Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

I I I семестр

Задание 3: «Наследование, полиморфизм»

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №3 |
| Студент: | Овечкин Виталий Андреевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 11.11.2019 |

Москва, 2019

1. **Тема**: Наследование, полиморфизм
2. **Цель работы**: Изучение механизмов работы с наследованием в С++
3. **Задание** (*вариант № 19* ):

Разработать классы согласно варианту задания. Классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать общий набор методов:

* Вычисление геометрического центра фигуры
* Вывод в стандартный поток std::cout координат вершин фигуры
* Вычисление площади фигуры

Создать программу, которая позволяет:

* Вводить из стандартного вода std::cin фигуры, согласно варианту задания
* Сохранять созданные фигуры в динамический массив std::vector<Figure\*>
* Вызывыть для всего массива общие функции (1 — 3)
* Необходимо уметь вычислять общую площадь фигур в массиве
* Удалять из массива фигуру по индексу

Фигуры (Вариант 19):

Прямоугольник,трапеция, ромб.

1. **Адрес репозитория на GitHub** <https://github.com/vitalouivi/oop_exercise_3>
2. **Код программы на С++**

main.cpp

#include <iostream>

#include "figure.h"

#include "rhombus.h"

#include "trapezoid.h"

#include "rectangle.h"

#include <vector>

#include <string>

// 2

//enter for trapezoid 2 3 rhombus 1 3 rectangle 2 3

// 1 4 4 1 4

void read\_figt(std::vector<Figure \*>& fig)

{

int figt;

Figure \*f = nullptr;

std::cout << "Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle\n";

std::cin >> figt;

switch (figt) {

case 1:

try {

f = new Trapezoid(std::cin);

}

catch (std::logic\_error& err) {

std::cout << err.what() << std::endl;

break;

}

break;

case 2:

try {

f = new Rhombus(std::cin);

}

catch (std::logic\_error& err) {

std::cout << err.what() << std::endl;

break;

}

break;

case 3:

try {

f = new Rectangle(std::cin);

}

catch (std::logic\_error& err) {

std::cout << err.what() << std::endl;

break;

}

break;

default:

std::cout << "Wrong. Try 1 - trapezoid, 2 - rhombus or 3 - rectangle\n";

}

fig.push\_back(dynamic\_cast<Figure\*>(f));

}

int main() {

unsigned int index;

double Tarea = 0;

std::string operation;

std::vector<Figure\*> fig;

std::cout << "Operations: add / delete / out / quit\n";

while (std::cin >> operation) {

if (operation == "add") {

read\_figt(fig);

}

else if (operation == "delete") {

std::cin >> index;

delete fig[index];

for (; index < fig.size() - 1; ++index) {

fig[index] = fig[index + 1];

}

fig.pop\_back();

}

else if (operation == "out") {

Tarea = 0;

for (unsigned int i = 0; i < fig.size(); i++) {

std::cout << i << ":\n";

std::cout << "Area: " << fig[i]->area() << std::endl;

std::cout << "Center: " << fig[i]->center() << std::endl;

std::cout << "Coordinates: ";

fig[i]->print(std::cout);

std::cout << std::endl;

Tarea += fig[i]->area();

}

std::cout << "Total area: " << Tarea << std::endl;

}

else if (operation == "quit") {

for (unsigned int i = 0; i < fig.size(); ++i) {

delete fig[i];

}

return 0;

}

else {

std::cout << "Wrong. Operations: add / delete / out / quit\n";

}

}

}

point.h

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(double x, double y);

double X() const;

double Y() const;

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

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

private:

double x;

double y;

};

#endif

point.cpp

#include "point.h"

#include <cmath>

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

{}

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

{}

double Point::X() const

{

return x;

}

double Point::Y() const

{

return y;

}

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

{

out << "(" << p.X() << ";" << p.Y() << ")";

return out;

}

std::istream& operator>> (std::istream& in, Point& p)

