Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 8**

**Тема: Асинхронное программирование**

|  |  |
| --- | --- |
| Cтудент: | Королев И.М. |
| Группа: | 8О-208Б |
| Преподаватель: | Чернышов Л.Н. |
| Дата: |  |
| Оценка: |  |

Москва 2020

Оглавление

[1. Постановка задачи 3](#_Toc60000773)

[2. Описание программы 5](#_Toc60000774)

[3. Тесты и результаты их выполнения 8](#_Toc60000781)

[5. Листинг программы 11](#_Toc60000782)

[6. Выводы 18](#_Toc60000783)

[Список используемых источников 19](#_Toc60000784)

# 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 введённым данным фигур;
3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задаётся параметром командной строки. Например, для буфера размеров 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:
7. Вывод информации о фигурах в буфере на экран;
8. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.
9. Оба обработчика должны обрабатывать каждый введённый буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл;
10. Обработчики должны быть реализованы в виде лямбда-функций и должны храниться в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике;
11. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
12. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик;
13. 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 11:** прямоугольник, трапеция, ромб

# 2. Описание программы

Программа выполняет чтение данных из консоли или файла (с помощью cat | ./исполняемый файл) и выполняет необходимые действия с заданными фигурами с помощью их параметров.

Программа осуществляет чтение фигур из стандартного ввода фигур. После того, как данные фигуры были прочитаны, она создаёт фигуру заданного класса. Программа содержит внутренний буфер, в который помещаются созданные фигуры. Размер буфера задаётся параметром командной строки. После того, как буфер был заполнен, над фигурами в буфере происходит асинхронная обработка, после чего буфер очищается. Обработка происходит в отдельном потоке. Программа имеет два потока (thread): основной (main); поток для обработчиков. Для обработки данных буфера были созданы два обработчика:

1. Обработчик вывода информации о фигурах на экран;
2. Обработчик вывода информации о фигурах, которые находятся в буфере, в файл. Для каждого буфера создаётся файл с уникальным именем.

Эти обработчики реализованы с помощью лямбда-функций и хранятся в специальном массиве обработчиков, откуда и последовательно вызываются в потоке-обработчике. При заполнении буфера выполняются два обработчика. Второй обработчик открывает файл для записи и записывает в него информацию о фигурах из заполненного буфера. После выполнения этих двух обработчиков буфер очищается. В программе есть шаблон Publish-Subscribe. Каждый обработчик реализован как отдельный подписчик. После отправки буфера на обработку основной поток ждёт пока поток-обработчик не выведет все данные на экран и не запишет их в файл.

Был создан абстрактный класс *Figure*, от которого наследуются классы фигур *Rectangle*, *Trapeze*, *Rhombus*.

Создание графических примитивов было вынесено в отдельный класс *Factory*. С помощью него можно создавать фигуры по введённым данным.

# Класс Factory

От класса *Factory* наследуются классы *Rectangle*, *Trapeze*, *Rhombus*.

*static std::shared\_ptr<Figure> CreateFigure()* – функция создания фигуры. Возвращает указатель на созданную фигуру.

*static std::shared\_ptr<Figure> Read()* – функция чтения фигуры из файла. Возвращает указатель на новую созданную фигуру.

# Класс Rectangle

*std::pair<T,T> Center* – координаты геометрического центра прямоугольника.

*T Length* – длина прямоугольника.

*T Width* – ширина прямоугольника.

*void Print(std::out &out)* – функция вывода информации о прямоугольнике.

*void Write(std::ofstream &file)* – записи прямоугольника в файл.

# Класс Trapeze

*std::pair<T,T> Center* – координаты геометрического центра трапеции.

*T UpperSide* – верхнее основание трапеции.

*T DownSide* – нижнее основание трапеции.

*T Height* – высота трапеции.

*void Print(std::out &out)* – функция вывода информации о трапеции.

*void Write(std::ofstream &file)* – записи трапеции в файл.

# Класс Rhombus

*std::pair<T,T> Center* – координаты геометрического центра ромба.

*T Side* – сторона ромба.

*float Angle* – один из углов ромба.

*void Print(std::out &out)* – функция вывода информации о трапеции.

*void Write(std::ofstream &file)* – записи трапеции в файл.

