**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 8**

Тема: Асинхронное программирование.

Студент: Петрухин Дмитрий

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. **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:

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 3:

Реализовать простейший графический редактор с возможностью сохранения документов в файлы, и загрузки из них. Применить паттерн “Factory”.

|  |  |  |  |
| --- | --- | --- | --- |
| 3. | Прямоугольник | Трапеция | Ромб |

1. **Ссылка на github**

[**https://github.com/dobb2/oop\_exercise\_08**](https://github.com/dobb2/oop_exercise_08)

1. **Описание программы**

Программа состоит из 4 файлов:

1) figures.h - содержит реализацию фигур и все операции связанные с ними.

2) factory.h - содержит класс для создания графических примитиве фигур.

3) subscriber.h - реализация класса, необходимого для передачи в поток как функтора, который необходим для выполнения обработки на отдельном потоке.

4) main.cpp - файл с взаимодействием с пользователем.

Пользователь при запуске указывает размер буфера фигур, то есть количество фигур, которое можно вбить буфер. В программе пользователь может полностью заполнить буфер различными фигурами, после чего будет показана содержимое буфера на экран и экспорт буфера в файл с уникальным именем. После экспорта буфер очищается и пользователь может вновь его заполнить различными фигурами. В программе имеются два потока, в поток subscriber\_thread передаем функтор класса Subscriber, который после заполнения буфера будет выводить информацию о фигурах из буфера в файл и на экран.

При неверном вводе параметров фигуры будет происходить исключения и пользователю нужно вновь выбрать фигуру с данными.

1. **Набор тестов**

*test\_01.txt*:

./oop\_exercise\_08 3

Add

Rec 0 0 3 0 3 5 0 5

Trapeze 0 0 4 0 3 5 1 5

Rhomb -2 0 0 -5 2 0 0 5

Menu

Add

Rec 5 5 10 5 10 10 5 10

Rhomb 0 0 2 -3 4 0 2 3

Rec 0 0 3 0 3 2 0 2

Exit

ls

cat 0

cat 1

Создаем буфер размером 3

*test\_02.txt*:

вводим неверно заданные фигуры

./oop\_exercise\_08 1

Add

Rec 0 0 1 1 2 2 3 3

Rhomb 5 10 3 3 9 0 0 9

Rec 0 0 5 0 5 10 0 10

Add

Trapeze sigsg

ls

cat 0

1. **Результаты выполнения тестов.**

***test\_01.txt*:**

➜ cmake-build-debug git:(master) ✗ ./oop\_exercise\_08 3

Menu

Add figure

Exit

Add

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rec

Enter the coordinates separated by a space

0 0 3 0 3 5 0 5

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Trapeze

Enter the coordinates separated by a space

0 0 4 0 3 5 1 5

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rhomb

Enter the coordinates separated by a space

-2 0 0 -5 2 0 0 5

Id: 0

Figure: Rectangle

Coords:

<0, 0>

<3, 0>

<3, 5>

<0, 5>

Id: 1

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 5>

<1, 5>

Id: 2

Figure: Trapezoid

Coords:

<-2, 0>

<0, -5>

<2, 0>

<0, 5>

Menu

Menu

Add figure

Exit

Add

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rec

Enter the coordinates separated by a space

5 5 10 5 10 10 5 10

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rhomb

Enter the coordinates separated by a space

0 0 2 -3 4 0 2 3

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rec

Enter the coordinates separated by a space

0 0 3 0 3 2 0 2

Id: 0

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 10>

<5, 10>

Id: 1

Figure: Trapezoid

Coords:

<0, 0>

<2, -3>

<4, 0>

<2, 3>

Id: 2

Figure: Rectangle

Coords:

<0, 0>

<3, 0>

<3, 2>

<0, 2>

Exit

➜ cmake-build-debug git:(master) ✗ ls

0 1 CMakeCache.txt CMakeFiles Makefile cmake\_install.cmake oop\_exercise\_08 oop\_exercise\_08.cbp

➜ cmake-build-debug git:(master) ✗ cat 0

Id: 0

Figure: Rectangle

Coords:

<0, 0>

<3, 0>

<3, 5>

<0, 5>

Id: 1

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 5>

<1, 5>

Id: 2

Figure: Trapezoid

Coords:

