**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» ИНЖЕНЕРНО- ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчет по лабораторной работе №4**

по курсу «Объектно ориентированное программирование»

**«Контейнеры STL»**

Выполнил: студент гр. КТбо2-7

Измайлов В.Н.

Таганрог 2020

**Оглавление**

[Техническое задание 3](#_TOC_250004)

[Выполнение задания 3](#_TOC_250003)

1. [Спецификация классов 3](#_TOC_250002)
2. [Диаграмма классов 4](#_TOC_250001)
3. [Листинг 5](#_TOC_250000)

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# Создать Класс “Расписание\_полетов (FlightShedule)” с полями: дата вылета, время вылета, пункт назначения, время прибытия, номер рейса. Вывести номера рейсов, время вылета и время прибытия на задаваемую дату и задаваемый пункт назначения.

# ВЫПОЛНЕНИЕ ЗАДАНИЯ

# Спецификация классов.

# Класс FlightBuilder парсит данные, считанные классом FlightParser. FlightsStorage – хранилище всех рейсов. Flight хранит всю информацию о каждом рейсе (DateTime (время вылета/прилета, дата вылета/прилета), FlightCode (код рейса), аэропорт прибытия, аэропорт отбытия). DateTime – класс содержащий всю информацию о времени рейса. FlightCode – класс содержащий информацию о коде рейса, arrival\_airport и departure\_airport содержат информацию о аэропорте прибытия и отбытия соответственно.

# 2. Диаграмма классов.

# Листинг

Main.cpp

#include "ConsoleInteractor.h"

#include <vld.h>

int main()

{

ConsoleInteractor start;

start.Start();

}

ConsoleInteractor.h

#pragma once

#include "FlightsStorage.h"

#include <memory>

class ConsoleInteractor {

public:

void Start();

private:

FlightsStorage \_flightsStorage;

std::vector<std::string> \_find\_commands = {

"Find flights by date and time",

"Find flights by airports and date",

"Find flights by arrival airport",

"Find flights by departure airport",

"Find flights by code"

};

std::vector<std::string> \_update\_commands = {

"Update date and time of flight",

"Update date of flight",

"Update airport of flight",

};

std::vector<std::string> \_info\_menu = {

"Read file ",

"Find Flights",

"Update Flights",

"Print Storage",

"Print Menu"

};

int ReadCommand() const;

void FindFlights();

void UpdateStorage();

void ReadFileAndBuildFlights();

void PrintFlightShedule();

void PrintCommands() const;

void UpdateDateTime();

void UpdateDate();

void UpdateAirport();

std::vector<Flight\*> FindFlightByDateTime();

std::vector<Flight\*> FindFlightByAirportsAndDate();

std::vector<Flight\*> FindFlightByArrivalAirport();

std::vector<Flight\*> FindFlightByDepartureAirport();

Flight\* FindFlightByFlightCode();

void ToUniqueFormat(std::string& typeForFormat) const;

void FindCommands() const;

void UpdateCommands() const;

template<typename T>

void ReadVar(T& var) const;

enum {

UPDATE\_DATE\_TIME = 1,

UPDATE\_DATE,

UPDATE\_AIRPORT,

DELETE\_FLIGHT

};

enum {

FIND\_BY\_DATE\_TIME = 1,

FIND\_BY\_AIRPORTS\_AND\_DATE,

FIND\_BY\_ARRIVAL,

FIND\_BY\_DEPARTURE,

FIND\_BY\_CODE

};

enum {

BUILD\_FLIGHT\_STORE = 1,

FIND\_FLIGHTS,

UPDATE\_FLIGHTS,

PRINT\_STORAGE,

PRINT\_MENU,

};

};

ConsoleInteractor.cpp

#include "ConsoleInteractor.h"

#include "FlightParser.h"

#include "FlightBuilder.h"

#include <string>

#include <sstream>

#include <iostream>

using namespace std;

template<typename T>

void ConsoleInteractor::ReadVar(T& var) const

{

string input = "";

cin >> input;

istringstream sin(input);

char c;

if (!(sin >> var) || (sin >> c))

{

throw std::runtime\_error("Input error. Failed to enter a variable");

}

}

void ConsoleInteractor::Start()

{

int command;

PrintCommands();

while (true) {

try {

command = ReadCommand();

switch (command) {

case 0: return;

case BUILD\_FLIGHT\_STORE: {

ReadFileAndBuildFlights();

break;

}

case FIND\_FLIGHTS: {

FindFlights();

break;

}

case UPDATE\_FLIGHTS: {

UpdateStorage();

break;

}

case PRINT\_STORAGE: {

PrintFlightShedule();

break;

}

case PRINT\_MENU: {

system("cls");

PrintCommands();

break;

}

default: {

throw std::runtime\_error(R"(Wrong command, try again)");

break;

