**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: Волков М. А.

Группа: М80-207Б-19

Преподаватель: Чернышев Л. Н.

Дата:

Оценка:

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:
   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.
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. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.
9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
11. 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: Прямоугольник, трапеция, ромб*

1. **Описание программы**

Задание разбито на несколько файлов: *main.cpp, document.h, factory.h, Rectangle.h, Rhombus.h, Trapeze.h, Figures.h, event.h, event-loop.h*

Файлы *Rectangle.h, Rhombus.h, Trapeze.h, Figures.h* были взяты из лабораторной работы 7 и были доработаны. Я сделал так потому, что распределение вариантов предполагает дорабатывать свою предыдущую программу. Я добавил файлы *event.h* и *event-loop.h.* В них сделана сложная обертка над очередью запросов. Сложность обосновывается тем, что данный код следует принципам универсальности. Также были в этих файлах написаны классы, в которых прослеживается шаблон Publish-Subscribe.

1. **Наборы и результаты выполнения тестов**

*Тест 1: размер буфера - 2*

> + 1 1 1 1 1 1

Enter rhombus as follows: x y d1 d2

x, y is a left bottom point cords

d1 and d2 are diagonals of rhombus

> + 1 1

Enter rhombus as follows: x y d1 d2

x, y is a left bottom point cords

d1 and d2 are diagonals of rhombus

1 1 1 1

..Rhombus vertices: [ (1, 1), (1.5, 1.5), (2, 1), (0.5, 0.5) ]

Rhombus vertices: [ (1, 1), (1.5, 1.5), (2, 1), (0.5, 0.5) ]

> + 1 1 2 2 2 2

Enter rhombus as follows: x y d1 d2

x, y is a left bottom point cords

d1 and d2 are diagonals of rhombus

> + 1 1

Enter rhombus as follows: x y d1 d2

x, y is a left bottom point cords

d1 and d2 are diagonals of rhombus

2 2 2 2

..Rhombus vertices: [ (2, 2), (3, 3), (4, 2), (1, 1) ]

Rhombus vertices: [ (2, 2), (3, 3), (4, 2), (1, 1) ]

> e

Process finished with exit code 0

*Тест 2: размер буфера - 0*

terminate called after throwing an instance of 'std::invalid\_argument'

what(): buffer\_size is less or equal than zero

Process finished with exit code 3

1. **Листинг программы**

**main.cpp**

/\*  
 \* Волков Матвей Андреевич  
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. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков.  
 Откуда и должны последовательно вызываться в потоке – обработчике.  
9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;  
10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как  
 отдельный подписчик.  
11. 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: Прямоугольник, трапеция, ромб  
\*/  
  
#include "event\_loop.h"  
#include <exception>  
#include <functional>  
//нужно, чтобы было 2 одинаковых типа  
#define yourTYPE double , double  
#define figTYPE <double>  
  
  
  
int main(int argc, char\* argv[]) {  
 try{  
 if(argc != 2){  
 throw std::invalid\_argument("Too many or too low arguments");  
 }  
 }  
 catch (std::invalid\_argument& er){  
 std::terminate();  
 }  
  
 int buffer\_size = std::stoi(argv[1]);  
 try{  
 if(buffer\_size <= 0){  
 throw std::invalid\_argument("buffer\_size is less or equal than zero");  
 }  
 }  
 catch (std::invalid\_argument& er){  
 std::terminate();  
 }  
  
 TDocument<yourTYPE> doc;  
 Event\_loop figTYPE eventLoop;  
 std::shared\_ptr<Handler figTYPE> handler\_printer(new Handler\_Printer figTYPE);  
 std::shared\_ptr<Handler figTYPE> handler\_saver(new Handler\_Saver figTYPE);  
 eventLoop.addHandler(eventType::print, handler\_printer);  
 eventLoop.addHandler(eventType::save, handler\_saver);  
  
