Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"

Лабораторная работа №7 по курсу “Объектно-ориентированное программирование”

Студент: Хисамутдинов Д.С.

Группа: М8О-208Б *Преподаватель:*

Журавлев А.А.

Вариант: 5

Оценка: Дата:

Москва
2019

1 Исходный код

figure.hpp

```
#pragma once

#include<iostream>

#include "point.hpp"

enum class Figures {Rhombus, Pentagon, Hexagon};

        class Figure {
        public:
        virtual Point Center() const = 0;
        virtual double Square() const = 0;
        virtual void Print(std::ostream& os) const = 0;
        virtual ~Figure() = default;
        };
```

rhombus.hpp

```
#pragma once

#include <array>

#include "figure.hpp" #include "point.hpp"

class Rhombus : public Figure { public:
        Rhombus(std::istream& is);
        Point Center() const override;
        double Square() const override; void Print(std::ostream& os) const override; private:
        std::array<Point, 4> points;
        double smallerDiagonal, biggerDiagonal;
        };
```

rhombus.cpp

```
#include "rhombus.hpp"

                double checkIfRhombus(const Point& p1, const Point& p2, const
                Point& p3, const Point& p4) {
                double d1 = calculateDistance(p1, p2);
                double d2 = calculateDistance(p1, p3);
                double d3 = calculateDistance(p1, p4);
                if(d1 == d2) {         return d3;
                 } else if(d1 == d3) {
                 return d2;
                 } else if(d2 == d3) {
                 return d1;
                 } else {
```

```
15      throw std::invalid_argument("Entered coordinates are not forming Rhombus. Try entering new
        coordinates");
16      }
17      }
18
19                              Rhombus::Rhombus(std::istream& is) {
20                              Point p1, p2, p3, p4;
21                              is >> p1 >> p2 >> p3 >> p4;
22                              try {
23                              double d1 = checkIfRhombus(p1, p2, p3, p4);
24                              double d2 = checkIfRhombus(p2, p1, p3, p4);
25                              double d3 = checkIfRhombus(p3, p1, p2, p4);
26                              double d4 = checkIfRhombus(p4, p1, p2, p3);
27                              if(d1 == d2 || d1 == d4) {
28                              if(d1 < d3) {
29                              smallerDiagonal = d1;
30                              biggerDiagonal = d3;
31
32                              } else {
33                              smallerDiagonal = d3;
34                              biggerDiagonal = d1;
35                              }
36                              } else if(d1 == d3) {
37                              if(d1 < d2) {
38                              smallerDiagonal = d1;
39                              biggerDiagonal = d2;
40                              } else {
41                              smallerDiagonal = d2;
42                              biggerDiagonal = d1;
43                              }
44                              }
45                              } catch(std::exception& e) {
46                              throw std::invalid_argument(e.what());
47                              return;
48                              }
49                              points[0] = p1;
50                              points[1] = p2;
51                              points[2] = p3;
52                              points[3] = p4;
53                              }
54
55              Point Rhombus::Center() const {
56              if(calculateDistance(points[0], points[1]) == smallerDiagonal
        ||
57              calculateDistance(points[0], points[1]) == biggerDiagonal) {
58              return {((points[0].x + points[1].x) / 2.0), ((points[0].y
         + points[1].y) / 2.0)};
59              } else if(calculateDistance(points[0], points[2]) == smallerDiagonal ||
60              calculateDistance(points[0], points[2]) == biggerDiagonal) {
61              return {((points[0].x + points[2].x) / 2.0), ((points[0].y
         + points[2].y) / 2.0)};
62              } else {
```

```
63                 return {((points[0].x + points[3].x) / 2.0), ((points[0].y
          + points[3].y) / 2.0)};
64                 }
65                 }
66
67          double Rhombus::Square() const {
68          return smallerDiagonal * biggerDiagonal / 2.0;
69          }
70
71 void Rhombus::Print(std::ostream& os) const {
72                 os << "Rhombus: ";
73                   for(const auto& p : points) {
74                   os << p << ' ';
75                   }
76                   os << std::endl;
77                   }
```

