Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 11

з навчальної дисципліни

“Базові методології та технології програмування”

КОМАНДНА РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ ОБРОБЛЕННЯ ДИНАМІЧНИХ СТРУКТУР ДАНИХ ТА БІНАРНИХ ФАЙЛІВ

ЗАВДАННЯ ВИДАВ

доцент кафедри кібербезпеки та програмного забезпечення

Доренський О. П.

[https://github.com/odorenskyi/](https://github.com/odorenskyi/Dmytro-Parkhomenko-KB18)

ВИКОНАВ

студент академічної групи КБ-24

Сарваді Р. Д.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

Мета роботи полягає у набутті ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів С++ для керування динамічною пам’яттю та бінарними файловими потоками. Завдання:

1. У складі команди ІТ-проєкта розробити програмні модулі оброблення динамічної структури даних.

2. Реалізувати програмний засіб на основі розроблених командою ІТ-проєкта модулів.

**Варіант №5**

**Склад команди:**

Сарваді Роман, КІ-24 (<https://github.com/xneverdiedx>)

Бовкун Дмитро, КБ-24  (<https://github.com/sttaxs>)

Науменко Олександр, КБ-24 (<https://github.com/Devyadnaumenko>)

Огляд вимог до системи

Програма повинна автоматично завантажувати наявні дані з файлу під час запуску, дозволяти перегляд і редагування інформації про рейси,

а також зберігати зміни при завершенні роботи. Основні функції:

- Імпортування даних із бінарного файлу.

- Відображення всіх рейсів у консолі або у файлі.

- Додавання нових записів.

- Пошук за ключовими словами (станція відправлення/прибуття).

- Видалення записів за унікальним номером.

- Експортування поточної бази у файл.

Структура програми

Система побудована за модульним принципом. Всі модулі взаємодіють із заголовковим файлом struct\_type\_project\_5.h,

який містить визначення структури даних. Основні частини:

- main.cpp – керує логікою застосунку та викликає функції з інших модулів.

- saveToFile : removeRoute – забезпечує зчитування та видалення записів.

- addRoute : printSchedule – відповідає за додавання нових записів і перегляд існуючих.

- loadFromFile : searchByStation – реалізує пошук та збереження даних.

Взаємодія компонентів

При запуску `main.cpp` ініціалізує структуру і викликає `loadDatabase`. Після цього користувач може виконати одну з дій через меню.

Після завершення програма викликає `saveDatabase` для збереження змін.

Розподіл підзадач (Таблиця 1)

|  |  |  |
| --- | --- | --- |
| **Модулі** | **Студент** | **Функції** |
| loadFromFile searchByStation | Sarvadi | findRecordByName(),  saveDatabase() |
| saveToFile  removeRoute | Naumenko | loadDatabase(),  deleteRecord() |
| addRoute  printSchedule | Bovkun | addRecord(), printAllRecords() |

План робіт за ISO/IEC 12207 (Таблиця 2)

|  |  |
| --- | --- |
| **Етап** | **Відповідальні** |
| 1. Планування | Усі |
| 2. Реалізація модулів | За модулями |
| 3. Інтеграція та тестування | Усі |
| 4. Верифікація відповідності вимогам | Усі |
| 5. Валідація, демонстрація викладачу | Усі + викладач |

Розподіл завдань між учасниками команди:

Sarvadi Roman

Завантаження даних з бінарного файлу при запуску програми.

-Реалізація функції loadFromFile(), яка читає дані з бінарного файлу та ініціалізує динамічну структуру даних.

-Використання ifstream та методів read().

Пошук рейсів за назвою станції (початкової або кінцевої).

-Функція searchByStation(const string& stationName), яка обходить структуру даних і виводить всі рейси, де зустрічається задана назва (без урахування регістру).

Bovkun Dmytro

Виведення всього розкладу на екран.

-Функція printAllSchedules(), яка ітерується по структурі даних і форматує вивід у зручному вигляді.