}

}

}

catch (std::exception& err) {

cout << err.what() << endl;

}

}

}

int ConsoleInteractor::ReadCommand() const

{

int command;

cout << "Input command: ";

ReadVar(command);

return command;

}

void ConsoleInteractor::FindFlights()

{

if (\_flightsStorage.IsEmpty()) {

throw std::runtime\_error("\nInclude file with flights first\n");

}

FindCommands();

int indexOfPredicate;

std::vector<Flight\*> finded;

cout << "\nInput index of command: ";

ReadVar(indexOfPredicate);

switch (indexOfPredicate) {

case FIND\_BY\_DATE\_TIME: {

finded = FindFlightByDateTime();

break;

}

case FIND\_BY\_AIRPORTS\_AND\_DATE: {

finded = FindFlightByAirportsAndDate();

break;

}

case FIND\_BY\_ARRIVAL: {

finded = FindFlightByArrivalAirport();

break;

}

case FIND\_BY\_DEPARTURE: {

finded = FindFlightByDepartureAirport();

break;

}

case FIND\_BY\_CODE: {

finded.push\_back(FindFlightByFlightCode());

break;

}

default: {

std::runtime\_error(R"(Wrong command, try again)");

}

}

for (auto& el : finded) {

cout << el->To\_String() << endl;

}

cout << endl;

}

void ConsoleInteractor::UpdateStorage()

{

if (\_flightsStorage.IsEmpty()) {

throw std::runtime\_error("\nInclude file with flights first\n");

}

UpdateCommands();

int indexOfPredicate;

std::vector<Flight\*> finded;

ReadVar(indexOfPredicate);

switch (indexOfPredicate) {

case UPDATE\_DATE\_TIME: {

UpdateDateTime();

break;

}

case UPDATE\_DATE: {

UpdateDate();

break;

}

case UPDATE\_AIRPORT: {

UpdateAirport();

break;

}

default: {

std::runtime\_error(R"(Wrong command, try again)");

}

}

}

void ConsoleInteractor::ReadFileAndBuildFlights()

{

std::string file;

cout << "Enter file name: ";

cin >> file;

FlightParser parse;

FlightBuilder builder;

auto parsedFile = parse.Parse(file);

auto buildedFlights = builder.Build(parsedFile);

for (auto& flight : buildedFlights) {

\_flightsStorage.Add(flight);

}

cout << "\nSuccess\n" << endl;

}

void ConsoleInteractor::PrintFlightShedule()

{

if (!\_flightsStorage.IsEmpty()) {

cout << \_flightsStorage.To\_String() << endl;

}

else {

throw std::runtime\_error("List is empty");

}

}

void ConsoleInteractor::FindCommands() const

{

cout << "\n";

for (unsigned long i = 0; i < \_find\_commands.size(); i++) {

cout << "#" << i+1 << "\t" << \_find\_commands[i] << endl;

}

}

void ConsoleInteractor::UpdateCommands() const

{

cout << "\n";

for (unsigned long i = 0; i < \_update\_commands.size(); i++) {

cout<< "#" << i+1 << "\t" << \_update\_commands[i] << endl;

}

}

void ConsoleInteractor::PrintCommands() const

{

system("cls");

cout << "\n";

for (unsigned long i = 0; i < \_info\_menu.size(); i++) {

cout << "#" << i+1 << "\t" << \_info\_menu[i] << endl;

}

}

void ConsoleInteractor::UpdateDateTime()

{

stringstream buf;

FlightBuilder builder;

std::string date\_time;

std::string code;

cout << "Input FlightCode: ";

ReadVar(code);

cout << "Input date and time in format HH:mm-HH:mm YY.MM.DD-YY.MM.DD" << endl;

ReadVar(date\_time);

buf << date\_time;

DateTime date\_time\_obj = builder.ParseDateTime(buf);

buf << code;

FlightCode flight\_code = builder.ParseCode(buf);

\_flightsStorage.UpdateDateTime(flight\_code, date\_time\_obj);

}

void ConsoleInteractor::UpdateDate()

{

stringstream buf;

std::string date;

std::string code;

char flag;

char skip;

std::vector<int> new\_date(3);

cout << "Input FlightCode: ";

ReadVar(code);

cout << "Input new date: ";

ReadVar(date);

cout << "Input flag: (D - Deparutre date, A - Arrival date)";

ReadVar(flag);

buf << date;

buf >> new\_date[0] >> skip >> new\_date[1] >> skip >> new\_date[2];

\_flightsStorage.UpdateDate(code, new\_date, flag);

}

void ConsoleInteractor::UpdateAirport()

{

std::string airport;

std::string flight\_code;

stringstream buf;

char flag;

FlightBuilder builder;

FlightCode flight\_code\_obj;

cout << "Input flight code: ";

