**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 8**

Тема: Асинхронное программирование

Студент: Лисин Роман Сергеевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

**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 27: Прямоугольник, трапеция, ромб

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.

***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.

***Работа программы***

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. **Набор тестов**

1 5 6 7 8 2 3 4 5 6 3 8 9 7

1. **Результаты выполнения тестов**

1

Enter coords of the center and lengths of diagonals

5 6 7 8

Successfully added

2

Enter coords of the left highest point, the lengths of height and width

3 4 5 6

Successfully added

3

Enter the lengths of bases and height

8 9 7

Successfully added

Rhombus {(1.5, 6); (5, 10); (8.5, 6); (5, 2)}

Rectangle {(3, 4); (9, 4); (9, -1); (3, -1)}

Trapezoid {( Bases: 8, 9; Height: 7)}

1. **Листинг программы**

**main.cpp**

#include <iostream>

#include <queue>

#include <ctime>

#include <sstream>

#include "factory.h"

#include "server.hpp"

/\* Лабораторная работа №8

Вариант 27: Прямоугольник, трапеция, ромб

Выполнил Лисин Роман, группа М8О-206Б-20

\*/

void print\_menu() {

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

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

std::cout << "3. Add trapezoid" << 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->write\_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 left highest point, the lengths of height and width" << 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 the lengths of bases and height" << 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();

}

**server.hpp**

#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 + ".txt";

}

};

**rectangle.h**

#ifndef OOP\_LAB7\_RECTANGLE\_H

#define OOP\_LAB7\_RECTANGLE\_H

#include "figure.h"

#include <iostream>

class rectangle : public figure {

public:

rectangle() = default;

rectangle(std::pair<double, double>& center, double h, double w) : figure(center), height(h), width(w) {}

double square() override {

return height \* width;

}

void print() override {

std::cout << \*this;

}

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

int id = 1;

out << "Rectangle {(" << center.first << ", " << center.second << "); "

<< "(" << center.first + width << ", " << center.second << "); "

<< "(" << center.first + width << ", " << center.second - height << "); "

<< "(" << center.first << ", " << center.second - height << ")}\n";

}

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

private:

double height = 0;

double width = 0;

};

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

out << "Rectangle {(" << r.center.first << ", " << r.center.second << "); "

<< "(" << r.center.first + r.width << ", " << r.center.second << "); "

<< "(" << r.center.first + r.width << ", " << r.center.second - r.height << "); "

<< "(" << r.center.first << ", " << r.center.second - r.height << ")}\n";

return out;

}

#endif //OOP\_LAB7\_RECTANGLE\_H

**rhombus.h**

#ifndef OOP\_LAB7\_RHOMBUS\_H

#define OOP\_LAB7\_RHOMBUS\_H

#include "figure.h"

#include <iostream>

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;

}

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

int id = 1;

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

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

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

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

}

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 << ")}\n";

return out;

}

#endif //OOP\_LAB7\_RHOMBUS\_H

**trapezoid.h**

#ifndef OOP\_LAB7\_TRAPEZOID\_H

#define OOP\_LAB7\_TRAPEZOID\_H

#include "figure.h"

#include <iostream>

class trapezoid : public figure {

public:

trapezoid() = default;

trapezoid(double a, double b, double h) : a(a), b(b), h(h) {}

double square() override {

return (a + b) / 2 \* h;

}

void print() override {

std::cout << \*this;

}

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

int id = 1;

out << "Trapezoid {(" << " Bases: " << a << ", " << b << "; Height: " << h << ")}\n";

}

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

private:

double a = 0;

double b = 0;

double h = 0;

};

std::ostream& operator<<(std::ostream& out, trapezoid& t) {

out << "Trapezoid {(" << " Bases: "<< t.a << ", " << t.b << "; Height: " << t.h << ")}\n";

return out;

}

#endif //OOP\_LAB7\_TRAPEZOID\_H

**factory.h**

#ifndef OOP\_LAB7\_FACTORY\_H

#define OOP\_LAB7\_FACTORY\_H

#include "rhombus.h"

#include "rectangle.h"

#include "trapezoid.h"

enum class figure\_type {

rhombus = 1,

rectangle = 2,

trapezoid = 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::rectangle: {

std::pair<double, double> center;

double h, w;

std::cin >> center.first >> center.second >> h >> w;

return std::make\_shared<rectangle>(center, h, w);

}

case figure\_type::trapezoid: {

std::pair<double, double> center;

double a, b, h;

std::cin >> a >> b >> h;

return std::make\_shared<trapezoid>(a, b, h);

}

default:

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

}

}

static std::shared\_ptr<figure> read\_from\_file(figure\_type t, std::ifstream& in) {

switch (t) {

case figure\_type::rhombus: {

std::pair<double, double> center;

double d1, d2;

in.read((char\*)&center.first, sizeof(double));

in.read((char\*)&center.second, sizeof(double));

in.read((char\*)&d1, sizeof(double));

in.read((char\*)&d2, sizeof(double));

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

}

case figure\_type::rectangle: {

std::pair<double, double> center;

double height, width;

in.read((char\*)&center.first, sizeof(double));

in.read((char\*)&center.second, sizeof(double));

in.read((char\*)&height, sizeof(double));

in.read((char\*)&width, sizeof(double));

return std::make\_shared<rectangle>(center, height, width);

}

case figure\_type::trapezoid: {

double a, b, h;

in.read((char\*)&a, sizeof(double));

in.read((char\*)&b, sizeof(double));

in.read((char\*)&h, sizeof(double));

return std::make\_shared<trapezoid>(a, b, h);

}

default:

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

}

}

};

#endif //OOP\_LAB7\_FACTORY\_H

**figure.h**

#ifndef OOP\_LAB7\_FIGURE\_H

#define OOP\_LAB7\_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 write\_to\_file(std::ofstream& out) = 0;

std::pair<double, double> get\_center() {

return center;

}

protected:

std::pair<double, double> center;

};

#endif //OOP\_LAB7\_FIGURE\_H

**ЛИТЕРАТУРА**

1. Паттерн Factory [Электронный ресурс].

URL: http://cpp-reference.ru/patterns/creational-patterns/factory-method/ (дата обращения: 16.12.2021)