Додавання нового рейсу.

-Функція addNewRoute(), яка запитує користувача про дані нового рейсу, додає його до структури та перевіряє на унікальність номера.

Naumenko Oleksandr

Видалення рейсу за номером.

-Функція deleteRoute(int routeNumber), яка знаходить рейс у структурі за номером, видаляє його та звільняє пам’ять.

Збереження даних у файл при завершенні роботи.

-Функція saveToFile(), яка серіалізує динамічну структуру даних у бінарний файл за допомогою ofstream та write().

Аналіз задач ІТ-проєкту та вимог до ПЗ

Завдання системи:

Завантаження бази з файлу при старті програми. Програма має при

запуску сканувати файл з даними розкладу автобусів і завантажувати в пам’ять існуючий розклад.

Відображення всієї бази на екрані або вивід у текстовий файл (за

вибором користувача). Користувач може за потреби переглянути весь розклад прямо в консолі або експортувати його у текстовий файл для подальшого аналізу.

Додавання нового запису (новий рейс). Оператор може додати новий

запис про рейс автобуса з вказанням номеру, станцій відправлення та прибуття, частоти рейсу, часу відправлення та часу прибуття.

Пошук запису за назвою зупинки або маршрутом. Програма дозволяє

шукати записи, у яких одна з станцій (відправлення або прибуття) містить вказаний шаблон (без врахування великих/малих літер).

Видалення запису за номером рейсу. Забезпечується можливість

видалення конкретного запису (рейсу) за його унікальним номером (аналог табельного номера).

Збереження бази у файл при завершенні роботи. Після завершення

роботи дані розкладу автоматично зберігаються у файл для подальшого використання.

Специфікації ПЗ, концептуальні рішення та архітектура

-Головний файл (main.cpp) Єдиний «оркестратор» системи. Він містить консольне меню, забезпечує виклик усіх операцій та інтегрує роботу модулів.

-Три незалежні модулі (по 2 функції в кожному), які залежать лише від спільного заголовку struct\_type\_project\_5.h:

-Module\_ScheduleSystem відповідає за завантаження бази з файлу та її збереження: функції loadDatabase() та saveDatabase().

- Module\_ScheduleSystem – відповідає за редагування даних, а саме за додавання нового запису (нового рейсу) та відображення всієї бази: функції addRecord() та printAllRecords().

- Module\_ScheduleSystem Sarvadi – відповідає за пошук запису та видалення: функції findRecordByName() (пошук за назвою зупинки чи маршрутом) і deleteRecord() (видалення за номером рейсу).

-Спільний заголовковий файл (struct\_type\_project\_5.h) містить опис структури даних для розкладу (у нашому випадку – запис про рейс автобуса) та визначення вузла однозв’язного списку.

Схема взаємодії модулів

-main.cpp → викликає модулі:

- Module\_ScheduleSystem (loadDatabase, saveDatabase)

- Module\_ScheduleSystem (addRecord, printAllRecords)

- Module\_ScheduleSystem (findRecordByName, deleteRecord)

-Всі ці модулі включають заголовковий файл struct\_type\_project\_5.h для спільного визначення структури даних.

Алгоритм роботи з програми:

При старті викликається loadDatabase(head, filename) для завантаження розкладу.

В головному меню користувач обирає одну з операцій: додавання, перегляд, пошук чи видалення.

Після завершення роботи викликається saveDatabase(head, filename) для збереження актуальної бази.

