# 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: Кочисов Алексей

Группа: М8О-201Б-20

Преподаватель: Чернышов Л.Н. Дата:

Оценка:

Москва, 2020

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

Программа должна:

Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;

Программа должна создавать классы, соответствующие введенным данных фигур;

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**

При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;

Обработка должна производиться в отдельном потоке;

Реализовать два обработчика, которые должны обрабатывать данные буфера:

* 1. Вывод информации о фигурах в буфере на экран;
  2. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.

Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. 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 хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.

В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;

В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.

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.

|  |  |  |
| --- | --- | --- |
| **Фигура №1** | **Фигура №2** | **Фигура №3** |
| 6-угольник | 8-угольник | Триугольник |

# Описание программы

Синхронизация процессов осуществляется посредством двух условных переменных и мьютекса.

# Набор тестов test\_1

r

hexagon 1 1 1 1 1 1 1 1 1 1 1 1

octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

triangle 1 1 1 1 1 1 q

# test\_2

r

hexagon 1 1 1 1 1 1 1 1 1 1 1 1

octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

triangle 1 1 0 0 1 0 q

# Результаты выполнения тестов

Input 'q' for quit, or 'r' to continue r

hexagon 1 1 1 1 1 1 1 1 1 1 1 1 Added

octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 Added

triangle 1 1 1 1 1 1 Added

hexagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

octagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

triangle: (1 1) (1 1) (1 1)

Input 'q' for quit, or 'r' to continue q

C:\Users\Daniel\source\repos\Lab8\Debug\Lab8.exe (процесс 14948) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно…

Input 'q' for quit, or 'r' to continue r

hexagon 1 1 1 1 1 1 1 1 1 1 1 1 Added

octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 Added

triangle 1 1 0 0 1 0 Added

hexagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

octagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

triangle: (1 1) (0 0) (1 0)

Input 'q' for quit, or 'r' to continue q

C:\Users\Daniel\source\repos\Lab8\Debug\Lab8.exe (процесс 17468) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно…

# Листинг программы

**Main.cpp**

#include <iostream> #include <vector> #include <memory> #include <string> #include <thread> #include <mutex>

#include <condition\_variable> #include "factory.h" #include "subscriber.h"

int main(int argc, char\*\* argv) { if (argc != 2) {

std::cout << "Wrong. \n"; return 0;

}

int Vecsize = std::atoi(argv[1]); std::vector<std::shared\_ptr<figures::Figure>> Vec; factory::Factory Factory;

std::condition\_variable cv; std::condition\_variable cv2; std::mutex mutex;

bool done = false; char cmd = 'd'; int in = 1;

std::vector<std::shared\_ptr<Sub>> subs; subs.push\_back(std::make\_shared<Print>()); subs.push\_back(std::make\_shared<Log>()); std::thread subscriber([&]() {

std::unique\_lock<std::mutex> subscriber\_lock(mutex); while (!done) {

cv.wait(subscriber\_lock); if (done) {

cv2.notify\_all(); break;

}

for (unsigned int i = 0; i < subs.size(); ++i) { subs[i]->output(Vec);

}

in++; Vec.resize(0); cv2.notify\_all();

}

});

while (cmd != 'q') {

std::cout << "Input 'q' for quit, or 'r' to continue" << std::endl; std::cin >> cmd;

if (cmd != 'q') {

std::unique\_lock<std::mutex> main\_lock(mutex); for (int i = 0; i < Vecsize; i++) {

Vec.push\_back(Factory.FigureCreate(std::cin)); std::cout << "Added" << std::endl;

}

cv.notify\_all(); cv2.wait(main\_lock);

}

}

done = true; cv.notify\_all(); subscriber.join(); return 0;

}

**Factory.h**

#ifndef FACTORY\_H #define FACTORY\_H

#include <iostream> #include "figure.h"

namespace factory {

class Factory { public:

std::shared\_ptr<figures::Figure> FigureCreate(std::istream& is) const

{

std::string type; std::cin >> type;

if (type == "hexagon") {

return std::shared\_ptr<figures::Figure>(new

figures::Hexagon(is));

}

else if (type == "octagon") {

return std::shared\_ptr<figures::Figure>(new

figures::Octagon(is));

}

else if (type == "triangle") {

return std::shared\_ptr<figures::Figure>(new

figures::Triangle(is));

}

throw std::logic\_error("Wrong. Figures: hexagon, octagon,

triangle");

}

};

}

#endif

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

#include <cmath>

#include "point.h"

namespace figures {

enum FigureType {

hexagon,

octagon,

triangle

};

class Figure {

public:

virtual std::ostream& print(std::ostream& out) const = 0;

~Figure() = default;

};

class Hexagon : public Figure {

public:

point A, B, C, D, E, F;

Hexagon() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0, 0 }, E{ 0, 0 }, F{ 0, 0 } {}

explicit Hexagon(std::istream& is) {

is >> A >> B >> C >> D >> E >> F;

}

std::ostream& print(std::ostream& os) const override {

os << "hexagon: " << A << " " << B << " " << C << " " << D << " " << E << " " << F << std::endl;

return os;

}

};

class Octagon : public Figure {

public:

point A, B, C, D, E, F, G, H;

Octagon() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0, 0 }, E{ 0, 0 }, F{ 0, 0 }, G{ 0, 0 }, H{ 0, 0 } {}

explicit Octagon(std::istream& is) {

is >> A >> B >> C >> D >> E >> F >> G >> H;

}

std::ostream& print(std::ostream& os) const override {

os << "octagon: " << A << " " << B << " " << C << " " << D << " " << E << " " << F << " " << G << " " << H << std::endl;;

return os;

}

};

class Triangle : public Figure {

public:

point A, B, C, D;

Triangle() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 } {}

explicit Triangle(std::istream& is) {

is >> A >> B >> C;

}

std::ostream& print(std::ostream& os) const override {

os << "triangle: " << A << " " << B << " " << C << std::endl;

return os;

}

};

}

#endif

**Point.h**

#ifndef OOP\_LAB7\_POINT\_H

#define OOP\_LAB7\_POINT\_H

#include <iostream>

struct point {

point() : x(0), y(0) {}

point(double a, double b) : x(a), y(b) {}

double x;

double y;

};

std::istream& operator>>(std::istream& is, point& p) {

is >> p.x >> p.y;

return is;

}

std::ostream& operator<<(std::ostream& os, point p) {

os << '(' << p.x << ' ' << p.y << ')';

return os;

}

#endif

**Subscriber.h**

#ifndef SUBSCRIBERS\_H

#define SUBSCRIBERS\_H

#include <fstream>

class Sub {

public:

virtual void output(std::vector<std::shared\_ptr<figures::Figure>>& Vec) = 0;

virtual ~Sub() = default;

};

class Print : public Sub {

public:

void output(std::vector<std::shared\_ptr<figures::Figure>>& Vec) override {

for (auto& figure : Vec) {

figure->print(std::cout);

}

}

};

class Log : public Sub {

public:

Log() : in(1) {}

void output(std::vector<std::shared\_ptr<figures::Figure>>& Vec) override {

std::string filename;

filename = std::to\_string(in);

filename += ".txt";

std::ofstream file;

file.open(filename);

for (auto& figure : Vec) {

figure->print(file);

}

in++;

}

private:

int in;

};

#endif

# Выводы

В ходе выполнения лабораторной работы мною были приобретены начальные навыки работы с асинхронным программированием. Также я научился работать с аргументами программы.