# PublishSubscribe

*Std::queue<T> QueueOfMessages* – очередь добавленных фигур. Нужна для печати фигур на экран и их записи в файл в нужном порядке.

Std::mutex QueueMutex – мьютекс. Нужен для блокировки потока для работы с очередью, чтобы в этот момент, другие потоки не имели к ней доступ. И после выполнения действий над очередью, происходит разблокировка для того, чтобы другие потоки имели доступ к очереди.

# Описание функций

*void ThreadFunction()* – содержит в себе лямбда-выражения в виде печати фигуры на экран и печати фигуры в файл. Функция предназначена для корректной работы потока-обработчика. Её главной задачей является печать фигур на экран и в файл. Выполняется, когда буфер полностью заполнен или, когда программа завершается, а фигуры, которые были добавлены, ещё не выведены на экран и в файл.

# 3. Тесты и результаты их выполнения

Были проведены тесты. Тесты представляют собой введённые в консоль и результаты выполнения части программы после каждого введённого действия. В качестве данных подаются параметры фигур, которые необходимо добавить.

Также при запуске каждого теста в качестве параметра командной строки вводилось количество фигур, которое может быть помещено в буфер, т.е. размер буфера. При после окончания каждого теста в файлы были помещены такие же сведения о фигурах, которые выводились на экран.

Введённая команда: ./oop\_exercise\_08 3

Таблица 1 – тест 1 и результаты его выполнения

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 1 0 0 43 45 | Figure rectangle has been added |
| 2 0 0 43 45 56 | Figure trapeze has been added |
| 3 0 0 43 90 | Figure rhombus has been added  Rectangle: {(21.5, 22.5), (-21.5, 22.5), (-21.5, -22.5), (21.5, -22.5)} Trapeze: {(21.5, -28), (22.5, 28), (-22.5, 28),  (-21.5, -28)} Rhombus: {(30.4056, 0), (0, 30.4056), (-30.4056, 0),  (0, -30.4056)} |
| 2 10 -1 100 50 43 | Figure trapeze has been added |
| 1 0 0 10 10 | Figure rectangle has been added |
| ^D | Trapeze: {(60, -22.5), (35, 20.5), (-15, 20.5),  (-40, -22.5)} Rectangle: {(5, 5), (-5, 5), (-5, -5), (5, -5)} |

Было создано 2 файла с фигурами.

Введённая команда: ./oop\_exercise\_08 5

Таблица 2 – тест 2 и результаты его выполнения

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 1 0 0 10 5 | Figure rectangle has been added |
| 2 0 0 10 100 40 | Figure trapeze has been added |
| 3 0 0 32 90 | Figure rhombus has been added |
| 2 9 5 1 10 3 | Figure trapeze has been added |
| 1 0 -1 90 43 | Figure rectangle has been added Rectangle: {(5, 2.5), (-5, 2.5), (-5, -2.5), (5, -2.5)} Trapeze: {(5, -20), (50, 20), (-50, 20), (-5, -20)} Rhombus: {(22.6274, 0), (0, 22.6274), (-22.6274, 0),  (0, -22.6274)} Trapeze: {(9.5, 3.5), (14, 6.5), (4, 6.5), (8.5, 3.5)} Rectangle: {(45, 20.5), (-45, 20.5), (-45, -22.5),  (45, -22.5)} |
| 3 0 0 100 90 | Figure rhombus has been added |
| 1 0 -10 32 34 | Figure rectangle has been added |
| 1 0 100 32 90 | Figure rectangle has been added |
| 1 54 776 3 5 | Figure rectangle has been added |
| 2 0 0 43 45 100 | Figure trapeze has been added Rhombus: {(70.7107, 0), (0, 70.7107), (-70.7107, 0),  (0, -70.7107)} Rectangle: {(16, 7), (-16, 7), (-16, -27), (16, -27)} Rectangle: {(16, 145), (-16, 145), (-16, 55), (16, 55)} Rectangle: {(55.5, 778.5), (52.5, 778.5), (52.5, 773.5), (55.5, 773.5)} Trapeze: {(21.5, -50), (22.5, 50), (-22.5, 50),  (-21.5, -50)} |
| 3 0 0 2 60 | Figure rhombus has been added |
| ^D | Rhombus: {(1.73205, 0), (0, 1), (-1.73205, 0), (0, -1)} |

Было создано 3 файла с фигурами.