 std::thread threadHandler(std::ref(eventLoop));  
 std::string s;  
 while ((std::cout << "> ") && (std::cin >> s)) {  
 if (s == "+") {  
 size\_t pos;  
 unsigned short type;  
 std::cin >> pos >> type;  
 doc.Add(pos, type);  
 }  
 else if (s == "-") {  
 size\_t pos;  
 std::cin >> pos;  
 doc.Delete(pos);  
 }  
 else if (s == "p") {  
 std::cout << "Printing document:" << std::endl;  
 std::cout << doc;  
 }  
 else if (s == "u") {  
 doc.Undo();  
 }  
 else if (s == "h") {  
 std::cout << "> \'+\' - add a figure\n"  
 "when you add figure:\nfirst - enter your position you want insert your figure\n"  
 "second - enter figure ID:\n"  
 " 1 - rhombus\n 2 - rectangle\n 3 - trapeze" << std::endl;  
 std::cout << "> \'-\' - remove a figure\n"  
 "when you remove figure, enter your position you want to delete" << std::endl;  
 std::cout << "> \'p\' - print document" << std::endl;  
 std::cout << "> \'u\' - undo changes" << std::endl;  
 std::cout << "> \'h\' - show this message" << std::endl;  
 std::cout << "> \'e\' - exit\n" << std::endl;  
 }  
  
 else if(s == "e"){  
 Event figTYPE ev (eventType::exit,  
 std::make\_shared<Event\_data>(),  
 std::make\_shared<Event\_Response>(),  
 [](auto){});  
 eventLoop.addEvent(ev);  
 break;  
 }  
 else {  
 std::cout << "Unknown command. Type \'h\' to show help" << std::endl;  
 }  
  
 bool printer\_done = false;  
 bool save\_done = false;  
 if(doc.getFigs().size() >= buffer\_size){  
 auto do\_lam\_ev1 = [](const std::shared\_ptr<Event\_Response>& response){  
 if(auto ptr = std::static\_pointer\_cast<Event\_Response\_Printer>(response)){  
 ptr->done = true;  
 }  
 };  
 auto do\_lam\_ev2 = [](const std::shared\_ptr<Event\_Response>& response){  
 if(auto ptr = std::static\_pointer\_cast<Event\_Response\_Saver>(response)){  
 ptr->done = true;  
 }  
 };  
  
  
 Event figTYPE ev1(eventType::print,  
 std::make\_shared<Event\_data\_Printer figTYPE>(doc.getFigs()),  
 std::make\_shared<Event\_Response\_Printer>(printer\_done),  
 do\_lam\_ev1);  
 Event figTYPE ev2(eventType::save,  
 std::make\_shared<Event\_data\_Saver figTYPE>(doc.getFigs()),  
 std::make\_shared<Event\_Response\_Saver>(save\_done),  
 do\_lam\_ev2);  
  
 eventLoop.addEvent(ev1);  
 eventLoop.addEvent(ev2);  
 while(!printer\_done && !save\_done){  
 std::cout << '.' << std::flush;  
 std::this\_thread::sleep\_for(std::chrono::milliseconds(250));  
 }  
 std::cout << std::endl;  
 doc.Clear\_list();  
 }  
 }  
 threadHandler.join();  
 return 0;  
}

**document.h**

#ifndef OOP\_EXERCISE\_07\_DOCUMENT\_H  
#define OOP\_EXERCISE\_07\_DOCUMENT\_H  
  
//тип, используемый вами  
#define tempTYPE class Tpoint, class Tside  
  
//если нет типа у глобального класса фигуы, нужно закомментировать  
//если имеется ваш собственный тип, обязательно нужно писать <>  
#define FIGURE\_TYPE <Tpoint>  
  
//здесь находится применение ваших типов  
#define TYPES Tpoint, Tside

#define FIGURE\_TYPE <Tpoint>

#include <list>  
#include <stack>  
  
#include "factory.h"  
  
template<tempTYPE>  
class TDocument {  
private:  
 struct IAction;  
  
 using figure\_pointer = std::shared\_ptr<Figures FIGURE\_TYPE>;  
 using action\_pointer = std::shared\_ptr<IAction>;  
 using const\_iterator = typename std::list< figure\_pointer >::const\_iterator;  
  
 std::list< figure\_pointer > FiguresList;  
 std::stack< action\_pointer > ActionStack;  
  
