Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 3

Тема: Наследование, полиморфизм

Студент: Кудинов Сергей

Преподаватель: Журавлев А.А.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Разработать классы фигур, представляющих пятиугольник, ромб и трапецию, классы которых должны наследоваться от базового класса Figure. Все классы должны поддерживать набор общих методов:

- Вычисление геометрического центра фигуры.
- Вывод точек фигуры в поток с помощью оператора <<
- Ввод точек фигуры из потока с помощью оператора >>
- Вычисление площади фигур

Также необходимо хранить созданные фигуры в векторе указателей на объекты базового класса.

2. **Репозиторий github**

https://github.com/ilya89099/oop_exercise_03/

3. **Описание программы**

Реализован базовый абстрактный класс Figure, имеющий чисто виртуальные функции для вычисления площади, центра и ввода/вывода из потоков. В конструкторах классов-наследников Trapeze, Rhombus и Rectangle реализованы проверки на корректность переданных точек, стоит заметить, что точки в конструкторы передаются в любом порядке. Площадь трапеции подсчитвается стандартной формулой (высота * полусумма длин оснований)/2. Площадь ромба подсчитывается как полупроизведение длин диагоналей. Площадь прямоугольника рассчитывается стандартной формулой произведения смежных ребер. Центры всех фигур вычисляются как сумма координат всех составляющих точек по координатам x и y, поделенная на количество точек.

Для удобства пользования создано меню с несколькими командами:

- 1 – создает новую фигуру типа 1 – трапеция, 2 – ромб, 3 – прямоугольник, по переданным точкам. Фигура добавляется в конец вектора фигур.
- area INDEX – выводит фигуру, находящуюся в векторе по данному индексу(при нумерации с 1), а также ее площадь
- center INDEX – выводит фигуру, находящуюся в векторе по данному индексу(при нумерации с 1), а также ее центр
- print INDEX – выводит все точки фигуры находящейся в векторе по данному индексу(при нумерации с 1).
- delete INDEX – удаляет из вектора фигур фигуру с заданным индексам(нумерация с 1).
- count – выводит количество фигур в векторе.

4. **Набор testcases**

Тестовые файлы: **test_01.test**, **test_02.test, test_03.test, test_04.test**

test_01.test:

1

1

5 0 10 0 7 5 10 5

3

1

Проверка на создание трапеций

Результат работы программы

Created figure

Trapeze 5 0 p1 7 5 p2 10 0 p3 10 5 p4

Trapeze 5 0 p1 7 5 p2 10 0 p3 10 5 p4

Area: 20

test_02.test:

1

3

0 5 10 5 0 0 10 0

3

1

4

1

count

Проверка на создание прямоугольников

Результат работы программы

Created figure

Rectangle p1: 0 5p1 10 5p2 0 0p3 10 0p4

Rectangle p1: 0 5p1 10 5p2 0 0p3 10 0p4

Area: 50

Rectangle p1: 0 5p1 10 5p2 0 0p3 10 0p4

Center: 5 2.5

Elements count: 1

test_03.test:

1

2

3 4 8 4 5 0 0 0

3

1

4

1

5

1

count

Проверка на создание ромбов

Результат работы программы

Created figure

Rhombus 3 4p1 8 4p2 5 0p3 0 0p4

Rhombus 3 4p1 8 4p2 5 0p3 0 0p4

Area: 20

Rhombus 3 4p1 8 4p2 5 0p3 0 0p4

Center: 4 2

Elements count: 0

5. Результаты выполнения тестов

Все тесты успешно пройдены, программа выдаёт верные результаты.