<-2, 0>

<0, -5>

<2, 0>

<0, 5>

➜ cmake-build-debug git:(master) ✗ cat 1

Id: 0

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 10>

<5, 10>

Id: 1

Figure: Trapezoid

Coords:

<0, 0>

<2, -3>

<4, 0>

<2, 3>

Id: 2

Figure: Rectangle

Coords:

<0, 0>

<3, 0>

<3, 2>

<0, 2>

***test\_02.txt*:**

➜ cmake-build-debug git:(master) ✗ ./oop\_exercise\_08 1

Menu

Add figure

Exit

Add

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rec

Enter the coordinates separated by a space

0 0 1 1 2 2 3 3

The entered coordinates of the vertices do not belong to the rectangle.

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rhomb

Enter the coordinates separated by a space

5 10 3 3 9 0 0 9

The entered coordinates of the vertices do not belong to the rhombus.

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Rec

Enter the coordinates separated by a space

0 0 5 0 5 10 0 10

Id: 0

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Add

Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]

Trapeze

Enter the coordinates separated by a space

sigsg

libc++abi.dylib: terminating with uncaught exception of type std::overflow\_error: Is not a number

[1] 16856 abort ./oop\_exercise\_08 1

➜ cmake-build-debug git:(master) ✗ ls

0 CMakeCache.txt CMakeFiles Makefile cmake\_install.cmake oop\_exercise\_08 oop\_exercise\_08.cbp

➜ cmake-build-debug git:(master) ✗ cat 0

Id: 0

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

➜ cmake-build-debug git:(master) ✗

1. **Листинг программы**

**main.cpp**

/\*

Асинхронное программирование

Вариант 3: Прямоугольник, трапеция, ромб.

Петрухин Дмитрий

М8О-201Б-18

\*/

#include <condition\_variable>

#include <fstream>

#include <iostream>

#include <memory>

#include <mutex>

#include <string>

#include <vector>

#include <thread>

#include <cstdlib>

#include "factory.h"

#include "figures.h"

#include "subscriber.h"

void menu() {

std::cout << "\nMenu\n";

std::cout << "Add figure\n";

std::cout << "Exit\n\n";

}

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

if (argc != 2) {

std::cout << "Enter size \n";

return 1;

}

size\_t vector\_size = std::atoi(argv[1]);

Factory factory;

Subscriber subscriber;

subscriber.buffer.reserve(vector\_size);

subscriber.processes.push\_back(std::make\_shared<Console\_process>());

subscriber.processes.push\_back(std::make\_shared<File\_process>());

std::thread subscriber\_thread(std::ref(subscriber));

menu();

std::string cmd;

while(std::cin >> cmd) {

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

if (cmd == "Menu") {

menu();

continue;

} else if (cmd == "Exit") {

subscriber.end = true;

subscriber.cv.notify\_all();

break;

} else if (cmd == "Add"){

std::string figure\_type;

for (size\_t id = 0; id < vector\_size; id++) {

std::cout << "Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]\n";

std::cin >> figure\_type;

if (figure\_type == "Rec"){

Vertex \*vertices = new Vertex[4];

std::cout << "Enter the coordinates separated by a space\n";

for (int i = 0; i < 4; i++) {

std::cin >> vertices[i];

}

try {

subscriber.buffer.push\_back(factory.FigureCreate(Rec, vertices, id));

} catch (std::logic\_error &e) {

std::cout << e.what() << "\n";

id--;

}

} else if (figure\_type == "Trapeze") {

Vertex \*vertices = new Vertex[4];

std::cout << "Enter the coordinates separated by a space\n";

for (int i = 0; i < 4; i++) {

std::cin >> vertices[i];

}

try {

subscriber.buffer.push\_back(factory.FigureCreate(Trapeze, vertices, id));

} catch (std::logic\_error &e) {

std::cout << e.what() << "\n";

id--;

}

} else if (figure\_type == "Rhomb") {

Vertex \*vertices = new Vertex[4];

std::cout << "Enter the coordinates separated by a space\n";

for (int i = 0; i < 4; i++) {

std::cin >> vertices[i];

}

try {

subscriber.buffer.push\_back(factory.FigureCreate(Rhomb, vertices, id));

} catch (std::logic\_error &e){

std::cout << e.what() << "\n";

id--;