ReadVar(flight\_code);

buf << flight\_code;

flight\_code\_obj = builder.ParseCode(buf);

cout << "Input aiport: ";

ReadVar(airport);

cout << "Input flag (D - Departure, A - Arrival) : ";

ReadVar(flag);

\_flightsStorage.UpdateAirport(flight\_code\_obj, airport, flag);

}

std::vector<Flight\*> ConsoleInteractor::FindFlightByDateTime()

{

std::string time;

std::string date;

stringstream buf;

DateTime date\_time\_obj;

FlightBuilder builder;

cout << "Input Date and Time " << endl;

ReadVar(time);

ReadVar(date);

buf << time << " " << date;

date\_time\_obj = builder.ParseDateTime(buf);

return \_flightsStorage.FindFlightsByDateTime(date\_time\_obj);

}

std::vector<Flight\*> ConsoleInteractor::FindFlightByAirportsAndDate()

{

stringstream buf;

FlightBuilder builder;

std::string arrival\_airport;

std::string departure\_airport;

std::string date;

char flag;

cout << "\nDeparture town: ";

ReadVar(departure\_airport);

cout << "\n";

cout << "Arrival town: ";

ReadVar(arrival\_airport);

cout << "\n";

cout << "Date: ";

ReadVar(date);

cout << "\n";

cout << "Input flag (A - Airports and arrival date, D - Airports and departure date) : ";

ReadVar(flag);

buf << date;

DateTime date\_time\_obj = builder.ParseDate(buf);

switch (flag) {

case'A': {

AirportsAndArrivalDate airports\_and\_date{ date\_time\_obj.GetArrivalDate(),date\_time\_obj.GetDaysCount(), arrival\_airport, departure\_airport };

return \_flightsStorage.FindFlightsByAirportAndDate(airports\_and\_date);

break;

}

case 'D': {

AirportsAndDepartureDate airports\_and\_date{ date\_time\_obj.GetDepartureDate(),date\_time\_obj.GetDaysCount(), arrival\_airport, departure\_airport };

return \_flightsStorage.FindFlightsByAirportAndDate(airports\_and\_date);

break;

}

}

}

std::vector<Flight\*> ConsoleInteractor::FindFlightByArrivalAirport()

{

cout << "Input arrival airport: ";

std::string arrival\_airport;

ReadVar(arrival\_airport);

return \_flightsStorage.FindFlightByArrivalAirports(arrival\_airport);

}

std::vector<Flight\*> ConsoleInteractor::FindFlightByDepartureAirport()

{

cout << "Input departure airport: ";

std::string departure\_airport;

ReadVar(departure\_airport);

return \_flightsStorage.FindFlightByDepartureAirports(departure\_airport);

}

Flight\* ConsoleInteractor::FindFlightByFlightCode()

{

cout << "Input flight code: ";

FlightCode flight\_code\_obj;

FlightBuilder builder;

stringstream buf;

std::string flight\_code;

ReadVar(flight\_code);

buf << flight\_code;

flight\_code\_obj = builder.ParseCode(buf);

return \_flightsStorage.FindFlightByCode(flight\_code\_obj);

}

void ConsoleInteractor::ToUniqueFormat(std::string& typeForFormat) const

{

for (auto& letter : typeForFormat) {

letter = toupper(letter);

}

}

DateTime.h

#pragma once

#include "IStringConvertable.h"

#include <chrono>

using namespace std::chrono;

struct ArrivalDate {

int \_arrival\_day, \_arrival\_month, \_arrival\_year;

};

struct DepartureDate {

int \_departure\_day, \_departure\_month, \_departure\_year;

};

struct ArrivalTime {

hours \_arrival\_time\_hours;

minutes \_arrival\_time\_minutes;

};

struct DepartureTime {

hours \_departure\_time\_hours;

minutes \_departure\_time\_minutes;

};

class DateTime: public IStringConvertable {

public:

DateTime() = default;

DateTime(int departure\_time\_hours, int departure\_time\_minutes,

int arrival\_time\_hours, int arrival\_time\_minutes,

int arrival\_day, int arrival\_month, int arrival\_year,

int departure\_day, int departure\_month, int departure\_year) {

\_arrival\_date.\_arrival\_day = arrival\_day;

\_arrival\_date.\_arrival\_month = arrival\_month;

\_arrival\_date.\_arrival\_year = arrival\_year;

\_departure\_date.\_departure\_day = departure\_day;

\_departure\_date.\_departure\_month = departure\_month;

\_departure\_date.\_departure\_year = departure\_year;

\_arrival\_time.\_arrival\_time\_hours = (hours)arrival\_time\_hours;

\_arrival\_time.\_arrival\_time\_minutes = (minutes)arrival\_time\_minutes;

\_departure\_time.\_departure\_time\_hours = (hours)departure\_time\_hours;

