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: Мариничев Иван Александрович
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата: 28.12.20
Оценка:

Москва, 2020

1. Постановка задачи. Вариант 13

   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 (например, координата центра, количество точек и радиус).

   Программа должна:

   1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;
   2. Программа должна создавать классы, соответствующие введенным данных фигур;
   3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки. Например, для буфера размером 10 фигур: **oop_exercise_08 10**
   4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;
   5. Обработка должна производиться в отдельном потоке;
   6. 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.
   7. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.
   8. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.
   9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
   10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.

11. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку

Фигуры:

- Ромб;
- Пятиугольник;
- Шестиугольник;

2. Описание программы

Таблица 1 – Функции и классы

Название	Описание
Класс **figure**	Класс figure - это абстрактный базовый класс для остальных фигур. Класс содержит в себе чисто виртуальные функции Area() для вычисления площади, Print() для печати фигуры, PrintToFile() 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 FigureType. 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 - мьютекс, ● 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.

3. Результаты выполнения тестов

Таблица 2 – Тесты и результаты работы с ними

Название тестового файла	Входные данные	Результат
test_01.txt	1 0 0 2 4 1 1 1 3 5 1 2 2 4 6 2 0 0 5 2 1 1 10 2 5 5 13 3 0 0 5 3 1 1 10 3 2 2 15 3 100 100 1000 0	1. Add rhombus 2. Add pentagon 3. Add hexagon 4. Help 0. Exit Select option: Enter coords of the center and lengths of diagonals Successfully added Select option: Enter coords of the center and lengths of diagonals Successfully added Select option: Enter coords of the center and lengths of diagonals Successfully added Select option: Enter coords of the center and length of radius Successfully added Select option: Enter coords of the center and length of radius Successfully added Select option: Enter coords of the center and length of radius Successfully added Select option: Enter coords of the center and length of side

		Successfully added Select option: Enter coords of the center and length of side Successfully added Select option: Enter coords of the center and length of side Successfully added Select option: Enter coords of the center and length of side Successfully added Select option: Rhombus: 1. Coordinates: (1; 0), (0; 2), (-1; 0), (0; -2) 2. Area of figure: 4 3. Center: (0; 0) Rhombus: 1. Coordinates: (2.5; 1), (1; 3.5), (-0.5; 1), (1; -1.5) 2. Area of figure: 7.5 3. Center: (1; 1) Rhombus: 1. Coordinates: (4; 2), (2; 5), (0; 2), (2; -1) 2. Area of figure: 12 3. Center: (2; 2) Pentagon: 1. Coordinates: (4.8; 1.5), (3.1e-16; 5), (-4.8; 1.5), (-2.9; -4), (2.9; -4) 2. Area of figure: 59 3. Center: (0; 0) Pentagon: 1. Coordinates: (11; 4.1), (1; 11), (-8.5; 4.1), (-4.9; -7.1), (6.9; -7.1) 2. Area of figure: 2.4e+02 3. Center: (1; 1) Pentagon: 1. Coordinates: (17; 9), (5; 18), (-7.4; 9), (-2.6; -5.5), (13; -5.5) 2. Area of figure: 4e+02 3. Center: (5; 5) Hexagon: 1. Coordinates: (5; 0), (2.5; 4.3), (-2.5; 4.3), (-5; 6.1e-16), (-2.5; -4.3), (2.5; -4.3) 2. Area of figure: 65 3. Center: (0; 0) Hexagon: 1. Coordinates: (11; 1), (6; 9.7), (-4; 9.7),

