

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:	Болдырев А.К.
Группа:	М8О-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	2
Оценка:	
Дата:	

Москва  
2019

## 1. Код программы на языке C++:

### figure.hpp

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

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

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

### rectangle.hpp

```
#ifndef RECTANGLE_HPP
#define RECTANGLE_HPP

#include "point.hpp"
#include "figure.hpp"
#include <cmath>

class TRectangle : public TFigure {

private:
    TPoint a, b, c, d;

public:
    double Area() const override;
    TPoint Center() const override;
    void Print(std::ostream&) const override;
    TRectangle();
    TRectangle(const TPoint p1, const TPoint p2, TPoint p3, const TPoint p4);
```

```
};
```

```
#endif
```

### **rectangle.cpp**

```
#include "rectangle.hpp"
```

```
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((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)) == 0 && (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)) == 0);
```

```
}
```

```
double TRectangle::Area () 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.hpp**

```

#ifndef RECTANGLE_HPP
#define RECTANGLE_HPP

```

```

#include "point.hpp"
#include "figure.hpp"
#include <cmath>

```

```

class TRectangle : public TFigure {

private:
    TPoint a, b, c, d;

public:
    double Area() const override;
    TPoint Center() const override;
    void Print(std::ostream&) const override;
    TRectangle();
    TRectangle(const TPoint p1, const TPoint p2, TPoint p3, const TPoint p4);
};

#endif

```

## **trapezoid.cpp**

```

#include "trapezoid.hpp"

```

```

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::Area() 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";
}

```

## square.hpp

```

#ifndef SQUARE_HPP
#define SQUARE_HPP

#include "point.hpp"
#include "figure.hpp"
#include <iostream>

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

#endif

```

## square.cpp

```
#include "square.hpp"
```

```
TSquare::TSquare (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( ( ab.x * da.x + ab.y * da.y ) / ( sqrt( ab.x * ab.x + ab.y * ab.y ) * sqrt( da.x
* da.x + da.y * da.y )) == 0 && ( bc.x * cd.x + bc.y * cd.y ) / ( sqrt( bc.x * bc.x +
bc.y * bc.y ) * sqrt( cd.x * cd.x + cd.y * cd.y )) == 0 && ( ab.x * bc.x + ab.y *
bc.y ) / ( sqrt( ab.x * ab.x + ab.y * ab.y ) * sqrt( bc.x * bc.x + bc.y * bc.y )) == 0 &&
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 TSquare::Area() const {
    double ans = sqrt( (b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y) ) * sqrt( (b.x -
a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y) );
    return fabs(ans);
}

TPoint TSquare::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 TSquare::Print(std::ostream& os) const {
```

```

    os << "square:\n" << a << " " << b << " " << c << " " << d << "\n";
}

```

## **main.cpp**

```

#include <vector>
#include <string>
#include <cstring>
#include "figure.hpp"
#include "point.hpp"
#include "rectangle.hpp"
#include "trapezoid.hpp"
#include "square.hpp"

int main()
{
    std::vector<TFigure*> v;
    int i, j;
    TPoint p1, p2, p3, p4;
    double A;
    std::cout << "Go:\na - to add figure\n" << "d - to delete figure\n" << "f - to print area and center of figure\n" << "t - to print total area of all figures\n" << "e - to complete programme execution\n" << "h - to show this manual\n";
    std::string cmd;
    while ( true ) {
        std::cin >> cmd;
        if ( cmd == "a" ) {
            std::cout << "chose figure:\n1 - rectangle\n2 - trapezoid\n3 - square\n";
            std::cin >> i;
            TFigure* f;
            if ( i == 1 ) {
                std::cin >> p1 >> p2 >> p3 >> p4;
                f = new TRectangle( p1, p2, p3, p4 );
            } else if ( i == 2 ) {
                std::cin >> p1 >> p2 >> p3 >> p4;
                f = new TTrapezoid( p1, p2, p3, p4 );
            } else if ( i == 3 ) {
                std::cin >> p1 >> p2 >> p3 >> p4;
                f = new TSquare( p1, p2, p3, p4 );
            } else {
                std::cout << "ERROR\ntry again\n";
                continue;
            }
            v.push_back(f);
        } else if ( cmd == "d" ) {
            std::cout << "enter index\n";

```

