Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ № 8

Классы контейнеры-итераторы.

STL-контейнеры

по дисциплине «Программирование на языках высокого уровня»

Выполнил ст. гр. 450503 А.П. Красько

Проверил асс. каф. ЭВМ И.Г. Скиба

Минск 2025

# 1 ПОСТАНОВКА ЗАДАЧИ

Реализовать классы контейнер и итератор для работы с двунаправленным кольцом. Реализовать класс алгоритма, в котором определить методы поиска и сортировки двунаправленного кольца. Выполнить программу.

# 2 ДИАГРАММА КЛАССОВ

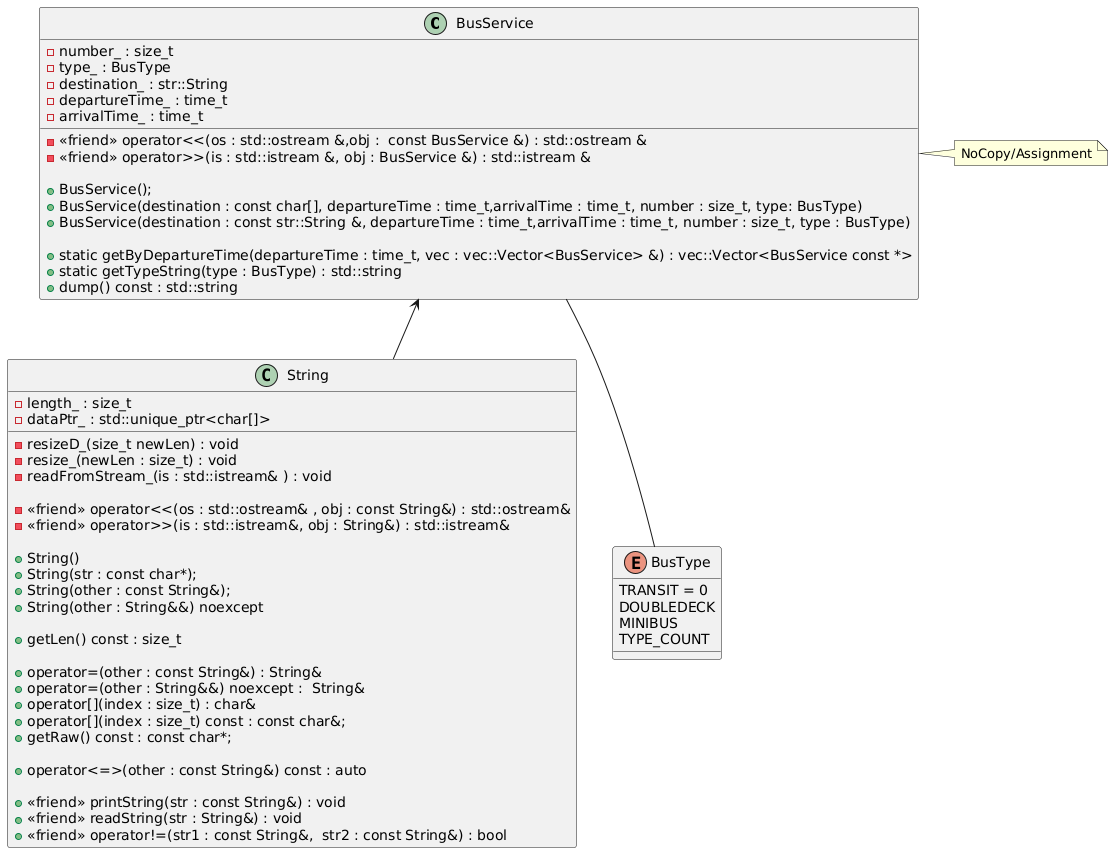


Рисунок 2.1 – Диаграмма классов

# 3 ЛИСТИНГ КОДА

Файл busService.hh:

#pragma once  
#include <ctime>  
#include <l2/include/string.hh>  
#include <l5/include/vector.hh>  
#include <utility>  
namespace bus\_service {  
enum class BusType { TRANSIT = 0, DOUBLEDECK, MINIBUS, TYPE\_COUNT };  
class BusService {  
 private:  
 size\_t number\_;  
 BusType type\_;  
 str::String destination\_;  
 time\_t departureTime\_;  
 time\_t arrivalTime\_;  
  
 friend std::ostream &operator<<(std::ostream &os, const BusService &obj) {  
 os.write(reinterpret\_cast<const std::istream::char\_type \*>(&obj.number\_), sizeof(obj.number\_));  
 os.write(reinterpret\_cast<const std::istream::char\_type \*>(&obj.type\_), sizeof(obj.type\_));  
  
 size\_t destinationLen = obj.destination\_.getLen();  
 os.write(reinterpret\_cast<const std::istream::char\_type \*>(&destinationLen), sizeof(destinationLen));  
 os.write(obj.destination\_.getRaw(), destinationLen);  
  
 os.write(reinterpret\_cast<const std::istream::char\_type \*>(&obj.departureTime\_), sizeof(obj.departureTime\_));  
 os.write(reinterpret\_cast<const std::istream::char\_type \*>(&obj.arrivalTime\_), sizeof(obj.arrivalTime\_));  
 return os;  
 };  
 friend std::istream &operator>>(std::istream &is, BusService &obj) {  
 is.read(reinterpret\_cast<std::istream::char\_type \*>(&obj.number\_), sizeof(obj.number\_));  
 is.read(reinterpret\_cast<std::istream::char\_type \*>(&obj.type\_), sizeof(obj.type\_));  
  
 size\_t destinationLen;  
 is.read(reinterpret\_cast<std::istream::char\_type \*>(&destinationLen), sizeof(destinationLen));  
 auto destination = std::make\_unique\_for\_overwrite<char[]>(destinationLen);  
 is.read(destination.get(), destinationLen);  
 obj.destination\_ = str::String(destination.get());  
  
 is.read(reinterpret\_cast<std::istream::char\_type \*>(&obj.departureTime\_), sizeof(obj.departureTime\_));  
 is.read(reinterpret\_cast<std::istream::char\_type \*>(&obj.arrivalTime\_), sizeof(obj.arrivalTime\_));  
 return is;  
 };  
  
 public:  
 BusService();  
 BusService(const str::String &destination, time\_t departureTime, time\_t arrivalTime, size\_t number,  
 BusType type = BusType::TRANSIT);  
 BusService(const char destination[], time\_t departureTime, time\_t arrivalTime, size\_t number,  
 BusType type = BusType::TRANSIT);  
 BusService(BusService &) = delete;  
  
 BusService &operator=(BusService &other) = delete;  
  
 static vec::Vector<BusService const \*> getByDepartureTime(time\_t departureTime, vec::Vector<BusService> &vec);  
 static std::string getTypeString(BusType type);  
 std::string dump() const;  
};  
  
} // namespace bus\_service

Файл screens.hh:

#pragma once  
#include <memory>  
#include <l5/include/vector.hh>  
#include "busService.hh"  
namespace screens {  
void printMainScreen();  
bool printFlights(vec::Vector<bus\_service::BusService> &vec);  
bool getByDepartureTime(vec::Vector<bus\_service::BusService> &vec);  
} // namespace screens

Файл screens.cc:

#include <consoleUtils.hh>  
#include <ctime>  
#include <iomanip>  
#include <l5/include/vector.hh>  
#include <l7/include/busService.hh>  
#include <print>  
#include <chrono>  
  
using namespace std;  
using namespace vec;  
using namespace str;  
using namespace bus\_service;  
using namespace console\_utils;  
  
namespace screens {  
void printMainScreen() {  
 auto [cols, rows] = getConsoleDimensions();  
 println("{:^{}}", "\x{1B}[48;5;35mLab 5\x{1B}[0m", cols);  
 println("Please select action:\n");  
 println(" 1.Print bus services");  
 println(" 2.Find service by departure time");  
 println(" 3.Exit");  
}  
bool printFlights(vec::Vector<BusService> &vec) {  
 for (size\_t i = 0, count = vec.count(); i < count; ++i) {  
 cout << vec[i].dump() << endl;  
 }  
 return true;  
}  
bool getByDepartureTime(Vector<BusService> &buses) {  
 String str;  
 std::chrono::sys\_seconds timePoint = {};  
 readT(timePoint, "Please enter time in DD-MM-YYYY HH:MM UTC+3 format: ", "%d-%m-%Y %H:%M");   
 time\_t departure = std::chrono::system\_clock::to\_time\_t(timePoint);  
 auto vec = BusService::getByDepartureTime(departure, buses);  
 print("Found {} Buses:\n", vec.count());  
 for (size\_t i = 0, count = vec.count(); i < count; ++i)  
 {  
 cout << vec[i]->dump() << endl;  
 }  
 cout << endl;  
   
 return true;  
}  
} // namespace screens

Файл main.cc:

#include <consoleUtils.hh>  
#include <fstream>  
#include <functional>  
#include <l5/include/vector.hh>  
#include <l7/include/busService.hh>  
#include <l7/include/screens.hh>  
#include <memory>  
  
using namespace std;  
using namespace console\_utils;  
using namespace screens;  
using namespace bus\_service;  
  
int main(void) {  
 auto tp = std::chrono::system\_clock::now();  
 time\_t time = std::chrono::system\_clock::to\_time\_t(tp);  
 ofstream out("data.bin", std::ios::binary);  
 if (!out.is\_open()) {  
 cout << "Cannot create data file" << endl;  
 return 1;  
 }  
 out << 3;  
 BusService b1{"Brest", time, time + 60 \* 60 \* 3, 1};  
 BusService b2{"Hrodno", time, time + 60 \* 60, 2};  
 BusService b3{"Hrodno", time + 60 \* 20, time + 60 \* 30, 3};  
 out << b1 << b2 << b3;  
 out.close();  
 std::ifstream in("data.bin", ios::binary);  
 if (!in.is\_open()) {  
 cout << "No data file provided. Please place one near executable" << endl;  
 return 1;  
 }  
 size\_t num;  
 in >> num;  
 if (in.fail()) {  
 cout << "Data file is not valid." << endl;  
 return 1;  
 }  
 vec::Vector<BusService> vec{num};  
 static array<function<bool()>, 4> actions = {  
 [&vec]() { return printFlights(vec); },  
 [&vec]() { return getByDepartureTime(vec); },  
 []() { return false; },  
 };  
  
 for (size\_t i = 0; i < num; ++i) {  
 in >> vec[i];  
 if (in.fail()) {  
 cout << "Data file is not valid." << endl;  
 return 1;  
 }  
 }  
 unsigned int response;  
 do {  
 printMainScreen();  
 readT(response, ">", [](unsigned int numb) { return numb > 0 && numb <= 3; });  
 cout << "\x{1B}[2J\x{1B}[H\n";  
 } while (actions[response - 1]());  
 in.close();  
  
 return 0;  
}

Файл busService.cc:

#include <chrono>  
#include <l7/include/busService.hh>  
#include <utility>  
  
using namespace vec;  
using namespace std;  
namespace bus\_service {  
  
BusService::BusService() : BusService("UNKNOWN", 0, 0, 0){};  
BusService::BusService(const str::String &destination, time\_t departureTime, time\_t arrivalTime, size\_t number,  
 BusType type)  
 : number\_{number},  
 type\_{type},  
 destination\_{destination},  
 departureTime\_{departureTime},  
 arrivalTime\_{arrivalTime} {};  
BusService::BusService(const char destination[], time\_t departureTime, time\_t arrivalTime, size\_t number, BusType type)  
 : BusService(str::String(destination), departureTime, arrivalTime, number, type){};  
  
Vector<BusService const \*> BusService::getByDepartureTime(time\_t departureTime, Vector<BusService> &vec) {  
 Vector<BusService const \*> res;  
 for (size\_t i = 0, count = vec.count(); i < count; ++i) {  
 time\_t epsilon = vec[i].departureTime\_ > departureTime ? vec[i].departureTime\_ - departureTime  
 : departureTime - vec[i].departureTime\_;  
 if (epsilon <= 60) {  
 res.pushBack(&vec[i]);  
 }  
 }  
 return res;  
}  
std::string BusService::getTypeString(BusType type) {  
 static const array<std::string, to\_underlying(BusType::TYPE\_COUNT)> strings = {"Transit", "Double deck",  
 "Mini bus"};  
 return strings[std::to\_underlying(type)];  
}  
std::string BusService::dump() const {  
 using namespace std::chrono;  
  
 std::stringstream stream;  
 time\_point tp1 = system\_clock::from\_time\_t(departureTime\_);  
 time\_point tp2 = system\_clock::from\_time\_t(arrivalTime\_);  
 stream << std::format(  
 "flight: [number: {}, type: {}, destination: {}, departure time: {:%d-%m-%Y %H:%M}, arrival time: "  
 "{:%d-%m-%Y %H:%M}]",  
 number\_, getTypeString(type\_), destination\_.getRaw(), tp1, tp2);  
 return stream.str();  
};  
  
