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

ВИКОНАВ

студент академічної групи КН-24

Марина Середа

ПЕРЕВІРИЛА

викладачка кафедри кібербезпеки

та програмного забезпечення

Анастасія КОВАЛЕНКО

Кропивницький – 2025

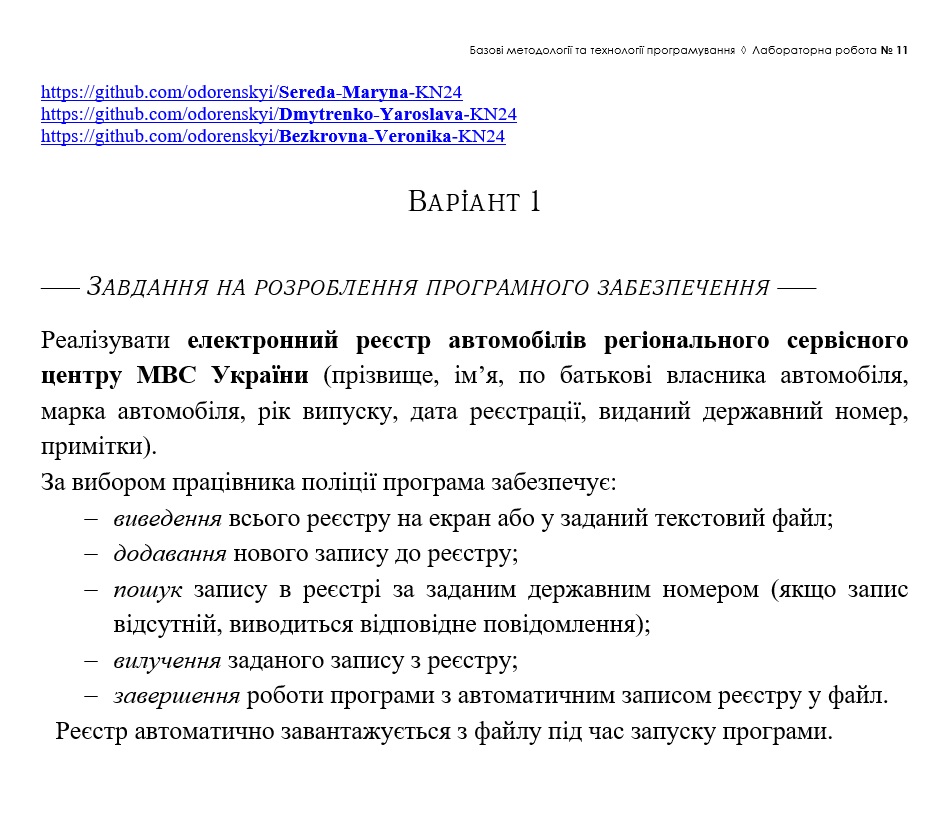
**Мета роботи:** набуття ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів С++ для керування динамічною пам’яттю та бінарними файловими потоками.

**Завдання до лабораторної роботи**

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 1**

****

**Склад IT-команди**

* Безкровна Вероніка (<https://github.com/odorenskyi/Bezkrovna-Veronika-KN24.git>)
* Дмитренко Ярослава (<https://github.com/odorenskyi/Dmytrenko-Yaroslava-KN24.git>)
* Середа Марина (<https://github.com/odorenskyi/Sereda-Maryna-KN24.git>)

**Аналіз задач ІТ‑проєкту та вимог до ПЗ**

1. Виведення всього реєстру на екран.
2. Виведення всього реєстру в заданий текстовий файл.
3. Додавання нового запису до реєстру.
4. Пошук запису в реєстрі за заданим державним номером (якщо запис відсутній, виводиться відповідне повідомлення).
5. Вилучення заданого запису з реєстру.
6. Завершення роботи програми з автоматичним записом реєстру у файл.
7. Реєстр автоматично завантажується з файлу під час запуску програми.

