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: Подоляка Елена

Группа: М8О-208Б-18

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/markisonka/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 и Rhombus реализованы проверки на корректность переданных точек, стоит заметить, что точки в конструкторы передаются в любом порядке. Для класса Pentagon проверок в конструкторе не предусмотрено. Площадь трапеции подсчитвается стандартной формулой (высота * полусумма длин оснований)/2. Площадь ромба подсчитывается как полупроизведение длин диагоналей. Площадь пятиугольника рассчитывается как сумма площадей составляющих его треугольников. 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:

add FIG_TYPE POINTS – создает новую фигуру данного типа 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:

add Trapeze 0 0 1 3 3 2 4 0

add Square 0 0 1 1 2 2 3 3

add Rhombus 0 0 3 4 8 4 5 1

count

add Rhombus 0 0 3 4 8 4 5 0

print 1

count

Проверка обработки фигур, не удовлетворяющих условиям.

Результат работы программы

At least 2 sides of trapeze must be parallel

Invalid figure type

This is not rhombus, sides arent equal

Elements count: 0

Created figure

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

Figure at index 1 - Rhombus, p1: 0 0, p2: 3 4, p3: 8 4, p4: 5 0

Elements count: 1

test_02.test:

add Rhombus 0 0 3 4 8 4 5 0

add Rhombus 3 4 0 0 8 4 5 0

add Rhombus 3 4 8 4 5 0 0 0

add Rhombus 8 4 5 0 3 4 0 0

add Rhombus 5 0 3 4 0 0 8 4

count

area 1

area 2

area 3

area 4

area 5

center 1

center 2

center 3

center 4

center 5

Проверка на правильность работы конструктора класса Rhombus, в частности проверяется коррекность фигуры, если точки передаются в случайном порядке.

Результат работы программы

Created figure

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

Created figure

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

Created figure

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

Created figure

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

Created figure

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

Elements count: 5

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

Area: 20

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

Area: 20

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

Area: 20

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

Area: 20

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

Area: 20

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

Center: 4 2

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

Center: 4 2

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

Center: 4 2

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

Center: 4 2

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

Center: 4 2

test_03.test:

add Trapeze 0 0 3 4 6 4 7 0

add Trapeze 3 4 6 4 7 0 0 0

add Trapeze 0 0 7 0 3 4 6 4

add Trapeze 3 4 0 0 6 4 7 0

add Trapeze 7 0 6 4 0 0 3 4

count

area 1

area 2

area 3

area 4

area 5

center 1

center 2

center 3

center 4

center 5

Проверка на правильность работы конструктора класса Rhombus, в частности проверяется коррекность фигуры, если точки передаются в случайном порядке.

Результат работы программы

Created figure

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Created figure

Trapeze p1:3 4, p2:6 4, p3:0 0, p4:7 0

Created figure

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Created figure

Trapeze p1:3 4, p2:6 4, p3:0 0, p4:7 0

Created figure

Trapeze p1:7 0, p2:0 0, p3:6 4, p4:3 4

Elements count: 5

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Area: 20

Trapeze p1:3 4, p2:6 4, p3:0 0, p4:7 0

Area: 20

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Area: 20

Trapeze p1:3 4, p2:6 4, p3:0 0, p4:7 0

Area: 20

Trapeze p1:7 0, p2:0 0, p3:6 4, p4:3 4

Area: 20

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Center: 4 2

Trapeze p1:3 4, p2:6 4, p3:0 0, p4:7 0

Center: 4 2

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Center: 4 2

Trapeze p1:3 4, p2:6 4, p3:0 0, p4:7 0

Center: 4 2

Trapeze p1:7 0, p2:0 0, p3:6 4, p4:3 4

Center: 4 2

test_04.test:

add Pentagon 0 0 2 3 7 2 7 -4 3 -3

add Trapeze 0 0 3 4 6 4 7 0

add Rhombus 3 4 8 4 0 0 5 0

count

delete 2

print 1

print 2

print 3

area 1

center 1

area 3

center 3

Проверка на правильность работы конструктора конструктора Pentagon, а также проверка удаления фигур.

Результат работы программы

Created figure

Pentagon, p1: 0 0, p2: 2 3, p3: 7 2, p4: 7 -4, p5: 3 -3

Created figure

Trapeze p1:0 0, p2:7 0, p3:3 4, p4:6 4

Created figure

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

Elements count: 3

Figure at index 1 - Pentagon, p1: 0 0, p2: 2 3, p3: 7 2, p4: 7 -4, p5: 3 -3

Figure at index 2 - Rhombus, p1: 3 4, p2: 8 4, p3: 5 0, p4: 0 0

No object at that index

Pentagon, p1: 0 0, p2: 2 3, p3: 7 2, p4: 7 -4, p5: 3 -3

Area: 34

Pentagon, p1: 0 0, p2: 2 3, p3: 7 2, p4: 7 -4, p5: 3 -3

Center: 3.8 -0.4

No object at that index

No object at that index

5. Результаты выполнения тестов

Все тесты успешно пройдены, программа выдаёт верные результаты.

