Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
«Наследование, полиморфизм»

Студент:	Ли А. И7
Группа:	М80-208Б-18
Преподаватель:	Журавлев А.А.
Вариант:	13
Оценка:	
Дата:	

Москва
2019

1. Постановка задачи

- Ознакомиться с теоретическим материалом.
- Разработать классы согласно варианту, все классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:
  - Вычисление геометрического центра фигура.
  - Вывод в стандартный поток вывода std::cout координат вершин фигуры.
  - Вычисление площади фигуры.

4) Программа должна позволять:
   - Выводить из стандартного ввода std::cin фигуры, согласно варианту задания.
   - Сохранять созданные фигуры в динамический массив std::vector<Figure*>
   - Вызывать общие функции для всего массива.
   - Вычислять общую площадь фигур в массиве.
   - Удалять из массива фигуру по индексу.

Вариант 13: Ромб, 5-угольник, 6-угольник.

2. Код программы на языке C++

main.cpp

```
#include <iostream>
#include "Rhombus.h"
#include "Pentagon.h"
#include "Hexagon.h"

void help1(){
   std::cout <<"What you want?"<<std::endl;
   std::cout <<"If you want to create Rhombus, press 1."<<std::endl;
```

```
    std::cout <<"If you want to create Pentagon, press 2."<<std::endl;
    std::cout <<"If you want to create Hexagon, press 3."<<std::endl;
    std::cout <<"If you want to push now figure in vector, press 4."<<std::endl;
    std::cout <<"If you want to delete figure in vector, press 5."<<std::endl;
    std::cout <<"If you want to choose figure in vector, press 6."<<std::endl;
    std::cout <<"If you want to exit, press 7." << std::endl;
    std::cout <<"If you want to look help, press 8." << std::endl;
    std::cout <<"If you want to check all square, press 9." << std::endl;
}

int main() {
    std::vector <Figure *> data;



    std::pair <double, double> a(0, 2);
    std::pair <double, double> b(4, 0);
    std::pair <double, double> c(2, 4);
    std::pair <double, double> d(-2, 6);
    std::vector <std::pair <double, double>> vertex = {a, b, c, d};
    Figure * element_rhomb = new Rhombus(vertex, "rhombus");
    data.push_back(element_rhomb);



    std::pair <double, double> a1(13, -92);
    std::pair <double, double> b1(44, 0);
    std::pair <double, double> c1(-800, 30);
    std::pair <double, double> d1(27, 2);
    std::pair <double, double> e1(1, 2);
    std::vector <std::pair <double, double>> vertex1 = {a1, b1, c1, d1, e1};
    Figure * element_pent = new Pentagon(vertex1, "pentagon");
```

```
data.push_back(element_pent);


std::pair <double, double> a2(-2, 0);
std::pair <double, double> b2(-1, 1);
std::pair <double, double> c2(1, 1);
std::pair <double, double> d2(2, 0);
std::pair <double, double> e2(1, -1);
std::pair <double, double> f2(-1, -1);
std::vector <std::pair <double, double>> vertex2 = {a2, b2, c2, d2, e2, f2};
Figure * element_hex = new Hexagon(vertex2, "hexagon");
data.push_back(element_hex);


for (int i = 0; i < data.size(); ++i) {
    std::cout << data[i]->who_i_am() << std::endl;
    std::cout << data[i]->square() << std::endl;
}

int choose;
help1();
std::cin >> choose;
Figure * element = nullptr;
while(choose != 7){
    if(choose == 8) {
        help1();
        continue;
    }
    else if(choose == 6){
        std::cout << "Enter index in vector" << std::endl;
```

```
            int index;
            std::cin >> index;
            std::cout << "If you want to check square, press 1."<< std::endl;
            std::cout << "If you want to check vertex, press 2."<< std::endl;
            std::cout << "If you want to check type, press 3."<< std::endl;
            std::cout << "If you want to check centr, press 4."<< std::endl;
            int ch;
            std::cin >> ch;
            if(ch == 1)
               data[index]->square();
            else if(ch == 2){
               std::vector <std::pair <double, double>> vertex_fig =
data[index]->get_vertex();
               for(int i = 0; i < vertex_fig.size();++i)
                  std::cout << vertex_fig[i].first << " " << vertex_fig[i].second;
            }
            else if (ch == 3)
               std::cout << data[index]->who_i_am() << std::endl;
            else if(ch == 4) {
               std::pair<double, double> c = data[index]->center();
               std::cout << c.first << " " << c.second;
            }

         }
         else if(choose == 5){
            std::cout << "Enter index in vector" << std::endl;
            int index;
            std::cin >> index;
            delete(data[index]);
            data.erase(data.begin() + index);
```

```
        }
        else if(choose == 4){
            data.push_back(element);
            element = nullptr;
        }
        else if(choose == 3){
            if(element != nullptr)
                delete(element);
            std::pair <double, double> p1;
            std::pair <double, double> p2;
            std::pair <double, double> p3;
            std::pair <double, double> p4;
            std::pair <double, double> p5;
            std::pair <double, double> p6;
            std::cout << "Enter coordinates" << std::endl;
            std::cin >> p1.first >> p1.second >> p2.first >> p2.second >> p3.first >> p3.second >> p4.first >> p4.second >> p5.first >> p5.second >> p6.first >> p6.second;
            std::vector <std::pair <double, double>> vertex_ = {p1, p2, p3, p4, p5, p6};
            element = new Hexagon(vertex_, "hexagon");
        }
        else if(choose == 2){
            if(element != nullptr)
                delete(element);
            std::pair <double, double> p1;
            std::pair <double, double> p2;
            std::pair <double, double> p3;
            std::pair <double, double> p4;
            std::pair <double, double> p5;
```

```
            std::cout << "Enter coordinates" << std::endl;
            std::cin >> p1.first >> p1.second >> p2.first >> p2.second >> p3.first >>
p3.second >> p4.first >> p4.second >> p5.first >> p5.second;
            std::vector <std::pair <double, double>> vertex_ = {p1, p2, p3, p4, p5};
            element = new Pentagon(vertex_, "pentagon");
        }
        else if(choose == 1){
            if(element != nullptr)
                delete(element);
            std::pair <double, double> p1;
            std::pair <double, double> p2;
            std::pair <double, double> p3;
            std::pair <double, double> p4;
            std::cout << "Enter coordinates" << std::endl;
            std::cin >> p1.first >> p1.second >> p2.first >> p2.second >> p3.first >>
p3.second >> p4.first >> p4.second;
            std::vector <std::pair <double, double>> vertex_ = {p1, p2, p3, p4};
            element = new Rhombus(vertex_, "rhombus");
        }
        else if(choose == 9) {
            double sum = 0;
            for(auto i : data)
                sum += i->square();
            std::cout << sum << std::endl;
        }
        std::cin >> choose;
    }
    return 0;
}
```

