Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа №3
по курсу «ООП»

Тема:
Наследование, полиморфизм

Студент:	Вахрамян К.О.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	3
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
#include <cassert>
#include <stdexcept>
#include "point.h"
#include <cmath>

class TFigure {
public:
	virtual void Print(std::ostream&) const = 0;
	virtual TPoint Center() const = 0;
	virtual double Square() const = 0;
	virtual ~TFigure(){};
};
#endif
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "figure.h"

#include <cmath>


class TRectangle : public TFigure {
private:
	TPoint a, b, c, d;
public:
	double Square() const override;
	TPoint Center() const override;
	void Print(std::ostream&) const override;
	TRectangle();
	TRectangle(const TPoint p1, const TPoint p2, const TPoint p3, const TPoint
p4);
};

#endif
```

rectangle.cpp

```
#include "rectangle.h"

TRectangle::TRectangle (const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4) {
	a = p1;
	b = p2;
	c = p3;
	d = p4;
	TPoint ab, ad, cb, cd;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	cb.x = b.x - c.x;
	cb.y = b.y - c.y;
	cd.x = d.x - c.x;
	cd.y = d.y - c.y;

	assert(acos((ab.x * ad.x + ab.y * ad.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(ad.x * ad.x + ad.y * ad.y))) / M_PI == 0.5 && acos((cb.x * cd.x + cb.y * cd.y) / (sqrt(cb.x * cb.x + cb.y * cb.y) * sqrt(cd.x * cd.x + cd.y * cd.y))) / M_PI == 0.5);
}

double TRectangle::Square () const {

	double ans = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
	return fabs(ans);
}

TPoint TRectangle::Center() const {
	TPoint p;
	double x = (a.x + b.x + c.x + d.x) / 4;
	double y = (a.y + b.y + c.y + d.y) / 4;
	p.x = x;
	p.y = y;
	return p;
}

void TRectangle::Print(std::ostream& os) const {
	os << "rectangle:\n" <<  a << " " << b << " " << c << " " << d << "\n";
}
```

trapezoid.h

```
#ifndef TRAPEZOID_H
```

```
#define TRAPEZOID_H

#include "figure.h"
#include <cmath>

class TTrapezoid : public TFigure{
private:
	TPoint a, b, c, d;
public:
	double Square() const override;
	TPoint Center() const override;
	void Print(std::ostream&) const override;
	TTrapezoid();
	TTrapezoid(const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4);
};
#endif
```

trapezoid.cpp

```
#include "trapezoid.h"

TTrapezoid::TTrapezoid (const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4) {
	a = p1;
	b = p2;
	c = p3;
	d = p4;
	TPoint ab, ad, bc, dc;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	dc.x = c.x - d.x;
	dc.y = c.y - d.y;
	assert(acos((ab.x * dc.x + ab.y * dc.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(dc.x * dc.x + dc.y * dc.y))) == 0 || acos((ad.x * bc.x + ad.y * bc.y) / (sqrt(ad.x * ad.x + ad.y * ad.y) * sqrt(bc.x * bc.x + bc.y * bc.y))) == 0);

}
TPoint TTrapezoid::Center() const {
	TPoint p;
	double x = (a.x + b.x + c.x + d.x) /4;
```

```
	double y = (a.y + b.y + c.y + d.y) /4;
	p.x = x;
	p.y = y;

	return p;

}
double TTrapezoid::Square() const {
	TPoint p = this->Center();
	double t1 = 0.5 * fabs((b.x - a.x) * (p.y - a.y) - (p.x - a.x) * (b.y - a.y));
	double t2 = 0.5 * fabs((c.x - b.x) * (p.y - b.y) - (p.x - b.x) * (c.y - b.y));
	double t3 = 0.5 * fabs((d.x - c.x) * (p.y - c.y) - (p.x - c.x) * (d.y - c.y));
	double t4 = 0.5 * fabs((a.x - d.x) * (p.y - d.y) - (p.x - d.x) * (a.y - d.y));
	return t1 + t2 + t3 + t4;
}

void TTrapezoid::Print(std::ostream& os) const {
	os <<"trapezoid:\n" << a << " " << b << " " << c << " " << d << "\n";
}
```

rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H

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

class TRhombus : public TFigure{
private:
	TPoint a, b, c, d;
public:
	double Square() const override;
	TPoint Center() const override;
	void Print(std::ostream&) const override;
	TRhombus();
	TRhombus(const TPoint p1, const TPoint p2, const TPoint p3, const TPoint
p4);
};


#endif
```

