Министерство образования Республики Беларусь
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ
О ЛАБОРАТОРНОЙ РАБОТЕ № 7
Потоки ввода/вывода. Работа с файлами

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 класс “Рейс автобуса”. Поля класса: номер рейса, тип автобуса, пункт назначения, время отправления, время прибытия на конечный пункт. Реализовать перегрузку операторов <<, >> для записи и чтения объектов в файл. Реализовать метод, который возвратит все рейсы, указанного времени отправления.

2 ДИАГРАММА КЛАССОВ

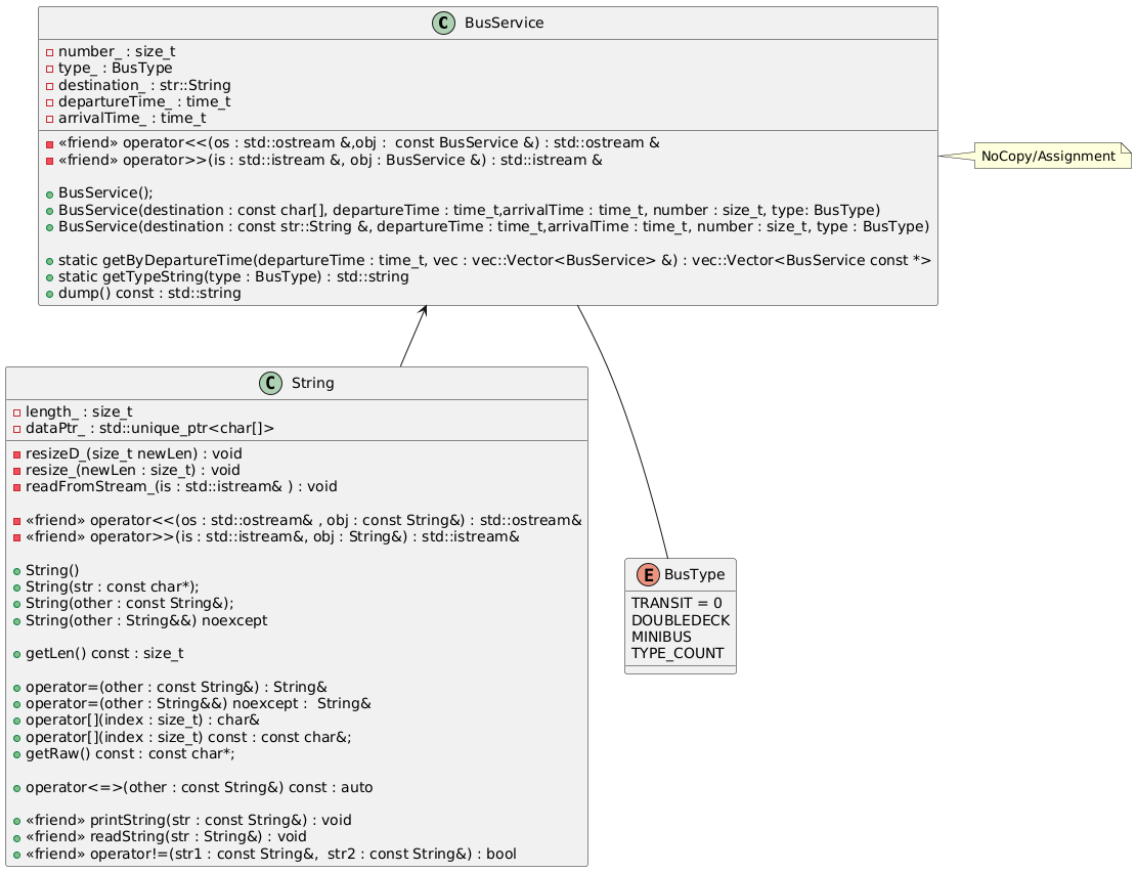

Рисунок 2.1 – Диаграмма классов

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

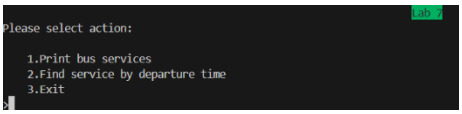

Рисунок 4.1 – Главное меню

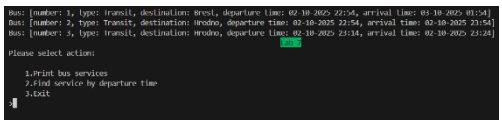

Рисунок 4.2 – Вывод всех автобусов

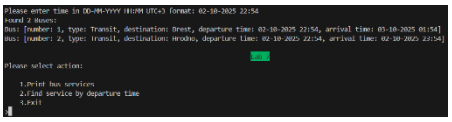

Рисунок 4.3 – Поиск по времени отправления

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.