pentagon.hpp

```
1 #pragma once
2
3  #include <iostream>
4  #include <array>
5
6 #include "figure.hpp" 7 #include "point.hpp"
8
9           class Pentagon : public Figure {
10          public:
11          Pentagon(std::istream& is);
12          Point Center() const override;
13          double Square() const override;
14          void Print(std::ostream& os) const override; 15 private:
16          std::array<Point, 5> points;
17          };
```

pentagon.cpp

```
1 #include <cmath>
2
3 #include "pentagon.hpp"
4
5         Pentagon::Pentagon(std::istream& is) {
6         is >> points[0] >> points[1] >> points[2] >> points[3] >> points[4];
7         }
8
9                                   Point Pentagon::Center() const {
10                                  Point insideFigure{0, 0};
11                                  Point result{0, 0};
12                                  double square = this->Square();
13                                  for(unsigned i = 0; i < points.size(); ++i) {
14                                  insideFigure.x += points[i].x;
15                                  insideFigure.y += points[i].y;
```

```
                        }
                        insideFigure.x /= points.size();
                        insideFigure.y /= points.size();
                        for(unsigned i = 0; i < points.size(); ++i) {
                        double tempSquare = triangleSquare(points[i], points[(i +
    1) % points.size()],
                        insideFigure);
                        result.x += tempSquare * (points[i].x + points[(i + 1) % points.size()].x
                        + insideFigure.x) / 3.0;
                        result.y += tempSquare * (points[i].y + points[(i + 1) % points.size()].y
                        + insideFigure.y) / 3.0;
                        }
                        result.x /= square;
                        result.y /= square;
                        return result;
                        }

            double Pentagon::Square() const {
            double result = 0;
            for(unsigned i = 0; i < points.size(); ++i) {
            Point p1 = i ? points[i - 1] : points[points.size() - 1];
            Point p2 = points[i];
            result += (p1.x - p2.x) * (p1.y + p2.y);
            }
            return fabs(result) / 2.0;
            }

void Pentagon::Print(std::ostream& os) const {
            os << "Pentagon: ";
            for(const auto& p : points) {
            os << p << ' ';
            }
            os << std::endl;
            }
```

hexagon.hpp

```
#pragma once

#include <iostream>
#include <array>

#include "figure.hpp" #include "point.hpp"

        class Hexagon : public Figure {
        public:
        Hexagon(std::istream& is);
        Point Center() const override;
        double Square() const override;
        void Print(std::ostream& os) const override; private:
        std::array<Point, 6> points;
        };
```

hexagon.cpp

```
#include <cmath>

#include "hexagon.hpp"

        Hexagon::Hexagon(std::istream& is) {
        is >> points[0] >> points[1] >> points[2] >> points[3] >> points[4] >> points[5];
        }

                                Point Hexagon::Center() const {
                                Point insideFigure{0, 0};
                                Point result{0, 0};
                                double square = this->Square();
                                for(unsigned i = 0; i < points.size(); ++i) {
                                insideFigure.x += points[i].x;
                                insideFigure.y += points[i].y;
                                }
                                insideFigure.x /= points.size();
                                insideFigure.y /= points.size();
                                for(unsigned i = 0; i < points.size(); ++i) {
                                double tempSquare = triangleSquare(points[i], points[(i +
         1) % points.size()],
                                insideFigure);
                                result.x += tempSquare * (points[i].x + points[(i + 1) % points.size()].x
                                + insideFigure.x) / 3.0;
                                result.y += tempSquare * (points[i].y + points[(i + 1) % points.size()].y
                                + insideFigure.y) / 3.0;
                                }
                                result.x /= square;
                                result.y /= square;
                                return result;
                                }

                double Hexagon::Square() const {
                double result = 0;
                for(unsigned i = 0; i < points.size(); ++i) {
                Point p1 = i ? points[i - 1] : points[points.size() - 1];
                Point p2 = points[i];
                result += (p1.x - p2.x) * (p1.y + p2.y);
                }
                return fabs(result) / 2.0;
                }

void Hexagon::Print(std::ostream& os) const {
                os << "Hexagon:";
                  for(const auto& p : points) {
                  os << p << ' ';
                  }
                  os << std::endl;
}
```

