МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 4**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Контейнеры STL»**

Выполнил:

Студент группы

КТбо2-6

Калитин А.В.

Проверил:

доцент кафедры

МОП ЭВМ

Тарасов С. А.

Таганрог 2020

# **1 Цель работы**

# Цель работы – ознакомление с классами-контейнерами библиотеки STL.

# **2 Задание**

# Класс «Поезд (Train)» с полями: пункт назначения, номер поезда, дата и время отправления, время прибытия. Вывести номера поездов в задаваемый пункт, отправляющихся после задаваемого времени.

# **3 Ход работы**

**3.1 Спецификации классов**

Диаграмма используемых классов приведена на Рисунке 1.

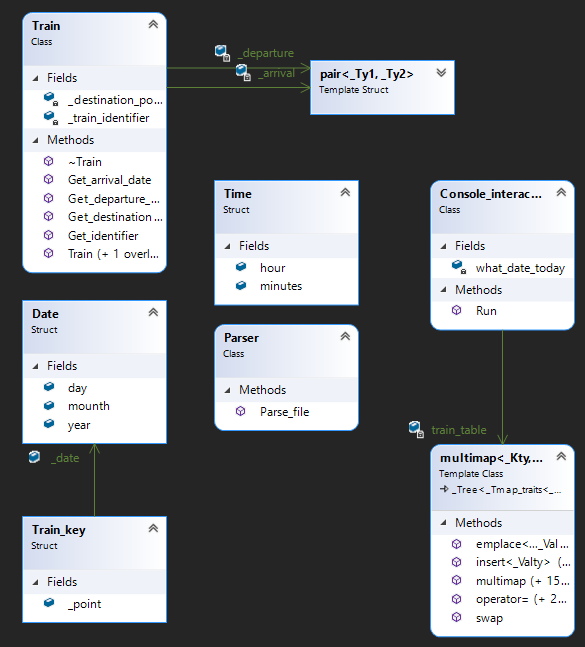


Рисунок 1 - Диаграмма классов

**3.2 Пояснение к коду**

# Класс Train содержит основную информацию об объекте поезда (время и дату прибытия и отбытия, уникальный идентификатор поезда, город в который он отправляется). Console\_interactor обеспечивает взаимодействие с пользователем и содержит в себе контейнер с таблицей. Parser с помощью метода Parse\_file осуществляет считывание данных с файла, который укажет пользователь. В файле имеется оперделенная структура записи информации о поезде: (пункт назначения, идентификатор, время отправления(hh:mm), дата отправления(dd.mm.yyyy), время прибытия(hh:mm), дата прибытия(dd.mm.yyyy)). Вся информация хранится в контейнере multimap. Ключ в multimap содержит структуру, которая содержит пункт назначения и время отправления. Так как может быть много поездов в один город, то для поиска я использую метод контейнера equal\_range(). А для прохода по этим записям итератор.

4. Листинг

Main.cpp

#define \_\_CRTDBG\_MAP\_ALLOC

#include <crtdbg.h>

#define DEBUG\_NEW new(\_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

#define new DEBUG\_NEW

#include "Console\_interactor.h"

int main()

{

\_CrtMemState \_ms;

\_CrtMemCheckpoint(&\_ms);

Console\_interactor console;

console.Run();

\_CrtMemDumpAllObjectsSince(&\_ms);

\_CrtSetReportMode(\_CRT\_WARN, \_CRTDBG\_MODE\_FILE);

\_CrtSetReportFile(\_CRT\_WARN, \_CRTDBG\_FILE\_STDOUT);

\_CrtDumpMemoryLeaks();

}

Console\_interactor.cpp

#include "Console\_interactor.h"

#define new new( \_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

void Console\_interactor::Run()

{

int command = 1;

try

{

std::string filename;

std::cout << "Enter path to file:\n";

std::cin >> filename;

Parser parser;

parser.Parse\_file(filename, &train\_table);

}

catch (std::exception& ex)

{

std::cout << ex.what();

exit(1);

}

std::cout << "1 - Find by city and time\n2 - Find by city(today)\n3 - Add train\n4 - Print all trains\n0 - Exit.\n";

while (command)

{

std::cout << "Enter a command: ";

std::cin >> command;

switch (command)

{

case 0:

{

break;

}

case 1:

{

std::cout << "Enter a town: \n";

std::string town;

std::cin >> town;

std::cout << "Enter the time and date(hh:mm dd.mm.yyyy):\n";

Time time;

Date date;

char symb;

std::cin >> time.hour >> symb >> time.minutes >> date.day >> symb >> date.mounth >> symb >> date.year;

Train\_key key;

key.\_date = date;

key.\_point = town;

auto ktown = train\_table.equal\_range(key);

std::cout << "Search result:\n";

for (auto it = ktown.first; it != ktown.second; it++)

{

if (time < (\*it).second.Get\_departure\_date().first)

{

std::cout << (\*it).second.Get\_identifier() << " " << (\*it).second.Get\_departure\_date().first.hour << ":"

<< (\*it).second.Get\_departure\_date().first.minutes << "\n";

}

}

}

break;

case 2:

{

std::cout << "Enter a town: \n";

Train\_key key;

key.\_date = what\_date\_today;

std::cin >> key.\_point;

auto ktown = train\_table.equal\_range(key);

std::cout << "Search result:\n";

for (auto it = ktown.first; it != ktown.second; it++)

{

std::cout << (\*it).second.Get\_identifier() << " " << (\*it).second.Get\_departure\_date().first.hour << ":"

