**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: 8 «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 8**

Тема: Асинхронное программирование

Студент: Моисеенков Илья Павлович

Группа: М8О-208Б-19

Преподаватель: Чернышов Л.Н.

Дата: 28.12.2020

Оценка: 15/15

Москва, 2020

1. **Постановка задачи**

Создать приложение, которое будет считывать из стандартного ввода данные фигур согласно варианту задания, выводить их характеристики на экран и записывать в файл. Фигуры могут задаваться как своими вершинами, так и другими характеристиками.

Требования к реализации:

* Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;
* Программа должна создавать классы, соответствующие введенным данным фигур;
* Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки;
* При накоплении буфера фигуры должны запускаться на асинхронную обработку, после чего буфер должен очищаться;
* Обработка должна производиться в отдельном потоке;
* Реализовать два обработчика, которые должны обрабатывать данные буфера:
  + Вывод информации о фигурах в буфере на экран;
  + Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.
* Оба обработчика должны обрабатывать каждый введенный буфер. После каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл;
* Обработчики должны быть реализованы в виде лямбда-функций и должны храниться в специальном массиве обработчиков, откуда и должны последовательно вызываться в потоке-обработчике;
* В программе должны быть ровно два потока. Один - основной, второй - для обработчиков;
* В программе должен прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик;
* Реализовать в основном потоке ожидание обработки буфера в потоке-обработчике. После отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

Вариант 5. Фигуры: ромб, пятиугольник, шестиугольник.

1. **Описание программы**

***Класс figure***

Класс figure - это абстрактный базовый класс для остальных фигур. Класс содержит в себе чисто виртуальные функции square() для вычисления площади, print() для печати фигуры, print\_to\_file() для записи в файл. Единственный атрибут - координаты центра фигуры.

***Классы rhombus, pentagon, hexagon***

Классы rhombus, pentagon и hexagon - это классы-наследники от figure, в которых описаны ромб, пятиугольник и шестиугольник соответственно. В этих классах переопределены все виртуальные функции из базового класса, а также переопределен оператор вывода. Класс rhombus дополнительно содержит два атрибута - длины диагоналей. Остальные классы содержат атрибут radius - радиус описанной окружности.

***Класс factory***

В данном классе реализован шаблон factory. Этот шаблон предназначен для упрощения создания новых объектов. Во время выполнения программы он сам определяет, какой объект необходимо создать, при помощи id фигуры. Фигуры и их id определены в enum class figure\_type. Класс возвращает умный указатель на созданную фигуру.

***Класс server***

Класс server представляет собой сервер для обработки фигур. При реализации класса использовались шаблоны проектирования singleton и publish-subscribe. Сервер создается в единственном экземпляре и работает в отдельном потоке. У сервера есть следующие атрибуты:

* std::vector<std::function<void(const MESSAGE\_T&)>> subscribers - вектор с “подписчиками”, т.е. с функциями-обработчиками,
* std::queue<std::shared\_ptr<figure>> message\_queue - очередь сообщений фигур,
* std::mutex mtx,
* std::string file\_name, std::ofstream fd - для работы с файлами,
* bool active - переменная, отвечающая за работу сервера.

Когда буфер с фигурами будет заполнен, сервер начинает обработку. Для каждой фигуры он вызывает все обработчики из массива, затем удаляет ее из буфера. Название файла для вывода генерируется случайным образом.

***Функция main***

В функции main пользователю предлагается интерфейс для добавления фигур в очередь. Для добавления фигуры нужно ввести её id, координаты центра и дополнительные атрибуты (для ромба - длины диагоналей, для пятиугольника и шестиугольника - длину радиуса описанной окружности). Обработка фигур производится автоматически при заполнении буфера. Размер буфера указывается в аргументах командной строки.

1. **Тестирование программы**

В качестве тестовых данных программе подается набор команд. Интерфейс для взаимодействия с программой:

1. Добавить ромб
2. Добавить пятиугольник
3. Добавить шестиугольник
4. Выход

Остальные команды игнорируются.

***test1.txt***

1 1 2 3 4 // Добавить ромб с центром (1,2) и диагоналями 3 и 4

2 1 2 3 // Добавить пятиугольник с центром (1,2) и радиусом 3

3 2 3 4 // Добавить шестиугольник с центром (2,3) и радиусом 4

3 5 1 2 // Добавить шестиугольник с центром (5,1) и радиусом 2

2 -1 -1 2 // Добавить пятиугольник с центром (-1,-1) и радиусом 2

1 1 4 2 9 // Добавить ромб с центром (1,4) и диагоналями 2 и 9

0 // Выход

**mosik@LAPTOP-69S778GL:~/oop\_lab8$** ./oop\_exercise\_08 3

1. Add rhombus

2. Add pentagon

3. Add hexagon

0. Exit

1

Enter coords of the center and lengths of diagonals

1 2 3 4

Successfully added

2

Enter coords of the center and length of radius

1 2 3

Successfully added

3

Enter coords of the center and length of side

2 3 4

Successfully added

Rhombus {(-0.5; 2), (1; 4), (2.5; 2), (1; 0)}

Square: 6

Center: (1; 2)