```

std::cin >> i;
if ( i >= v.size() or i < 0 ) {
    std::cout << "ERROR\ntry again\n";
    continue;
} else {
    delete v[i];
    v.erase( v.begin() + i);
}
} else if ( cmd == "f" ) {
    for ( auto tmp : v ) {
        std::cout << "Center: " << tmp->Center() << "\n";
        std::cout << "Area: " << tmp->Area() << "\n";
    }
} else if ( cmd == "t" ) {
    double A = 0;
    for ( auto tmp : v ) {
        A += tmp->Area();
    }
    std::cout << "total area: " << A << "\n";
} else if ( cmd == "h" ) {
    std::cout << "Go:\na - to add figure\nd - to delete figure\nf - to print area and
center of figure\nt - to print total area of all figures\ne - to complite programme
execution\nh - to show this manual\n";
} else if ( cmd == "e" ) {
    for ( auto& c : v ) {
        delete c;
    }
    break;
} else {
    std::cout << "ERROR\ntry again\n";
    continue;
}
}
}

```

## 2. Ссылка на репозиторий на GitHub.

[https://github.com/Anton-Boldyrev/oop\\_exercise\\_03/](https://github.com/Anton-Boldyrev/oop_exercise_03/)



### 3. Пример работы программы.

test\_1:

Go:

a - to add figure

d - to delete figure

f - to print area and center of figure

t - to print total area of all figures

e - to complete programme execution

h - to show this manual

a

chose figure:

1 - rectangle

2 - trapezoid

3 - square

1

0 0 0 4 2 4 2 0

a

chose figure:

1 - rectangle

2 - trapezoid

3 - square

2

0 0

0 5

2 4

2 1

a

chose figure:

1 - rectangle

2 - trapezoid

3 - square

3

0 0

0 1

1 1

1 0

h

Go:

a - to add figure

d - to delete figure

f - to print area and center of figure

t - to print total area of all figures

e - to complete programme execution

h - to show this manual

f

Center: 1 2

Area: 8  
Center: 1 2.5  
Area: 8  
Center: 0.5 0.5  
Area: 1  
t  
total area: 17  
d  
enter index  
1  
t  
total area: 9  
f  
Center: 1 2  
Area: 8  
Center: 0.5 0.5  
Area: 1  
e

## **test\_2:**

Go:  
a - to add figure  
d - to delete figure  
f - to print area and center of figure  
t - to print total area of all figures  
e - to complete programme execution  
h - to show this manual  
h  
Go:  
a - to add figure  
d - to delete figure  
f - to print area and center of figure  
t - to print total area of all figures  
e - to complete programme execution  
h - to show this manual  
h  
Go:  
a - to add figure  
d - to delete figure  
f - to print area and center of figure  
t - to print total area of all figures  
e - to complete programme execution  
h - to show this manual  
f  
t  
total area: 0

a  
chose figure:  
1 - rectangle  
2 - trapezoid  
3 - square  
2  
0 0  
0 1  
1 0.8  
1 0.2

a  
chose figure:  
1 - rectangle  
2 - trapezoid  
3 - square  
3  
0 0  
0 2  
2 2  
2 0

a  
chose figure:  
1 - rectangle  
2 - trapezoid  
3 - square  
3  
1 1  
1 2  
2 2  
2 1

a  
chose figure:  
1 - rectangle  
2 - trapezoid  
3 - square  
1  
0.1 0.1  
0.1 0.4  
0.3 0.4  
0.3 0.1

a  
chose figure:  
1 - rectangle  
2 - trapezoid  
3 - square

3

5 5

5 6

6 6

6 5

h

Go:

a - to add figure

d - to delete figure

f - to print area and center of figure

t - to print total area of all figures

e - to complete programme execution

h - to show this manual

f

Center: 0.5 0.5

Area: 0.8

Center: 1 1

Area: 4

Center: 1.5 1.5

Area: 1

Center: 0.2 0.25

Area: 0.06

Center: 5.5 5.5

Area: 1

t

total area: 6.86

d

enter index

3

f

Center: 0.5 0.5

Area: 0.8

Center: 1 1

Area: 4

Center: 1.5 1.5

Area: 1

Center: 5.5 5.5

Area: 1

t

total area: 6.8

a

chose figure:

1 - rectangle

2 - trapezoid

3 - square

```
3
0 0
1 0
1 1
0 1
f
Center: 0.5 0.5
Area: 0.8
Center: 1 1
Area: 4
Center: 1.5 1.5
Area: 1
Center: 5.5 5.5
Area: 1
Center: 0.5 0.5
Area: 1
t
total area: 7.8
e
```

#### 4. Объяснение результатов работы программы.

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

В первом тесте мы сначала добавляем новую фигуру, а конкретно — прямоугольник. Далее добавляем трапецию и еще квадрат. Далее вызываем мануал, просим вывести центры и площади всех фигур и общую площадь. Пробуем удалить фигуру с индексом 1, потом выводим общую площадь оставшихся и площади и центры по отдельности. Нажимаем e(exit), программа успешно завершается.

Во втором тесте мы пробуем, не добавив никаких фигур, вывести общую площадь, неудивительно выводится — 0. Добавляем трапецию, три квадрата и прямоугольник. Выводим все площади и центры, затем общую площадь, пробуем удалить 3 элемент и выводим площади и центры оставшихся и их общую площадь. Еще добавляем квадрат и опять выводим площади и центры всех фигур и их общую площадь. Пишем e(exit), программа успешно завершается.

## 5. Вывод.

Выполняя данную лабораторную, я получил опыт работы с механизмами наследования классов в C++ и полиморфизмом (общие методы для различных фигур: `Center()`, `Area()`, `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.