}

}

}

if (subscriber.buffer.size() == vector\_size) {

subscriber.cv.notify\_all();

subscriber.cv.wait(main\_lock, [&subscriber]() {

return subscriber.success == true;

});

subscriber.success = false;

}

}

}

subscriber\_thread.join();

return 0;

}

**subscriber.h**

#ifndef SUBSCRIBER\_H

#define SUBSCRIBER\_H

#include <string>

struct Subscriber;

struct Subscribers\_process {

virtual void Process(std::vector<std::shared\_ptr<Figure>> &buffer) = 0;

virtual ~Subscribers\_process() = default;

};

struct Console\_process : Subscribers\_process {

void Process(std::vector<std::shared\_ptr<Figure>> &buffer) override {

for (const auto figure : buffer) {

figure->Print(std::cout);

}

}

};

struct File\_process : Subscribers\_process {

size\_t name = 0;

void Process(std::vector<std::shared\_ptr<Figure>> &buffer) override {

std::ofstream os(std::to\_string(name));

for (const auto figure : buffer) {

figure->Print(os);

}

name++;

}

};

struct Subscriber {

std::mutex mtx;

std::condition\_variable cv;

std::vector<std::shared\_ptr<Figure>> buffer;

std::vector<std::shared\_ptr<Subscribers\_process>> processes;

std::condition\_variable sv;

bool end;

bool success;

void operator()() {

for(;;) {

std::unique\_lock<std::mutex> guard(mtx); // lock mutex

cv.wait(guard, [&](){ // вошедший поток на этом месте ждет пока не выполнится условия

return buffer.size() == buffer.capacity() || end;

});

if (end) {

break;

}

for (size\_t i = 0; i < processes.size(); i++) {

processes[i]->Process(buffer);

}

buffer.clear();

success = true;

cv.notify\_all();

}

}

};

#endif //SUBSCRIBER\_H

**factory.h**

#ifndef FACTORY\_H

#define FACTORY\_H 1

#include "figures.h"

class Factory {

public:

std::shared\_ptr<Figure> FigureCreate(FigureType type) const {

std::shared\_ptr<Figure> res;

if (type == Rec) {

res = std::make\_shared<Rectangle>();

} else if (type == Rhomb) {

res = std::make\_shared<Rhombus>();

} else if (type == Trapeze) {

res = std::make\_shared<Trapezoid>();

}

return res;

}

std::shared\_ptr<Figure> FigureCreate(FigureType type, Vertex \*vertices, int id) const {

std::shared\_ptr<Figure> res;

if (type == Rec) {

res = std::make\_shared<Rectangle>(vertices[0], vertices[1], vertices[2], vertices[3], id);

} else if (type == Rhomb) {

res = std::make\_shared<Rhombus>(vertices[0], vertices[1], vertices[2], vertices[3], id);

} else if (type == Trapeze) {

res = std::make\_shared<Trapezoid>(vertices[0], vertices[1], vertices[2], vertices[3], id);

}

return res;

}

};

#endif //FACTORY\_H

**figures.h**

#ifndef FIGURES\_H

#define FIGURES\_H 1

#include <iostream>

#include <fstream>

#include <utility> // for pair

#include <memory>

#include <cmath>

#include <stdexcept>

enum FigureType {

Rec,

Rhomb,

Trapeze

};

using Vertex = std::pair<double, double>;

class Figure {

public:

virtual double Area() const = 0;

virtual Vertex Center() const = 0;

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

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

virtual void Deserialize(std::ifstream &is) = 0;

virtual int getId() const = 0;

virtual ~Figure() = default;

};

Vertex Get\_Center(const Vertex \*vertices, int n) {

double x = 0, y = 0;

for (int i = 0; i < n; i++) {

x += vertices[i].first;

y += vertices[i].second;

}

return std::make\_pair(x / n, y / n);

}

Vertex operator-(const Vertex &p1, const Vertex &p2) {

return {p1.first - p2.first, p1.second - p2.second};

}

bool oneline(const Vertex &a, const Vertex &b, const Vertex &c) {

return ( (c.first - a.first) / (b.first - a.first) == (c.second - a.second) / (b.second - a.second) );

}

bool collinear(const Vertex &a, const Vertex &b, const Vertex &c, const Vertex &d){

return (b.second-a.second)\*(d.first-c.first) - (d.second-c.second)\*(b.first-a.first) <= 1e-9;