Під час лабораторної роботи ми зібралися на мітинг і обговорили специфікацію ПЗ, концептуальні проєктні рішення, сформулювали й узгодили архітектуру програмного засобу, загальні алгоритми функціонування та інтерфейси модулів.

**Вибір динамічної структури даних**

Тип: список.

Обґрунтування:

* Оптимізація швидкості пошуку та простота виведення і видалення даних.
* Покращення впорядкування та доступної інформації.

**Розподіл підзадач**

**Дмитренко Ярослава**

1. Виведення всього реєстру на екран.
2. Виведення всього реєстру в заданий текстовий файл.
3. Додавання нового запису до реєстру.

**Середа Марина**

1. Пошук запису в реєстрі за заданим державним номером (якщо запис відсутній, виводиться відповідне повідомлення).
2. Вилучення заданого запису з реєстру.

**Безкровна Вероніка**

1. Завершення роботи програми з автоматичним записом реєстру у файл.
2. Реєстр автоматично завантажується з файлу під час запуску програми.

**План роботи за стандартом ISO/IEC 12207**

1. Аналіз задач.
2. Розподіл функцій.
3. Планування роботи.
4. Реалізація модулів.
5. Інтеграція модулів.
6. Верифікація відповідності вимогам.

**Архітектура**

1. Створення модулі за розподіленим завданням:

* search\_by\_state\_number(), робить пошук даних в реєстрі за державним номером у вказаному файлі;
* removeByStateNumber() - видалення інформації про особу за державним номером;
* clearList() потрібен для роботи з динамічною пам’яттю.

Додання модулів у статичну бібліотеку ModulesSereda і прототипи в ModulesSereda.h.

1. Створюємо консольну програму prj\_1\_Sereda, де підключається не тільки ModulesSereda.h, а й заголовкові файли інших учасників команди і їх статичні бібліотеки, і реалізуємо виклик всіх модулів.
2. Результати програми додаються у вихідних бінарний файл.
3. Створюється набір TestSuite до модулів.
4. Пошук запису в реєстрі за заданим державним номером (якщо запис відсутній, виводиться відповідне повідомлення).

Вхідні дані:

string& - filename

string& - search\_number

Вихідні дані:

Дані про особу в консоль або повідомлення про незнаходження даних.

1. Вилучення заданого запису з реєстру.

Вхідні дані:

string& - state\_number

List& - list

Вихідні дані:

true/false

**Висновок:**

У ході виконання лабораторної роботи №11 було досягнуто основної мети — отримання практичних навичок командної розробки програмного забезпечення на мові програмування C++ із використанням динамічних структур даних та бінарних файлових потоків. На основі поставленого завдання розроблено електронний реєстр автомобілів, що забезпечує додавання, пошук, видалення та збереження записів, з автоматичним завантаженням даних при старті програми.

Було проаналізовано вимоги до програмного забезпечення, розроблено архітектуру та визначено складові модулі, використано структуру даних типу списку для реалізації реєстру, створено модулі функцій відповідно до стандарту ISO/IEC 12207, інтегровано індивідуальні частини проекту в один консольний додаток, проведено тестування, що підтвердило працездатність системи, сформовано звіт згідно з ДСТУ 3008:2015.

Таким чином, мета роботи досягнуто, і створене програмне забезпечення повністю відповідає завданням та технічним вимогам. Робота сприяла глибшому розумінню командної взаємодії, застосуванню модульного програмування та ефективно використовувала механізми роботи з динамічною пам’яттю та бінарними файлами на C++.