6. Листинг программы

main.cpp

```
#pragma once

#include "Figure.h"

#include <iostream>
#include <vector>
#include <algorithm>
#include <exception>
#include "Rectangle.h"
#include "Rhombus.h"
#include "Trapeze.h"
int main() {
  std::vector<Figure*> figures;
    std::string command;
    while (std::cin >> command) {
        if (command == "1") {
            std::string fig_type;
            std::cin >> fig_type;
            Figure* new_fig;
            if (fig_type == "1") {
```

```
                new_fig = new Trapeze;
            } else if (fig_type == "2") {
                new_fig = new Rhombus;
            } else if (fig_type == "3") {
                new_fig = new Rectangle;
            } else {
                std::cout << "Invalid figure type\n";
                std::cin.ignore(30000, '\n');
                continue;
            }
            try {
                std::cin >> (*new_fig);
            } catch (std::exception& e) {
                std::cout << e.what() << "\n";
                delete new_fig;
                continue;
            }
            figures.push_back(new_fig);
            std::cout << "Created figure\n";
            std::cout << *new_fig << "\n";
        } else if (command == "2") {
            int index;
            std::cin >> index;
            index--;
            if (index < 0 || index >= figures.size()) {
                std::cout << "No object at that index\n";
                continue;
            }
            std::cout << "Figure at index " << index + 1 << " - " <<
*figures[index] << "\n";
        } else if (command == "3") {
            int index;
            std::cin >> index;
            index--;
            if (index < 0 || index >= figures.size()) {
                std::cout << "No object at that index\n";
                continue;
            }
            std::cout << *figures[index] << "\n";
            std::cout << "Area: " << figures[index]->getSquare() << "\n";
        } else if (command == "4") {
            int index;
            std::cin >> index;
            index--;
            if (index < 0 || index >= figures.size()) {
                std::cout << "No object at that index\n";
                continue;
            }
            std::cout << *figures[index] << "\n";
            std::cout << "Center: " << figures[index]->getCenter() << "\n";
        } else if (command == "5") {
            int index;
            std::cin >> index;
```

```
            index--;
            if (index < 0 || index >= figures.size()) {
                std::cout << "No object at that index\n";
                continue;
            }
            delete figures[index];
            figures.erase(figures.begin() + index);
        } else if (command == "count") {
            std::cout << "Elements count: " << figures.size() << "\n";
        }
    }
    for (Figure* ptr : figures) {
        delete ptr;
    }
    return 0;

    return 0;
}
```

Trapeze.h

```
#pragma once

#include <vector>
#include "Figure.h"
class Trapeze: public Figure {
    protected:
        std::vector<Point> points;
    public:
        Trapeze() = default;
        Trapeze(Point p1, Point p2, Point p3, Point p4);

         double getSquare();
        Point getCenter();
         void print(std::ostream& os) const;
         void scan(std::istream &is);
};
```

Trapeze.cpp

```
#pragma once

#include "Trapeze.h"


    Trapeze::Trapeze(Point p1, Point p2, Point p3, Point p4) {
       if (IsParallel(p1,p2,p3,p4)
        && !IsParallel(p1,p3,p2,p4)) {
        std::swap(p2, p3);

    } else if (!IsParallel(p1,p2,p3,p4)
        && IsParallel(p1,p3,p2,p4)) {


    } else {
        throw std::logic_error("not Trapeze");
```

```
    }
    points.push_back(p1);
        points.push_back(p2);
    points.push_back(p3);
    points.push_back(p4);

    }
    double Trapeze::getSquare() {
        double res = 0;
          for (unsigned i=0; i<this->points.size(); i++) {
            Point p1 = i ? this->points[i-1] : this->points.back(),
                  p2 = this->points[i];
                res += (p1.x - p2.x) * (p1.y + p2.y);
          }
        return (length(this->points[0],this->points[2])+length(this-
>points[1],this->points[3]))*fabs((this->points[0].y-this-
>points[1].y))*(0.5);
    }
    void Trapeze::print(std::ostream& os) const {
    os << "Trapeze ";
    for (int i = 0; i < this->points.size(); ++i) {
        os << this->points[i] << " p" << i+1 <<" ";
    }
        os << std::endl;

    }

void Trapeze::scan(std::istream &is) {
    Point p1,p2,p3,p4;
    is >> p1 >> p2 >> p3 >> p4;
    *this = Trapeze(p1,p2,p3,p4);
}
Point Trapeze::getCenter()  {

    Point p;
    p.x = 0;
    p.y = 0;
    for (int i = 0; i < points.size(); ++i) {
        p = p+(points[i]/points.size());
    }
    return p;
}
```