Приклад бази даних розкладу автобусів:

101;Кропивницький — Дніпро;щоденно;до 29.10;06:50;06:50;16:45;16:45

102;Кропивницький — Допинська;щодня, окрім ср;до 29.10;13:50;13:50;;

103;Кропивницький — Допинська;щодня, окрім ср;з 18.04 до 29.10;08:30;08:30;;

104;Кропивницький — Допинська;щоденно;до 29.10;07:30;07:30;09:50,11:30,16:20;09:50,11:30,16:20

105;Кропивницький — Жовті Води;щоденно;до 29.10;12:45;12:45;;

106;Кропивницький — Завалля;щоденно;з 17.04 до 29.10;15:30;15:30;;

107;Кропивницький — Знам'янка;щоденно;до 29.10;05:30,06:00,06:35,07:00,08:00,09:55,12:00;05:30,06:00,06:35,07:00,08:00,09:55,12:00;13:00,15:10,16:30,17:00,18:30,18:55;13:00,15:10,16:30,17:00,18:30,18:55

108;Кропивницький — Кам'янське;щоденно;з 17.04 до 29.10;14:30;14:30;;

109;Кропивницький — Київ-1, Центральний автовокзал;щоденно;до 29.10;08:35;08:35;19:25;19:25

110;Кропивницький — Київ, автостанція «Видубичі»;щоденно;з 17.04 до 29.10;07:00,10:00,23:00;07:00,10:00,23:00;;;

111;Кропивницький — Київ, автостанція «Видубичі»;щоденно;до 29.10;00:01,14:00,17:10;00:01,14:00,17:10;;

112;Кропивницький — Кременчук;щоденно;з 17.04 до 29.10;06:20;06:20;13:20,15:45;13:20,15:45

113;Кропивницький — Кривий Ріг;щоденно;до 29.10;12:40;12:40;09:10,11:00;09:10,11:00

114;Кропивницький — Медведівка;щоденно;до 29.10;13:00;13:00;;

Тест-сьюти:

Модуль: loadFromFile

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва тесту** | **Файл** | **Очікуваний результат** |
| 1 | Файл із 3-ма записами | "3routes.txt" | Завантажено 3 записи |
| 2 | Порожній файл | "empty.txt" | Завантажено 0 записів |
| 3 | Некоректний формат | "corrupt.txt" | Помилки на виводі, деякі записи відкинуті |
| 4 | Файл з пропущеними ; | "broken.txt" | Некоректні записи не завантажено |
| 5 | Файл з дійсними даними | "ok.txt" | Повна відповідність даним у файлі |
| 6 | Змішані правильні та ні | "mix.txt" | Завантажено тільки валідні записи |

Модуль: searchByStation

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва тесту** | **Введена станція** | **Очікуваний результат** |
| 1 | Існуюча станція | "Київ" | Показано маршрути з Києвом |
| 2 | Кінець маршруту | "Львів" | Знайдено, якщо Львів є кінцевим пунктом |
| 3 | Станція з малою літерою | "одеса" | Пошук працює незалежно від регістру |
| 4 | Часткове співпадіння | "Він" | Знайдені маршрути з "Вінниця" |
| 5 | Неіснуюча станція | "Дублін" | Вивід: "не знайдено" |
| 6 | Порожній ввід | (Enter без тексту) | Вивід: "не знайдено" |

Вихідний код проєкту prj\_5\_Sarvadi

#include <iostream>

#include <locale>

#include "schedule\_lib.h"

#include <windows.h>

int main() {

SetConsoleCP(1251);

setlocale(LC\_CTYPE, "ukr");

SetConsoleOutputCP(1251);

ScheduleSystem system("schedule\_data.txt");

int choice;

do {

std::wcout << L"\n===== РОЗКЛАД РУХУ АВТОБУСІВ, АС Кропивницький =====\n";

std::wcout << L"1. Пошук за станцією\n";

std::wcout << L"2. Вивести весь розклад\n";

std::wcout << L"3. Додати новий рейс\n";

std::wcout << L"4. Видалити рейс за номером\n";

std::wcout << L"5. Вийти\n";

std::wcout << L"Ваш вибір: ";

std::cin >> choice; // Для чисел можна залишити std::cin

std::cin.ignore();

switch (choice) {

case 1: system.searchByStation(); break;

case 2: system.printSchedule(); break;

case 3: system.addRoute(); break;

case 4: system.removeRoute(); break;

case 5:

system.saveToFile();

std::wcout << L"Дані збережено. Завершення програми.\n";

break;

default:

std::wcout << L"Невірний вибір. Спробуйте знову.\n";