6. Листинг программы

main.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <exception>
#include "Figure.h"
#include "Trapeze.h"
#include "Rhombus.h"
#include "Pentagon.h"
int main() {
  std::vector<Figure*> figures;
  std::string command;
  while (std::cin >> command) {
    if (command == "add") {
      std::string fig_type;
      std::cin >> fig_type;
      Figure* new_fig;
      if (fig_type == "Trapeze") {
        new_fig = new Trapeze;
      } else if (fig_type == "Rhombus") {
        new_fig = new Rhombus;
      } else if (fig_type == "Pentagon") {
        new_fig = new Pentagon;
      } 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 == "print") {
        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 == "area") {
        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]->Area() << "\n";
    } else if (command == "center") {
        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]->Center() << "\n";
    } else if (command == "delete") {
        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;}
```

Trapeze.h

```
#pragma once
#include "Figure.h"
#include <exception>
class Trapeze : public Figure {
public:
  Trapeze() = default;
  Trapeze(Point p1, Point p2, Point p3, Point p4);
  Point Center() const override;
  double Area() const override;
  void Print(std::ostream& os) const override;
  void Scan(std::istream& is) override;
private:
  Point p1_, p2_, p3_, p4_;
};
```

Trapeze.cpp

```
#include "Trapeze.h"

Trapeze::Trapeze(Point p1, Point p2, Point p3, Point p4)
:  p1_(p1), p2_(p2), p3_(p3), p4_(p4){
  Vector v1(p1_, p2_), v2(p3_, p4_);
  if (v1 = Vector(p1_, p2_), v2 = Vector(p3_, p4_), is_parallel(v1, v2)) {
    if (v1 * v2 < 0) {
      std::swap(p3_, p4_);
    }
  } else if (v1 = Vector(p1_, p3_), v2 = Vector(p2_, p4_), is_parallel(v1, v2)) {
    if (v1 * v2 < 0) {
      std::swap(p2_, p4_);
    }
    std::swap(p2_, p3_);
  } else if (v1 = Vector(p1_, p4_), v2 = Vector(p2_, p3_), is_parallel(v1, v2)) {
    if (v1 * v2 < 0) {
      std::swap(p2_, p3_);
    }
    std::swap(p2_, p4_);
    std::swap(p3_, p4_);
  } else {
    throw std::logic_error("At least 2 sides of trapeze must be parallel");
  }
}
Point Trapeze::Center() const {
```

```
  return (p1_ + p2_ + p3_ + p4_) / 4;
}
double Trapeze::Area() const {
  double height = point_and_straight_distance(p1_,p3_,p4_);
  return (Vector(p1_, p2_).length() + Vector(p3_, p4_).length()) * height / 2;
}
void Trapeze::Print(std::ostream& os) const {
  os << "Trapeze p1:" << p1_ << ", p2:" << p2_ << ", p3:" << p3_ << ", p4:" <<
p4_;
}
void Trapeze::Scan(std::istream &is) {
  Point p1,p2,p3,p4;
  is >> p1 >> p2 >> p3 >> p4;
  *this = Trapeze(p1,p2,p3,p4);
}
```

Trapeze.h

```
#pragma once
#include "Figure.h"
#include <exception>
class Trapeze : public Figure {
public:
  Trapeze() = default;
  Trapeze(Point p1, Point p2, Point p3, Point p4);
  Point Center() const override;
  double Area() const override;
  void Print(std::ostream& os) const override;
  void Scan(std::istream& is) override;
private:
  Point p1_, p2_, p3_, p4_;
};
```

Rhombus.h

```
#pragma once
#include "Figure.h"
class Rhombus : public Figure {
public:
  Rhombus() = default;
  Rhombus(Point p1_, Point p2_, Point p3_, Point p4_);
  Point Center() const override;
  double Area() const override;
  void Print(std::ostream& os) const override;
  void Scan(std::istream& is) override;
private:
  Point p1_, p2_, p3_, p4_;
};
```

Rhombus.cpp