		(-9; 1), (-4; -7.7), (6; -7.7) 2. Area of figure: 2.6e+02 3. Center: (1; 1) Hexagon: 1. Coordinates: (17; 2), (9.5; 15), (-5.5; 15), (-13; 2), (-5.5; -11), (9.5; -11) 2. Area of figure: 5.8e+02 3. Center: (2; 2) Hexagon: 1. Coordinates: (1.1e+03; 1e+02), (6e+02; 9.7e+02), (-4e+02; 9.7e+02), (-9e+02; 1e+02), (-4e+02; -7.7e+02), (6e+02; -7.7e+02) 2. Area of figure: 2.6e+06 3. Center: (1e+02; 1e+02)
test_02.txt	1 0 0 10 100 2 10 10 100 3 4 4 10 0	1. Add rhombus 2. Add pentagon 3. Add hexagon 4. Help 0. Exit Select option: Enter coords of the center and lengths of diagonals Successfully added Select option: Enter coords of the center and length of radius Successfully added Select option: Enter coords of the center and length of side Successfully added Select option: Rhombus: 1. Coordinates: (5; 0), (0; 50), (-5; 0), (0; -50) 2. Area of figure: 500 3. Center: (0; 0) Pentagon: 1. Coordinates: (1.1e+02; 41), (10; 1.1e+02), (-85; 41), (-49; -71), (69; -71) 2. Area of figure: 2.4e+04 3. Center: (10; 10) Hexagon: 1. Coordinates: (14; 4), (9; 13), (-1; 13), (-6; 4), (-1; -4.7), (9; -4.7) 2. Area of figure: 2.6e+02 3. Center: (4; 4)

4. Листинг программы

main.cpp

```
/*
 *  Мариничев И. А.
 *  М8О-208Б-19
 *  github.com/IvaMarin/oop_exercise_08
 *  Вариант 13:
 *  Фигура:
 *  - ромб
 *  - пятиугольник
 *  - шестиугольник
 */

#include <iostream>
#include <queue>
#include <ctime>
#include <sstream>

#include "factory.hpp"
#include "server.hpp"

void Menu() {
	std::cout << "1. Add rhombus" << std::endl;
	std::cout << "2. Add pentagon" << std::endl;
	std::cout << "3. Add hexagon" << std::endl;
	std::cout << "4. Help" << 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().RegisterSubscriber([](const
std::shared_ptr<Figure> fig) {
		fig->Print();
		});

	server_t::Get().RegisterSubscriber([](const
std::shared_ptr<Figure> fig) {
		fig->PrintToFile(server_t::Get().GetFD());
		});

	// starting handler
	std::thread th([buf_size]() {
		server_t::Get().Run(buf_size);
		});

	Menu();
```

```
	int cmd;

	std::cout << "Select option: ";
	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((FigureType)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((FigureType)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((FigureType)cmd);
			server_t::Get().Publish(fig);
			std::cout << "Successfully added" << std::endl;
		} else if (cmd == 4) {
		Menu();
		}
		else if (cmd == 0) {
			server_t::Get().Stop();
			break;
		}
		else {
			std::cout << "Incorrect cmd" << std::endl;
		}
		std::cout << std::endl;
	std::cout << "Select option: "; // Repeat input
	}

	th.join();

}
```

server.hpp

```
#ifndef SERVER_HPP
#define 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; // Guaranteed to be destroyed
                                // Instantiated on first use
        return instance;
    }

    // Subscriber - function to handle buffer
    void RegisterSubscriber(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 = GenerateFileName();
                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 {
                // Provides a hint to the implementation to reschedule
                // the execution of threads, allowing other threads to
run
                std::this_thread::yield;
            }
        }
    }

    void Stop() {
        active = false;
    }

    // Gets file descriptor
    std::ofstream& GetFD() {
        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 GenerateFileName() {
        std::string file_name = "file_";
        srand(time(NULL));
        for (int i = 0; i < 3; ++i) {
            file_name.push_back(rand() % 10 + '0');
        }
        return file_name;
    }
};

#endif //SERVER_HPP
```

factory.hpp

```
#ifndef FACTORY_HPP
#define FACTORY_HPP

#include <memory>

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

enum class FigureType {
    RHOMBUS = 1,
    PENTAGON = 2,
    HEXAGON = 3
};

struct factory {
    static std::shared_ptr<Figure> Create(FigureType t) {
        switch (t) {
        case FigureType::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 FigureType::PENTAGON: {
            std::pair<double, double> center;
            double r;
            std::cin >> center.first >> center.second >> r;
            return std::make_shared<Pentagon>(center, r);
        }
        case FigureType::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");
        }
    }
};

#endif //FACTORY_HPP
```

figure.hpp

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

#include <cmath>
#include <fstream>

class Figure {
public:
    Figure() = default;

    Figure(std::pair<double, double>& center_) : center(center_) {}

    virtual double Area() = 0;

    virtual void Print() = 0;

    virtual void PrintToFile(std::ofstream&) = 0;

    std::pair<double, double> GetCenter() {
        return center;
    }

protected:
    std::pair<double, double> center;
};