\_departure\_time.\_departure\_time\_minutes = (minutes)departure\_time\_minutes;

}

DateTime(int day, int month, int year) {

\_departure\_date.\_departure\_day = day;

\_departure\_date.\_departure\_month = month;

\_departure\_date.\_departure\_year = year;

\_arrival\_date.\_arrival\_day = day;

\_arrival\_date.\_arrival\_month = month;

\_arrival\_date.\_arrival\_year = year;

}

bool operator<(const DateTime& date) const;

int GetDaysCount() const;

static int GetDaysCountInMonth(int month, int year);

static int GetDaysCountInYear(int year);

std::string To\_String() const override;

const ArrivalDate& GetArrivalDate() const {

return \_arrival\_date;

}

const DepartureDate& GetDepartureDate() const {

return \_departure\_date;

}

const ArrivalTime& GetArrivalTime() const {

return \_arrival\_time;

}

const DepartureTime& GetDepartureTime() const {

return \_departure\_time;

}

void SetArrivalDate(int arrival\_day, int arrival\_month, int arrival\_year) {

\_arrival\_date.\_arrival\_day = arrival\_day;

\_arrival\_date.\_arrival\_month = arrival\_month;

\_arrival\_date.\_arrival\_year = arrival\_year;

}

void SetDepartureDate(int departure\_day, int departure\_month, int departure\_year) {

\_departure\_date.\_departure\_day = departure\_day;

\_departure\_date.\_departure\_month = departure\_month;

\_departure\_date.\_departure\_year = departure\_year;

}

void SetDepartureTime(hours departure\_time\_hours, minutes departure\_time\_minutes) {

\_departure\_time.\_departure\_time\_hours = departure\_time\_hours;

\_departure\_time.\_departure\_time\_minutes = departure\_time\_minutes;

}

void SetArrivalTime(hours arrival\_time\_hours, minutes arrival\_time\_minutes) {

\_arrival\_time.\_arrival\_time\_hours = arrival\_time\_hours;

\_arrival\_time.\_arrival\_time\_minutes = arrival\_time\_minutes;

}

private:

ArrivalDate \_arrival\_date;

DepartureDate \_departure\_date;

ArrivalTime \_arrival\_time;

DepartureTime \_departure\_time;

};

DateTime.cpp

#include "DateTime.h"

#include <iostream>

#include <sstream>

#include <iomanip>

std::string DateTime::To\_String() const

{

std::stringstream buf;

buf << std::setfill('0') << std::setw(2) << \_departure\_time.\_departure\_time\_hours.count() << ":"

<< std::setfill('0') << std::setw(2) << \_departure\_time.\_departure\_time\_minutes.count() << "-"

<< std::setfill('0') << std::setw(2) << \_arrival\_time.\_arrival\_time\_hours.count() << ":"

<< std::setfill('0') << std::setw(2) << \_arrival\_time.\_arrival\_time\_minutes.count()

<< "\t"

<< std::setfill('0') << std::setw(2) << \_departure\_date.\_departure\_day << "."

<< std::setfill('0') << std::setw(2) << \_departure\_date.\_departure\_month << "."

<< std::setfill('0') << std::setw(2) << \_departure\_date.\_departure\_year << "-"

<< std::setfill('0') << std::setw(2) << \_arrival\_date.\_arrival\_day << "."

<< std::setfill('0') << std::setw(2) << \_arrival\_date.\_arrival\_month << "."

<< std::setfill('0') << std::setw(4) << \_arrival\_date.\_arrival\_year;

return buf.str();

}

bool DateTime::operator<(const DateTime& date) const

{

int date1 = GetDaysCount();

int date2 = date.GetDaysCount();

if (date1 < date2)

{

return true;

}

return false;

}

int DateTime::GetDaysCount() const

{

int res = \_arrival\_date.\_arrival\_day;

for (int i = 1; i < \_arrival\_date.\_arrival\_month; i++)

{

res += GetDaysCountInMonth(\_arrival\_date.\_arrival\_month, \_arrival\_date.\_arrival\_year);

}

int leapYearsCount = \_arrival\_date.\_arrival\_year / 4;

int yearsCount = \_arrival\_date.\_arrival\_year - leapYearsCount;

res += leapYearsCount \* 366 + yearsCount \* 365;

return res;

}

int DateTime::GetDaysCountInMonth(int month, int year)

{

if (month == 4 || month == 6 || month == 9 || month == 11)

{

return 30;

}

else if (month == 2)

{

if (GetDaysCountInYear(year) == 365)

{

return 28;

}

else

{

return 29;

}

}

else

{

return 31;

}

}

int DateTime::GetDaysCountInYear(int year)

{

if (year % 4 != 0 || year % 100 == 0 && year % 400 != 0)

{

return 365;

}

return 366;

}

Flight.h

#pragma once

#include "DateTime.h"