}

bool perpendic(const Vertex &a, const Vertex &b, const Vertex &c, const Vertex &d){

using vect = std::pair<double, double>;

vect AC = c-a;

vect BD = d-b;

double dotProduct = AC.first\*BD.first + AC.second\*BD.second;

if(dotProduct <= 1e-9 && dotProduct >= -1e-9) return true;

else return false;

}

double dist(const Vertex &a, const Vertex &b){

return sqrt( ((b.first - a.first) \* (b.first - a.first)) + ((b.second - a.second) \* (b.second - a.second)));

}

bool operator==(const Vertex &a, const Vertex &b){

return (a.first == b.first) && (a.second == b.second);

}

std::ostream& operator<<(std::ostream &o, const Vertex &p){

o << "<" << p.first << ", " << p.second << ">";

return o;

}

bool isNumber(const std::string& s){

return !s.empty() && s.find\_first\_not\_of("-.0123456789") == std::string::npos;

}

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

std::string checker;

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.first = static\_cast<double>(std::stod(checker));

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.second = static\_cast<double>(std::stod(checker));

return is;

}

class Rectangle : public Figure {

int Id;

Vertex \*vertices;

public:

Rectangle() : Id{0}, vertices{new Vertex[4]} {

for (int i = 0; i < 4; i++){

vertices[i] = std::make\_pair(0,0);

}

}

Rectangle(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :

Id{id}, vertices{new Vertex[4]} {

if (a == b || a == c || b == c || a == d || oneline(a,b,c) || oneline(a,b,d) || oneline(b,c,d) ||

!(perpendic(a, b, a, d)) || !collinear(a, d, c, b)

|| !collinear(a, b, d, c)) {

throw std::logic\_error("The entered coordinates of the vertices do not belong to the rectangle.");

} else {

vertices[0] = a;

vertices[1] = b;

vertices[2] = c;

vertices[3] = d;

}

}

~Rectangle() override {

delete [] vertices;

vertices = nullptr;

}

Vertex Center() const override {

return Get\_Center(vertices,4);

}

double Area() const override {

auto AB = dist(vertices[0], vertices[1]);

auto AD = dist(vertices[0],vertices[3]);

return AD\*AB;

}

std::ostream &Print(std::ostream &out) const override{

out << "Id: " << Id << "\n";

out << "Figure: Rectangle\n";

out << "Coords:\n";

for (int i = 0; i < 4; i++) {

out << vertices[i] << "\n";

}

return out;

}

void Serialize(std::ofstream &os) const override{

FigureType type = Rec;

os.write((char \*) &type, sizeof(type));

os.write((char \*) &Id, sizeof(Id));

for (int i = 0; i < 4; i++) {

os.write((char\*) &(vertices[i].first), sizeof(vertices[i].first));

os.write((char\*) &(vertices[i].second), sizeof(vertices[i].second));

}

}

void Deserialize(std::ifstream &is) override {

is.read((char \*) &Id, sizeof(Id));

for (int i = 0; i<4; i++) {

is.read((char \*) &(vertices[i].first), sizeof(vertices[i].first));

is.read((char \*) &(vertices[i].second), sizeof(vertices[i].second));

}

}

int getId() const override {

return Id;

}

};

class Trapezoid : public Figure {

int Id;

Vertex \*vertices;

public:

Trapezoid() : Id{0}, vertices{new Vertex[4]} {

for (int i = 0; i < 4; i++){

vertices[i] = std::make\_pair(0,0);

}

}

Trapezoid(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :

Id{id}, vertices{new Vertex[4]} {

auto AB = dist(a, b);

auto AD = dist(a, d);

if (a == b || a == c || b == c || a == d || b == d || c == d || oneline(a,b,c) ||

oneline(a,b,d) || oneline(b,c,d) ||

collinear(a, b, c, a) || collinear(a, b, d, a) || collinear(a, c, d, a)

|| collinear(b, c, d, b)) {

throw std::logic\_error("The entered coordinates of the vertices do not belong to the trapezoid.");

} else {

vertices[0] = a;

vertices[1] = b;

vertices[2] = c;

vertices[3] = d;

}

}

~Trapezoid() override {

delete [] vertices;

vertices = nullptr;

}

Vertex Center() const override {

return Get\_Center(vertices,4);

