Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 8: «Асинхронное программирование»

Группа:	М8О-108Б-18, №6
Студент:	Васильева Василиса Евгеньевна
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	28.12.2019

Москва, 2019

1. Задание

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:

a. Вывод информации о фигурах в буфере на экран;

b. 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. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
9. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
10. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

Вариант 6: пятиугольник, шестиугольник, восьмиугольник

2. Адрес репозитория на GitHub

https://github.com/vasilisavasileva/oop_exercise_08

3. Код программы на C++

Pentagon.h

```
#pragma once
#include"figure.h"
#include<array>

struct Pentagon : figure {
private:
	std::array<vertex, 5> vertices_;
public:
	void Read(std::istream& is) override;
	void Print(std::ostream& os) const override;

};
```

Octagon.h

```
#pragma once
#include"figure.h"
#include<array>

struct Octagon : figure {
private:
	std::array<vertex, 8> vertices_;
public:
	void Read(std::istream& is) override;
	void Print(std::ostream& os) const override;

};
```

Hexagon.h

```
#pragma once
#include"figure.h"
#include<array>

struct Hexagon : figure {
private:
	std::array<vertex, 6> vertices_;
public:
	void Read(std::istream& is) override;
	void Print(std::ostream& os) const override;

};
```

Pentagon.cpp

```
#include"Pentagon.h"
#include<iostream>
#include<fstream>


void Pentagon::Read(std::istream& is) {
	for (int i = 0; i < 5; i++) {
		is >> vertices_[i].x >> vertices_[i].y;
	}
}

void Pentagon::Print(std::ostream& os) const{
```

```
	os << "Pentagon" << std::endl;
	for (int i = 0; i < 5; i++) {
		os << vertices_[i].x << ' ' << vertices_[i].y << std::endl;
	}
}
```

Octagon.cpp

```
#include"Octagon.h"
#include<iostream>
#include<fstream>


void Octagon::Read(std::istream& is) {
	for (int i = 0; i < 8; i++) {
		is >> vertices_[i].x >> vertices_[i].y;
	}
}

void Octagon::Print(std::ostream& os) const {
	os << "Octagon" << std::endl;
	for (int i = 0; i < 8; i++) {
		os << vertices_[i].x << ' ' << vertices_[i].y << std::endl;
	}
}
```

Hexagon.cpp

```
#include"Hexagon.h"
#include<iostream>
#include<fstream>


void Hexagon::Read(std::istream& is) {
	for (int i = 0; i < 6; i++) {
		is >> vertices_[i].x >> vertices_[i].y;
	}
}

void Hexagon::Print(std::ostream& os) const {
	os << "Octagon" << std::endl;
	for (int i = 0; i < 6; i++) {
		os << vertices_[i].x << ' ' << vertices_[i].y << std::endl;
	}
}
```

Figure.h

```
#pragma once
#include<iostream>

struct figure {
	virtual void Read(std::istream& is) = 0;
	virtual void Print(std::ostream& os) const = 0;
	virtual ~figure() = default;
};

struct vertex {
	int x, y;
};
```

Factory.h

```
#pragma once
#include<iostream>
#include"figure.h"
#include"Pentagon.h"
#include"Hexagon.h"
#include"Octagon.h"
#include<memory>

struct factory {
public:
	virtual std::unique_ptr<figure> factoring(std::istream& is) = 0;
	virtual ~factory() = default;
};

struct pentagon_factory : factory{
	std::unique_ptr<figure> factoring(std::istream& is) override {
		std::unique_ptr<Pentagon> t_pent;
		t_pent = std::make_unique<Pentagon>();
		t_pent->Read(is);
		return std::move(t_pent);
	}
};

struct hexagon_factory : factory {
	std::unique_ptr<figure> factoring(std::istream& is) override {
		std::unique_ptr<Hexagon> t_hex;
		t_hex = std::make_unique<Hexagon>();
		t_hex->Read(is);
		return std::move(t_hex);
	}
};

struct octagon_factory : factory {
	std::unique_ptr<figure> factoring(std::istream& is) override {
		std::unique_ptr<Octagon> t_oct;
		t_oct = std::make_unique<Octagon>();
		t_oct->Read(is);
		return std::move(t_oct);
	}
};
```