{

in >> p.x >> p.y;

return in;

}

figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

#include "point.h"

class Figure {

public:

virtual double area() const = 0;

virtual Point center() const = 0;

virtual std::ostream& print(std::ostream& out) const = 0;

virtual ~Figure() = default;

};

#endif

figure.cpp

#include "figure.h"

rhombus.h

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include "figure.h"

#include "point.h"

class Rhombus : public Figure {

public:

Rhombus();

Rhombus(std::istream& in);

double area() const override;

Point center() const override;

std::ostream& print(std::ostream& out) const override;

private:

Point A;

Point B;

Point C;

Point D;

};

#endif

rhombus.cpp

#include "rhombus.h"

#include <cmath>

Rhombus::Rhombus() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0,0 } {}

Rhombus::Rhombus(std::istream& in) {

in >> A >> B >> C >> D;

double a, b, c, d;

a = sqrt((B.X() - A.X()) \* (B.X() - A.X()) + (B.Y() - A.Y()) \* (B.Y() - A.Y()));

b = sqrt((C.X() - B.X()) \* (C.X() - B.X()) + (C.Y() - B.Y()) \* (C.Y() - B.Y()));

c = sqrt((C.X() - D.X()) \* (C.X() - D.X()) + (C.Y() - D.Y()) \* (C.Y() - D.Y()));

d = sqrt((D.X() - A.X()) \* (D.X() - A.X()) + (D.Y() - A.Y()) \* (D.Y() - A.Y()));

if (a != b || a != c || a != d)

throw std::logic\_error("It`s not a rhombus");

}

double Rhombus::area() const {

double d1 = sqrt((C.X() - A.X()) \* (C.X() - A.X()) + (C.Y() - A.Y()) \* (C.Y() - A.Y()));;

double d2 = sqrt((B.X() - D.X()) \* (B.X() - D.X()) + (B.Y() - D.Y()) \* (B.Y() - D.Y()));;

return d1 \* d2 / 2;

}

Point Rhombus::center() const

{

return Point{ (A.X() + B.X() + C.X() + D.X()) / 4, (A.Y() + B.Y() + C.Y() + D.Y()) / 4 };

}

std::ostream& Rhombus::print(std::ostream& out) const

{

out << A << " " << B << " " << C << " " << D;

return out;

}

rectangle.h

#ifndef RECTNAGLE\_H

#define RECTNAGLE\_H

#include "figure.h"

#include "point.h"

class Rectangle : public Figure {

public:

Rectangle();

Rectangle(std::istream& in);

double area() const override;

Point center() const override;

std::ostream& print(std::ostream& out) const override;

private:

Point A;

Point B;

Point C;

Point D;

};

#endif

rectangle.cpp

#include "rectangle.h"

#include <cmath>

Rectangle::Rectangle() : A{ 0, 0 }, B{ 0, 0 }, C{ 0,0 }, D{ 0,0 } {}

Rectangle::Rectangle(std::istream& in) {

in >> A >> B >> C >> D;

double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;

a = sqrt((B.X() - A.X()) \* (B.X() - A.X()) + (B.Y() - A.Y()) \* (B.Y() - A.Y()));

b = sqrt((C.X() - B.X()) \* (C.X() - B.X()) + (C.Y() - B.Y()) \* (C.Y() - B.Y()));

c = sqrt((C.X() - D.X()) \* (C.X() - D.X()) + (C.Y() - D.Y()) \* (C.Y() - D.Y()));

d = sqrt((D.X() - A.X()) \* (D.X() - A.X()) + (D.Y() - A.Y()) \* (D.Y() - A.Y()));

d1 = sqrt((B.X() - D.X()) \* (B.X() - D.X()) + (B.Y() - D.Y()) \* (B.Y() - D.Y()));

d2 = sqrt((C.X() - A.X()) \* (C.X() - A.X()) + (C.Y() - A.Y()) \* (C.Y() - A.Y()));