Введённая команда: ./oop\_exercise\_08 2

Таблица 3 – test\_03.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 1 0 0 34 90 | Figure rectangle has been added |
| 1 0 0 34 56 | Figure rectangle has been added Rectangle: {(17, 45), (-17, 45), (-17, -45), (17, -45)} Rectangle: {(17, 28), (-17, 28), (-17, -28), (17, -28)} |
| 2 0 10 100 150 45 | Figure trapeze has been added |
| 3 0 0 32 50 | Figure rhombus has been added Trapeze: {(50, -12.5), (75, 32.5), (-75, 32.5),  (-50, -12.5)} Rhombus: {(29.0019, 0), (0, 13.5238), (-29.0019, 0),  (0, -13.5238)} |
| 1 0 0 65 78 | Figure rectangle has been added |
| 3 10 -10 56 90 | Figure rhombus has been added Rectangle: {(32.5, 39), (-32.5, 39), (-32.5, -39),  (32.5, -39)} Rhombus: {(49.598, 0), (0, 29.598), (-29.598, 0),  (0, -49.598)} |
| 2 0 0 21 32 70 | Figure trapeze has been added |
| 1 90 -10 50 32 | Figure rectangle has been added Trapeze: {(10.5, -35), (16, 35), (-16, 35), (-10.5, -35)} Rectangle: {(115, 6), (65, 6), (65, -26), (115, -26)} |
| 3 100 -60 43 60 | Figure rhombus has been added |
| 2 0 8 32 45 10 | Figure trapeze has been added Rhombus: {(137.239, 0), (0, -38.5), (62.7609, 0),  (0, -81.5)} Trapeze: {(16, 3), (22.5, 13), (-22.5, 13), (-16, 3)} |
| 2 10 80 100 150 90 | Figure trapeze has been added |
| 2 5 6 3 4 5 | Figure trapeze has been added Trapeze: {(60, 35), (85, 125), (-65, 125), (-40, 35)} Trapeze: {(6.5, 3.5), (7, 8.5), (3, 8.5), (3.5, 3.5)} |
| ^D | Программа завершилась |

Было создано 6 файлов с фигурами.

# 5. Листинг программы

**CMakeLists.txt**

cmake\_minimum\_required (VERSION 3.2)

set(CMAKE\_CXX\_FLAGS "-g -pedantic -pthread -Wall -Werror")

set(CMAKE\_CXX\_STANDART 17)

project (OOP)

add\_executable (oop\_exercise\_08 main.cpp)

**figure.hpp**

#pragma once

#include <iostream>

#include <fstream>

class Figure {

public:

    virtual void Print() = 0;

    virtual void Write(std::ofstream &file) = 0;

    virtual ~Figure() = default;

};

**factory.hpp**

#pragma once

#include <memory>

#include "figure.hpp"

#include "rectangle.hpp"

#include "rhombus.hpp"

#include "trapeze.hpp"

// Class for creating graphical primitives

template <class T, class FIGURE>

class Factory;

template <class T>

class Factory<T, Rectangle<T>> {

public:

    static std::shared\_ptr<Figure> CreateFigure() {

        std::pair<T,T> center;

        T length;

        T width;

        std::cout << "Enter the center coordinates: ";

        std::cin >> center.first >> center.second;

        std::cout << "Enter lenght and width: ";

        std::cin >> length >> width;

        Rectangle<T> \*rectangle = new Rectangle<T>(center, length, width);

        return std::shared\_ptr<Figure>(rectangle);

    }

};

template <class T>

class Factory<T, Rhombus<T>> {

public:

    static std::shared\_ptr<Figure> CreateFigure() {

        std::pair<T,T> center;

        T side;

        float angle;

        std::cout << "Enter the center coordinates: ";

        std::cin >> center.first >> center.second;

        std::cout << "Enter the side: ";

        std::cin >> side;

        std::cout << "Enter an angle: ";

        std::cin >> angle;

        Rhombus<T> \*rhombus = new Rhombus<T>(center, side, angle);

        return std::shared\_ptr<Figure>(rhombus);

    }

};

template <class T>

class Factory<T, Trapeze<T>> {

public:

    static std::shared\_ptr<Figure> CreateFigure() {

        std::pair<T,T> center;

        T down\_side;

        T upper\_side;