 struct IAction {  
 virtual void PerformAction(TDocument & fact) = 0;  
 virtual ~IAction() {}  
 };  
  
 class TDeleteAction : public IAction {  
 private:  
 size\_t DeletePos;  
 public:  
 TDeleteAction(const size\_t & pos) : DeletePos(pos) {}  
 void PerformAction(TDocument & fact) override {  
 fact.Delete(DeletePos);  
 }  
 };  
  
 class TAddAction : public IAction {  
 private:  
 size\_t AddPos;  
 figure\_pointer AddFigure;  
 public:  
 TAddAction(const size\_t & pos, const figure\_pointer & fig) : AddPos(pos), AddFigure(fig) {}  
 void PerformAction(TDocument & fact) override {  
 fact.AddFigure(AddPos, AddFigure);  
 }  
 };  
  
  
public:  
 void CreateNew() {  
 while (!ActionStack.empty()) {  
 ActionStack.pop();  
 }  
 FiguresList.clear();  
 }  
  
 void LoadFromFile(std::ifstream& in) {  
 CreateNew();  
 size\_t n;  
 in.read((char\*) &(n), sizeof (size\_t));  
 for (size\_t i = 0; i < n; ++i) {  
 unsigned int type;  
 in.read((char\*) &(type), sizeof(unsigned));  
 if (type == RHOMBUS\_TYPE\_ID) {  
 FiguresList.push\_back(TFactory< TYPES, Rhombus<TYPES>>::Read(in));  
 } else if (type == RECTANGLE\_TYPE\_ID) {  
 FiguresList.push\_back(TFactory< TYPES, Rectangle<TYPES>>::Read(in));  
 } else if (type == TRAPEZE\_TYPE\_ID) {  
 FiguresList.push\_back(TFactory< TYPES, Trapeze<TYPES>>::Read(in));  
 }  
 }  
 }  
  
 void SaveToFile(std::ofstream& out) {  
 size\_t n = FiguresList.size();  
 out.write((const char\*) &n, sizeof (size\_t));  
 for (const\_iterator it = FiguresList.begin(); it != FiguresList.end(); ++it) {  
 (\*it)->Write(out);  
 }  
 }  
  
 void Add(const size\_t& pos, const unsigned int& figureID) {  
 if (figureID == RHOMBUS\_TYPE\_ID) {  
 AddFigure(pos, TFactory<TYPES, Rhombus<TYPES> >::CreateFigure());  
 } else if (figureID == RECTANGLE\_TYPE\_ID) {  
 AddFigure(pos, TFactory<TYPES, Rectangle<TYPES> >::CreateFigure());  
 } else if (figureID == TRAPEZE\_TYPE\_ID) {  
 AddFigure(pos, TFactory<TYPES, Trapeze<TYPES> >::CreateFigure());  
 }  
 }  
  
 void AddFigure(const size\_t & pos, const figure\_pointer & fig) {  
 if (pos > FiguresList.size()) {  
 FiguresList.push\_back(fig);  
 auto delAct = new TDeleteAction(FiguresList.size());  
 ActionStack.push(action\_pointer(delAct));  
 } else {  
 size\_t cur = 0;  
 const\_iterator it = FiguresList.begin();  
 while (cur < pos) {  
 ++cur;  
 ++it;  
 }  
 FiguresList.insert(it, fig);  
 auto delAct = new TDeleteAction(pos + 1);  
 ActionStack.push(action\_pointer(delAct));  
 }  
 }  
  
 void Delete(const size\_t & pos) {  
 if (FiguresList.empty()) {  
 std::cout << "Nothing to delete!" << std::endl;  
 return;  
 }  
 if (pos > FiguresList.size()) {  
 auto addAct = new TAddAction(FiguresList.size() - 1, FiguresList.back());  
 ActionStack.push(action\_pointer(addAct));  
 FiguresList.pop\_back();  
 } else {  
 size\_t cur = 1;  
 const\_iterator it = FiguresList.begin();  
 while (cur < pos) {  
 ++cur;  
 ++it;  
 }  
 auto addAct = new TAddAction(cur - 1, \*it);  
 ActionStack.push(action\_pointer(addAct));  
 FiguresList.erase(it);  
 }  
 }  
  