#endif // FIGURE_HPP
```

rhombus.hpp

```
#ifndef RHOMBUS_HPP
#define RHOMBUS_HPP

#include "figure.hpp"

class Rhombus : public Figure {
public:
    Rhombus() = default;

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

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

    void Print() override {
        std::cout << *this << std::endl;
        std::cout << "2. Area of figure: " << Area() << std::endl;
        auto center = GetCenter();
        std::cout << "3. Center: (" << center.first << "; " <<
center.second << ")" << std::endl;
    }

    void PrintToFile(std::ofstream& out) override {
        out << *this << std::endl;
        out << "2. Area of figure: " << Area() << std::endl;
        auto center = GetCenter();
        out << "3. Center: (" << center.first << "; " << center.second
<< ")" << 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 << "\nRhombus:\n";
    out << "1. Coordinates: (";
    out << 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;
}

#endif //RHOMBUS_HPP
```

pentagon.hpp

```
#ifndef PENTAGON_HPP
#define PENTAGON_HPP

#include "figure.hpp"

class Pentagon : public Figure {
public:
    Pentagon() = default;

    Pentagon(std::pair<double, double>& center, double rad) :
Figure(center), radius(rad) {}

    double Area() 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 << "2. Area of figure: " << Area() << std::endl;
        auto center = GetCenter();
        std::cout << "3. Center: (" << center.first << "; " <<
center.second << ")" << std::endl;
    }

    void PrintToFile(std::ofstream& out) override {
        out << *this << std::endl;
        out << "2. Area of figure: " << Area() << std::endl;
        auto center = GetCenter();
        out << "3. Center: (" << center.first << "; " << center.second
<< ")" << std::endl;
    }

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

```
private:
    double radius = 0;
};

std::ostream& operator<<(std::ostream& out, Pentagon& p) {
    out << "\nPentagon:\n";
    out << "1. Coordinates: ";
    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 << ", ";
        }
    }
    return out;
}

#endif //PENTAGON_HPP
```

hexagon.hpp

```
#ifndef HEXAGON_HPP
#define HEXAGON_HPP

#include "figure.hpp"

class Hexagon : public Figure {
public:
    Hexagon() = default;

    Hexagon(std::pair<double, double>& center, double rad) :
Figure(center), radius(rad) {}

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

    void Print() override {
        std::cout << *this << std::endl;
        std::cout << "2. Area of figure: " << Area() << std::endl;
        auto center = GetCenter();
        std::cout << "3. Center: (" << center.first << "; " <<
center.second << ")" << std::endl;
    }

    void PrintToFile(std::ofstream& out) override {
        out << *this << std::endl;
        out << "2. Area of figure: " << Area() << std::endl;
        auto center = GetCenter();
        out << "3. Center: (" << center.first << "; " << center.second
<< ")" << std::endl;
    }

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

private:
    double radius = 0;
```

```
};

std::ostream& operator<<(std::ostream& out, Hexagon& h) {
    out << "\nHexagon:\n";
    out << "1. Coordinates: ";
    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 << ", ";
        }
    }
    return out;
}

#endif //HEXAGON_HPP
```

Makefile

```
CC=g++
FLAGS=-std=c++17 -pedantic -Wall -Wextra -Wno-unused-variable -pthread
OUTPUT=oop_exercise_08

all: main.cpp
      $(CC) $(FLAGS) main.cpp -o $(OUTPUT)
clean:
      rm *.o $(OUTPUT)
```

5. Выводы

В ходе лабораторной работы я познакомился с асинхронным программированием, получил практические навыки в параллельной обработке данных и синхронизации потоков.

Список литературы

1. Стефан К. Дьюхэрст Скользкие места C++. Как избежать проблем при проектировании и компиляции ваших программ — ISBN: 5-94074-083-9 — 266 с.
2. Руководство по языку C++ [Электронный ресурс]. URL: https://www.cplusplus.com/ (дата обращения 25.12.2020).
3. Шаблон publish-subscribe [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber (дата обращения 26.12.2020).
4. Статья про асинхронное программирование [Электронный ресурс]. URL: https://habr.com/ru/company/jugru/blog/446562/ (дата обращения 26.12.2020).