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-208Б-18, №19 |
| Студент: | Овечкин Виталий Андреевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: |  |

Москва, 2019

1. **Тема**: Асинхронное программирование
2. **Цель работы**: Знакомство с асинхронным программировнаием; Получение точечных навыков в параллельной обработке данных; Получение практических навыков в синхронизации потоков.
3. **Задание** (*вариант № 19* ):

Фигуры — прямоугольник, трапеция, ромб.

1. **Адрес репозитория на GitHub** [https://github.com/vitalouivi/oop\_exercise\_0](https://github.com/wAlienUFOx/oop_exercise_01)8
2. **Код программы на С++**

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. Try ./oop\_exercise\_08 size\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';

std::shared\_ptr<Print> print;

std::shared\_ptr<Log> log;

std::thread subscriber([&]() {

std::unique\_lock<std::mutex> subscriber\_lock(mutex);

while(!done) {

cv.wait(subscriber\_lock);

if (done) {

cv2.notify\_all();

break;

}

print->output(Vec);

log->output(Vec);

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;

}

figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

#include <cmath>

#include "point.h"

namespace figures{

enum FigureType {

rhombus,

rectangle,

trapezoid

};

class Figure {

public:

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

~Figure() = default;

};

class Rectangle : public Figure {

public:

point A , B, C, D;

Rectangle(): A{0, 0}, B{0, 0}, C{0, 0}, D{0,0} {}

explicit Rectangle(std::istream& is) {

is >> A >> B >> C >> D;

double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;

a = sqrt((B.x- A.x) \* (B.x - A.x) + (B.y - A.y) \* (B.y - A.y));

b = sqrt((C.x- B.x) \* (C.x - B.x) + (C.y - B.y) \* (C.y - B.y));

c = sqrt((C.x- D.x) \* (C.x - D.x) + (C.y - D.y) \* (C.y - D.y));

d = sqrt((D.x- A.x) \* (D.x - A.x) + (D.y - A.y) \* (D.y - A.y));

d1 = sqrt((B.x- D.x) \* (B.x - D.x) + (B.y - D.y) \* (B.y - D.y));

d2 = sqrt((C.x- A.x) \* (C.x - A.x) + (C.y - A.y) \* (C.y - A.y));

ABC = (a \* a + b \* b - d2 \* d2) / 2 \* a \* b;

BCD = (b \* b + c \* c - d1 \* d1) / 2 \* b \* c;

CDA = (d \* d + c \* c - d2 \* d2) / 2 \* d \* c;

DAB = (a \* a + d \* d - d1 \* d1) / 2 \* a \* d;

if(ABC != BCD || ABC != CDA || ABC != DAB)

throw std::logic\_error("It`s not a rectangle");

}

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

os << "rectangle: " << A << " " << B << " " << C << " " << D << std::endl;

return os;

}

};

class Trapezoid : public Figure {

public:

point A, B, C, D;

Trapezoid(): A{0, 0}, B{0, 0}, C{0, 0}, D{0,0} {}

explicit Trapezoid(std::istream& is){

is >> A >> B >> C >> D;

if((C.y - B.y) / (C.x - B.x) != (D.y - A.y) / (D.x - A.x))

throw std::logic\_error("It`s not a trapezoid");

}

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

os << "trapezoid: " << A << " " << B << " " << C << " " << D << std::endl;

return os;

}

};

class Rhombus : public Figure {

public:

point A, B, C, D;

Rhombus(): A{0, 0}, B{0, 0}, C{0, 0}, D{0,0} {}

explicit Rhombus(std::istream& is){

is >> A >> B >> C >> D;

double a, b, c, d;

a = sqrt((B.x - A.x) \* (B.x - A.x) + (B.y - A.y) \* (B.y - A.y));

b = sqrt((C.x - B.x) \* (C.x - B.x) + (C.y - B.y) \* (C.y - B.y));

c = sqrt((C.x - D.x) \* (C.x - D.x) + (C.y - D.y) \* (C.y - D.y));

d = sqrt((D.x - A.x) \* (D.x - A.x) + (D.y - A.y) \* (D.y - A.y));

if(a != b || a != c || a != d)

throw std::logic\_error("It`s not a rhombus");