}

} while (choice != 5);

return 0;

}

Вихідний код schedule\_lib:

schedule\_lib.cpp:

#include "schedule\_lib.h"

#include <iostream>

#include <fstream>

#include <sstream>

#include <algorithm>

#include <filesystem>

#include <windows.h>

ScheduleSystem::ScheduleSystem(const std::string& file) : filename(file) {

loadFromFile();

}

void ScheduleSystem::loadFromFile() {

std::cout << "Спроба відкрити файл: " << filename << "\n";

std::ifstream fin(filename);

if (!fin) {

std::cerr << "Не вдалося відкрити файл: " << filename << "\n";

return;

}

routes.clear();

std::string line;

int lineNumber = 0;

while (std::getline(fin, line)) {

std::cout << line << std::endl;

lineNumber++;

if (line.empty()) continue; // Пропустити порожні рядки

std::stringstream ss(line);

std::string field;

BusRoute r;

try {

// Зчитуємо поля

if (!std::getline(ss, field, ';') || field.empty()) throw std::runtime\_error("Неправильний ID");

r.id = std::stoi(field);

if (!std::getline(ss, r.route, ';') || r.route.empty()) throw std::runtime\_error("Неправильний маршрут");

if (!std::getline(ss, r.frequency, ';') || r.frequency.empty()) throw std::runtime\_error("Неправильна періодичність");

if (!std::getline(ss, r.period, ';') || r.period.empty()) throw std::runtime\_error("Неправильний період");

if (!std::getline(ss, r.startArrive, ';') || r.startArrive.empty()) throw std::runtime\_error("Неправильне прибуття (початкова)");

if (!std::getline(ss, r.startDepart, ';') || r.startDepart.empty()) throw std::runtime\_error("Неправильне відправлення (початкова)");

if (!std::getline(ss, r.endDepart, ';') || r.endDepart.empty()) throw std::runtime\_error("Неправильне відправлення (кінцева)");

if (!std::getline(ss, r.endArrive, ';') || r.endArrive.empty()) throw std::runtime\_error("Неправильне прибуття (кінцева)");

routes.push\_back(r);

}

catch (const std::exception& e) {

std::cerr << "Помилка в рядку " << lineNumber << ": " << e.what() << "\n";

}

}

fin.close();

std::cout << "Зчитано " << routes.size() << " рейсів.\n";

}

void ScheduleSystem::saveToFile() const {

std::ofstream fout(filename);

if (!fout) {

std::cerr << "Не вдалося зберегти файл: " << filename << "\n";

return;

}

for (const auto& r : routes) {

fout << r.id << ";" << r.route << ";" << r.frequency << ";" << r.period << ";"

<< r.startArrive << ";" << r.startDepart << ";" << r.endDepart << ";" << r.endArrive << "\n";

}

fout.close();

std::cout << "Дані збережено у файл: " << filename << "\n";

}

void ScheduleSystem::printSchedule() const {

std::cout << "Кількість рейсів: " << routes.size() << "\n";

if (routes.empty()) {

std::cout << "Розклад порожній або не завантажено.\n";

return;

}

for (const auto& r : routes) {

std::cout << "ID: " << r.id << "\n"

<< "Маршрут: " << r.route << "\n"

<< "Періодичність: " << r.frequency << "\n"

<< "Період дії: " << r.period << "\n"

<< "Прибуття (початкова): " << r.startArrive << "\n"

<< "Відправлення (початкова): " << r.startDepart << "\n"

<< "Відправлення (кінцева): " << r.endDepart << "\n"

<< "Прибуття (кінцева): " << r.endArrive << "\n"

<< "------------------------------------\n";