Figure.h

```
#include <utility>
#include <vector>
#include <cmath>
#include <string>

#ifndef OOP_FIGURE_H
#define OOP_FIGURE_H

class Figure{
public:
   explicit Figure(std::vector <std::pair <double, double>> n_vertex, std::string
n_i_am){
       vertex = std::move(n_vertex);
       i_am = std::move(n_i_am);
   }
   virtual std::pair<double, double> center() = 0;
   virtual std::vector <std::pair <double, double>> get_vertex() = 0;
   virtual double square() = 0;
   virtual std::string who_i_am() = 0;

protected:
   std::vector <std::pair <double, double>> vertex;
   std::string i_am;
};

#endif
```

Hexagon.h

```
#ifndef OOP_HEXAGON_H
```

```
#define OOP_HEXAGON_H
#include "Figure.h"

class Hexagon : public Figure{
public:
    explicit Hexagon(std::vector<std::pair<double, double>> nVertex, std::string
me) : Figure(std::move(nVertex), std::move(me)) {}
    std::pair<double, double> center() override;
    std::vector <std::pair <double, double>> get_vertex() override;
    double square() override;
    std::string who_i_am() override;
};
#endif
```

Hexagon.cpp

```
#include "Hexagon.h"

std::string Hexagon::who_i_am() {
    return this->i_am;
}


std::pair<double, double> Hexagon::center(){
    std::pair<double, double> answer(0, 0);
    for(auto i : vertex){
        answer.first += i.first;
        answer.second += i.second;
    }
    answer.first /= vertex.size();
    answer.second /= vertex.size();
```

```
    return answer;
}

std::vector <std::pair <double, double>> Hexagon::get_vertex(){
    return this->vertex;
}

double Hexagon::square(){
    double res = 0;
    std::pair <double, double> point2;
    std::pair <double, double> point1 = vertex[0];
    for(int i = 1; i < vertex.size();++i){
        point2 = vertex[i];
        res += (point1.first + point2.first) * (point2.second - point1.second);
        point1 = point2;
    }
    res += (vertex[0].first + vertex[vertex.size() - 1].first) * (vertex[0].second -
vertex[vertex.size() - 1].second);
    return std::abs(res) / 2;
}
```