        T height;

        std::cout << "Enter the center coordinates: ";

        std::cin >> center.first >> center.second;

        std::cout << "Enter down side, upper side and height: ";

        std::cin >> down\_side >> upper\_side;

        std::cin >> height;

        Trapeze<T> \*trapeze = new Trapeze<T>(center, down\_side, upper\_side,

height);

        return std::shared\_ptr<Figure>(trapeze);

    }

};

**rectangle.hpp**

#pragma once

#include <iostream>

#include "figure.hpp"

const unsigned int RECTANGLE\_ID\_TYPE = 1;

template <class T>

class Rectangle: public Figure {

private:

    std::pair<T,T> Center;

    T Length;

    T Width;

public:

    Rectangle() {}

    Rectangle(std::pair<T,T> center, T length, T width): Center(center),

Length(length), Width(width) {}

    // Print rectangle’s parameters

    void Print() override {

        std::cout << \*this;

    }

    // Write the rectangle in file

    void Write(std::ofstream &file) {

        file << \*this;

    }

    template <class U>

    friend std::ostream &operator << (std::ostream &out, const Rectangle<U> &rectangle) {

        out << "Rectangle: ";

        out << "{(" << rectangle.Center.first + rectangle.Length / 2.0 << ", "

<< rectangle.Center.second + rectangle.Width / 2.0 <<"), ";

        out << "(" << rectangle.Center.first - rectangle.Length / 2.0 << ", " << rectangle.Center.second + rectangle.Width

/ 2.0 << "), ";

        out << "(" << rectangle.Center.first - rectangle.Length / 2.0 << ", " << rectangle.Center.second - rectangle.Width / 2.0 << "), ";

        out << "(" << rectangle.Center.first + rectangle.Length / 2.0 << ", "

<< rectangle.Center.second - rectangle.Width / 2.0 << ")}\n";

        return out;

    }

};

**trapeze.hpp**

#pragma once

#include <iostream>

#include "figure.hpp"

const unsigned int TRAPEZE\_ID\_TYPE = 2;

template <class T>

class Trapeze: public Figure {

private:

    std::pair<T,T> Center;

    T DownSide;

    T UpperSide;

    T Height;

public:

    Trapeze() {}

    Trapeze(std::pair<T,T> center, T down\_side, T upper\_side, T height):

Center(center), DownSide(down\_side), UpperSide(upper\_side), Height(height) {}

    // Print trapeze parameters

    void Print() override {

        std::cout << \*this;

    }

    // Write trapeze in file

    void Write(std::ofstream &file) override {

        file << \*this;

    }

    template <class U>

    friend std::ostream &operator<< (std::ostream &out, const Trapeze<U>

&trapeze) {

        out << "Trapeze: ";

        out << "{(" << trapeze.Center.first + trapeze.DownSide / 2.0 << ", " << trapeze.Center.second - trapeze.Height / 2.0 <<"), ";

        out << "(" << trapeze.Center.first + trapeze.UpperSide / 2.0 << ", " << trapeze.Center.second + trapeze.Height / 2.0 <<"), ";

        out << "(" << trapeze.Center.first - trapeze.UpperSide / 2.0 << ", " << trapeze.Center.second + trapeze.Height / 2.0 <<"), ";

        out << "(" << trapeze.Center.first - trapeze.DownSide / 2.0 << ", " << trapeze.Center.second - trapeze.Height / 2.0 <<")}\n";

        return out;

    }

};

**rhombus.hpp**

#pragma once

#include <iostream>

#include <cmath>

#include "figure.hpp"

#define PI 3.14159265

const unsigned int RHOMBUS\_ID\_TYPE = 3;

template <class T>

class Rhombus: public Figure {

private:

    std::pair<T,T> Center;

    T Side;

    float Angle;

public:

    Rhombus(): Center(), Side(), Angle(90) {}

    Rhombus(std::pair<T,T> center, T side, float angle): Center(center),

Side(side), Angle(angle) {}

    void Print() override {

        std::cout << \*this;

    }

    void Write(std::ofstream &file) override {

        file << \*this;