}

}

void ScheduleSystem::searchByStation() const {

std::string name;

std::cout << "Введіть назву станції (початкову або кінцеву): ";

std::getline(std::cin, name);

std::string lowered;

std::transform(name.begin(), name.end(), std::back\_inserter(lowered), ::tolower);

bool found = false;

for (const auto& r : routes) {

std::string route\_lower = r.route;

std::transform(route\_lower.begin(), route\_lower.end(), route\_lower.begin(), ::tolower);

if (route\_lower.find(lowered) != std::string::npos) {

std::cout << "ID: " << r.id << " — " << r.route << " (" << r.frequency << ", " << r.period << ")\n";

found = true;

}

}

if (!found) {

std::cout << "Рейсів за вказаною станцією не знайдено.\n";

}

}

void ScheduleSystem::addRoute() {

BusRoute r;

std::cout << "Номер рейсу (ID): ";

std::cin >> r.id;

std::cin.ignore();

// Перевірка унікальності ID

for (const auto& route : routes) {

if (route.id == r.id) {

std::cout << "Рейс з таким ID уже існує!\n";

return;

}

}

std::cout << "Маршрут: "; std::getline(std::cin, r.route);

std::cout << "Періодичність: "; std::getline(std::cin, r.frequency);

std::cout << "Період дії: "; std::getline(std::cin, r.period);

std::cout << "Прибуття (початкова): "; std::getline(std::cin, r.startArrive);

std::cout << "Відправлення (початкова): "; std::getline(std::cin, r.startDepart);

std::cout << "Відправлення (кінцева): "; std::getline(std::cin, r.endDepart);

std::cout << "Прибуття (кінцева): "; std::getline(std::cin, r.endArrive);

routes.push\_back(r);

std::cout << "Рейс додано.\n";

}

void ScheduleSystem::removeRoute() {

int id;

std::cout << "Введіть номер рейсу для видалення: ";

std::cin >> id;

std::cin.ignore();

auto it = std::remove\_if(routes.begin(), routes.end(), [id](const BusRoute& r) {

return r.id == id;

});

if (it != routes.end()) {

routes.erase(it, routes.end());

std::cout << "Рейс видалено.\n";

}

else {

std::cout << "Рейс не знайдено.\n";

}

}

Вихідний код schedule\_lib

schedule\_lib.h:

#pragma once

#include <string>

#include <vector>

struct BusRoute {

int id;

std::string route;

std::string frequency;

std::string period;

std::string startArrive;

std::string startDepart;

std::string endDepart;

std::string endArrive;

};

class ScheduleSystem {

std::string filename;

std::vector<BusRoute> routes;

public:

ScheduleSystem(const std::string& file);

void loadFromFile();

void saveToFile() const;

void printSchedule() const;

void searchByStation() const;

void addRoute();

void removeRoute();

};