 void Undo() {  
 if (ActionStack.empty()) {  
 std::cout << "Nothing to undo!" << std::endl;  
 } else {  
 ActionStack.top()->PerformAction(\*this);  
 ActionStack.pop();  
 ActionStack.pop();  
 }  
 }  
  
  
 friend std::ostream & operator << (std::ostream & of, const TDocument<TYPES> & fact) {  
 TDocument::const\_iterator it = fact.FiguresList.begin();  
 for (size\_t i = 0; i < fact.FiguresList.size(); ++i) {  
 of << "[" << i + 1 << "] ";  
 (\*it)->Print(of);  
 of << std::endl;  
 ++it;  
 }  
 return of;  
 }  
};  
  
#endif //OOP\_EXERCISE\_07\_DOCUMENT\_H

**factory.h**

#ifndef OOP\_EXERCISE\_07\_FACTORY\_H  
#define OOP\_EXERCISE\_07\_FACTORY\_H  
  
#include <memory>  
  
#include "Rectangle.h"  
#include "Rhombus.h"  
#include "Trapeze.h"  
  
template<class Tpoint,class Tside, class FIGURE>  
class TFactory;  
  
  
template<class Tpoint, class Tside>  
class TFactory< Tpoint, Tside, Rectangle<Tpoint, Tside> > {  
public:  
 static std::shared\_ptr<Figures<Tpoint>> CreateFigure() {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curHeight, curWidth;  
 std::cout << "Enter rectangle as follows: x y a b" << std::endl;  
 std::cout << "x, y is a left bottom point cords" << std::endl;  
 std::cout << "a and b are width and heigth" << std::endl;  
 std::cin >> curCords.first >> curCords.second >> curHeight >> curWidth;  
 auto rect = new Rectangle<Tpoint,Tside>(curCords, curHeight, curWidth);  
 return std::shared\_ptr<Figures<Tpoint>>(rect);  
 }  
  
 static std::shared\_ptr<Figures<Tpoint>> Read(std::ifstream& file) {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curHeight, curWidth;  
 file.read((char\*) &(curCords.first), sizeof(Tpoint));  
 file.read((char\*) &(curCords.second), sizeof (Tpoint));  
 file.read((char\*) &(curHeight), sizeof (Tside));  
 file.read((char\*) &(curWidth), sizeof (Tside));  
 auto rect = new Rectangle<Tpoint,Tside>(curCords, curHeight, curWidth);  
 return std::shared\_ptr<Figures<Tpoint>>(rect);  
 }  
};  
  
template<class Tpoint, class Tside>  
class TFactory< Tpoint, Tside, Trapeze<Tpoint,Tside> > {  
public:  
 static std::shared\_ptr<Figures<Tpoint>> CreateFigure() {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curGreaterBase, curSmallerBase, curHeight;  
 std::cout << "Enter trapeze as follows: x y a b c" << std::endl  
 << "x, y is a left bottom point cords" << std::endl  
 << "a, b and c are larger, smaller base and height" << std::endl;  
 std::cin >> curCords.first >> curCords.second >> curGreaterBase >> curSmallerBase >> curHeight;  
 auto trap = new Trapeze<Tpoint,Tside>(curCords, curGreaterBase, curSmallerBase, curHeight);  
 return std::shared\_ptr<Figures<Tpoint>>(trap);  
 }  
  
 static std::shared\_ptr<Figures<Tpoint>> Read(std::ifstream& in) {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curGreaterBase, curSmallerBase, curHeight;  
 in.read((char\*) &(curCords.first), sizeof (Tpoint));  
 in.read((char\*) &(curCords.second), sizeof (Tpoint));  
 in.read((char\*) &(curGreaterBase), sizeof (Tside));  
 in.read((char\*) &(curSmallerBase), sizeof (Tside));  
 in.read((char\*) &(curHeight), sizeof (Tside));  
 auto trap = new Trapeze<Tpoint,Tside>(curCords, curGreaterBase, curSmallerBase, curHeight);  
 return std::shared\_ptr<Figures<Tpoint>>(trap);  
 }  
};  
  
