Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа**

**по курсу «ООП»**

**Тема:**

**Асинхронное программирование.**

|  |  |
| --- | --- |
| Студент: | Николаев В.А. |
| Группа: | М80-206Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 14 |
| Оценка: |  |
| Дата: |  |

Москва

2019

**1. Код программы на языке C++:**

**point.h:**

#include <iostream>

struct point {

double x, y;

point (double a,double b) { x = a, y = b;};

point() = default;

};

std::istream& operator >> (std::istream& npt,point& p );

std::ostream& operator << (std::ostream& out,const point& p);

**point.cpp:**

#include "point.h"

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

return npt >> p.x >> p.y;

}

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

return out << p.x << ' ' << p.y << '\n';

}

**figure.h:**

#pragma once

#include <iostream>

#include "point.h"

struct figure {

virtual point center() const = 0;

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

virtual double area() const = 0;

virtual void printFile(std::ofstream&) const = 0 ;

virtual ~figure() = default;

};

**pentagon.h:**

#pragma once

#include "figure.h"

struct pentagon : figure{

point a1,a2,a3,a4,a5;

point center() const override;

void print(std::ostream& out) override;

double area() const override;

pentagon() = default;

pentagon(std::istream& is);

pentagon(std::ifstream& is);

};

**pentagon.cpp:**

#include "pentagon.h"

point pentagon::center() const {

double x,y;

x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;

y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;

point p(x,y);

return p;

}

void pentagon::print(std::ostream& out) {

out << "Coordinates are:\n"<<"{\n"<< a1 << a2 << a3 << a4 << a5 << "}\n";

}

double pentagon::area() const {

return (0.5) \* std::abs((a1.x\*a2.y + a2.x\*a3.y + a3.x\*a4.y + a4.x\*a5.y + a5.x\*a1.y) - ( a1.y\*a2.x + a2.y\*a3.x + a3.y\*a4.x + a4.y\*a5.x + a5.y\*a1.x ));

}

pentagon::pentagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5;

}

pentagon::pentagon(std::ifstream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5;

}

**hexagon.h:**  
#pragma once

#include "figure.h"

struct hexagon : figure

{

point a1, a2, a3, a4, a5, a6;

point center() const override;

void print(std::ostream& out) override;

double area() const override;

hexagon() = default;

hexagon(std::istream& is);

hexagon(std::ifstream& npt);

};

**hexagon.cpp:**

#include "hexagon.h"

point hexagon::center() const {

double x,y;

x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x) / 6;

y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y) / 6;

point p(x,y);

return p;

}

void hexagon::print(std::ostream& out) {

out << "Coordinates are:\n{\n"<< a1 << a2 << a3 << a4 << a5 << a6 << "}\n";

}

double hexagon::area() const {

return 0.5 \* std::abs((a1.x\*a2.y + a2.x\*a3.y + a3.x\*a4.y + a4.x\*a5.y + a5.x\*a6.y + a6.x\*a1.y) - ( a1.y\*a2.x + a2.y\*a3.x + a3.y\*a4.x + a4.y\*a5.x + a5.y\*a6.x + a6.y\*a1.x ));

}

hexagon::hexagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;

}

hexagon::hexagon(std::ifstream& npt) {

npt >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;

}

**octagon.h:**

#pragma once

#include "figure.h"

struct octagon : figure

{

point a1, a2, a3, a4, a5, a6, a7, a8;

point center() const override;

void print(std::ostream& out) override;

double area() const override;

octagon() = default;

octagon(std::istream& is);

octagon(std::ifstream& is);

};

**octagon.cpp:**

#include "octagon.h"

point octagon::center() const {

double x,y;

x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x + a7.x + a8.x) / 8;

y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y + a7.y + a8.y) / 8;

point p(x,y);

return p;

}

void octagon::print(std::ostream& out) {

out << "Coordinates are:\n{\n"<< a1 << a2 << a3 << a4 << a5 << a6 << a7 << a8 << "}\n";

}

double octagon::area() const {

return 0.5 \* std::abs((a1.x\*a2.y + a2.x\*a3.y + a3.x\*a4.y + a4.x\*a5.y + a5.x\*a6.y + a6.x\*a7.y + a7.x\*a8.y + a8.x\*a1.y) - ( a1.y\*a2.x + a2.y\*a3.x + a3.y\*a4.x + a4.y\*a5.x + a5.y\*a6.x + a6.y\*a7.x + a7.y\*a8.x + a8.y\*a1.x ));

}

octagon::octagon(std::istream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;

}

octagon::octagon(std::ifstream& is) {

is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;

}

**factory.h:**

#pragma once

#include <memory>

#include <iostream>

#include <fstream>

#include "hexagon.h"

#include "octagon.h"

#include "pentagon.h"

#include <string>

struct factory {

std::shared\_ptr<figure> fig(std::istream& is);

std::shared\_ptr<figure> fig\_from\_file(std::ifstream& is);

};

**factory.cpp:**

#include "factory.h"

std::shared\_ptr<figure> factory::fig(std::istream &is) {

std::string name;

is >> name;

if ( name == "pentagon" ) {

return std::shared\_ptr<figure> ( new pentagon(is));

