Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Асинхронное программирование.

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
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

struct TPoint
{
	double x, y;
};

std::istream& operator >> (std::istream& is,TPoint& p );
std::ostream& operator << (std::ostream& os,const TPoint& p);

#endif
```

point.cpp

```
#include "point.h"

std::istream& operator >> (std::istream& is, TPoint& p) {
	return is >> p.x >> p.y;
}

std::ostream& operator << (std::ostream& os, const TPoint& p) {
	return os << p.x << " " << p.y;
}
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "figure.h"

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


#endif // RECTANGLE_H
```

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;
	if (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) {
		throw std::logic_error("it's not rectangle\n");
	}
	//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);
}
```

```
TRectangle::TRectangle(std::istream& is) {
	is >> a >> b >> c >> d;
	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;
	if (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) {
		throw std::logic_error("it's not rectangle\n");
	}
}

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 <<  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);
	TRhombus(std::istream& is);
};


#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;
	if (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)) {
		throw std::logic_error("it's not rhombus\n");
	}
	//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));

}

TRhombus::TRhombus(std::istream& is) {
	is >> a >> b >> c >> d;
	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;
	if (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)) {
```

```
                throw std::logic_error("it's not rhombus\n");
        }
}

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 << 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:
        std::string name = "trapezoid ";
        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);
        TTrapezoid(std::istream& is);
};
#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;
	if (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) {
		throw std::logic_error("it's not trapezoid\n");
	}

	//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);

}

TTrapezoid::TTrapezoid(std::istream& is) {
	is >> a >> b >> c >> d;
	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;
	if (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) {
		throw std::logic_error("it's not trapezoid\n");
	}

}

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 << a << " " << b << " " << c << " " << d << "\n";
}
```

factory.h

```
#pragma once

#include <iostream>
#include <string>
#include <memory>
#include "figure.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"

class Factory {
public:
	std::shared_ptr<TFigure> FigureCreate(std::istream& is) {
		std::string type;
		std::cin >> type;
		if (type == "rectangle") {
			 std::shared_ptr<TFigure> f(new TRectangle(is));
			return f;
		} else if (type == "rhombus") {
			std::shared_ptr<TFigure> f(new TRhombus(is));
			return f;
		} else if (type == "trapezoid") {
			std::shared_ptr<TFigure> f(new TTrapezoid(is));
			return f;
		} else {
			throw std::logic_error("Wrong figure\n");
		}
	}

};
```

subscriber.h

```
#pragma once
#include "subscriber.h"
#include <string>
#include <fstream>
#include <time.h>
#include <string>



class ConsolePrint : public sub {
public:
	void Print(std::vector<std::shared_ptr<TFigure>>& v) override {
		for (unsigned int i = 0; i < v.size(); i++) {
			v[i]->Print(std::cout);
		}
	}
};

int rndm() {
		srand (time(NULL));
		int name = rand() % 600000 + 1;
		return name;
}

class FilePrint : public sub {
public:
	void Print(std::vector<std::shared_ptr<TFigure>>& v) override {
		std::string filename = "";
		filename = "file_" + std::to_string(rndm()) + ".txt";
		std::ofstream file(filename);
		for (unsigned int i = 0; i < v.size(); i++) {
			v[i]->Print(file);
		}
	}

};
```

subscriber.cpp

```
#pragma once


#include "figure.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"
#include <vector>
#include <memory>
```

```
class  sub {
public:
	virtual void Print(std::vector<std::shared_ptr<TFigure>>& v) = 0;
	virtual ~sub() = default;
};
```

subscribers.cpp

```
#pragma once
#include "subscriber.h"
#include <string>
#include <fstream>
#include <time.h>
#include <string>


class ConsolePrint : public sub {
public:
	void Print(std::vector<std::shared_ptr<TFigure>>& v) override {
		for (unsigned int i = 0; i < v.size(); i++) {
			v[i]->Print(std::cout);
		}
	}
};

int rndm() {
		srand (time(NULL));
		int name = rand() % 600000 + 1;
		return name;
}

class FilePrint : public sub {
public:
	void Print(std::vector<std::shared_ptr<TFigure>>& v) override {
		std::string filename = "";
		filename = "file_" + std::to_string(rndm()) + ".txt";
		std::ofstream file(filename);
		for (unsigned int i = 0; i < v.size(); i++) {
			v[i]->Print(file);
		}
	}

};
```

2. Ссылка на репозиторий на GitHub.

https://github.com/vebcreatex7/oop_exercise_08

3. Набор тестов.

```
courage@courage-X550LC:~/oop/oop_exercise_08/build$ ./oop_exercise_08 4
enter - to enter 4 figures
exit - to finish execution of program
enter
rectangle 0 0 0 3 6 3 6 0
trapezoid 0 0 3 3 6 3 9 0
rhombus 0 0 0 1 1 1 1 0
rectangle 0 0 0 3 6 3 6 0
Buffer is full!
0 0 0 3 6 3 6 0
0 0 3 3 6 3 9 0
0 0 0 1 1 1 1 0
0 0 0 3 6 3 6 0
exit
courage@courage-X550LC:~/oop/oop_exercise_08/build$ cat file_440624.txt
0 0 0 3 6 3 6 0
0 0 3 3 6 3 9 0
0 0 0 1 1 1 1 0
0 0 0 3 6 3 6 0
```

4. Объяснение результатов работы программы.

Для создания фигур реализован отдельный класс Factory, в классе ConsolePrint написан метод печати фигур в консоли, в классе FilePrint — метод записи в файл с рандомным именем. Оба класса наследуют метод Print из класса родителя Sub. В main работают 2 потока, один считывает и сохраняет в буфер фигуры, а другой печатает их в файл и на консоль.

5. Вывод.

Выполняя данную лабораторную работу, я обрел базовые навыки многопоточного программирования, научился использовать мьютексы и условные переменные. Понял, как нужно работать с потоками и какие могут при этом возникнуть трудности.