  
template<class Tpoint, class Tside>  
class TFactory< Tpoint, Tside, Rhombus<Tpoint,Tside> > {  
public:  
 static std::shared\_ptr<Figures<Tpoint>> CreateFigure() {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curD1, curD2;  
 std::cout << "Enter rhombus as follows: x y d1 d2" << std::endl  
 << "x, y is a left bottom point cords" << std::endl  
 << "d1 and d2 are diagonals of rhombus" << std::endl;  
 std::cin >> curCords.first >> curCords.second >> curD1 >> curD2;  
 auto pRhombus = new Rhombus<Tpoint,Tside>(curCords, curD1, curD2);  
 return std::shared\_ptr<Figures<Tpoint>>(pRhombus);  
 }  
  
 static std::shared\_ptr<Figures<Tpoint>> Read(std::ifstream& in) {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curD1, curD2;  
 in.read((char\*) &(curCords.first), sizeof (Tpoint));  
 in.read((char\*) &(curCords.second), sizeof (Tpoint));  
 in.read((char\*) &(curD1), sizeof (Tside));  
 in.read((char\*) &(curD2), sizeof (Tside));  
 auto pRhombus = new Rhombus<Tpoint,Tside>(curCords, curD1, curD2);  
 return std::shared\_ptr<Figures<Tpoint>>(pRhombus);  
 }  
};  
  
#endif //OOP\_EXERCISE\_07\_FACTORY\_H

**Figures.h**

#ifndef OOP\_EXERSICE\_03\_FIGURES\_H  
#define OOP\_EXERSICE\_03\_FIGURES\_H  
#include <fstream>  
#include <iostream>  
#include <cmath>  
  
template<class T1,class T2>  
struct is\_int : std::false\_type {};  
template<>  
struct is\_int<int,int>:std::true\_type {};  
  
template<class T1,class T2>  
struct is\_double : std::false\_type {};  
template<>  
struct is\_double<double,double>:std::true\_type {};  
  
template<class T>  
class Figures {  
protected:  
 using coord = std::pair<T,T>;  
 coord point;  
  
public:  
 virtual void Print(std::ostream& out) = 0;  
 virtual void Write(std::ofstream& file){};  
 virtual ~Figures() = default;  
};  
  
template<class T1, class T2>  
inline std::ostream & operator<<(std::ostream& out, const std::pair<T1, T2>& pair){  
 out << "(" << pair.first << ", " << pair.second << ")";  
 return out;  
}  
  
#endif //OOP\_EXERSICE\_03\_FIGURES\_H

**Rectangle.h**

#ifndef RECTANGLE\_HPP  
#define RECTANGLE\_HPP  
#include "Figures.h"  
  
const unsigned int RECTANGLE\_TYPE\_ID = 2;  
  
template<class T1, class T2>  
class Rectangle : public Figures<T1> {  
private:  
 T2 A, B;  
public:  
 Rectangle(std::pair<T1,T2> \_point, const T2 &a, const T2 &b): A(a), B(b){// x and y are left-bottom point of figure  
 this->point.first = \_point.first;  
 this->point.second = \_point.second;  
  
 if (A < 0 or B < 0) {  
 throw std::invalid\_argument("Invalid rectangle parameters!");  
 }  
 }  
  
  
 void Print(std::ostream& out) override {  
 out << \*this << std::endl;  
 }  
  
 void Write(std::ofstream& file) override{  
 file.write((const char\*) &(RECTANGLE\_TYPE\_ID), sizeof(unsigned));  
 file.write((const char\*) &(this->point.first), sizeof (T1));  
 file.write((const char\*) &(this->point.second), sizeof(T1));  
 file.write((const char\*) &(this->A), sizeof(T2));  
 file.write((const char\*) &(this->B), sizeof (T2));  
 }  
  
 template<class U>  
 friend std::ostream & operator << (std::ostream & out, const Rectangle<U,U> & rectangle){  
 out << "Rectangle vertices: [";  
 out << std::pair<U,U> (rectangle.point.first, rectangle.point.second) << ", ";  
 out << std::pair<U,U> (rectangle.point.first, rectangle.point.second + rectangle.B) << ", ";  
 out << std::pair<U,U> (rectangle.point.first + rectangle.A, rectangle.point.second + rectangle.B) << ", ";  
 out << std::pair<U,U> (rectangle.point.first + rectangle.A, rectangle.point.second);  
 out << "]";  
 return out;  
 }  
};  
  
