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:	Симонов С.Я.
Группа:	М8О-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	21
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

figure.h

```
#ifndef D_FIGURE_H_
#define D_FIGURE_H_

#include <iostream>

#include "point.h"

struct figure {
        //РґРѕР±РІРёС‚СЊ РїСЂРѕРІРµСЂРєСѓ PSP° РєРѕСЂРµРєС‚РЅРѕСЃС‚СЊ
        virtual point center() const = 0;
        virtual void print(std::ostream& os) const = 0;
        virtual double area() const = 0; // PIC‹C‡РёСЃР»РµРЅРЅРѕРµ РїСЂРѕС‰Р°РґРё

        virtual ~figure() {}
};

#endif
```

point.h

```
#ifndef D_POINT_H_

#define D_POINT_H_

#include <iostream>

struct point {
        double x, y;
};

std::istream& operator>> (std::istream& is, point& p);
std::ostream& operator<< (std::ostream& os, const point& p);
point operator+ (point p1, point p2);
point& operator/ (point& p, int num);

#endif
```

point.cpp

```
#include "point.h"

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

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

```
point operator+ (point p1, point p2) {
        point p;
        p.x = p1.x + p2.x;
        p.y = p1.y + p2.y;
        return p;
}

point& operator/ (point& p, int num) {
        p.x = p.x / num;
        p.y = p.y / num;
        return p;
}
```

rhombus.h

```
#ifndef D_RHOMBUS_H_
#define D_RHOMBUS_H_ 1

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

struct rhombus : figure
{
        rhombus(std::istream& is);
        rhombus(const point& a1, const point& a2, const point& a3, const point&
a4);

        point center() const override;
        void print(std::ostream& os) const override;
        double area() const override;
private:
        point a1, a2, a3, a4;
};

#endif
```

rhombus.cpp

```
#include <iostream>
#include <cmath>

#include "rhombus.h"

rhombus::rhombus(std::istream & is) {
        is >> a1 >> a2 >> a3 >> a4;
        double str1, str2, str3, str4;
        str1 = sqrt((a1.x - a2.x) * (a1.x - a2.x) + (a1.y - a2.y) * (a1.y -
a2.y));
        str2 = sqrt((a2.x - a3.x) * (a2.x - a3.x) + (a2.y - a3.y) * (a2.y -
a3.y));
        str3 = sqrt((a3.x - a4.x) * (a3.x - a4.x) + (a3.y - a4.y) * (a3.y -
a4.y));
        str4 = sqrt((a4.x - a1.x) * (a4.x - a1.x) + (a4.y - a1.y) * (a4.y -
a1.y));
        if (str1 != str2 || str2 != str3 || str3 != str4) {
```

```
                throw std::logic_error("Is not rhombus");
        }
}

rhombus::rhombus(const point& a1, const point& a2, const point& a3, const point&
a4) {
        double str1, str2, str3, str4;
        str1 = sqrt((a1.x - a2.x) * (a1.x - a2.x) + (a1.y - a2.y) * (a1.y -
a2.y));
        str2 = sqrt((a2.x - a3.x) * (a2.x - a3.x) + (a2.y - a3.y) * (a2.y -
a3.y));
        str3 = sqrt((a3.x - a4.x) * (a3.x - a4.x) + (a3.y - a4.y) * (a3.y -
a4.y));
        str4 = sqrt((a4.x - a1.x) * (a4.x - a1.x) + (a4.y - a1.y) * (a4.y -
a1.y));
        if (str1 != str2 || str2 != str3 || str3 != str4) {
                throw std::logic_error("Is not rhombus");
        }
}

point rhombus::center() const {
        point result;
        result = a1 + a2 + a3 + a4;
        result = result / 4;
        return result;
}

void rhombus::print(std::ostream & os) const {
        os << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " <<
a4 << "\n";
}

double rhombus::area() const {
        point v_1;
        v_1.x = a1.x - a3.x;
        v_1.y = a1.y - a3.y;
        point v_2;
        v_2.x = a2.x - a4.x;
        v_2.y = a2.y - a4.y;
        double result = 0.5 * (sqrt(v_1.x * v_1.x + v_1.y * v_1.y) * sqrt(v_2.x
* v_2.x + v_2.y * v_2.y));
        return result;
}
```

pentagon.h

```
#ifndef D_PENTAGON_H_
#define D_PENTAGON_H_

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

struct pentagon : public figure
{
        pentagon(std::istream& is);

        point center() const override;
        void print(std::ostream& os) const override;
        double area() const override;
```

```
private:
        point a1, a2, a3, a4, a5;
};