```
#include "Rhombus.h"
```

```
Rhombus::Rhombus(Point p1, Point p2, Point p3, Point p4)
: p1_(p1), p2_(p2), p3_(p3), p4_(p4) {
  if (Vector(p1_, p2_).length() == Vector(p1_, p4_).length()
    && Vector(p3_, p4_).length() == Vector(p2_, p3_).length()
    && Vector(p1_, p2_).length() == Vector(p2_, p3_).length()) {
  } else if (Vector(p1_, p4_).length() == Vector(p1_, p3_).length()
      && Vector(p2_, p3_).length() == Vector(p2_, p4_).length()
      && Vector(p1_, p4_).length() == Vector(p2_, p4_).length()) {
    std::swap(p2_, p3_);
  } else if (Vector(p1_, p3_).length() == Vector(p1_, p2_).length()
        && Vector(p2_, p4_).length() == Vector(p3_, p4_).length()
        && Vector(p1_, p2_).length() == Vector(p2_, p4_).length()) {
    std::swap(p3_, p4_);
  } else {
    throw std::logic_error("This is not rhombus, sides arent equal");
  }
}
double Rhombus::Area() const {
  return Vector(p1_, p3_).length() * Vector(p2_, p4_).length() / 2;
}
Point Rhombus::Center() const {
  return (p1_ + p3_) / 2;
}
void Rhombus::Print(std::ostream& os) const {
  os << "Rhombus, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ << ", p4:
" << p4_;
}
void Rhombus::Scan(std::istream &is) {
  Point p1,p2,p3,p4;
  is >> p1 >> p2 >> p3 >> p4;
  *this = Rhombus(p1,p2,p3,p4);
}
```

Pentagon.h

```
#pragma once
#include "Figure.h"
class Pentagon : public Figure {
public:
  Pentagon() = default;
  explicit Pentagon(const Point& p1, const Point& p2, const Point& p3, const Point&
p4, const Point& p5);
  Point Center() const override;
  double Area() const override;
  void Print(std::ostream& os) const override;
  void Scan(std::istream& is) override;
private:
  Point p1_, p2_, p3_, p4_, p5_;
};
```

Pentagon.cpp

```
#include "Pentagon.h"
Pentagon::Pentagon(const Point& p1, const Point& p2, const Point& p3, const Point& p4, const Point& p5)
    : p1_(p1), p2_(p2), p3_(p3), p4_(p4), p5_(p5) {}
double Pentagon::Area() const {
  return
  point_and_straight_distance(p1_, p2_, p3_) * Vector(p2_, p3_).length() / 2
  + point_and_straight_distance(p1_, p3_, p4_) * Vector(p3_, p4_).length() / 2
  + point_and_straight_distance(p1_, p4_, p5_) * Vector(p4_, p5_).length() / 2;
}
Point Pentagon::Center() const {
  return (p1_ + p2_ + p3_ + p4_ + p5_) / 5;
}
void Pentagon::Print(std::ostream& os) const {
  os << "Pentagon, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ << ", p4: " << p4_ << ", p5: " << p5_;
}
void Pentagon::Scan(std::istream &is) {
  Point p1, p2, p3, p4, p5;
  is >> p1 >> p2 >> p3 >> p4 >> p5;
  *this = Pentagon(p1,p2,p3,p4,p5);
}
```

Figure.h

```
#pragma once
#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
struct Point {
  double x;
  double y;
};
Point operator + (const Point& lhs, const Point& rhs);
Point operator - (const Point& lhs, const Point& rhs);
Point operator * (const Point& p, double d);
Point operator / (const Point& p, double d);
std::ostream& operator << (std::ostream& os, const Point& p);
std::istream& operator >> (std::istream& is, Point& p);
double point_and_straight_distance(Point p1, Point p2, Point p3);
double straight_lenght(Point p1, Point p2);
class Figure {
public:
  virtual Point Center() const = 0;
```

```
    virtual double Area() const = 0;
    virtual void Print(std::ostream& os) const = 0;
    virtual ~Figure() = default;
};
std::ostream& operator << (std::ostream& str, const Figure& fig);
```

Figure.cpp

```
#include "Figure.h"
Point operator + (const Point& lhs, const Point& rhs) {
    return {lhs.x + rhs.x, lhs.y + rhs.y};
}
Point operator - (const Point& lhs, const Point& rhs) {
    return {lhs.x - rhs.x, lhs.y - rhs.y};
}
Point operator * (const Point& p, double d) {
    return {p.x * d, p.y * d};
}
Point operator / (const Point& p, double d) {
    return {p.x / d, p.y / d};
}
std::ostream& operator << (std::ostream& os, const Point& p) {
    return os << p.x << " " << p.y;
}
std::istream& operator >> (std::istream& is, Point& p) {
    return is >> p.x >> p.y;
}
std::ostream& operator << (std::ostream& os, const Figure& fig) {
    fig.Print(os);
    return os;
}
double point_and_straight_distance(Point p1, Point p2, Point p3) {
    double A = p2.y - p3.y;
    double B = p3.x - p2.x;
    double C = p2.x*p3.y - p3.x*p2.y;
    return (std::abs(A*p1.x + B*p1.y + C) / std::sqrt(A*A + B*B));
}
double straight_lenght(Point p1, Point p2) {
    return std::sqrt(std::pow(p2.x - p1.x, 2) + std::pow(p2.y - p1.y, 2));
}
```

7. Вывод

В результате данной работы я научилась работать с Cmake, создавать базовые абстрактные классы и их классы наследники, а также узнала больше о принципах объектно ориентированного программирования