length();

if (!(s1 == s2 && s3 == s4)) {

throw std::logic\_error("Это не прямоугольник, соответствующие стороны не равны");

}

}

double Rectangle::Area() const {

return Vector(p1\_,p3\_).length() \* Vector(p1\_,p2\_).length();

}

Point Rectangle::Center() const {

return (p1\_ + p2\_ + p3\_ + p4\_) / 4;

}

void Rectangle::Print(std::ostream& os) const {

os << "Прямоугольник, точки - " << "(" << p1\_ << ") "

<< "(" << p2\_ << ") "

<< "(" << p3\_ << ") "

<< "(" << p4\_ << ") ";

}

void Rectangle::Scan(std::istream &is) {

Point p1,p2,p3,p4;

is >> p1 >> p2 >> p3 >> p4;

\*this = Rectangle(p1,p2,p3,p4);

}

Square.cpp

#include "Square.h"

Square::Square(Point p1, Point p2, Point p3, Point p4)

: p1\_(p1), p2\_(p2), p3\_(p3), p4\_(p4) {

if (is\_perpendecular(Vector(p1\_, p2\_), Vector(p1\_,p3\_))

&& is\_perpendecular(Vector(p4\_, p2\_), Vector(p4\_,p3\_))

&& is\_perpendecular(Vector(p1\_, p3\_), Vector(p3\_,p4\_))

&& is\_perpendecular(Vector(p1\_, p2\_), Vector(p2\_,p4\_))) {

} else if (is\_perpendecular(Vector(p1\_, p4\_), Vector(p1\_,p3\_))

&& is\_perpendecular(Vector(p2\_, p4\_), Vector(p2\_,p3\_))

&& is\_perpendecular(Vector(p1\_, p3\_), Vector(p3\_,p2\_))

&& is\_perpendecular(Vector(p1\_, p4\_), Vector(p2\_,p4\_))){

std::swap(p2\_,p4\_);

} else if (is\_perpendecular(Vector(p1\_, p2\_), Vector(p1\_,p4\_))

&& is\_perpendecular(Vector(p3\_, p2\_), Vector(p3\_,p4\_))

&& is\_perpendecular(Vector(p1\_, p2\_), Vector(p2\_,p3\_))

&& is\_perpendecular(Vector(p1\_, p4\_), Vector(p4\_,p3\_))) {

std::swap(p3\_,p4\_);

} else {

throw std::logic\_error("Это не квадрат, стороны не перпендикулярны");

}

double s1 = Vector(p1\_, p2\_).length();

double s2 = Vector(p3\_, p4\_).length();

double s3 = Vector(p1\_, p3\_).length();

double s4 = Vector(p2\_, p4\_).length();

if (s1 != s2 || s2 != s3 || s3 != s4 || s4 != s1) {

throw std::logic\_error("Это не квадрат, стороны не равны");

}

}

double Square::Area() const {

return std::pow(Vector(p1\_, p2\_).length(), 2);

}

Point Square::Center() const {

return (p1\_ + p2\_ + p3\_ + p4\_) / 4;

}

void Square::Print(std::ostream& os) const {

os << "Квадрат, точки - " << "(" << p1\_ << ") "

<< "(" << p2\_ << ") "

<< "(" << p3\_ << ") "

<< "(" << p4\_ << ") ";

}

void Square::Scan(std::istream &is) {

Point p1,p2,p3,p4;

is >> p1 >> p2 >> p3 >> p4;

\*this = Square(p1,p2,p3,p4);

}

Trapeze.cpp

#include "Trapeze.h"

Trapeze::Trapeze(Point p1, Point p2, Point p3, Point p4)

: p1\_(p1), p2\_(p2), p3\_(p3), p4\_(p4){

Vector v1(p1\_, p2\_), v2(p3\_, p4\_);

if (v1 = Vector(p1\_, p2\_), v2 = Vector(p3\_, p4\_), is\_parallel(v1, v2)) {

if (v1 \* v2 < 0) {

std::swap(p3\_, p4\_);

}

} else if (v1 = Vector(p1\_, p3\_), v2 = Vector(p2\_, p4\_), is\_parallel(v1, v2)) {

if (v1 \* v2 < 0) {

std::swap(p2\_, p4\_);

}

std::swap(p2\_, p3\_);

} else if (v1 = Vector(p1\_, p4\_), v2 = Vector(p2\_, p3\_), is\_parallel(v1, v2)) {

if (v1 \* v2 < 0) {

std::swap(p2\_, p3\_);

}

std::swap(p2\_, p4\_);

std::swap(p3\_, p4\_);

} else {

throw std::logic\_error("At least 2 sides of trapeze must be parallel");

}

}

Point Trapeze::Center() const {

return (p1\_ + p2\_ + p3\_ + p4\_)/4;

}

double Trapeze::Area() const {

double A = p2\_.y - p3\_.y;

double B = p3\_.x - p2\_.x;

double C = p2\_.x\*p3\_.y - p3\_.x\*p2\_.y;

double height = (std::abs(A\*p1\_.x + B\*p1\_.y + C) / sqrt(A\*A + B\*B));

return (Vector(p1\_, p2\_).length() + Vector(p3\_, p4\_).length()) \* height / 2;