Pentagon {(3.9; 2.9), (1; 5), (-1.9; 2.9), (-0.76; -0.43), (2.8; -0.43)}

Square: 21

Center: (1; 2)

Hexagon {(6; 3), (4; 6.5), (8.9e-16; 6.5), (-2; 3), (-1.8e-15; -0.46), (4; -0.46)}

Square: 42

Center: (2; 3)

3

Enter coords of the center and length of side

5 1 2

Successfully added

2

Enter coords of the center and length of radius

-1 -1 2

Successfully added

1

Enter coords of the center and lengths of diagonals

1 4 2 9

Successfully added

Hexagon {(7; 1), (6; 2.7), (4; 2.7), (3; 1), (4; -0.73), (6; -0.73)}

Square: 10

Center: (5; 1)

Pentagon {(0.9; -0.38), (-1; 1), (-2.9; -0.38), (-2.2; -2.6), (0.18; -2.6)}

Square: 9.5

Center: (-1; -1)

Rhombus {(0; 4), (1; 8.5), (2; 4), (1; -0.5)}

Square: 9

Center: (1; 4)

0

**mosik@LAPTOP-69S778GL:~/oop\_lab8$** cat 303

Rhombus {(-0.5; 2), (1; 4), (2.5; 2), (1; 0)}

Square: 6

Center: (1; 2)

Pentagon {(3.9; 2.9), (1; 5), (-1.9; 2.9), (-0.76; -0.43), (2.8; -0.43)}

Square: 21

Center: (1; 2)

Hexagon {(6; 3), (4; 6.5), (8.9e-16; 6.5), (-2; 3), (-1.8e-15; -0.46), (4; -0.46)}

Square: 42

Center: (2; 3)

**mosik@LAPTOP-69S778GL:~/oop\_lab8$** cat 662

Hexagon {(7; 1), (6; 2.7), (4; 2.7), (3; 1), (4; -0.73), (6; -0.73)}

Square: 10

Center: (5; 1)

Pentagon {(0.9; -0.38), (-1; 1), (-2.9; -0.38), (-2.2; -2.6), (0.18; -2.6)}

Square: 9.5

Center: (-1; -1)

Rhombus {(0; 4), (1; 8.5), (2; 4), (1; -0.5)}

Square: 9

Center: (1; 4)

***test2.txt***

2 1 2 3 // Добавить пятиугольник с центром (1,2) и радиусом 3

3 2 5 8 // Добавить шестиугольник с центром (2,5) и радиусом 8

3 1 1 1 // Добавить шестиугольник с центром (1,1) и радиусом 1

0 // Выход

**mosik@LAPTOP-69S778GL:~/oop\_lab8$** ./oop\_exercise\_08 2

1. Add rhombus

2. Add pentagon

3. Add hexagon

0. Exit

2

Enter coords of the center and length of radius

1 2 3

Successfully added

3

Enter coords of the center and length of side

2 5 8

Successfully added

Pentagon {(3.9; 2.9), (1; 5), (-1.9; 2.9), (-0.76; -0.43), (2.8; -0.43)}

Square: 21

Center: (1; 2)

Hexagon {(10; 5), (6; 12), (-2; 12), (-6; 5), (-2; -1.9), (6; -1.9)}

Square: 1.7e+02

Center: (2; 5)

3

Enter coords of the center and length of side

1 1 1

Successfully added

0

**mosik@LAPTOP-69S778GL:~/oop\_lab8$** cat 170

Pentagon {(3.9; 2.9), (1; 5), (-1.9; 2.9), (-0.76; -0.43), (2.8; -0.43)}

Square: 21

Center: (1; 2)

Hexagon {(10; 5), (6; 12), (-2; 12), (-6; 5), (-2; -1.9), (6; -1.9)}

Square: 1.7e+02

Center: (2; 5)

Программа выдала корректные результаты на всех тестах.