ABC = (a \* a + b \* b - d2 \* d2) / 2 \* a \* b;

BCD = (b \* b + c \* c - d1 \* d1) / 2 \* b \* c;

CDA = (d \* d + c \* c - d2 \* d2) / 2 \* d \* c;

DAB = (a \* a + d \* d - d1 \* d1) / 2 \* a \* d;

if (ABC != BCD || ABC != CDA || ABC != DAB)

throw std::logic\_error("It`s not a rectangle");

}

double Rectangle::area() const {

double a = sqrt((B.X() - A.X()) \* (B.X() - A.X()) + (B.Y() - A.Y()) \* (B.Y() - A.Y()));

double b = sqrt((C.X() - B.X()) \* (C.X() - B.X()) + (C.Y() - B.Y()) \* (C.Y() - B.Y()));;

return a \* b;

}

Point Rectangle::center() const {

return Point{ (A.X() + B.X() + C.X() + D.X()) / 4, (A.Y() + B.Y() + C.Y() + D.Y()) / 4 };

}

std::ostream& Rectangle::print(std::ostream& out) const {

out << A << " " << B << " " << C << " " << D;

return out;

}

trapezoid.h

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include "figure.h"

#include "point.h"

class Trapeze : public Figure {

public:

Trapeze();

Trapeze(std::istream& in);

double area() const override;

Point center() const override;

std::ostream& print(std::ostream& out) const override;

private:

Point A;

Point B;

Point C;

Point D;

};

#endif

trapezoid.cpp

#include "trapezoid.h"

#include <cmath>

Trapezoid::Trapezoid() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0,0 } {}

Trapezoid::Trapezoid(std::istream& in) {

in >> A >> B >> C >> D;

double b = sqrt((C.X() - B.X()) \* (C.X() - B.X()) + (C.Y() - B.Y()) \* (C.Y() - B.Y()));

double d = sqrt((D.X() - A.X()) \* (D.X() - A.X()) + (D.Y() - A.Y()) \* (D.Y() - A.Y()));

if ((C.Y() - B.Y()) / (C.X() - B.X()) != (D.Y() - A.Y()) / (D.X() - A.X()))

throw std::logic\_error("It`s not a trapezoid");

}

double Trapezoid::area() const {

return 0.5 \* std::abs(A.X() \* B.Y() + B.X() \* C.Y() + C.X() \* D.Y() + D.X() \* A.Y() - B.X() \* A.Y() - C.X() \* B.Y() - D.X() \* C.Y() - A.X() \* D.Y());

}

Point Trapezoid::center() const

{

return Point{ (A.X() + B.X() + C.X() + D.X()) / 4, (A.Y() + B.Y() + C.Y() + D.Y()) / 4 };

}

std::ostream& Trapezoid::print(std::ostream& out) const

{

out << A << " " << B << " " << C << " " << D;

return out;

}

CMakeLists.txt

cmake\_minimum\_required (VERSION 3.5)

project(lab3)

add\_executable(oop\_exercise\_03

main.cpp

figure.cpp

rhombus.cpp

point.cpp

rectangle.cpp

trapeze.cpp)

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -Wall -Wextra")

set\_target\_properties(oop\_exercise\_03 PROPERTIES CXX\_STANDART 14 CXX\_STANDART\_REQUIRED ON)

1. **Набор testcases**

test\_01.txt Координаты вершин Ожидаемый результат

Трапеция (0;0) (0;3) (3;3) (4;0) Площадь 10.5

0 0 Центр (1.75; 1.5)

0 3

3 3

4 0

Ромб (0:0)(1;1)(2;0)(-1;-1) не ромб

0 0

1 1

2 0

-1-1

Ромб (0:0)(1;1)(2;0)(-1;-1) Площадь 2

0 0

1 1 Центр (1,0)