#include "Time.h"

#include "FlightCode.h"

#include "IStringConvertable.h"

class Flight : public IStringConvertable {

public:

Flight() = default;

Flight(const DateTime& date\_time, const std::string& arrival\_airport, const std::string& departure\_airport, const FlightCode& code) {

\_departure\_airport = departure\_airport;

\_arrival\_airport = arrival\_airport;

\_date\_time = date\_time;

\_code = code;

}

Flight(const Flight& flight) {

\_date\_time = flight.\_date\_time;

\_code = flight.\_code;

\_arrival\_airport = flight.\_arrival\_airport;

\_departure\_airport = flight.\_departure\_airport;

}

Flight& operator = (const Flight& flight) {

if (&flight == this) {

return \*this;

}

\_arrival\_airport = flight.\_arrival\_airport;

\_departure\_airport = flight.\_departure\_airport;

\_date\_time = flight.\_date\_time;

\_code = flight.\_code;

}

std::string To\_String() const override;

const DateTime& GetDateTime() const {

return \_date\_time;

}

const FlightCode& GetInitialCode() const {

return \_code;

}

const std::string& GetArrivalAirport() const {

return \_arrival\_airport;

}

const std::string& GetDepartureAirport() const {

return \_departure\_airport;

}

void SetDateTime(const DateTime& new\_date\_time) {

\_date\_time = new\_date\_time;

}

void SetArrivalAirport(const std::string& new\_arrival\_airport) {

\_arrival\_airport = new\_arrival\_airport;

}

void SetDepartureAirport(const std::string& new\_departure\_airport) {

\_departure\_airport = new\_departure\_airport;

}

void SetFlightCode(const FlightCode& new\_flight\_code) {

\_code = new\_flight\_code;

}

void SetDate(int arrival\_day, int arrival\_month, int arrival\_year) {

\_date\_time.SetArrivalDate(

arrival\_day, arrival\_month, arrival\_year

);

}

private:

DateTime \_date\_time;

FlightCode \_code;

std::string \_arrival\_airport;

std::string \_departure\_airport;

};

Flight.cpp

#include "Flight.h"

#include <sstream>

#include <iomanip>

std::string Flight::To\_String() const

{

std::ostringstream str;

str << "\n" << std::setw(20) << \_date\_time.To\_String() << "\t" << std::left << std::setw(40) << \_departure\_airport + "->" + \_arrival\_airport << std::setw(20) <<\_code.To\_String() << "\n";

return str.str();

}

FlightBuilder.h

#pragma once

#include "Flight.h"

#include <vector>

class FlightBuilder {

public:

Flight Build(const std::string& data) const;

std::vector<Flight> Build(const std::vector<std::string> &data) const;

DateTime ParseDateTime(std::istream& sin) const;

DateTime ParseDate(std::istream& sin) const;

std::vector<std::string> ParseAirports(std::istream& sin) const;

FlightCode ParseCode(std::istream& sin) const;

};

FlightBuilder.cpp

#include "FlightBuilder.h"

#include <istream>

#include <sstream>

#include <stdexcept>

#include <chrono>

Flight FlightBuilder::Build(const std::string& data) const

{

std::istringstream sin(data);

auto date\_time = ParseDateTime(sin);

auto airports = ParseAirports(sin);

auto code = ParseCode(sin);

return Flight(date\_time, airports[1], airports[0], code);

}

std::vector<Flight> FlightBuilder::Build(const std::vector<std::string>& data) const

{

std::vector<Flight> flights;

for (auto& elem : data) {

flights.push\_back(Build(elem));

}

flights.shrink\_to\_fit();

return flights;

}

DateTime FlightBuilder::ParseDateTime(std::istream& sin) const

{

char skipSymbol;

int partOfDateStructure = 0;

int partOfTimeStructure = 0;

int date[6];

int time[4];

for (int i = 0; i < 3; i++) {

sin >> time[partOfTimeStructure++];

sin >> skipSymbol;

}

sin >> time[partOfTimeStructure++];

for (int i = 0; i < 5; i++) {

sin >> date[partOfDateStructure++];

sin >> skipSymbol;

}

sin >> date[partOfDateStructure++];

return DateTime(time[0], time[1], time[2], time[3], date[0], date[1], date[2], date[3], date[4],

date[5]);

}

DateTime FlightBuilder::ParseDate(std::istream& sin) const

{

int date[3];

int partOfDateStructure = 0;

char skipSymbol;

for (int i = 0; i < 2; i++) {

sin >> date[partOfDateStructure++];

sin >> skipSymbol;

}

sin >> date[partOfDateStructure++];

return DateTime(date[0], date[1], date[2]);