}

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

os << "rhombus: " << A << " " << B << " " << C << " " << D << std::endl;

return os;

}

};

}

#endif

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 == "rhombus") {

return std::shared\_ptr<figures::Figure>(new figures::Rhombus(is));

} else if (type == "rectangle") {

return std::shared\_ptr<figures::Figure>(new figures::Rectangle(is));

} else if (type == "trapezoid") {

return std::shared\_ptr<figures::Figure>(new figures::Trapezoid(is));

}

throw std::logic\_error("Wrong. Figures: rhombus, rectangle, trapezoid");

}

};

}

#endif

subscriber.h

#ifndef SUBSCRIBERS\_H

#define SUBSCRIBERS\_H

#include <fstream>

class Print {

public:

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

for (auto& figure : Vec) {

figure->print(std::cout);

}

}

~Print() = default;

};

class Log {

public:

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

std::string filename;

std::cout << "Input filename" << std::endl;

std::cin >> filename;

std::ofstream file;

file.open(filename);

for (auto &figure : Vec) {

figure->print(file);

}

}

~Log() = default;

};

#endif

CMakeLists.txt

cmake\_minimum\_required (VERSION 3.5)

project(lab8)

add\_executable(oop\_exercise\_08

main.cpp)

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -Wall -g3 -Wextra -thread")

set\_target\_properties(oop\_exercise\_08 PROPERTIES CXX\_STANDART 14 CXX\_STANDART\_REQUIRED ON)

1. **Набор testcases**

test\_01.txt

r

trapezoid 0 0 1 1 2 1 3 0

rectangle 0 0 0 0 0 0 0 0

rhombus 1 1 1 1 1 1 1 1

r

rectangle 1 1 1 1 1 1 1 1

rhombus 1 1 1 1 1 1 1 1

rectangle 2 2 2 2 2 2 2 2

q

test\_02.txt

r

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

r

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

rectangle 2 2 2 2 2 2 2 2

q

1. **Результаты выполнения тестов**

$ ./oop\_exercise\_08 3

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

r

rectangle 0 0 0 0 0 0 0 0

Added

rhombus 1 1 1 1 1 1 1 1

Added

trapezoid 0 0 1 1 2 1 3 0

Added

rectangle: (0 0) (0 0) (0 0) (0 0)

rhombus: (1 1) (1 1) (1 1) (1 1)

trapezoid: (0 0) (1 1) (2 1) (3 0)

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

r

rectangle 1 1 1 1 1 1 1 1

Added

rhombus 1 1 1 1 1 1 1 1

Added

rectangle 2 2 2 2 2 2 2 2

Added

rectangle: (1 1) (1 1) (1 1) (1 1)

rhombus: (1 1) (1 1) (1 1) (1 1)

rectangle: (2 2) (2 2) (2 2) (2 2)

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

q

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

r

rectangle 2 2 2 2 2 2 2 2

Added

rectangle 2 2 2 2 2 2 2 2

Added

rectangle 2 2 2 2 2 2 2 2

Added

rectangle 2 2 2 2 2 2 2 2

Added

rectangle 2 2 2 2 2 2 2 2

Added

rectangle 2 2 2 2 2 2 2 2

Added

rectangle: (2 2) (2 2) (2 2) (2 2)

rectangle: (2 2) (2 2) (2 2) (2 2)

rectangle: (2 2) (2 2) (2 2) (2 2)

rectangle: (2 2) (2 2) (2 2) (2 2)

rectangle: (2 2) (2 2) (2 2) (2 2)

rectangle: (2 2) (2 2) (2 2) (2 2)

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

1. **Объяснение результатов работы программы - вывод**

В subscriber.h реализованы два подпсичика — обработчика. Один осуществляет вывод в файл, другой в текстовый файл.

Синхронизация процессов осуществляется посредством двух условных переменных и мьютекса.

В ходе выполнения лабораторной работы были приобретены начальные навыки работы с асинхронным программировнаием, получены точечные навыки в параллельной обработке данных, получены практические навыки в синхронизации потоков.

Была на конкретном примере разобрана работа условной переменной и классов - подписчиков.