#endif
```

pentagon.cpp

```
#include <iostream>
#include <cmath>

#include "pentagon.h"

pentagon::pentagon(std::istream& is) {
        is >> a1 >> a2 >> a3 >> a4 >> a5;
}

point pentagon::center() const {
        point result;
        result = a1 + a2 + a3 + a4 + a5;
        result = result / 5;
        return result;
}

void pentagon::print(std::ostream & os) const {
        os << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " <<
a4 << " a5 = " << a5 << "\n";
}

double pentagon::area() const {
        double s1 = (a2.x * a3.y - a1.x * a3.y - a2.x * a1.y - a3.x * a2.y +
a1.x * a2.y + a1.y * a3.x) / 2;
        double s2 = (a3.x * a4.y - a1.x * a4.y - a3.x * a1.y - a4.x * a3.y +
a1.x * a3.y + a1.y * a4.x) / 2;
        double s3 = (a4.x * a5.y - a1.x * a5.y - a4.x * a1.y - a5.x * a4.y +
a1.x * a4.y + a1.y * a5.x) / 2;
        double result = s1 + s2 + s3;
        if (result >= 0) {
                return result;
        } else {
                return -result;
        }
}
```

hexagon.h

```
#ifndef D_HEXAGON_H_
#define D_HEXAGON_H_

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

struct hexagon : figure
{
```

```
        hexagon(std::istream& is);

        point center() const override;
        void print(std::ostream& os) const override;
        double area() const override;
private:
        point a1, a2, a3, a4, a5, a6;
};

#endif
```

hexagon.cpp

```
#include "hexagon.h"

#include <iostream>
#include <cmath>

hexagon::hexagon(std::istream & is) {
        is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}

point hexagon::center() const {
        point result;
        result = (a1 + a2 + a3 + a4 + a5 + a6);
        result = result / 6;
        return result;
}

void hexagon::print(std::ostream & os) const {
        os << "a1 = " << a1 << " a2 = " << a2 << " a3 = " << a3 << " a4 = " <<
a4 << " a5 = " << a5 << " a6 = " << a6 << "\n";
}

double hexagon::area() const {
        double s1 = (a2.x * a3.y - a1.x * a3.y - a2.x * a1.y - a3.x * a2.y +
a1.x * a2.y + a1.y * a3.x) / 2;
        double s2 = (a3.x * a4.y - a1.x * a4.y - a3.x * a1.y - a4.x * a3.y +
a1.x * a3.y + a1.y * a4.x) / 2;
        double s3 = (a4.x * a5.y - a1.x * a5.y - a4.x * a1.y - a5.x * a4.y +
a1.x * a4.y + a1.y * a5.x) / 2;
        double s4 = (a5.x * a6.y - a1.x * a6.y - a5.x * a1.y - a6.x * a5.y +
a1.x * a5.y + a1.y * a6.x) / 2;
        double result = s1 + s2 + s3 + s4;
        if (result >= 0) {
                return result;
        } else {
                return -result;
        }
}
```

main.cpp:

```
#include <iostream>
#include <vector>
```

```
#include "figure.h"
#include "point.h"

#include "pentagon.h"
#include "hexagon.h"
#include "rhombus.h"

int main() {
    std::vector<figure*> figures;
    for (;;) {
        int command;
        std::cin >> command;
        if (command == 0) {
            break;
        } else if (command == 1) {
            int figure_type;
            std::cin >> figure_type;
            figure* ptr;
            if (figure_type == 0) {
                ptr = new rhombus(std::cin);
            } else if (figure_type == 1) {
                ptr = new pentagon(std::cin);
            }  else {
                    ptr = new hexagon(std::cin);
            }
            figures.push_back(ptr);
        } else if (command == 2) {
            int id;
            std::cin >> id;
            delete figures[id];
            figures.erase(figures.begin() + id);
            std::cout << std::endl;
        } else if (command == 3) {
            std::cout << "Centers:\n";
            for (figure* ptr: figures) {
                //std::cout << ptr->center() << std::endl;
                point p = ptr->center();
                std::cout << p << "\n" << std::endl;
            }
        } else if (command == 4) {
            std::cout << "Areas:\n";
            for (figure* ptr: figures) {
                std::cout << ptr->area() << std::endl << std::endl;
            }
        } else if (command == 5) {
            std::cout << "Figures:\n";
            for (figure* ptr: figures) {
                ptr->print(std::cout);
                std::cout << std::endl;
            }
        }
    }
    //for (figure* ptr: figures) {
      //  delete ptr;
    //}
    for (size_t i = 0; i < figures.size(); ++i) {
        delete figures[i];
    }
}
```

CmakeLists.txt:

```
cmake_minimum_required(VERSION 3.5)

project(lab1)

add_executable(lab1
  main.cpp
  point.cpp
  pentagon.cpp
  hexagon.cpp
  rhombus.cpp

)

set_property(TARGET lab1 PROPERTY CXX_STANDART 11)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")
```

2. Ссылка на репозиторий на GitHub.

https://github.com/keoni02032/oop_exercise_03

3. Набор testcases.

test_01.test:

```
1
0
-1 0
0 1
1 0
0 -1
5
4
3
1
1
-1 0
0 1
1 1
1 0
0 -1
5
4
3
```

1
2
-1 0
0 1
1 1
2 0
1 -1
0 -1
5
4
3
2

test_02.test:

1
1
-1 0
-0.5 1
1 0.5
0.5 -0.5
-0.5 -1
5
4
3
1
0
0 0
1 2
2 0
1 -2
5
4
3
1
2
-1 0
0 0.5
1 1
2 0
1 -1
0 -1
5
4

3
0

test_03.test:

1
2
0 0.5
1 1
1 0
1 -1
0 -0.5
0 0
5
4
3
1
1
0 0.5
1.5 0
0.5 -0.5
-1 -1
-0.5 0
5
4
3
1
0
-1 1
1.5 1
1 -1
-1.5 -1
5
4
3
0

4. Результаты выполнения тестов.

test_01.result:

Figures:
a1 = -1 0 a2 = 0 1 a3 = 1 0 a4 = 0 -1

Areas:
2

Centers:
0 0

Figures:
a1 = -1 0 a2 = 0 1 a3 = 1 0 a4 = 0 -1

a1 = -1 0 a2 = 0 1 a3 = 1 1 a4 = 1 0 a5 = 0 -1

Areas:
2

2.5

Centers:
0 0

0.2 0.2

Figures:
a1 = -1 0 a2 = 0 1 a3 = 1 0 a4 = 0 -1

a1 = -1 0 a2 = 0 1 a3 = 1 1 a4 = 1 0 a5 = 0 -1

a1 = -1 0 a2 = 0 1 a3 = 1 1 a4 = 2 0 a5 = 1 -1 a6 = 0 -1

Areas:
2

2.5

4

Centers:
0 0

0.2 0.2

0.5 0

Figures:
a1 = -1 0 a2 = 0 1 a3 = 1 0 a4 = 0 -1

a1 = -1 0 a2 = 0 1 a3 = 1 1 a4 = 1 0 a5 = 0 -1

Areas:
2

2.5

Centers:
0 0

0.2 0.2

Figures:
a1 = -1 0 a2 = 0 1 a3 = 1 0 a4 = 0 -1

Areas:
2

Centers:
0 0

Figures:
Areas:
Centers:

test_02.result:

Figures:
a1 = -1 0 a2 = -0.5 1 a3 = 1 0.5 a4 = 0.5 -0.5 a5 = -0.5 -1

Areas:
2.375

Centers:
-0.1 0

Figures:
a1 = -1 0 a2 = -0.5 1 a3 = 1 0.5 a4 = 0.5 -0.5 a5 = -0.5 -1

a1 = 0 0 a2 = 1 2 a3 = 2 0 a4 = 1 -2

Areas:
2.375

4

Centers:
-0.1 0

1 0

Figures:
a1 = -1 0 a2 = -0.5 1 a3 = 1 0.5 a4 = 0.5 -0.5 a5 = -0.5 -1

a1 = 0 0 a2 = 1 2 a3 = 2 0 a4 = 1 -2

a1 = -1 0 a2 = 0 0.5 a3 = 1 1 a4 = 2 0 a5 = 1 -1 a6 = 0 -1

Areas:
2.375

4

3.5

Centers:
-0.1 0

1 0

0.5 -0.0833333

test_03.result:

Figures:
a1 = 0 0.5 a2 = 1 1 a3 = 1 0 a4 = 1 -1 a5 = 0 -0.5 a6 = 0 0

Areas:
1.5

Centers:
0.5 0

Figures:
a1 = 0 0.5 a2 = 1 1 a3 = 1 0 a4 = 1 -1 a5 = 0 -0.5 a6 = 0 0

a1 = 0 0.5 a2 = 1.5 0 a3 = 0.5 -0.5 a4 = -1 -1 a5 = -0.5 0

Areas:
1.5

1.625

Centers:
0.5 0

0.1 -0.2

5. Объяснение результатов работы программы.

1) При запуске программы пользователю предоставляется на выбор шесть действий: 0 — завершение работы программы, 1 — задать фигуру (если вводи 0 — задается ромб, 1 — задаются вершины пятиугольника, 2 — задаются вершины шестиугольника), 2 — удалить фигуру по индексу из вектора, 3 — вывод геометрических центров, 4 — вывод площадей фигур, 5 — вывод вершин фигур в заданном порядке.
2) Далее происходит ввод команд в бесконечном цикле (все заданные фигуры заносятся в массив)
3) После введения команды «завершение работы программы» из созданного вектора удаляются указатели на фигуры.

6. Вывод.

При выполнении данной лабораторной работы, я получил опыт работы с наследованием классов и полиоморфизмом в C++. В ходе работы я создал базовый класс и три общих метода фигур, поразному определенные в классах фигур. Также было изучено понятие, как виртуальная функция.