}

std::vector<std::string> FlightBuilder::ParseAirports(std::istream& sin) const

{

std::vector<std::string> airports(2);

std::string buf\_airports;

sin >> buf\_airports;

int i = 0;

while (buf\_airports[i] != '>') {

airports[0].push\_back(buf\_airports[i]);

i++;

}

i++;

while (buf\_airports[i] != '\0') {

airports[1].push\_back(buf\_airports[i]);

i++;

}

return airports;

}

FlightCode FlightBuilder::ParseCode(std::istream& sin) const

{

std::string code;

sin >> code;

return FlightCode(code);

}

FlightCode.h

#pragma once

#include <string>

#include "IStringConvertable.h"

class FlightCode :public IStringConvertable {

public:

FlightCode() = default;

FlightCode(const std::string& code) {

\_code = code;

}

std::string To\_String() const override;

const std::string& GetCode() {

return \_code;

}

bool operator < (const FlightCode& flight\_code) const;

private:

std::string \_code;

};

FlightCode.cpp

#include "FlightCode.h"

#include <sstream>

std::string FlightCode::To\_String() const

{

std::ostringstream str(\_code);

return str.str();

}

bool FlightCode::operator<(const FlightCode& flight\_code) const

{

if (\_code < flight\_code.\_code) {

return true;

}

return false;

}

FlightParser.h

#pragma once

#include <vector>

#include <string>

class FlightParser {

public:

std::vector<std::string> Parse(std::string& filename);

};

FlightParser.cpp

#include "FlightParser.h"

#include <fstream>

std::vector<std::string> FlightParser::Parse(std::string& filename)

{

std::ifstream file;

file.open(filename, std::ios\_base::in);

if (!file.good()) {

throw std::runtime\_error("Can't open this file");

}

std::vector<std::string> textFromFile;

std::string textLine;

while (std::getline(file, textLine)) {

textFromFile.push\_back(std::move(textLine));

}

textFromFile.shrink\_to\_fit();

return textFromFile;

}

FlightStorage.h

#include "Flight.h"

#include <sstream>

#include <string>

#include <unordered\_map>

#include <map>

#include <set>

#include <chrono>

struct AirportsAndDepartureDate {

DepartureDate \_date\_time;

int \_days\_count;

std::string \_arrival\_airport;

std::string \_departure\_airport;

};

struct AirportsAndArrivalDate {

ArrivalDate \_date\_time;

int \_days\_count;

std::string \_arrival\_airport;

std::string \_departure\_airport;

};

class FlightsStorage {

public:

FlightsStorage() = default;

void Add(const Flight& flight);

std::string To\_String() const;

void UpdateDateTime(const FlightCode& flight\_code, const DateTime& new\_date\_time);

void UpdateDate(const FlightCode& flight\_code, const std::vector<int>& new\_arrival\_date, char flag);

void UpdateTime(const FlightCode& flight\_code, const std::chrono::hours new\_hours, const std::chrono::minutes new\_minutes, char flag);

void UpdateAirport(const FlightCode& flight\_code, const std::string& new\_arrival\_airport, char flag);

std::vector<Flight\*> FindFlightsByDateTime(const DateTime& date\_time);

std::vector<Flight\*> FindFlightsByAirportAndDate(const AirportsAndArrivalDate& airports\_and\_arrival\_date);

std::vector<Flight\*> FindFlightsByAirportAndDate(const AirportsAndDepartureDate& airports\_and\_departure\_date);

std::vector<Flight\*> FindFlightByArrivalAirports(const std::string& arrival\_airport);

std::vector<Flight\*> FindFlightByDepartureAirports(const std::string& departure\_airport);

Flight\* FindFlightByCode(const FlightCode& flight\_code);

bool IsEmpty();

private:

std::map<FlightCode, Flight> \_flightRegistry;

std::map<DateTime, std::set<FlightCode>> \_indexByDateTime;

std::map<AirportsAndDepartureDate, std::set<FlightCode>> \_indexByAirportsAndDepartureDate;

std::map<AirportsAndArrivalDate, std::set<FlightCode>> \_indexByAirportsAndArrivalDate;

std::map<std::string, std::set<FlightCode>> \_indexByArrivalAirports;

std::map<std::string, std::set<FlightCode>> \_indexByDepartureAirports;

};

FlightStorage.cpp

#include "FlightsStorage.h"

#include <iomanip>

void FlightsStorage::Add(const Flight& flight) {

\_flightRegistry.insert(std::make\_pair(flight.GetInitialCode(), flight));

auto itDateTime = \_indexByDateTime.find(flight.GetDateTime());

if (itDateTime != \_indexByDateTime.end()) {

itDateTime->second.insert(flight.GetInitialCode());

}

else {

\_indexByDateTime.insert(std::make\_pair(flight.GetDateTime(), std::set<FlightCode>()));

\_indexByDateTime[flight.GetDateTime()].insert(flight.GetInitialCode());