#endif /\* RECTANGLE\_HPP \*/

**Rhombus.h**

#ifndef OOP\_EXERSICE\_03\_RHOMBUS\_H  
#define OOP\_EXERSICE\_03\_RHOMBUS\_H  
#include "Figures.h"  
  
const unsigned int RHOMBUS\_TYPE\_ID = 1;  
  
template<class T1, class T2>  
class Rhombus : public Figures<T1> {  
private:  
 T2 D1, D2;  
public:  
 Rhombus(const std::pair<T1,T2> &\_point, const T2 &d1, const T2 &d2): D1(d1), D2(d2){  
 //point of left edge  
 this->point.first = \_point.first;  
 this->point.second = \_point.second;  
  
 if (D1 < 0 || D2 < 0) {  
 throw std::invalid\_argument("Invalid Rhombus parameters!");  
 }  
 }  
  
 void Print(std::ostream& out) override{  
 out << \*this << std::endl;  
 }  
  
 void Write(std::ofstream& file) override{  
 file.write((const char\*) &(RHOMBUS\_TYPE\_ID), sizeof(unsigned));  
 file.write((const char\*) &(this->point.first), sizeof (T1));  
 file.write((const char\*) &(this->point.second), sizeof(T1));  
 file.write((const char\*) &(this->D1), sizeof(T2));  
 file.write((const char\*) &(this->D2), sizeof (T2));  
 }  
  
 template<class U>  
 friend std::ostream& operator<<(std::ostream& out, const Rhombus<U,U>& rhombus){  
 out << "Rhombus vertices: [ "  
 << std::pair<U,U> (rhombus.point.first, rhombus.point.second) << ", "  
 << std::pair<U,U> (rhombus.point.first + rhombus.D1 / 2, rhombus.point.second + rhombus.D2 / 2) << ", "  
 << std::pair<U,U> (rhombus.point.first + rhombus.D1, rhombus.point.second) << ", "  
 << std::pair<U,U> (rhombus.point.first - rhombus.D1 / 2, rhombus.point.second - rhombus.D2 / 2) << " ]" << std::endl;  
  
 return out;  
 }  
};  
  
#endif //OOP\_EXERSICE\_03\_RHOMBUS\_H

**Trapeze.h**

#ifndef OOP\_EXERSICE\_03\_TRAPEZE\_H  
#define OOP\_EXERSICE\_03\_TRAPEZE\_H  
#include "Figures.h"  
  
const unsigned int TRAPEZE\_TYPE\_ID = 3;  
  
template<class Tpoint, class Tside>  
class Trapeze : public Figures<Tpoint>{  
private:  
 Tside A, B, C;  
public:  
 Trapeze(const std::pair<Tpoint,Tside> &\_point, const Tside &a, const Tside &b, const Tside &c): A(a), B(b), C(c){  
 this->point.first = \_point.first;  
 this->point.second = \_point.second;  
  
 if (A < 0 or B < 0 or C < 0) {  
 throw std::invalid\_argument("Invalid trapeze parameters!");  
 }  
 if (B > A) {  
 std::swap(A, B);  
 }  
 }  
  
 void Print(std::ostream& out) override{  
 out << \*this << std::endl;  
 }  
  
 void Write(std::ofstream& file) override{  
 file.write((const char\*) &(TRAPEZE\_TYPE\_ID), sizeof(unsigned));  
 file.write((const char\*) &(this->point.first), sizeof (Tpoint));  
 file.write((const char\*) &(this->point.second), sizeof(Tpoint));  
 file.write((const char\*) &(this->A), sizeof(Tside));  
 file.write((const char\*) &(this->B), sizeof (Tside));  
 file.write((const char\*) &(this->C), sizeof (Tside));  
 }  
  