2 0

1-1

Прямоугольник (-1;0)(1;0)(1;2)(-1;2) Площадь 4

-1 0 Центр (0;1)

1 0

1 2

-1 2

out

test\_02.txt Координаты вершин Ожидаемый результат

Трапеция (0;-2) (1;0) (4;0) (5;-2) Площадь 8

0 3 Центр (4; 4)

4 5

5 5

7 3

Прямоугольник (-2;-2)(-2;-1)(2;-1)(2;-2) Площадь 2

-1 0 Центр (0;0)

0 1

1 0

0 -1

Ромб (0:-2)(-1;0)(0;2)(1;0) Площадь 4

-2 0 Центр (0;0)

0 3

2 0

0 -3

Прямоугольник (-2;-2)(-2;-1)(2;-1)(2;-2) Площадь 4

-3 2 Центр (-0.5;1.5)

-1 4

2 1

0 -1

out

delete

1

out

test\_03.txt Координаты вершин Ожидаемый результат

Трапеция (0;0) (0;1) (1;1) (1;0) не трапеция

0 0

0 1

1 1

1 0

1. **Результаты выполнения тестов**

Operations: add / delete / out / quit

1

Wrong. Operations: add / delete / out / quit

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

1

0

0

0

3

3

3

4

0

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

3

-1

0

1

0

1

2

-1

2

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

2

0

0

1

1

2

0

-1

-1

It`s not a rhombus

0

Wrong. Operations: add / delete / out / quit

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

2

0

0

1

1

2

0

1

-1

out

0:

Area: 10.5

Center: (1.75;1.5)

Coordinates: (0;0) (0;3) (3;3) (4;0)

1:

Area: 4

Center: (0;1)

Coordinates: (-1;0) (1;0) (1;2) (-1;2)

2:

Area: 2

Center: (1;0)

Coordinates: (0;0) (1;1) (2;0) (1;-1)

Total area: 16.5

Operations: add / delete / out / quit

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

1

0

3

4

5

5

5

7

3

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

3

-1

0

0

1

1

0

0

-1

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

2

-2

0

0

3

2

0

0

-3

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

3

-3

2

-1

4

2

1

0

-1

out

0:

Area: 8

Center: (4;4)

Coordinates: (0;3) (4;5) (5;5) (7;3)

1:

Area: 2

Center: (0;0)

Coordinates: (-1;0) (0;1) (1;0) (0;-1)

2:

Area: 12

Center: (0;0)

Coordinates: (-2;0) (0;3) (2;0) (0;-3)

3:

Area: 12

Center: (-0.5;1.5)

Coordinates: (-3;2) (-1;4) (2;1) (0;-1)

Total area: 34

delete

1

out

0:

Area: 8

Center: (4;4)

Coordinates: (0;3) (4;5) (5;5) (7;3)

1:

Area: 12

Center: (0;0)

Coordinates: (-2;0) (0;3) (2;0) (0;-3)

2:

Area: 12

Center: (-0.5;1.5)

Coordinates: (-3;2) (-1;4) (2;1) (0;-1)

Total area: 32

Operations: add / delete / out / quit

add

Fig types: 1 - trapezoid; 2 - rhombus; 3 - rectangle

1

0

0

0

1

1

1

1

0

It`s not a trapezoid

out

Total area: 0

1. **Объяснение результатов работы программы - вывод**

В figure.h задаётся базовый класс Figure задающий структуру для классов — наследников — Rectangle, Trapezoid и Rhombus.

Наследование позволяет избежать дублирования кода при написании классов, т. к. класс может использовать переменные и методы другого класса как свои собственные.

В данном случае класс Figure является абстрактным — он определяет интерфейс для переопределения методов классами Rectangle, Trapezoid и Rhombus.

Полиморфизм позволяет использовать одно и то же имя для различных действий, похожих, но технически отличающихся. В данной лаборторной работе он осуществляется посредством виртуальных функций.