<< (\*it).second.Get\_departure\_date().first.minutes << "\n";

}

break;

}

case 3:

{

std::cout << "Type point(string) identifier(int) departure time(hh:mm) departure date(dd.mm.yyyy) arrival time(hh:mm) arrival date(dd.mm.yyyy)\n";

std::string point;

int identifier;

char symbol;

std::pair<Time, Date> \_departure;

std::pair<Time, Date> \_arrival;

Train\_key key;

std::cin >> point >> identifier >> \_departure.first.hour >> symbol >> \_departure.first.minutes

>> \_departure.second.day >> symbol >> \_departure.second.mounth >> symbol >> \_departure.second.year

>> \_arrival.first.hour >> symbol >> \_arrival.first.minutes

>> \_arrival.second.day >> symbol >> \_arrival.second.mounth >> symbol >> \_arrival.second.year;

Train temp(point, identifier, \_departure, \_arrival);

key.\_date = \_departure.second;

key.\_point = point;

train\_table.insert(std::pair<Train\_key, Train>(key, temp));

break;

}

case 4:

{

for (auto it = train\_table.begin(); it != train\_table.end(); it++)

{

std::cout << (\*it).first.\_point<<" "<< (\*it).second.Get\_identifier()<<" " << (\*it).second.Get\_departure\_date().first.hour << ":" << (\*it).second.Get\_departure\_date().first.minutes << "\n";

}

break;

}

default:

std::cout << "Invalid command. Try again.\n";

break;

}

}

}

Console\_interactor.h

#pragma once

#include <iostream>

#include <ctime>

#include <typeinfo>

#include <fstream>

#include <map>

#include "Parser.h"

#include "Train.h"

class Console\_interactor

{

public:

void Run();

private:

Date what\_date\_today = { 12, 12, 2020 };

std::multimap<Train\_key, Train> train\_table;

};

Parser.cpp

#include "Parser.h"

void Parser::Parse\_file(const std::string &path, std::multimap<Train\_key, Train> \*train\_table) const

{

std::fstream input(path);

if (!input.is\_open())

{

throw std::exception("File doesn`t exist!");

}

int how\_trains;

std::string point;

int identifier;

char symbol;

std::pair<Time, Date> \_departure;

std::pair<Time, Date> \_arrival;

Train\_key key;

input >> how\_trains;

for (auto i = 0; i < how\_trains; i++)

{

input >> point >> identifier >> \_departure.first.hour >> symbol >> \_departure.first.minutes >> \_departure.second.day >> symbol >> \_departure.second.mounth >> symbol >> \_departure.second.year

>> \_arrival.first.hour >> symbol >> \_arrival.first.minutes >> \_arrival.second.day >> symbol >> \_arrival.second.mounth >> symbol >> \_arrival.second.year;

Train temp(point, identifier, \_departure, \_arrival);

key.\_date = \_departure.second;

key.\_point = point;

train\_table->emplace(key, temp);

}

}

Time.h

#include "Time.h"

bool operator<(const Time& first, const Time& second)

{

return (first.hour\*60+first.minutes) <= (second.hour \* 60 + second.minutes);

}

Train.h

#pragma once

#include <string>

#include "Time.h"

#include "Date.h"

class Train

{

public:

Train() = default;

Train(std::string destination\_point, int train\_identifier, std::pair<Time, Date> \_departure, std::pair<Time, Date> \_arrival);

~Train() = default;

std::string Get\_destination\_point() const;

friend bool operator<(const Train& first, const Train& second);

friend bool operator>(const Train& first, const Train& second);

int Get\_identifier() const;

std::pair<Time, Date> Get\_departure\_date() const;

std::pair<Time, Date> Get\_arrival\_date() const;

private:

std::string \_destination\_point = "";

int \_train\_identifier = -1;

std::pair<Time, Date> \_departure;

std::pair<Time, Date> \_arrival;

};

Train.cpp

#include "Train.h"

Train::Train(std::string destination\_point, int train\_identifier, std::pair<Time, Date> departure, std::pair<Time, Date> arrival)

:

\_destination\_point(destination\_point),

\_train\_identifier(train\_identifier),

\_departure(departure),

\_arrival(arrival)

{}

std::string Train::Get\_destination\_point() const

{

return \_destination\_point;

}

int Train::Get\_identifier() const

{

return \_train\_identifier;

}

std::pair<Time, Date> Train::Get\_departure\_date() const

{

return \_departure;

}

std::pair<Time, Date> Train::Get\_arrival\_date() const

{

return \_arrival;

}

bool operator<(const Train& first, const Train& second)

{

return (first.\_departure.first.hour \* 60 + first.\_departure.first.minutes)

< (second.\_departure.first.hour \* 60 + second.\_departure.first.minutes);

}

bool operator>(const Train& first, const Train& second)

{

return (first.\_departure.first.hour \* 60 + first.\_departure.first.minutes)

> (second.\_departure.first.hour \* 60 + second.\_departure.first.minutes);

}

Train\_key.h

#pragma once

#include <string>

#include "Date.h"

struct Train\_key

{

std::string \_point = "";

Date \_date;

friend bool operator<(const Train\_key& a, const Train\_key& b)

{

return a.\_point < b.\_point;

}

};

Date.h

#pragma once

struct Date

{

int mounth = 0;

int day = 0;

int year = 0;

};

4 Заключение

В процессе выполнения лабораторной работы №4 мы ознакомились с контейнерами библиотеки STL, увеличили свои теоретические и практические навыки работы с контейнерами STL.