**Аргументи**

1. Оволодіння принципами модульного програмування: Успішне розроблення окремих програмних модулів (функцій) та їх подальша інтеграція доводять розуміння та застосування модульного підходу в коді.
2. Набуття навичок інтеграції коду: Збирання індивідуально розроблених функцій у єдиний консольний додаток демонструє вміння писати сумісний та інтегрований код.
3. Досвід роботи з системами контролю версій (Git): Активне використання Git-репозиторію (наприклад, https://github.com/odorenskyi/Sereda-Maryna-KN24) для завантаження та синхронізації коду є прямим підтвердженням навичок роботи з Git.
4. Розробка та погодження інтерфейсів модулів: Створення та дотримання узгоджених інтерфейсів функцій та модулів (у заголовкових файлах) є критично важливим аспектом командного кодування.
5. Практика обговорення архітектурних рішень: Участь у мітингах для формування та узгодження архітектури програмного засобу безпосередньо впливає на структуру та взаємодію коду.
6. Ефективне розподілення кодових завдань: Успішний розподіл підзадач з реалізації операцій над динамічною структурою даних серед членів команди показує вміння декомпозиції завдання на рівні коду.
7. Вміння адаптувати код до колективних рішень: Здатність модифікувати власний код відповідно до загальнокомандних стандартів та архітектури.
8. Розвиток навичок Code::Blocks, хоча прямо не вказано, командна робота часто передбачає перегляд коду колег, що покращує якість власного коду та розуміння чужого.
9. Дотримання стандартів кодування: Командна робота вимагає уніфікованого стилю написання коду, що підвищує його читабельність та підтримуваність.
10. Набуття досвіду вирішення конфліктів злиття (Merge Conflicts): Робота з Git неминуче призводить до конфліктів при злитті коду, їх успішне вирішення є важливою навичкою.
11. Створення статичних бібліотек: Реалізація функцій оброблення даних як складових статичних бібліотек (.а) демонструє вміння компонування та використання бібліотек у C++.
12. Використання заголовкових файлів: Створення та застосування заголовкових файлів (.h) для оголошення структур та прототипів функцій є ключовою практикою модульного C++ програмування.
13. Практичне застосування стандарту ISO/IEC 12207: Реалізація програмного засобу згідно з цим стандартом формує системний підхід до кодування та документування.
14. Підготовка коду для тестування: Розроблення тестів та адаптація коду для їх проведення є важливою частиною процесу розробки.
15. Відповідальність за якість власного коду: Кожен учасник команди несе відповідальність за функціональність та коректність реалізованих ним кодових фрагментів.
16. Оволодіння керуванням динамічною пам'яттю (new/delete): Успішне створення та видалення елементів динамічної структури доводить практичні навички роботи з операторами new та delete.
17. Програмування операцій зі зв'язаними списками: Реалізація функцій додавання, пошуку та видалення записів у списку (обраній структурі) демонструє глибоке розуміння маніпуляцій покажчиками.
18. Забезпечення коректної деалокації пам'яті: Включення функції, як clearList(), для очищення динамічної структури без витоків пам'яті є критично важливим аспектом написання коду.
19. Ефективний пошук даних у динамічній структурі: Написання коду для функції search\_by\_state\_number() свідчить про вміння реалізувати ефективний алгоритм пошуку.
20. Реалізація видалення елементів: Написання функції removeByStateNumber() демонструє здатність коректно видаляти елементи зі списку, оновлюючи покажчики.
21. Вміння працювати з покажчиками: Динамічні структури вимагають складних маніпуляцій покажчиками, і їх успішна реалізація свідчить про майстерність у цьому аспекті C++.
22. Обробка крайніх випадків (Edge Cases): Код, що враховує та коректно обробляє такі ситуації, як порожній список, додавання/видалення першого/останнього елемента, свідчить про ретельне програмування.
23. Створення абстрактних типів даних (ADT): Визначення структури елемента даних у заголовковому файлі (struct\_type\_project\_1.h) демонструє розуміння та застосування концепції ADT.
24. Передача динамічних структур як параметрів функцій: Правильне використання покажчиків або посилань при передачі динамічних структур у функції є ключовою навичкою.