} // namespace bus\_service

Файл consoleUtils.hh:

#pragma once  
#include <functional>  
#include <iostream>  
#include <limits>  
#include <iomanip>  
#include <chrono>  
namespace console\_utils {  
std::pair<int, int> getConsoleDimensions();  
  
  
template <typename T, typename CT>  
void readT(T& data, const std::string& massage, CT validator) {  
 std::cout << massage;  
 while (((std::cin >> data).fail()) || !validator(data)) {  
 std::cout << "Invalid input. Reread input requierments\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << massage;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
template <typename T>  
void readT(T& data, const std::string& massage) {  
 std::cout << massage;  
 while ((std::cin >> data).fail()) {  
 std::cout << "Invalid input. Reread input requierments\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << massage;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
template <typename T, typename CT>  
void readT(T& data, const std::string& message, CT validator, const std::string& errmess) {  
 std::cout << message;  
 while (((std::cin >> data).fail()) || !validator(data)) {  
 std::cout << errmess;  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << message;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
template <typename T = std::chrono::sys\_seconds, typename CT = const char \* >  
void readT(std::chrono::sys\_seconds &data, const char \* massage, const char \* format) {  
 std::cout << massage;  
 while ((std::cin >> std::chrono::parse(format, data)).fail()) {  
 std::cout << "Invalid input. Reread input requierments\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 std::cout << massage;  
 }  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
}  
} // namespace console\_utils

Файл consoleUtils.cc:

#include <iostream>  
  
#ifdef \_\_linux\_\_  
#include <sys/ioctl.h>  
#include <unistd.h>  
#endif  
  
#ifdef \_WIN32  
#include <Windows.h>  
#endif  
  
namespace console\_utils {  
std::pair<int, int> getConsoleDimensions() {  
#ifdef \_WIN32  
 CONSOLE\_SCREEN\_BUFFER\_INFO csbi;  
 GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);  
 return std::make\_pair(csbi.srWindow.Right - csbi.srWindow.Left + 1, csbi.srWindow.Bottom - csbi.srWindow.Top + 1);  
#endif  
#ifdef \_\_linux\_\_  
  
 struct winsize w;  
 ioctl(STDOUT\_FILENO, TIOCGWINSZ, &w);  
 return std::make\_pair(w.ws\_col, w.ws\_row);  
  
#endif  
}  
} // namespace console\_utils

Файл string.hh:

#pragma once  
#include <memory>  
#include <type\_traits>  
  
namespace str {  
class String {  
 private:  
 size\_t length\_;  
 std::unique\_ptr<char[]> dataPtr\_;  
  
 void resizeD\_(size\_t newLen);  
 void resize\_(size\_t newLen);  
 void readFromStream\_(std::istream& is);  
  
 friend std::ostream& operator<<(std::ostream& os, const String& obj) {  
 os << obj.dataPtr\_.get();  
 return os;  
 };  
  
 friend std::istream& operator>>(std::istream& is, String& obj) {  
 obj.readFromStream\_(is);  
 return is;  
 }  
  
 public:  
 explicit String(const char\* str);  
  
 String();  
 String(const String& other);  
 String(String&& other) noexcept;  
 ~String() { dataPtr\_.release(); }  
 size\_t getLen() const;  
  
 String& operator=(const String& other);  
 String& operator=(String&& other) noexcept;  
  
 char& operator[](size\_t index);  
 const char& operator[](size\_t index) const;  
 const char\* getRaw() const;  
  
 auto operator<=>(const String& other) const { return length\_ <=> other.length\_; }  
  
 friend void printString(const String& str);  
 friend void readString(String& str);  
 friend bool operator!=(const String& str1, const String& str2);  
};  
void printString(const String& str);  
void readString(String& str);  
} // namespace str