}

void Trapeze::Print(std::ostream& os) const {

os << "Трапеция, точки - " << "(" << p1\_ << ") "

<< "(" << p2\_ << ") "

<< "(" << p3\_ << ") "

<< "(" << p4\_ << ") ";

}

void Trapeze::Scan(std::istream &is) {

Point p1,p2,p3,p4;

is >> p1 >> p2 >> p3 >> p4;

\*this = Trapeze(p1,p2,p3,p4);

}

main.cpp

#include <iostream>

#include <string>

#include <vector>

#include "Trapeze.h"

#include "Square.h"

#include "Rectangle.h"

int main() {

std::string command;

std::vector<Figure\*> figures;

while (std::cin >> command) {

if (command == "add") {

std::string fig\_name;

std::cin >> fig\_name;

Figure\* new\_figure = nullptr;

try {

if (fig\_name == "trapeze") {

new\_figure = new Trapeze;

} else if (fig\_name == "square") {

new\_figure = new Square;

} else if (fig\_name == "rectangle") {

new\_figure = new Rectangle;

}

std::cin >> \*new\_figure;

} catch (std::exception& e) {

std::cout << e.what() << "\n";

delete new\_figure;

continue;

}

figures.push\_back(new\_figure);

std::cout << "Успешно добавлено\n";

continue;

} else if (command == "remove") {

size\_t index;

std::cin >> index;

if (index >= figures.size()) {

std::cout << "Слишком большой индекс\n";

continue;

}

figures.erase(figures.begin() + index);

continue;

} else if (command == "area") {

size\_t index;

std::cin >> index;

if (index >= figures.size()) {

std::cout << "Слишком большой индекс\n";

continue;

}

std::cout << \*figures[index] << "\n" << figures[index]->Area() << "\n";

continue;

} else if (command == "center") {

size\_t index;

std::cin >> index;

if (index >= figures.size()) {

std::cout << "Слишком большой индекс\n";

continue;

}

std::cout << \*figures[index] << "\n" << figures[index]->Center() << "\n";

continue;

} else if (command == "print") {

size\_t index;

std::cin >> index;

if (index >= figures.size()) {

std::cout << "Слишком большой индекс\n";

continue;

}

std::cout << \*figures[index] << "\n";

continue;

} else if (command == "printall") {

for (Figure\* ptr : figures) {

std::cout << \*ptr << "\n";

}

continue;

} else if (command == "exit") {

break;

} else {

std::couta << "Неизвестная команда\n";

std::cin.ignore(32768, '\n');

continue;

}

}

for (Figure\* ptr : figures) {

delete ptr;

}

return 0;

}

# 4. Входные данные

Сначала программа ожидает от пользователя введения команды. Возможны следующие варианты:

add <figure> <point1> <point2> <point3> <point4> – добавить фигуру. Point – это координаты (x, y) фигуры

remove <index> - удалить фигуру

area <index> - вывести площадь фигуры

center <index> - вывести координаты точки центра фигуры

print <index> - вывести фигуру (ее название и координаты ее точек)

printall – вывести все добавленные фигуры (их названия и координаты их точек)

exit – выход из программы

Test01.txt

add square 0 2 2 0 0 0 2 2

add trapeze 0 0 0 6 1 3 2 3

add square 1 1 2 2 1 2 2 1

exit

Test02.txt

add square 2 0 0 2 -2 0 0 -2

add square 2 0 0 1 -2 0 0 -2

printall

exit

Test03.txt

add rectangle 0 0 4 0 0 4 4 4

add trapeze -5 -4 -3 -2 -3 -5 -1 -4

print 2

print 1

printall

remove 0

printall

Трапеция, точки - (-5 -4) (-3 -2) (-3 -5) (-1 -4)

exit

# 5. Результаты выполнения тестов

Result\_test01.txt

add square 0 2 2 0 0 0 2 2

Успешно добавлено

add trapeze 0 0 0 6 1 3 2 3

Успешно добавлено

add square 1 1 2 2 1 2 2 1

Успешно добавлено

exit

Result\_test02.txt

add square 2 0 0 2 -2 0 0 -2

Успешно добавлено

add square 2 0 0 1 -2 0 0 -2

Это не квадрат, стороны не перпендикулярны

printall

Квадрат, точки - (2 0) (0 2) (0 -2) (-2 0)

exit

Result\_test03.txt

add rectangle 0 0 4 0 0 4 4 4

Успешно добавлено

add trapeze -5 -4 -3 -2 -3 -5 -1 -4

Успешно добавлено

print 2

Слишком большой индекс

print 1

Трапеция, точки - (-5 -4) (-3 -2) (-3 -5) (-1 -4)

printall

Прямоугольник, точки - (0 0) (4 0) (0 4) (4 4)

Трапеция, точки - (-5 -4) (-3 -2) (-3 -5) (-1 -4)

remove 0

printall

Трапеция, точки - (-5 -4) (-3 -2) (-3 -5) (-1 -4)

exit

# 6. Вывод

С помощью наследования программист может использовать универсальные классы и подстраивать их под себя, добавляя или изменяя функционал подкласса, для этого у программиста есть целый ряд функций и возможностей.