1. **Листинг программы**

***figure.h***

#include <cmath>

#include <fstream>

class figure {

public:

figure() = default;

figure(std::pair<double, double>& center\_) : center(center\_) {}

virtual double square() = 0;

virtual void print() = 0;

virtual void print\_to\_file(std::ofstream&) = 0;

std::pair<double, double> get\_center() { return center;}

protected:

std::pair<double, double> center;

};

***rhombus.h***

#include "figure.h"

class rhombus : public figure {

public:

rhombus() = default;

rhombus(std::pair<double, double>& center, double d1, double d2) : figure(center), diag1(d1), diag2(d2) {}

double square() override { return diag1 \* diag2 \* 0.5;}

void print() override {

std::cout << \*this << std::endl;

std::cout << "Square: " << square() << std::endl;

auto center = get\_center();

std::cout << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

void print\_to\_file(std::ofstream& out) override {

out << \*this << std::endl;

out << "Square: " << square() << std::endl;

auto center = get\_center();

out << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

friend std::ostream& operator<<(std::ostream& out, rhombus& r);

private:

double diag1 = 0;

double diag2 = 0;

};

std::ostream& operator<<(std::ostream& out, rhombus& r) {

out << "Rhombus {(" << r.center.first - r.diag1 \* 0.5 << "; " << r.center.second << "), (";

out << r.center.first << "; " << r.center.second + r.diag2 \* 0.5 << "), (";

out << r.center.first + r.diag1 \* 0.5 << "; " << r.center.second << "), (";

out << r.center.first << "; " << r.center.second - r.diag2 \* 0.5 << ")}";

return out;

}

***pentagon.h***

#include "figure.h"

class pentagon : public figure {

public:

pentagon() = default;

pentagon(std::pair<double, double>& center, double rad) : figure(center), radius(rad) {}

double square() override {

double pi = acos(-1);

double side = radius \* cos(13 \* pi / 10) - radius \* cos(17 \* pi / 10);

return sqrt(25 + 10 \* sqrt(5)) \* pow(side, 2) \* 0.25;

}

void print() override {

std::cout << \*this << std::endl;

std::cout << "Square: " << square() << std::endl;

auto center = get\_center();

std::cout << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

void print\_to\_file(std::ofstream& out) override {

out << \*this << std::endl;

out << "Square: " << square() << std::endl;

auto center = get\_center();

out << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

friend std::ostream& operator<<(std::ostream& out, pentagon& p);

private:

double radius = 0;

};

std::ostream& operator<<(std::ostream& out, pentagon& p) {

out << "Pentagon {";

double pi = acos(-1);

for (int i = 0; i < 5; ++i) {

double angle = 2 \* pi \* i / 5;

out.precision(2);

out << "(" << p.center.first + p.radius \* cos(angle + pi / 10) << "; "

<< p.center.second + p.radius \* sin(angle + pi / 10) << ")";

if (i != 4) {

out << ", ";

}

}

out << "}";

return out;

}

***hexagon.h***

#include "figure.h"

class hexagon : public figure {

public:

hexagon() = default;

hexagon(std::pair<double, double>& center, double rad) : figure(center), radius(rad) {}

double square() override {return pow(radius, 2) \* 3 \* sqrt(3) \* 0.5;}

void print() override {

std::cout << \*this << std::endl;

std::cout << "Square: " << square() << std::endl;

auto center = get\_center();

std::cout << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

void print\_to\_file(std::ofstream& out) override {

out << \*this << std::endl;

out << "Square: " << square() << std::endl;

auto center = get\_center();

out << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

friend std::ostream& operator<<(std::ostream& out, hexagon& h);

private:

double radius = 0;

};

std::ostream& operator<<(std::ostream& out, hexagon& h) {

out << "Hexagon {";

double pi = acos(-1);

for (int i = 0; i < 6; ++i) {

double angle = pi \* i / 3;

out.precision(2);

out << "(" << h.center.first + h.radius \* cos(angle) << "; "

<< h.center.second + h.radius \* sin(angle) << ")";

if (i != 5) {

out << ", ";

}

}

out << "}";

return out;

}

***factory.h***

#include <memory>

#include "rhombus.h"

#include "pentagon.h"

#include "hexagon.h"

enum class figure\_type {

rhombus = 1,

pentagon = 2,

hexagon = 3

};

struct factory {

static std::shared\_ptr<figure> create(figure\_type t) {

switch (t) {

case figure\_type::rhombus: {

std::pair<double, double> center;

double d1, d2;

std::cin >> center.first >> center.second >> d1 >> d2;

return std::make\_shared<rhombus>(center, d1, d2);

}

case figure\_type::pentagon: {

std::pair<double, double> center;

double r;

std::cin >> center.first >> center.second >> r;

return std::make\_shared<pentagon>(center, r);

}

case figure\_type::hexagon: {

std::pair<double, double> center;

double r;

std::cin >> center.first >> center.second >> r;

return std::make\_shared<hexagon>(center, r);

}

default:

throw std::logic\_error("Wrong figure id");

}

}

};

***server.h***

#include <vector>

#include <queue>

#include <mutex>

#include <thread>

#include <functional>

#include <fstream>

template <class MESSAGE\_T>

class server {

public:

using subscriber\_t = std::function<void(const MESSAGE\_T&)>;