}

double Area() const override {

double x1,x2,x3,x4, y1,y2,y3,y4;

x1 = vertices[0].first; y1 = vertices[0].second;

x2 = vertices[1].first; y2 = vertices[1].second;

x3 = vertices[2].first; y3 = vertices[2].second;

x4 = vertices[3].first; y4 = vertices[3].second;

auto area = ( (x1\*y2-x2\*y1)+(x2\*y3-x3\*y2)+(x3\*y4-x4\*y3) ) / 2;

return std::abs(area);

}

std::ostream &Print(std::ostream &out) const override{

out << "Id: " << Id << "\n";

out << "Figure: Trapezoid\n";

out << "Coords:\n";

for (int i = 0; i < 4; i++) {

out << vertices[i] << "\n";

}

return out;

}

void Serialize(std::ofstream &os) const override{

FigureType type = Trapeze;

os.write((char \*) &type, sizeof(type));

os.write((char \*) &Id, sizeof(Id));

for (int i = 0; i < 4; i++) {

os.write((char \*) &(vertices[i].first),sizeof(vertices[i].first));

os.write((char \*) &(vertices[i].second),sizeof(vertices[i].second));

}

}

void Deserialize(std::ifstream &is) override {

is.read((char \*) &Id, sizeof(Id));

for (int i = 0; i < 4; i++) {

is.read((char \*) &(vertices[i].first),sizeof(vertices[i].first));

is.read((char \*) &(vertices[i].second),sizeof(vertices[i].second));

}

}

int getId() const override {

return Id;

}

};

class Rhombus: public Figure {

int Id;

Vertex \*vertices;

public:

Rhombus() : Id{0}, vertices{new Vertex[4]} {

for (int i = 0; i < 4; i++){

vertices[i] = std::make\_pair(0,0);

}

}

Rhombus(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :

Id{id}, vertices{new Vertex[4]} {

auto AB = dist(a, b);

auto AD = dist(a, d);

auto BC = dist(b,c);

auto CD = dist(c,d);

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

!(AB == AD) || !(CD == BC) || !(AB == CD) ||

oneline(a,b,c) || oneline(a,b,d) || oneline(b,c,d) ||

!perpendic(a,b,c,d) || !collinear(a, b, c, d)

|| !collinear(a, d, c, b) ) {

throw std::logic\_error("The entered coordinates of the vertices do not belong to the rhombus.");

} else {

vertices[0] = a;

vertices[1] = b;

vertices[2] = c;

vertices[3] = d;

}

}

~Rhombus() override {

delete [] vertices;

vertices = nullptr;

}

Vertex Center() const override {

return Get\_Center(vertices,4);

}

double Area() const override {

double AC = dist(vertices[0],vertices[2]);

double DB = dist(vertices[3],vertices[1]);

return (AC\*DB) /2;

}

std::ostream &Print(std::ostream &out) const override{

out << "Id: " << Id << "\n";

out << "Figure: Trapezoid\n";

out << "Coords:\n";

for (int i = 0; i < 4; i++) {

out << vertices[i] << "\n";

}

return out;

}

void Serialize(std::ofstream &os) const override{

FigureType type = Rhomb;

os.write((char \*) &type, sizeof(type));

os.write((char \*) &Id, sizeof(Id));

for (int i = 0; i < 4; i++) {

os.write((char \*) &(vertices[i].first),sizeof(vertices[i].first));

os.write((char \*) &(vertices[i].second),sizeof(vertices[i].second));

}

}

void Deserialize(std::ifstream &is) override {

is.read((char \*) &Id, sizeof(Id));

for (int i = 0; i < 4; i++) {

is.read((char \*) &(vertices[i].first),

sizeof(vertices[i].first));

is.read((char \*) &(vertices[i].second),

sizeof(vertices[i].second));

}

}

int getId() const override {

return Id;

}

};

#endif //FIGURES\_H

1. **Выводы:**

Познакомился с потоками и их блокировкой с помощью мьютекса. Научился работать с многопоточностью в C++ с асинхронной обработкой данных.

**СПИСОК ЛИТЕРАТУРЫ**

1. std::thread [электронный ресурс]: URL <https://ru.cppreference.com/w/cpp/thread/thread>
2. Потоки, блокировки и условные переменные в C++11 [электронный ресурс]. URL: <https://habr.com/ru/post/182610/>