Handlers.h

```
#pragma once
#include<vector>
#include<string>
#include<fstream>
#include"figure.h"


struct handler {
	virtual void exec(std::vector<std::unique_ptr<figure>>& figures) = 0;
	virtual ~handler() = default;
};

struct file_handler : handler {
	void exec(std::vector<std::unique_ptr<figure>>& figures) override {
		static int count_file = 0;
		std::string filename = "";
		++count_file;

		filename = "file_" + std::to_string(count_file) + ".txt";
		std::ofstream file(filename);
```

```
		for (int i = 0; i < figures.size(); ++i) {
			figures[i]->Print(file);
		}
	}
};
struct console_handler : handler {
	void exec(std::vector<std::unique_ptr<figure>>& figures) override {
		for (int i = 0; i < figures.size(); ++i) {
			figures[i]->Print(std::cout);
		}
	}
};
```

main.cpp

```
#include<memory>
#include<iostream>
#include<vector>
#include<thread>
#include<mutex>
#include<Windows.h>
#include<future>
#include<condition_variable>
#include<string>
#include"figure.h"
#include"Pentagon.h"
#include"Hexagon.h"
#include"Octagon.h"
#include"Factory.h"
#include"Handlers.h"


void handle(std::vector<std::unique_ptr<figure>>& figures, int buffer_size,
std::condition_variable& cv_mtx1, std::condition_variable& cv_mtx2, std::mutex& mtx,
bool& stop_thrd) {
	std::unique_lock<std::mutex> lock(mtx);
	cv_mtx2.notify_all();
	std::vector<std::unique_ptr<handler>> handlers;

	handlers.push_back(std::make_unique<file_handler>());
	handlers.push_back(std::make_unique<console_handler>());
	while (!(stop_thrd)) {
		cv_mtx1.wait(lock);
		if (figures.size() != 0) {
			for (int i = 0; i < handlers.size(); ++i) {
				handlers[i]->exec(figures);
			}
		}		figures.clear();
		cv_mtx2.notify_all();
	}
	return;
}
int main(int argc, char* argv[]) {
	if (argc != 2)
		return 1;
	std::condition_variable cv_mtx1;
	std::condition_variable cv_mtx2;
	std::vector<std::unique_ptr<figure>> figures;
	std::unique_ptr<factory> my_factory;
	std::mutex mtx;
	std::unique_lock<std::mutex> lock(mtx);
	int buffer_size, menu;
```

```
	buffer_size = std::stoi(argv[1]);
	bool stop_thrd = false;
	std::thread handler(handle, std::ref(figures), buffer_size, std::ref(cv_mtx1),
std::ref(cv_mtx2), ref(mtx), std::ref(stop_thrd));
	cv_mtx2.wait(lock);
	while (true) {
		for (int i = 0; i < buffer_size; ++i) {
			std::cout << "1. Pentagon" << std::endl;
			std::cout << "2. Hexagon" << std::endl;
			std::cout << "3. Octagon" << std::endl;
			std::cin >> menu;
			switch (menu) {
			case 1:
				my_factory = std::make_unique<pentagon_factory>();
				figures.push_back(my_factory->factoring(std::cin));
				break;
			case 2:
				my_factory = std::make_unique<hexagon_factory>();
				figures.push_back(my_factory->factoring(std::cin));
				break;
			case 3:
				my_factory = std::make_unique<octagon_factory>();
				figures.push_back(my_factory->factoring(std::cin));
				break;
			}
		}
		cv_mtx1.notify_all();
		cv_mtx2.wait(lock);
		std::cout << "The buffer is filled" << std::endl;
		std::cout << "Want to create a new one?" << std::endl;
		std::cout << "y for Yes, n for No" << std::endl;
		char choise;
		std::cin >> choise;
		if (choise != 'y')
			break;
	}
	stop_thrd = true;
	cv_mtx1.notify_all();
	lock.unlock();
	handler.join();
	return 0;
}
```

4. Тест

PS C:\Users\Administrator.LAPTOP-C7V2HJKO\source\repos\oop_lab8\Debug> .\oop_lab8.exe 2
1. Pentagon
2. Hexagon
3. Octagon
1
0 0 0 0 0 0 0 0 0 0
1. Pentagon
2. Hexagon
3. Octagon
2
0 0 0 0 0 0 0 0 0 0 0

Pentagon
0 0
0 0
0 0
0 0
0 0
Octagon
0 0
0 0
0 0
0 0
0 0
0 0
The buffer is filled
Want to create a new one?
y for Yes, n for No
n

5. Объяснение результатов работы программы

При запуске программы мы вводим размер буфера, в который будем пушить наши элементы. Потом выбираем фигуру, координаты которой будем вводить. Вводим фигуры, пока наш буфер не будет заполнен. Когда буфер заполняется, то отправляет вектор с указателями на фигуры на обработку во второй поток. Этот поток создается при запуске основной программы и ожидает заполнения буфера. Перед тем как приступить к обработке второй поток ожидает освобождения мьютекса. О событии сообщает cv_mtx1. После обработки пользователю предлагается продолжить работу или завершить. При завершении работы обработка завершается, вектор чистится, мьютекс отпускается и вся работа прекращается.

6. Вывод

Асинхронное программирование позволяет продуктивнее задействовать временной ресурс, тк одновременно на выполнение можно подать два потока, независящих друг от друга. Для стыковки результатов работы возможно использование мьютекса и условных переменных.