    }

    template <class U>

    friend std::ostream &operator << (std::ostream &out, const Rhombus<U>

&rhombus) {

        float angle = rhombus.Angle / 2;

        float radius\_x = cos(angle \* PI / 180) \* rhombus.Side;

        float radius\_y = sin(angle \* PI / 180) \* rhombus.Side;

        out << "Rhombus: ";

        out << "{";

        for (int i = 0; i < 4; i++) {

            if (i % 2 == 0) {

                out << "(" << rhombus.Center.first + radius\_x << ", " << 0

<< "), ";

                radius\_x \*= -1;

            }

            else {

                out << "(" << 0 << ", " << rhombus.Center.second + radius\_y

<< ")";

                if (i != 3) {

                    out << ", ";

                }

                radius\_y \*= -1;

            }

        }

        out << "}\n";

        return out;

    }

};

**publish-subscribe.hpp**

#pragma once

#include <mutex>

#include <queue>

template <class T>

class PublishSubscribe {

private:

    std::queue<T> QueueOfMessages;

    std::mutex QueueMutex; // For lock and unlock the queue

public:

    explicit PublishSubscribe() noexcept: QueueOfMessages(), QueueMutex() {}

    bool Empty() {

        QueueMutex.lock();

        bool result = QueueOfMessages.empty();

        QueueMutex.unlock();

        return result;

    }

    T Front() {

        QueueMutex.lock();

        T top\_element = QueueOfMessages.front();

        QueueMutex.unlock();

        return top\_element;

    }

    void Push(const T &message) {

        QueueMutex.lock();

        QueueOfMessages.push(message);

        QueueMutex.unlock();

    }

    void Pop() {

        QueueMutex.lock();

        QueueOfMessages.pop();

        QueueMutex.unlock();

    }

    ~PublishSubscribe() {}

};

**main.cpp**

#include <iostream>

#include <thread>

#include <vector>

#include <functional>

#include "factory.hpp"

#include "figure.hpp"

#include "publish-subscribe.hpp"

// Queue of figures to print in console and file

PublishSubscribe<std::vector<std::shared\_ptr<Figure>>> message\_queue;

const std::string FILE\_NAME = "result\_file";

int number\_file = 1; // To number each file for each buffer

std::ofstream file;

// Function for outputting figures to the console and to a file after filling

// the buffer

void ThreadFunction() {

    using function = std::function<void(std::shared\_ptr<Figure> figure)>;

    function PrintOut = [](std::shared\_ptr<Figure> figure) {

        figure->Print();

    };

    function WriteInFile = [](std::shared\_ptr<Figure> figure) {

        figure->Write(file);

    };

    std::vector<function> functions({PrintOut, WriteInFile});

    bool message\_not\_empty = true;

    while (message\_not\_empty) {

        if (!message\_queue.Empty()) {

            std::vector<std::shared\_ptr<Figure>> message =

message\_queue.Front();

            if (message.empty()) {

                message\_not\_empty = false;

                break;

            }

            file.open(FILE\_NAME + std::to\_string(number\_file));

            while (!message.empty()) {

                std::shared\_ptr<Figure> figure\_pointer = message.front();

                message.erase(message.begin());

                for (auto function: functions) {

                    function(figure\_pointer);

                }

            }

            file.close();

            number\_file++;

            message\_queue.Pop();

        }

    }

}

int main(int argc, char\*\* argv) {

    if (argc < 2) {

        std::cout << "No enough parameters!\n";

        return -1;

    }

    size\_t buffer\_size;

    try {

        buffer\_size = std::stoi(std::string(argv[1])); // Conversation the parameter to integer

    }

    catch (std::exception &exeption) {

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

        return -1;

    }

    std::thread thread(ThreadFunction);

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

    unsigned short figure\_type;

    std::cout << "Number of figure type (1 - Rectangle; 2 - Trapeze;

3 - Rhombus)\n";

    while (std::cin >> figure\_type) {

        if (figure\_type == RECTANGLE\_ID\_TYPE) {

            figures.push\_back(Factory<int, Rectangle<int>>::CreateFigure());

            std::cout << "Figure rectangle has been added\n";

        } else if (figure\_type == TRAPEZE\_ID\_TYPE) {

            figures.push\_back(Factory<int, Trapeze<int>>::CreateFigure());

            std::cout << "Figure trapeze has been added\n";

        } else if (figure\_type == RHOMBUS\_ID\_TYPE) {

            figures.push\_back(Factory<int, Rhombus<int>>::CreateFigure());

            std::cout << "Figure rhombus has been added\n";