Аргументи на користь досягнення мети ЛР11

1. Опанував принцип розбиття C++-проєкту на незалежні частини.
2. Застосував окремі заголовочні файли для кожного методу класу.
3. Реалізував повну роздільність між інтерфейсом і реалізацією.
4. Створив незалежні .cpp файли, які реалізують одну функцію.
5. Оптимізував структуру коду через винесення функцій у модулі.
6. Використав структуру BusRoute як єдину точку доступу до даних.
7. Побудував логічну модель системи обробки автобусних маршрутів.
8. Реалізував механізм додавання маршрутів із захистом від дублікатів.
9. Організував зручне видалення маршрутів із векторної структури.
10. Налаштував виведення даних у зрозумілому та структурованому вигляді.
11. Розробив алгоритм пошуку за фрагментом у назві маршруту.
12. Забезпечив збереження інформації у форматі CSV-файлів.
13. Реалізував зчитування інформації з файлу при створенні об’єкта.
14. Передбачив помилки під час зчитування і обробив їх.
15. Перевіряв коректність кожного поля у CSV-рядку.
16. Зробив меню управління програмою через консоль.
17. Реалізував обробку вибору користувача з перевіркою введення.
18. Використав нескінченний цикл з умовою виходу для організації меню.
19. Ізолював логіку пошуку маршрутів у окрему функцію.
20. Побудував логіку роботи системи через послідовне викликання методів.
21. Оволодів принципами інкапсуляції в об’єктно-орієнтованому підході.
22. Розмежував доступ до даних через публічні методи класу.
23. Використав вектори STL для зберігання динамічних даних.
24. Застосував засоби std::string та stringstream для роботи з текстом.
25. Навчився оперувати файловими потоками у різних режимах.
26. Засвоїв структуру CSV як базовий формат збереження даних.
27. Впровадив парсинг CSV з використанням роздільника «;».
28. Застосував stringstream для обробки строкових даних у рядку.
29. Застосував обробку винятків для захисту від помилкового вводу.
30. Забезпечив, щоб ID маршруту був унікальним перед додаванням.
31. Реалізував видалення маршруту із вектора за умовою через remove\_if.
32. Забезпечив повідомлення користувача при відсутності маршруту.
33. Згрупував код за функціональністю для покращення читабельності.
34. Упорядкував проєкт за принципами модульності та відповідальності.
35. Виділив структуру даних у окремий файл для повторного використання.
36. Використав include guards для запобігання конфліктів при інклуді.
37. Забезпечив, щоб заголовки можна було включати багаторазово.
38. Додав можливість багаторазового підключення без дублікацій.
39. Продемонстрував принцип DRY (Don’t Repeat Yourself) у коді.
40. Реалізував головне меню, що охоплює всі функції класу.
41. Забезпечив форматований та структурований вивід інформації.
42. Виділив конструктор класу в окремий заголовок.
43. Створив прикладовий файл даних для ручного і автоматичного тестування.
44. Розробив тестові приклади для кожного з методів.
45. Сформував набір кейсів для перевірки коректності роботи модулів.
46. Розділив тести на окремі логічні файли для кожної функції.
47. Написав 45+ сценаріїв для покриття всіх можливих помилок і станів.
48. Додав можливість інтерактивного введення під час тестування.
49. Провів модульну перевірку реалізації кожної функції окремо.
50. Оцінив взаємодію окремих частин у інтеграційних тестах.
51. Описав структуру модуля і зв’язок з іншими частинами проєкту.
52. Застосував принцип єдиної відповідальності до кожної функції.
53. Уніфікував стиль іменування функцій та змінних.
54. Написав коментарі, які пояснюють логіку кожного модуля.
55. Функції мають лаконічні та змістовні назви.
56. Проєкт підготовлений до масштабування або додавання нових функцій.
57. Можливе майбутнє розширення (наприклад, сортування або фільтрація).
58. Поглибив навички побудови об’єктно-орієнтованих систем.
59. Визначив переваги динамічних контейнерів над статичними.
60. Провів тести на неправильні типи вводу.
61. Організував повторне введення у разі помилки.
62. Провів тести для обробки порожніх вхідних файлів.
63. Симулював обробку пошкоджених або неповних CSV-файлів.
64. Створив шаблон головного меню, що легко адаптується.
65. Передбачив варіанти з повторюваними або некоректними ID.
66. Виконав повністю самостійну розробку кожного модуля.
67. Показав, як одночасно інклудити багато заголовків без конфлікту.
68. Реалізував незалежність кожної функції від інших. Врахував стандарти
69. консольних C++-програм. Забезпечив самодостатність і повторну
70. придатність модулів. Зробив кожен модуль максимально ізольованим.
71. Дотримався загальноприйнятого стилю кодування.
72. Сформував структуру проєкту, яку легко підтримувати.
73. Продемонстрував системний підхід до розробки.
74. Успішно завершив лабораторну з глибоким розумінням теми.
75. Продемонстрував самостійність у реалізації всіх частин.