// singleton

static server& get()

{

static server instance;

return instance;

}

// subscriber - function to handle buffer

void register\_subscriber(const subscriber\_t& sub) {

subscribers.push\_back(sub);

}

// publisher - element of a buffer (figure)

void publish(const MESSAGE\_T& msg) {

std::lock\_guard<std::mutex> lck(mtx);

message\_queue.push(msg);

}

// starting handler

void run(size\_t max\_size) {

while (active) {

if (message\_queue.size() == max\_size) {

// handling

std::string file\_name = generate\_file\_name();

fd.open(file\_name);

while (!message\_queue.empty()) {

std::lock\_guard<std::mutex> lck(mtx);

MESSAGE\_T val = message\_queue.front();

message\_queue.pop();

for (auto sub : subscribers) { sub(val); }

}

fd.flush();

fd.close();

} else { std::this\_thread::yield; }

}

}

void stop() { active = false;}

std::ofstream& get\_fd() { return fd; }

private:

std::vector<subscriber\_t> subscribers;

std::queue<MESSAGE\_T> message\_queue;

std::mutex mtx;

std::string file\_name;

std::ofstream fd;

server() {};

bool active = true;

std::string generate\_file\_name() {

std::string file\_name;

srand(time(NULL));

for (int i = 0; i < 3; ++i) {

file\_name.push\_back(rand() % 10 + '0');

}

return file\_name;

}

};

***main.cpp***

/\* Моисеенков И.П. М8О-208Б-19

\* github.com/mosikk/oop\_exercise\_08

\* Реализовать приложение для асинхронной обработки фигур

\*/

#include <iostream>

#include <queue>

#include <ctime>

#include <sstream>

#include "factory.h"

#include "server.h"

void print\_menu() {

std::cout << "1. Add rhombus" << std::endl;

std::cout << "2. Add pentagon" << std::endl;

std::cout << "3. Add hexagon" << std::endl;

std::cout << "0. Exit" << std::endl << std::endl;

}

using server\_t = server<std::shared\_ptr<figure>>;

int main(int argc, char\* argv[]) {

if (argc != 2) {

std::cout << "Syntax: ./oop\_exercise\_08 buffer\_size" << std::endl;

return 1;

}

if (std::stoi(argv[1]) <= 0) {

std::cout << "Incorrect buffer size" << std::endl;

return 2;

}

size\_t buf\_size = std::stoul(argv[1]);

// adding subscribers (handler functions)

server\_t::get().register\_subscriber([](const std::shared\_ptr<figure> fig) {

fig->print();

});

server\_t::get().register\_subscriber([](const std::shared\_ptr<figure> fig) {

fig->print\_to\_file(server\_t::get().get\_fd());

});

// starting handler

std::thread th([buf\_size]() {

server\_t::get().run(buf\_size);

});

print\_menu();

int cmd;

while (true) {

std::cin >> cmd;

if (cmd == 1) {

std::cout << "Enter coords of the center and lengths of diagonals" << std::endl;

std::shared\_ptr<figure> fig = factory::create((figure\_type)cmd);

server\_t::get().publish(fig);

std::cout << "Successfully added" << std::endl;

} else if (cmd == 2) {

std::cout << "Enter coords of the center and length of radius" << std::endl;

std::shared\_ptr<figure> fig = factory::create((figure\_type)cmd);

server\_t::get().publish(fig);

std::cout << "Successfully added" << std::endl;

} else if (cmd == 3) {

std::cout << "Enter coords of the center and length of side" << std::endl;

std::shared\_ptr<figure> fig = factory::create((figure\_type)cmd);

server\_t::get().publish(fig);

std::cout << "Successfully added" << std::endl;

} else if (cmd == 0) {

server\_t::get().stop();

break;

} else { std::cout << "Incorrect cmd" << std::endl; }

}

th.join();

}

1. **Выводы**

Данная лабораторная работа была направлена на изучение основ асинхронного программирования. Мной были изучены механизмы работы с потоками, возможные проблемы, которые могут возникнуть при многопоточной обработке данных, а также их решения.

Я изучил примитив синхронизации мьютекс, который обеспечивает взаимное исключение исполнения критических участков кода, а также шаблон std::lock\_guard, который упрощает работу с мьютексами. Также был изучен еще один шаблон проектирования - Publish-Subscribe.

**Список используемых источников**

1. Руководство по языку С++ [Электронный ресурс]. URL: <https://www.cplusplus.com/> (дата обращения 25.12.2020).
2. Шаблон publish-subscribe [Электронный ресурс]. URL: <https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber> (дата обращения 25.12.2020).
3. Статья про асинхронное программирование [Электронный ресурс]. URL: <https://habr.com/ru/company/jugru/blog/446562/> (дата обращения 25.12.2020).