 template<class U>  
 friend typename std::enable\_if<is\_double<U,U>::value, std::ostream &>::type  
 operator<<(std::ostream &out, const Trapeze<U,U> &trapeze) {  
 Tside diff = (trapeze.A - trapeze.B) / 2.0;  
 double height = std::sqrt(fabs(trapeze.C \* trapeze.C - diff \* diff));  
 out << "Trapeze vertices: [";  
 out << std::pair<U,U> (trapeze.point.first, trapeze.point.second) << ", ";  
 out << std::pair<U,U> (trapeze.point.first + diff, trapeze.point.second + height) << ", ";  
 out << std::pair<U,U> (trapeze.point.first + trapeze.A - diff, trapeze.point.second + height) << ", ";  
 out << std::pair<U,U> (trapeze.point.first + trapeze.A, trapeze.point.second);  
 out << "]";  
 return out;  
 }  
  
 template<class U>  
 friend typename std::enable\_if<is\_int<U,U>::value, std::ostream &>::type  
 operator<<(std::ostream &out, const Trapeze<U,U> &trapeze) {  
 out << "Trapeze: [ I cant calculate coordinates, because the type is int ]" << std::endl;  
 return out;  
 }  
};  
#endif //OOP\_EXERSICE\_03\_TRAPEZE\_H

**event.h**

#ifndef OOP\_EXERCISE\_08\_EVENT\_H  
#define OOP\_EXERCISE\_08\_EVENT\_H  
#include <utility>  
#include <vector>  
#include <string>  
#include <memory>  
#include <functional>  
#include <fstream>  
#include <iostream>  
#include <unordered\_map>  
#include "document.h"  
  
unsigned FILE\_COUNT = 1;  
  
std::string str\_gen(){  
 std::string result = "log";  
 result += std::to\_string(FILE\_COUNT++);  
 result += ".bin";  
 return result;  
}  
  
enum class eventType{  
 save,print,exit  
};  
  
struct Event\_Response{  
 virtual ~Event\_Response() = default;  
};  
  
struct Event\_Response\_Printer: public Event\_Response{  
 bool& done;  
 explicit Event\_Response\_Printer(bool& \_done):done(\_done){}  
};  
  
struct Event\_Response\_Saver: public Event\_Response{  
 bool& done;  
 explicit Event\_Response\_Saver(bool& \_done):done(\_done){}  
};  
  
  
  
struct Event\_data{  
 virtual ~Event\_data() = default;  
};  
  
template<tempFigTYPE>  
struct Event\_data\_Printer: public Event\_data{  
 const std::list<std::shared\_ptr<Figures FIGURE\_TYPE>> figures;  
  
 explicit Event\_data\_Printer(std::list<std::shared\_ptr<Figures FIGURE\_TYPE>> \_figures) : figures(\_figures){}  
};  
  
template<tempFigTYPE>  
struct Event\_data\_Saver: public Event\_data{  
 const std::list<std::shared\_ptr<Figures FIGURE\_TYPE>> figures;  
  
 explicit Event\_data\_Saver(const std::list<std::shared\_ptr<Figures FIGURE\_TYPE>>& \_figures) : figures(\_figures){}  
};  
  
  
template<tempFigTYPE>  
struct Event {  
 eventType type;  
 std::shared\_ptr<Event\_Response> response;  
 std::shared\_ptr<Event\_data> data;  
 std::function<void(std::shared\_ptr<Event\_Response>)> callback;  
  
 Event(eventType \_type,  
 std::shared\_ptr<Event\_data> \_data,  
 std::shared\_ptr<Event\_Response> \_response,  
 std::function<void(std::shared\_ptr<Event\_Response>)> \_callback): type(\_type),  
 data(std::move(\_data)),  
 response(std::move(\_response)),  
 callback(std::move(\_callback)){}  
  