point.hpp

```
#pragma once
```

```
#include <iostream>

        struct Point {
        double x, y;
        };

 double calculateDistance(const Point& lhs, const Point& rhs);
 bool operator<(const Point& lhs, const Point& rhs);
 std::istream& operator>>(std::istream& is, Point& p);
 std::ostream& operator<<(std::ostream& os, const Point& p);
 double triangleSquare(const Point& p1, const Point& p2, const
        Point& p3);
```

pubsub.hpp

```
#pragma once

#include        <filesystem>
#include        <fstream>
#include        <memory>
#include        <vector>
#include        <queue>
#include        <map>
#include        <thread>
#include        <mutex>
#include        <condition_variable>

#include        "figure.hpp"

enum class TaskType {print, exit};

class Task {
public:
                Task(TaskType type, const std::vector<std::shared_ptr<Figure
        >>& data) : type(type), data(data) {};
                TaskType getType() const {
                return type;
                }
                std::vector<std::shared_ptr<Figure>> getData() const {
                return data;
                }
                private:
                TaskType type;
                std::vector<std::shared_ptr<Figure>> data;
                };

class Subscriber {
public:
          virtual void print(const Task& task) const = 0;
          virtual ~Subscriber() = default;
          };

                        class ConsolePrinter : public Subscriber {
                        public:
                        void print(const Task& task) const override {
```

```
                auto data = task.getData();
                for(const auto& figure : data) {
                figure->Print(std::cout);
                }
                }
                };

        class FilePrinter : public Subscriber {
        public:
        void print(const Task& task) const override {
        auto data = task.getData();
        std::ofstream os(std::to_string(rand() % 1337) + ".txt");
        if(os.bad()) {
                        std::cout << "Couldn’t open file\n";
                }
                for(const auto& figure : data) {
                figure->Print(os);
                }
                }
                };


class TaskChanel { public:
        void subscribe(std::shared_ptr<Subscriber>& s) { subscribers.push_back(s);
          }

          void notify(const Task& task) {
          for(const auto& subscriber : subscribers) { subscriber->print(task);
                }
                }
                private:
                std::vector<std::shared_ptr<Subscriber>> subscribers;
                };
```