rhombus.cpp

```
#include "rhombus.h"

TRhombus::TRhombus (const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4) {
	a = p1;
	b = p2;
	c = p3;
	d = p4;
	TPoint ab, bc, cd, da;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	cd.x = d.x - c.x;
	cd.y = d.y - c.y;
	da.x = a.x - d.x;
	da.y = a.y - d.y;
	assert(sqrt(ab.x * ab.x + ab.y * ab.y) ==  sqrt(bc.x * bc.x + bc.y * bc.y) && sqrt(bc.x * bc.x + bc.y * bc.y) == sqrt(cd.x * cd.x + cd.y * cd.y) && sqrt(cd.x * cd.x + cd.y * cd.y) == sqrt(da.x * da.x + da.y * da.y));

}

double TRhombus::Square() const {
	double ans = 0.5 * sqrt(pow(a.x - c.x, 2) + pow(a.y - c.y, 2)) * sqrt(pow(b.x - d.x, 2) + pow(b.y - d.y, 2));
	return fabs(ans);
}

TPoint TRhombus::Center() const {
	TPoint p;
	double x = (a.x + b.x + c.x + d.x) / 4;
	double y = (a.y + b.y + c.y + d.y) / 4;
	p.x = x;
	p.y = y;
	return p;
}

void TRhombus::Print(std::ostream& os) const {
	os << "rombus:\n"<< a << " " << b << " " << c << " " << d << "\n";
}
```

main.cpp

```
#include <vector>
```

```
#include <string>
#include <cstring>
#include "figure.h"
#include "point.h"
#include "rectangle.h"
#include "trapezoid.h"
#include "rhombus.h"

int main()
{
	std::vector<TFigure*> v;
	int i, j;
	TPoint p1, p2, p3, p4;
	double S;
	std::cout << "Enter:\nadd - to add figure\ndelete - to delete figure\nfigure - to print square and center of figure\ntotal - to print total area of all figures\nexit - to complite programme execution\nhelp - to show this manual\n";
	std::string cmd;
	while (true) {
		std::cin >> cmd;
		if (cmd == "add") {
			std::cout << "chose figure:\n1 - rectangle\n2 - trapezoid\n3 - rhombus\n";
			std::cin >> i;
			std::cin >> p1 >> p2 >> p3 >> p4;
			TFigure* f;
			if (i == 1) {
				f = new TRectangle(p1, p2, p3, p4);
				v.push_back(f);
			} else if (i == 2) {
				f = new TTrapezoid(p1, p2, p3, p4);
				v.push_back(f);
			} else if (i == 3) {
				f = new TRhombus(p1, p2, p3, p4);
				v.push_back(f);
			} else {
				std::cout << "wrong index, try again\n";
				continue;
			}
		} else if (cmd == "delete") {
			std::cout << "enter index\n";
			std::cin >> i;
			if (i >= v.size() || i < 0) {
				std::cout << "wrong index, try again\n";
				continue;
```

```
                } else {
                        delete v[i];
                        v.erase(v.begin() + i);
                }
        } else if (cmd == "figure") {
                for (auto tmp : v) {
                        std::cout << "Center: " << tmp->Center() << "\n";
                        std::cout << "Area: " << tmp->Square() << "\n";

                }
        } else if (cmd == "total") {
                double S = 0;
                for (auto tmp : v) {
                        S += tmp->Square();

                }
                std::cout << "total area: " << S << "\n";
        } else if (cmd == "help") {
                std::cout << "Enter:\nadd - to add figure\ndelete - to delete figure\
nfigure - to print square and center of figure\ntotal - to print total area of all figure\
nexit - to complite programme execution\nhelp - to show this manual\n";
        } else if (cmd == "exit") {
                for (auto& c : v) {
                        delete c;
                }
                break;
        } else {
                std::cout << "wrong comand, try again\n";
                continue;
        }

    }



}
```

2. Ссылка на репозиторий на GitHub.

https://github.com/vebcreatex7/oop_exercise_03

3. Пример работы программы.

```
courage@courage-X550LC:~/oop/oop_exercise_03$ ./oop_exercise_03
Enter:
add - to add figure
delete - to delete figure
figure - to print square and center of figure
total - to print total area of all figure
exit - to complite programme execution
help - to show this manual
add
chose figure:
1 - rectangle
2 - trapezoid
3 - rhombus
2
1 1 2 2 3 2 2 1
figure
Center: 2 1.5
Area: 1
add
chose figure:
1 - rectangle
2 - trapezoid
3 - rhombus
1
1 1 1 2 2 2 2 1
figure
Center: 2 1.5
Area: 1
Center: 1.5 1.5
Area: 1
total
total area: 2
delete
enter index
1
figure
Center: 2 1.5
Area: 1
delete 2
enter index
wrong index, try again
exit
```

4. Объяснение результатов работы программы.

Фигуры вводятся покоординатно со стандартного потока ввода затем для каждой отдельной фигуры находится центр и площадь при помои методов класса наследника. Если координаты введены не верно, то происходит остановка программы. Так же осуществлена процедура подсчета общей площади всех введенных фигур.

5. Вывод.

Выполняя данную лабораторную, я получил опыт работы с механизмами наследования классов в C++ и полиморфизмом (общие методы для различных фигур: center(), square(), print(), по-разному определенные в самих классах фигур, что позволяет работать с ними в едином интерфейсе), кроме того было изучено такое понятие, как полностью виртуальная функция (метод), т.е. метод, который в классе родителе не определен, но определяется в классах наследниках.