}

AirportsAndDepartureDate airports\_and\_departure\_date = {

flight.GetDateTime().GetDepartureDate(),

flight.GetDateTime().GetDaysCount(),

flight.GetArrivalAirport(),

flight.GetDepartureAirport()

};

auto itAirportsAndDepartureDate = \_indexByAirportsAndDepartureDate.find(airports\_and\_departure\_date);

if (itAirportsAndDepartureDate != \_indexByAirportsAndDepartureDate.end()) {

itAirportsAndDepartureDate->second.insert(flight.GetInitialCode());

}

else {

\_indexByAirportsAndDepartureDate.insert(std::make\_pair(airports\_and\_departure\_date, std::set<FlightCode>()));

\_indexByAirportsAndDepartureDate[airports\_and\_departure\_date].insert(flight.GetInitialCode());

}

AirportsAndArrivalDate airports\_and\_arrival\_date = {

flight.GetDateTime().GetArrivalDate(),

flight.GetDateTime().GetDaysCount(),

flight.GetArrivalAirport(),

flight.GetDepartureAirport()

};

auto itAirportsAndArrivalDate = \_indexByAirportsAndArrivalDate.find(airports\_and\_arrival\_date);

if (itAirportsAndArrivalDate != \_indexByAirportsAndArrivalDate.end()) {

itAirportsAndArrivalDate->second.insert(flight.GetInitialCode());

}

else {

\_indexByAirportsAndArrivalDate.insert(std::make\_pair(airports\_and\_arrival\_date, std::set<FlightCode>()));

\_indexByAirportsAndArrivalDate[airports\_and\_arrival\_date].insert(flight.GetInitialCode());

}

auto itArrivalAirports = \_indexByArrivalAirports.find(flight.GetArrivalAirport());

if (itArrivalAirports != \_indexByArrivalAirports.end()) {

itArrivalAirports->second.insert(flight.GetInitialCode());

}

else {

\_indexByArrivalAirports.insert(std::make\_pair(flight.GetArrivalAirport(), std::set<FlightCode>()));

\_indexByArrivalAirports[flight.GetArrivalAirport()].insert(flight.GetInitialCode());

}

auto itDepartureAirports = \_indexByDepartureAirports.find(flight.GetDepartureAirport());

if (itDepartureAirports != \_indexByDepartureAirports.end()) {

itDepartureAirports->second.insert(flight.GetInitialCode());

}

else {

\_indexByDepartureAirports.insert(std::make\_pair(flight.GetDepartureAirport(), std::set<FlightCode>()));

\_indexByDepartureAirports[flight.GetDepartureAirport()].insert(flight.GetInitialCode());

}

}

std::string FlightsStorage::To\_String() const

{

std::stringstream str;

str << "\nTime" << std::setw(12) << "\tDate" << std::setw(22) << "\tTowns" << std::setw(40) << "\tCode\n";

str << std::setfill('-') << std::setw(100) << "\n";

for (auto& flight : \_flightRegistry) {

str << flight.second.To\_String() << "\n";

}

return str.str();

}

void FlightsStorage::UpdateDateTime(const FlightCode& flight\_code, const DateTime& new\_date\_time)

{

auto it = \_flightRegistry.find(flight\_code);

if (it != \_flightRegistry.end()) {

\_indexByDateTime[it->second.GetDateTime()].erase(flight\_code);

\_indexByDateTime[new\_date\_time].insert(flight\_code);

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

it->second.SetDateTime(new\_date\_time);

}

void FlightsStorage::UpdateDate(const FlightCode& flight\_code, const std::vector<int>& new\_date, char flag)

{

auto itDate = \_flightRegistry.find(flight\_code);

if (itDate != \_flightRegistry.end()) {

DateTime new\_date\_time = itDate->second.GetDateTime();

switch (flag) {

case 'A': {

new\_date\_time.SetArrivalDate(

new\_date[0],

new\_date[1],

new\_date[2]

);

break;

}

case 'D': {

new\_date\_time.SetDepartureDate(

new\_date[0],

new\_date[1],

new\_date[2]

);

break;

}

}

\_indexByDateTime[itDate->second.GetDateTime()].erase(flight\_code);

\_indexByDateTime[new\_date\_time].insert(flight\_code);

itDate->second.SetDateTime(new\_date\_time);

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

}

void FlightsStorage::UpdateTime(const FlightCode& flight\_code, const std::chrono::hours new\_hours, const std::chrono::minutes new\_minutes, char flag)

{

auto itTime = \_flightRegistry.find(flight\_code);

if (itTime != \_flightRegistry.end()) {

DateTime new\_date\_time = itTime->second.GetDateTime();

switch (flag) {

case 'A': {

new\_date\_time.SetArrivalTime(

new\_hours,

new\_minutes

);

break;