Файл string.cc:

#include <iostream>  
#include <l2/include/screens.hh>  
#include <limits>  
  
using namespace std;  
namespace str {  
String::String(const char\* str) : length\_{0} {  
 for (; str[length\_]; length\_++);  
 ++length\_;  
 dataPtr\_ = make\_unique\_for\_overwrite<char[]>(length\_);  
 ranges::copy(str, str + length\_, dataPtr\_.get());  
}  
  
String::String() : length\_{1}, dataPtr\_{make\_unique<char[]>(length\_)} {};  
  
String::String(const String& other) : length\_{other.length\_}, dataPtr\_{make\_unique<char[]>(length\_)} {  
 ranges::copy(other.dataPtr\_.get(), other.dataPtr\_.get() + length\_, dataPtr\_.get());  
};  
String::String(String&& other) noexcept : length\_{other.length\_}, dataPtr\_{make\_unique\_for\_overwrite<char[]>(length\_)} {  
 dataPtr\_ = std::move(other.dataPtr\_);  
};  
  
String& String::operator=(const String& other) {  
 length\_ = other.length\_;  
 resizeD\_(length\_);  
 ranges::copy(other.dataPtr\_.get(), other.dataPtr\_.get() + length\_, dataPtr\_.get());  
 return \*this;  
};  
String& String::operator=(String&& other) noexcept {  
 length\_ = other.length\_;  
 dataPtr\_ = std::move(other.dataPtr\_);  
 return \*this;  
};  
  
size\_t String::getLen() const { return length\_; }  
  
char& String::operator[](size\_t index) {  
 if (index >= length\_) throw invalid\_argument("Index out of range");  
 return dataPtr\_[index];  
}  
const char& String::operator[](size\_t index) const {  
 if (index >= length\_) throw invalid\_argument("Index out of range");  
 return dataPtr\_[index];  
}  
  
void String::resizeD\_(size\_t newLen) {  
 length\_ = newLen;  
 auto tmp = make\_unique<char[]>(length\_);  
 dataPtr\_ = std::move(tmp);  
}  
void String::resize\_(size\_t newLen) {  
 length\_ = newLen;  
 auto tmp = make\_unique<char[]>(length\_);  
 std::ranges::copy(dataPtr\_.get(), dataPtr\_.get() + length\_, tmp.get());  
 dataPtr\_ = std::move(tmp);  
}  
void String::readFromStream\_(istream& is) {  
 char tmp;  
 size\_t counter = 0;  
 while (is.get(tmp) && tmp != '\n') {  
 if (counter >= length\_ - 1) {  
 resize\_(length\_ \* 2);  
 }  
 dataPtr\_[counter] = tmp;  
 counter++;  
 }  
 resize\_(counter + 1);  
}  
const char\* String::getRaw() const { return dataPtr\_.get(); }  
bool operator!=(const String& str1, const String& str2) { return str1.length\_ != str2.length\_; }  
  
void printString(const String& str) { cout << str.dataPtr\_.get(); }  
void readString(String& str) { str.readFromStream\_(cin); }  
} // namespace str

**4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

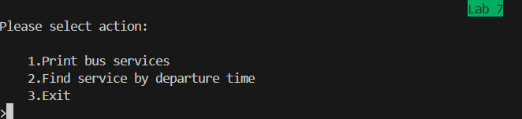


Рисунок 4.1 – Главное меню

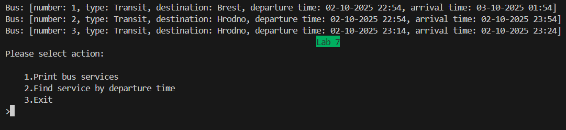


Рисунок 4.2 – Вывод всех автобусов

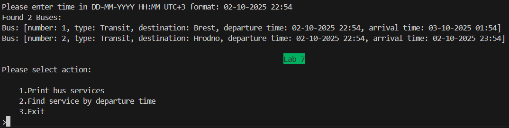


Рисунок 4.3 – Поиск по времени отправления

**5 ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы был успешно реализован класс "Рейс автобуса" с полями: номер рейса, тип автобуса, пункт назначения, время отправления и прибытия. Была продемонстрирована работа с потоками ввода/вывода и файлами в бинарном режиме.