        } else {

            std::cout << "Entered invalid type of figure!\n\n";

            continue;

        }

        if (figures.size() == buffer\_size) { // Running the handler thread

            message\_queue.Push(figures);

            bool thread\_work\_done = false;

            while (!thread\_work\_done) {

                if (message\_queue.Empty()) {

                    thread\_work\_done = true;

                }

            }

            figures.clear();

        }

        std::cout << "\nNumber of figure type (1 - Rectangle; 2 - Trapeze;

3 - Rhombus)\n";

    }

    message\_queue.Push(figures);

    figures.clear();

    message\_queue.Push(figures);

    figures.clear();

    thread.join();

    return 0;

}

Ссылка на github репозиторий: <https://github.com/HarryLiker/oop_exercise_08>

# 6. Выводы

Была написана программа, в которой реализованы шаблонные классы прямоугольник, трапеция и ромб. Был спроектирован простейший «графический» векторный редактор.

Была написана программа, в которой реализована обработка введённых фигур с помощью двух потоков: основного и потока-обработчика. Программа получает на вход параметры фигуры, создаёт фигуру по этим параметрам и посылает фигуру в буфер. После того, как буфер становится полностью заполненным, фигуры из буфера с помощью потока-обработчика выводятся на экран и записываются в файл. В потоке-обработчике реализованы два лямбда-выражения, которые выполняют запись информации о фигурах на экран и в файл. Если выполняется завершение программы, а буфер заполнен, но не полностью, то поток-обработчик всё равно выполняет вывод информации о фигурах на экран и в файл с помощью функции join(), после чего, буфер очищается, и программа завершается.

Выполняя лабораторную работу, были получены навыки работы с несколькими потоками. Были получены знания о том, как работать с std::thread и std::mutex. std::thread позволяет нескольким фрагментам кода работать асинхронно и одновременно. Были получены навыки работы с потоками, их взаимодействия друг с другом. std::mutex позволяет выполнять один поток, блокируя остальные. Это позволяет корректно обрабатывать информацию. Также были получены навыки работы с умными указателями std::shared\_ptr, написания лямбда-выражений, записи информации в файл в виде строк, преобразования параметров, введённых при запуске программы, в целочисленный тип.

# Список используемых источников

1. Р. Лафоре «Объектно-ориентированное программирование в C++» – Москва: «Питер», 2004.

2. Информация об умном указателе std::shared\_ptr – [Электронный ресурс]. – URL: <https://ravesli.com/urok-194-std-shared_ptr/> (дата обращения: 25.12.2020).

3. Информация о перегрузке операторов ввода /вывода – [Электронный ресурс]. – URL: <https://ravesli.com/urok-133-peregruzka-operatorov-vvoda-i-vyvoda/>

(дата обращения: 25.12.2020).

4. Информация о шаблоне std::pair – [Электронный ресурс]. – URL: <https://en.cppreference.com/w/cpp/utility/pair> (дата обращения: 25.12.2020).

5. Информация о функции std::stoi() – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/string/basic_string/stol> (дата обращения: 26.12.2020).

6. Информация об обработке исключений – [Электронный ресурс]. – URL: <https://ravesli.com/urok-182-obrabotka-isklyuchenij/> (дата обращения: 26.12.2020).

7. Информация о записи в файл – [Электронный ресурс]. – URL: <https://metanit.com/cpp/tutorial/8.3.php> (дата обращения: 26.12.2020).

8. Информация о классе std::thread – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/thread/thread> (дата обращения: 26.12.2020).

9. Информация о классе std::mutex – [Электронный ресурс]. – URL: <https://en.cppreference.com/w/cpp/thread/mutex> (дата обращения: 26.12.2020).

10. Информация о классе std::mutex – [Электронный ресурс]. – URL: <https://medium.com/nuances-of-programming/c-мьютекс-пишем-наш-первый-код-для-многопоточной-среды-543a3d60ef30> (дата обращения: 26.12.2020).

11. Информация о классе std::mutex – [Электронный ресурс]. – URL: <https://medium.com/nuances-of-programming/c-мьютекс-пишем-наш-первый-код-для-многопоточной-среды-543a3d60ef30> (дата обращения: 26.12.2020).

12. Информация о функции join() – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/thread/thread/join> (дата обращения: 26.12.2020).