main.cpp

```
#include <iostream>

#include "pubsub.hpp"
#include "figure.hpp"
#include "rhombus.hpp"
#include "hexagon.hpp"
#include "pentagon.hpp"


class ThreadFunc { public:
ThreadFunc(const TaskChanel& taskChanel) : taskChanel( taskChanel) {};

          void addTask(const Task& task) {
          std::lock_guard<std::mutex> lock(queueMutex); tasks.push(task);
           }
```

```
void startWorking() { working = true;
        }

void stopWorking() { working = false;
        }

                bool isWorking() {
                return working;
                }

                std::condition_variable& getVar1() {
                return var1;
                }

                std::condition_variable& getVar2() {
                return var2;
                }

        std::mutex& getReadMutex() {   return
readMutex;
        }

                                        void operator()() {
                                        while(true) {
                                        std::unique_lock<std::mutex> mainLock(readMutex);
                                        while(!working) {
                                        var2.wait(mainLock);
                                        }
                                        if(!tasks.empty()) {
                                        {
                                        std::lock_guard<std::mutex> lock(queueMutex);
                                        Task currentTask = tasks.front();
                                        tasks.pop();
                                        if(currentTask.getType() == TaskType::exit) {
                                        break;
                                        } else {
                                        taskChanel.notify(currentTask);
                                        }
                                        this->stopWorking();
                                        var1.notify_one();
                                        }
                                        }

                }
                }
                private:
                TaskChanel taskChanel;
                std::mutex readMutex;
                std::condition_variable var1;
                std::condition_variable var2;
                std::mutex queueMutex;
                std::queue<Task> tasks;
```

```
73              bool working = false;
74              };
75
76      int main(int argc, char** argv) {
77      unsigned bufferSize;
78      if(argc != 2) {
79      std::cout << "Did this so argc wouldn't be highlited red/ check your input" << std::endl;
80              return -1;
81              }
82              bufferSize = std::atoi(argv[1]);
83              std::vector<std::shared_ptr<Figure>> figures;
84              std::string command; 85      int command2;
86
87      std::shared_ptr<Subscriber> consolePrint(new ConsolePrinter())
        ;
88      std::shared_ptr<Subscriber> filePrint(new FilePrinter());
89
90      TaskChanel taskChanel;
91      taskChanel.subscribe(consolePrint);
92      taskChanel.subscribe(filePrint);
93
94      ThreadFunc func(taskChanel);
95      std::thread thread(std::ref(func));
96
97                      while(std::cin >> command) {
98                      if(command == "exit") {
99                      func.addTask({TaskType::exit, figures});
100                     func.startWorking();
101                     func.getVar2().notify_one();
102                     break;
103                     } else if(command == "add") {
104                     std::shared_ptr<Figure> f;
105                                     std::cout << "1 - Rhombus, 2 - Pentagon, 3 - Hexagon"
        << std::endl;
106                                     std::cin >> command2;
107                                     try {
108                                     if(command2 == 1) {
109                                     f = std::make_shared<Rhombus>(Rhombus(std::cin
        ));
110                                     } else if(command2 == 2) {
111                                     f = std::make_shared<Pentagon>(Pentagon{std::
        cin});
112                                     } else if(command2 == 3) {
113                                     f = std::make_shared<Hexagon>(Hexagon{std::cin
        });
114                                     } else {
115                                     std::cout << "Wrong input" << std::endl;
116                                     }
117                                     figures.push_back(f);
118                                     } catch(std::exception& e) {
119                                     std::cerr << e.what() << std::endl;
120                                     }
121                                     if(figures.size() == bufferSize) {
```

```
                                func.addTask({TaskType::print, figures});
                                func.startWorking();
                                func.getVar2().notify_one();
                                std::unique_lock<std::mutex> lock(func.
        getReadMutex());
                                while(func.isWorking()) {
                                func.getVar1().wait(lock);
                                }
                                figures.resize(0);
                                }
                                } else {
                                std::cout << "Unknown command" << std::endl;
                                }
                                }
                                thread.join();
                                return 0;
                                }
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab8)

find_package(Threads)

        add_executable(lab8
        main.cpp
        point.cpp
        rhombus.cpp
        pentagon.cpp
        hexagon.cpp)

set_property(TARGET lab8 PROPERTY CXX_STANDARD 17)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -g -Wall -Wextra -Werror")

target_link_libraries(lab8 ${CMAKE_THREAD_LIBS_INIT})
```

2 Тестирование

test_1_argv.txt:

Создадим буфер размера один и добавим туда ромб, пятиугольник и шестиугольник.

1 - Rhombus, 2 - Pentagon, 3 - Hexagon

Rhombus: [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000]

1 - Rhombus, 2 - Pentagon, 3 - Hexagon

Pentagon: [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000]

1 - Rhombus, 2 - Pentagon, 3 - Hexagon

Hexagon:[0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000]

Полученные файлы:

qelderdelta@qelderdelta-UX331UA:~/Study/oop_exercise_08/build$ cat 1198.txt

Rhombus: [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000]

qelderdelta@qelderdelta-UX331UA:~/Study/oop_exercise_08/build$ cat 214.txt

Pentagon: [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000]

qelderdelta@qelderdelta-UX331UA:~/Study/oop_exercise_08/build$ cat 807.txt

Hexagon:[0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000] [0.000, 0.000]

3 Объяснение результатов работы программы

При вводе координат для создания ромба производится проверка этих координат, ведь они могут не образовывать ромб. Для этого реализована функция checkIfRhombus, которая вычисляет расстояния от одной точки до трёх остальных, а поскольку фигура является ромбом, то два из низ должны быть равны. Третье же значение функция возвращает ведь оно равно длине одной из диагоналей. Площадь ромба вычисляется как половина произведения диагоналей, центр - точка пересечения диагоналей. Методы вычисления площади и центра для пяти- и шестиугольника совпадают. Чтобы найти площадь необходимо перебрать все ребра и сложить площади трапеций, ограниченных этими ребрами. Чтобы найти центр необходимо разбить фигуры на треугольники(найти одну точку внутри фигуры), для каждого треугольника найти центроид и площадь и перемножить их, просуммировать полученные величины и разделить на общую площадь фигуры.

4 Выводы

В ходе выполнения работы я познакомился с тем, как устроены встроенные механизмы языка для разработки многопоточных программ, а также получил навыки их написания.