} else if ( name == "hexagon") {

return std::shared\_ptr<figure> ( new hexagon(is));

} else if ( name == "octagon") {

return std::shared\_ptr<figure> ( new octagon(is));

} else {

throw std::logic\_error("There is no such figure\n");

}

}

std::shared\_ptr<figure> factory::fig\_from\_file(std::ifstream &is) {

std::string name;

is >> name;

if ( name == "pentagon" ) {

return std::shared\_ptr<figure> ( new pentagon(is));

} else if ( name == "hexagon") {

return std::shared\_ptr<figure> ( new hexagon(is));

} else if ( name == "octagon") {

return std::shared\_ptr<figure> ( new octagon(is));

} else {

throw std::logic\_error("There is no such figure\n");

}

}

**sub.h:**

#pragma once

#include "figure.h"

#include "pentagon.h"

#include "hexagon.h"

#include "octagon.h"

#include <vector>

#include <memory>

class sub {

public:

virtual void Print(std::vector<std::shared\_ptr<figure>>& v) = 0;

};

**subscriber.h:**

#pragma once

#include "sub.h"

#include <string>

#include <fstream>

#include <time.h>

#include <string>

class ConsolePrint : public sub {

public:

void Print(std::vector<std::shared\_ptr<figure>>& v) override {

for (unsigned int i = 0; i < v.size(); i++) {

v[i]->print(std::cout);

}

}

};

class FilePrint : public sub {

private:

unsigned int count = 1;

public:

void Print(std::vector<std::shared\_ptr<figure>>& v) override {

std::string filename = "";

filename = "file\_" + std::to\_string(count) + ".txt";

count++;

std::ofstream file(filename);

for (unsigned int i = 0; i < v.size(); i++) {

v[i]->printFile(file);

}

}

};

**main.cpp:**

#include <iostream>

#include <thread>

#include <vector>

#include <memory>

#include <mutex>

#include <condition\_variable>

#include "figure.h"

#include "point.h"

#include "factory.h"

#include "hexagon.h"

#include "pentagon.h"

#include "octagon.h"

#include "subscriber.h"

#include "printer.h"

int main(int argc, char\*\* argv) {

if (argc != 2) {

std::cout << "To check test\_01.txt put in ./lab8 3 < test\_01.txt";

return 0;

}

unsigned int BufSize = std::atoi(argv[1]);

std::vector<std::shared\_ptr<figure>> f;

std::string cmd;

factory factory;

bool done = false;

std::condition\_variable rd;

std::condition\_variable hd;

std::mutex mutex;

int in = 1;

std::vector<std::shared\_ptr<sub>> s;

s.push\_back(std::make\_shared<ConsolePrint>());

s.push\_back(std::make\_shared<FilePrint>());

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

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

while (!done) {

rd.wait(sub\_lock);

if (done) {

hd.notify\_all();

break;

}

for (unsigned int i = 0; i < s.size(); i ++) {

s[i]->Print(f);

}

in++;

f.resize(0);

hd.notify\_all();

}

});

std::cout << "1 - to add " << BufSize << " figures\n"

"2 - to finish execution of program\n";

while(cmd != "2") {

std::cin >> cmd;

if (cmd != "2") {

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

for (unsigned int i = 0; i < BufSize; i++) {

f.push\_back(factory.FigureCreate(std::cin));

if (f.size() == BufSize) {

std::cout << "Buffer is full!\n";

}

}

rd.notify\_all();

hd.wait(main\_lock);

}

}

done = true;

rd.notify\_all();

sub.join();

return 0;

}

**Makefile:**

all:

g++ -pthread main.cpp hexagon.cpp pentagon.cpp octagon.cpp point.cpp factory.cpp -o lab8

**2. Ссылка на репозиторий на GitHub.**

**https://github.com/a1dv/oop\_exercise\_08.git**

**3. Набор тестов.**

1 pentagon 0 0 2 0 2 2 1 3 0 2

hexagon 0 0 1 -1 2 0 2 2 1 3 0 2

octagon 0 0 1 -1 2 0 3 1 2 2 1 3 0 2 -1 1

2

**4. Результаты выполнения тестов.**

test\_01.result:

Buffer is full!

Coordinates are:

{

0 0

2 0

2 2

1 3

0 2

}

Coordinates are:

{

0 0

1 -1

2 0

2 2

1 3

0 2

}

Coordinates are:

{

0 0

1 -1

2 0

3 1

2 2

1 3

0 2

-1 1

}

**5. Объяснение результатов работы программы.**

Запуская программу, пользователь вводит размер буфера, в котором будут храниться фигуры, то есть количество фигур, затем заполняет буфер. Когда буфер заполнен, начинается асинхронная обработка фигур, а буфер очищается. Затем, используя оба обработчика, указанных в задании, результат записывается в файл и выводится в терминал.

**6. Вывод.**

Ознакомился с асинхронным программированием, расширил свои знания по ОС, узнал о библиотеке condition\_variable.