Pentagon.h

```
#ifndef OOP_PENTAGON_H
#define OOP_PENTAGON_H
#include "Figure.h"

class Pentagon : public Figure{
public:
    explicit Pentagon(std::vector<std::pair<double, double>> nVertex, std::string
me) : Figure(std::move(nVertex), std::move(me)) {}
```

```
    std::pair<double, double> center() override;
    std::vector <std::pair <double, double>> get_vertex() override;
    double square() override;
    std::string who_i_am() override;
};
#endif
```

Pentagon.cpp

```
#include "Pentagon.h"



std::string Pentagon::who_i_am() {
    return this->i_am;
}

std::pair<double, double> Pentagon::center(){
    std::pair<double, double> answer(0, 0);
    for(auto i : vertex){
        answer.first += i.first;
        answer.second += i.second;
    }
    answer.first /= vertex.size();
    answer.second /= vertex.size();
    return answer;
}

std::vector <std::pair <double, double>> Pentagon::get_vertex(){
    return this->vertex;
}
```

```
double Pentagon::square(){
    double res = 0;
    std::pair <double, double> point2;
    std::pair <double, double> point1 = vertex[0];
    for(int i = 1; i < vertex.size();++i){
        point2 = vertex[i];
        res += (point1.first + point2.first) * (point2.second - point1.second);
        point1 = point2;
    }
    res += (vertex[0].first + vertex[vertex.size() - 1].first) * (vertex[0].second - vertex[vertex.size() - 1].second);
    return std::abs(res) / 2;
}
```

Rhombus.h

```
#ifndef OOP_RHOMBUS_H
#define OOP_RHOMBUS_H
#include <utility>

#include "Figure.h"

class Rhombus : public Figure{
public:
    explicit Rhombus(std::vector<std::pair<double, double>> nVertex, std::string me) : Figure(std::move(nVertex), std::move(me)) {}
    std::pair<double, double> center() override;
    std::vector <std::pair <double, double>> get_vertex() override;
    double square() override;
    std::string who_i_am() override;
};
```

```
#endif
```

Rhombus.cpp

```
#include "Rhombus.h"


std::string Rhombus::who_i_am() {
    return this->i_am;
}

std::pair <double, double> calculate_vector(const std::pair <double, double> &
a, const std::pair <double, double> & b){
    std::pair <double, double> answer;
    answer.first = b.first - a.first;
    answer.second = b.second - a.second;
    return answer;
}

double length_vector(std::pair <double, double> a){
    return sqrt(a.first * a.first + a.second * a.second);
}

std::pair<double, double> Rhombus::center(){
    std::pair<double, double> answer(0, 0);
    for(auto i : vertex){
        answer.first += i.first;
        answer.second += i.second;
    }
    answer.first /= vertex.size();
    answer.second /= vertex.size();
```

```
    return answer;
}

std::vector <std::pair <double, double>> Rhombus::get_vertex(){
    return this->vertex;
}

double Rhombus::square(){
    return 0.5 * length_vector(calculate_vector(this->vertex[0], this->vertex[2]))
* length_vector(calculate_vector(this->vertex[1], this->vertex[3]));
}
```

2. Ссылка на репозиторий на GitHub.

https://github.com/elips0n/oop_exercise_03

4. Набор testcases.

```
rhombus
12
pentagon
1446
hexagon
6
What you want?
If you want to create Rhombus, press 1.
If you want to create Pentagon, press 2.
If you want to create Hexagon, press 3.
If you want to push now figure in vector, press 4.
If you want to delete figure in vector, press 5.
If you want to choose figure in vector, press 6.
```

If you want to exit, press 7.

If you want to look help, press 8.

If you want to check all square, press 9.

9

1464

6. Объяснение программы

В самом начале создается базовый класс фигур Figure. Далее объявляется функции в этом классе виртуальными (чтобы в классе наследнике их можно было предопределить), данные (координаты вершин), которые лежат в этом классе, объявляются защищенными. Потом создается три класса для поставленных в задаче трех фигур с необходимым функционалом. Для вычисления фигур в 5 и 6-угольниках используется специальная формула, а для ромба стандартная. В main реализовано общение с пользователем.

7. Вывод

В данной лабораторной работе мною были получены навыки работы с наследованием классов и виртуальными функциями. Было лучше освоена работа с математическими задачами, а также повторены знания и применение на практике создание классов.