Rhombus.h

```
#pragma once

#include <vector>
#include "Figure.h"
class Rhombus: public Figure {
    protected:
        std::vector<Point> points;
    public:
        Rhombus() = default;
```

```
        Rhombus(Point p1, Point p2, Point p3, Point p4);

         double getSquare();
        Point getCenter();
         void print(std::ostream& os) const;
         void scan(std::istream &is);
};
```

Rhombus.cpp

```
#pragma once

#include "Rhombus.h"

    Rhombus::Rhombus(Point p1, Point p2, Point p3, Point p4) {
       if (length(p1, p2) == length(p1, p4)
        && length(p3, p4) == length(p2, p3)
        && length(p1, p2) == length(p2, p3)) {

    } else if (length(p1, p4) == length(p1, p3)
           && length(p2, p3) == length(p2, p4)
           && length(p1, p4) == length(p2, p4)) {
        std::swap(p2, p3);
    } else if (length(p1, p3) == length(p1, p2)
              && length(p2, p4) == length(p3, p4)
              && length(p1, p2)== length(p2, p4)) {
        std::swap(p3, p4);
    } else {
        throw std::logic_error("not rhombus");
    }
        points.push_back(p1);
        points.push_back(p2);
    points.push_back(p3);
    points.push_back(p4);
    }
    double Rhombus::getSquare() {
      return length(this->points[1],this->points[3])*length(this-
>points[0],this->points[2])*0.5;
    }
        void Rhombus::print(std::ostream& os) const {
    os << "Rhombus ";
    for (int i = 0; i < this->points.size(); ++i) {
        os << this->points[i] << "p" << i+1 <<" ";
    }
        os << std::endl;

    }

void Rhombus::scan(std::istream &is) {
    Point p1,p2,p3,p4;
    is >> p1 >> p2 >> p3 >> p4;
    *this = Rhombus(p1,p2,p3,p4);
}
Point Rhombus::getCenter()  {
```

```
    Point p;
    p.x = 0;
    p.y = 0;
    for (int i = 0; i < points.size(); ++i) {
        p = p+(points[i]/points.size());
    }
    return p;
}
```

Rectangle.h

```
#pragma once

#include <vector>
#include "Figure.h"
class Rectangle: public Figure {
    protected:
        std::vector<Point> points;
    public:
     Rectangle() = default;
        Rectangle(Point p1, Point p2, Point p3, Point p4);
         double getSquare();
        Point getCenter();
         void print(std::ostream& os) const;
         void scan(std::istream &is);
};
```

Rectangle.cpp

```
#pragma once

#include "Rectangle.h"

    Rectangle::Rectangle(Point p1, Point p2, Point p3, Point p4) {
       if (length(p1, p2) == length(p3, p4)
        && length(p3, p1) == length(p2, p4)
        && IsOrthogonal(p2,p1,p3)) {

    } else if (length(p1, p4) == length(p2, p3)
           && length(p2, p4) == length(p1, p3)
           && IsOrthogonal(p1,p4,p2)) {
        std::swap(p2, p4);
    } else {
        throw std::logic_error("not rectangle");
    }
        points.push_back(p1);
        points.push_back(p2);
    points.push_back(p3);
    points.push_back(p4);
    }
    double Rectangle::getSquare() {
        double res = 0;
          for (unsigned i=0; i<this->points.size(); i++) {
            Point p1 = i ? this->points[i-1] : this->points.back(),
                  p2 = this->points[i];
                res += (p1.x - p2.x * (p1.y + p2.y));
          }
      return           length(this->points[0],this->points[1])*length(this-
```

```
>points[0],this->points[2]);
    }
            void Rectangle::print(std::ostream& os) const {
    os << "Rectangle p1: ";
    for (int i = 0; i < this->points.size(); ++i) {
        os << this->points[i] << "p" << i+1 <<" ";
    }
    os << std::endl;
    }

void Rectangle::scan(std::istream &is) {
    Point p1,p2,p3,p4;
    is >> p1 >> p2 >> p3 >> p4;
    *this = Rectangle(p1,p2,p3,p4);
}
Point Rectangle::getCenter()  {

    Point p;
    p.x = 0;
    p.y = 0;
    for (int i = 0; i < points.size(); ++i) {
        p = p+(points[i]/points.size());
    }
    return p;
}
```