};  
  
template<tempFigTYPE>  
struct Handler{  
 virtual bool do\_event(Event FIGURE\_TYPE& event) = 0;  
 virtual ~Handler() = default;  
};  
  
template<tempFigTYPE>  
struct Handler\_Printer: public Handler FIGURE\_TYPE {  
 bool do\_event(Event FIGURE\_TYPE& event) override{  
 auto data = std::static\_pointer\_cast<Event\_data\_Printer FIGURE\_TYPE>(event.data);  
 //если данный нет, то мы ничего и не сделали  
 if(!data){  
 return false;  
 }  
  
 std::this\_thread::sleep\_for(std::chrono::milliseconds(500));  
 for (auto& fig : data->figures) {  
 fig->Print(std::cout);  
 }  
 event.callback(event.response);  
 return true;  
 }  
};  
  
template<tempFigTYPE>  
struct Handler\_Saver: public Handler FIGURE\_TYPE {  
 bool do\_event(Event FIGURE\_TYPE& event) override{  
 std::ofstream fout;  
 fout.open(str\_gen(), std::ios\_base::out);  
 auto data = std::static\_pointer\_cast<Event\_data\_Saver FIGURE\_TYPE>(event.data);  
 //если данный нет, то мы ничего и не сделали  
 if(!data){  
 fout.close();  
 return false;  
 }  
  
 std::this\_thread::sleep\_for(std::chrono::milliseconds(500));  
 for (auto& fig : data->figures) {  
 fig->Print(fout);  
 }  
 event.callback(event.response);  
 fout.close();  
 return true;  
 }  
};  
  
#endif //OOP\_EXERCISE\_08\_EVENT\_H

**event-loop.h**

#ifndef OOP\_EXERCISE\_08\_EVENT\_LOOP\_H  
#define OOP\_EXERCISE\_08\_EVENT\_LOOP\_H  
  
#include <vector>  
#include <queue>  
#include <utility>  
#include <algorithm>  
#include <unordered\_map>  
#include <memory>  
#include <thread>  
#include <mutex>  
#include "event.h"  
  
template<tempFigTYPE>  
class Event\_Manager{  
private:  
 std::unordered\_multimap<eventType, std::shared\_ptr<Handler FIGURE\_TYPE>> hendlers;  
  
public:  
 void subscribe(eventType \_type, std::shared\_ptr<Handler FIGURE\_TYPE>& \_hendler){  
 hendlers.emplace(\_type,\_hendler);  
 }  
  
 void notify(eventType type, Event FIGURE\_TYPE& event){  
 auto keys = hendlers.equal\_range(type);  
 auto doEvent = [&](auto pointer){  
 pointer.second->do\_event(event);  
 };  
 std::for\_each(keys.first, keys.second,doEvent);  
 }  
};  
  
  
template <tempFigTYPE>  
class Event\_loop{  
private:  
 bool exit = false;  
 std::queue<Event FIGURE\_TYPE> events;  
 Event\_Manager FIGURE\_TYPE eventManager;  
 std::mutex wait;  
  
public:  
 void addHandler(eventType \_type, std::shared\_ptr<Handler FIGURE\_TYPE>& handler){  
 eventManager.subscribe(\_type,handler);  
 }  
  
 void addEvent(Event FIGURE\_TYPE& \_event){  
 std::lock\_guard<std::mutex> guard(wait);  
 events.push(\_event);  
 }  
  
 void operator()() {  
 while (!exit) {  
 if (!events.empty()) {  
 Event ev = events.front();  
 events.pop();  
  
 switch(ev.type) {  
 case eventType::exit:  
 exit = true;  
 break;  
 default:  
 eventManager.notify(ev.type, ev);  
 }  
 } else {  
 std::this\_thread::sleep\_for(std::chrono::microseconds(10));  
 }  
 }  
 }  
};  
#endif //OOP\_EXERCISE\_08\_EVENT\_LOOP\_H

1. **Вывод**

В ходе выполнения работы я на практике смог понять принцип создания асинхронного кода в C++ и также научился делать программу прослеживая шаблон Publish-Subscribe.

**Список литературы**

1. *std::mutex — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/thread/mutex> (дата обращения 16.12.2020).

1. *std::thread — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/thread/thread> (дата обращения 16.12.2020).

1. *Потоки, блокировки и условные переменные в C++11**— habr.com*

URL: <https://habr.com/ru/post/182610/> (дата обращения 16.12.2020)