}

case 'D': {

new\_date\_time.SetDepartureTime(

new\_hours,

new\_minutes

);

break;

}

}

\_indexByDateTime[itTime->second.GetDateTime()].erase(flight\_code);

\_indexByDateTime[new\_date\_time].insert(flight\_code);

itTime->second.SetDateTime(new\_date\_time);

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

}

void FlightsStorage::UpdateAirport(const FlightCode& flight\_code, const std::string& new\_arrival\_airport, char flag)

{

auto it = \_flightRegistry.find(flight\_code);

if (it != \_flightRegistry.end()) {

switch (flag) {

case'A': {

\_indexByArrivalAirports[it->second.GetArrivalAirport()].erase(flight\_code);

\_indexByArrivalAirports[new\_arrival\_airport].insert(flight\_code);

break;

}

case 'D': {

\_indexByDepartureAirports[it->second.GetArrivalAirport()].erase(flight\_code);

\_indexByDepartureAirports[new\_arrival\_airport].insert(flight\_code);

break;

}

}

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

it->second.SetArrivalAirport(new\_arrival\_airport);

}

std::vector<Flight\*> FlightsStorage::FindFlightsByDateTime(const DateTime& date\_time)

{

auto it = \_indexByDateTime.find(date\_time);

std::vector<Flight\*> finded\_flights;

if (it != \_indexByDateTime.end()) {

for (auto itByFlights = it->second.begin(); itByFlights != it->second.end(); itByFlights++) {

finded\_flights.push\_back(&\_flightRegistry[\*itByFlights]);

}

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

return finded\_flights;

}

std::vector<Flight\*> FlightsStorage::FindFlightsByAirportAndDate(const AirportsAndDepartureDate& airports\_and\_arrival\_date)

{

auto it = \_indexByAirportsAndDepartureDate.find(airports\_and\_arrival\_date);

std::vector<Flight\*> finded\_flights;

if (it != \_indexByAirportsAndDepartureDate.end()) {

for (auto itByFlights = it->second.begin(); itByFlights != it->second.end(); itByFlights++) {

finded\_flights.push\_back(&\_flightRegistry[\*itByFlights]);

}

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

return finded\_flights;

}

std::vector<Flight\*> FlightsStorage::FindFlightsByAirportAndDate(const AirportsAndArrivalDate& airports\_and\_arrival\_date)

{

auto it = \_indexByAirportsAndArrivalDate.find(airports\_and\_arrival\_date);

std::vector<Flight\*> finded\_flights;

if (it != \_indexByAirportsAndArrivalDate.end()) {

for (auto itByFlights = it->second.begin(); itByFlights != it->second.end(); itByFlights++) {

finded\_flights.push\_back(&\_flightRegistry[\*itByFlights]);

}

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

return finded\_flights;

}

std::vector<Flight\*> FlightsStorage::FindFlightByArrivalAirports(const std::string& arrival\_airport)

{

auto it = \_indexByArrivalAirports.find(arrival\_airport);

std::vector<Flight\*> finded\_flights;

if (it != \_indexByArrivalAirports.end()) {

for (auto itByFlights = it->second.begin(); itByFlights != it->second.end(); itByFlights++) {

finded\_flights.push\_back(&\_flightRegistry[\*itByFlights]);

}

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

return finded\_flights;

}

std::vector<Flight\*> FlightsStorage::FindFlightByDepartureAirports(const std::string& departure\_airport)

{

auto it = \_indexByDepartureAirports.find(departure\_airport);

std::vector<Flight\*> finded\_flights;

if (it != \_indexByDepartureAirports.end()) {

for (auto itByFlights = it->second.begin(); itByFlights != it->second.end(); itByFlights++) {

finded\_flights.push\_back(&\_flightRegistry[\*itByFlights]);

}

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

return finded\_flights;

}

Flight\* FlightsStorage::FindFlightByCode(const FlightCode& flight\_code)

{

if (\_flightRegistry.find(flight\_code) != \_flightRegistry.end()) {

return &\_flightRegistry.find(flight\_code)->second;

}

else {

throw std::runtime\_error("\nNo such flight in list\n");

}

}

bool FlightsStorage::IsEmpty()

{

return \_flightRegistry.empty();

}

bool operator<(const AirportsAndDepartureDate& ref1, const AirportsAndDepartureDate& ref2) {

int day1 = ref1.\_days\_count;

int day2 = ref2.\_days\_count;

if (day1 < day2) {

return true;

}

return false;

}

bool operator<(const AirportsAndArrivalDate& ref1, const AirportsAndArrivalDate& ref2) {

int day1 = ref1.\_days\_count;

int day2 = ref2.\_days\_count;

if (day1 < day2) {

return true;

}

return false;

}

IStringConvertable.h

#pragma once

#include <string>

class IStringConvertable {

public:

virtual std::string To\_String() const = 0;

};