Figure.h

```
#pragma once

#include <vector>
#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
struct Point {
   double x = 0;
   double y = 0;
};
std::istream& operator >> (std::istream& is, Point& p);
std::ostream& operator<< (std::ostream& os, const Point& p);
Point operator+(Point left, Point right);
Point operator+(Point left, int right);
Point operator-(Point left, Point right);
Point operator-(Point left, double right);
Point operator/(Point left, double right);

Point operator*(Point left, double right);
bool IsOrthogonal(Point a, Point b, Point c);
bool IsParallel(Point a, Point b, Point c, Point d);
double length(Point left, Point right);
class Figure {
   protected:
```

```
    std::vector<Point> points;
  public:
    virtual double getSquare() = 0;
    virtual Point getCenter() = 0;
      virtual ~Figure() = default;

    virtual void print(std::ostream& os) const = 0;
    virtual void scan(std::istream &is) = 0;
};
std::ostream& operator << (std::ostream& os, const Figure& fig);
std::istream& operator >> (std::istream& is, Figure& fig);
```

Figure.cpp

```
#pragma once

#include "Figure.h"

std::ostream& operator << (std::ostream& os, const Figure& fig) {
  fig.print(os);
  return os;
}

std::istream& operator >> (std::istream& is, Figure& fig) {
  fig.scan(is);
  return is;
}
```

Point.cpp

```
#pragma once

#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
#include "Figure.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 left, Point right) {
    Point p;
    p.x = left.x+right.x;
    p.y = left.y+right.y;
        return p;

}
Point operator+(Point left, int right) {
```

```
    Point p;
    p.x = left.x+right;
    p.y = left.y+right;
        return p;

}
Point operator-(Point left, Point right) {
    Point p;
    p.x = left.x-right.x;
    p.y = left.y-right.y;
        return p;

}
Point operator-(Point left, double right) {
    Point p;
    p.x = left.x-right;
    p.y = left.y-right;
        return p;

}
Point operator/(Point left, double right) {
    Point p;
    p.x = left.x/right;
    p.y = left.y/right;
        return p;

}
Point operator*(Point left, double right) {
    Point p;
    p.x = left.x*right;
    p.y = left.y*right;
    return p;
}
double length(Point left, Point right) {
    return sqrt((left.x-right.x)*(left.x-right.x)+(left.y-right.y)*(left.y-
right.y));
}
bool IsOrthogonal(Point a, Point b, Point c)
{
    return (b.x - a.x) * (b.x - c.x) + (b.y - a.y) * (b.y - c.y) == 0;
}
bool IsParallel(Point a, Point b, Point c, Point d)
{
    Point a1 = a-b;
    Point a2 = c-d;
    return ((a1.x*a2.x+a1.y*a2.y)/(length(a,b)*length(c,d))<=-1 ||
(a1.x*a2.x+a1.y*a2.y)/(length(a,b)*length(c,d))>=1);
}
```

7. Вывод

В результате данной работы я научился работать с Cmake, создавать базовые абстрактные классы и их классы наследники, а так же узнал

больше о принципах объектно ориентированного программирования