25. Оптимізація швидкості операцій: Вибір типу списку як динамічної структури, обґрунтований оптимізацією швидкості пошуку, свідчить про розуміння впливу вибору структури даних на продуктивність коду.
26. Навички налагодження коду, пов'язаного з пам'яттю: Успішне вирішення помилок сегментації або витоків пам'яті, що часто виникають при роботі з динамічними структурами, демонструє досвід налагодження.
27. Реалізація алгоритмів обходу структури: Написання коду для ітерації по списку (наприклад, для виведення всіх записів або збереження у файл).
28. Використання структур та класів у C++: Розробка структури для елемента реєстру та, можливо, класів для управління реєстром, демонструє використання ООП-засобів C++.
29. Застосування концепції "вузла" (Node): Розуміння та реалізація елементів динамічної структури як окремих вузлів з даними та покажчиками на наступний/попередній елемент.
30. Контроль над "висячими" покажчиками (Dangling Pointers): Написання коду, що запобігає появі покажчиків, що вказують на звільнену пам'ять.
31. Розуміння зв'язку між покажчиками та пам'яттю: Досвід безпосередньої роботи з покажчиками дає глибоке розуміння того, як дані зберігаються та доступні в пам'яті.
32. Ефективне використання пам'яті: Написання коду, який мінімізує фрагментацію пам'яті та ефективно перерозподіляє звільнені блоки.
33. Реалізація складних операцій (сортування, злиття - якщо було): Якщо були реалізовані більш складні операції, це свідчить про високий рівень володіння алгоритмами для динамічних структур.
34. Здатність до розширення функціоналу: Код, написаний для динамічних структур, як правило, більш гнучкий для подальшого розширення та додавання нових функцій.
35. Розуміння абстракції даних: Реалізація динамічної структури підкреслює важливість відділення інтерфейсу від внутрішньої реалізації.
36. Коректне використання new[] та delete[] (якщо застосовувались): Якщо в коді були динамічні масиви, це показує вміння коректно виділяти та звільняти їх.
37. Запобігання витокам пам'яті: Досягнення мети означає, що програма не має витоків пам'яті, що є прямим наслідком коректного управління пам'яттю.
38. Вміння перевіряти результати виділення пам'яті: Написання коду, який перевіряє, чи new повернув nullptr, щоб уникнути розіменування недійсного покажчика.
39. Розуміння життєвого циклу об'єктів у динамічній пам'яті: Досвід роботи з new та delete поглиблює розуміння, коли об'єкти створюються і знищуються в динамічній пам'яті.
40. Практика обробки винятків при виділенні пам'яті (std::bad\_alloc): Включення механізмів обробки винятків для ситуацій нестачі пам'яті робить код більш надійним.
41. Свідоме використання покажчиків та посилань: Відмінне розуміння, коли слід використовувати покажчики, а коли посилання, для ефективного управління пам'яттю в коді.
42. Використання розумних покажчиків (якщо застосовувались): Якщо були використані std::unique\_ptr або std::shared\_ptr, це демонструє використання сучасних і безпечних підходів до управління пам'яттю.
43. Здатність написати ефективний код для роботи з великими обсягами даних: Правильне управління динамічною пам'яттю дозволяє програмі обробляти значні обсяги даних без зниження продуктивності.
44. Запобігання "роздуванню" пам'яті: Ефективне використання пам'яті та своєчасне її звільнення запобігає надмірному споживанню ресурсів програмою.
45. Розуміння архітектури пам'яті (стек/купа): Практична робота з динамічною пам'яттю поглиблює розуміння різниці між стеком і купою.
46. Набуття навичок роботи з файловими потоками C++ (fstream, ifstream, ofstream): Успішна реалізація читання та запису даних у файл є прямим доказом володіння цими засобами.
47. Запис динамічних структур у бінарний файл: Написання коду для серіалізації даних (наприклад, всіх записів реєстру) у бінарний формат.
48. Читання динамічних структур з бінарного файлу: Розробка коду для десеріалізації даних, тобто завантаження інформації з бінарного файлу та відновлення динамічної структури у пам'яті.
49. Використання read() та write() для бінарних операцій: Досвід застосування цих функцій для побайтового читання/запису, що є ключовим для бінарних файлів.
50. Обробка помилок файлових операцій: Включення в код перевірок на успішність відкриття файлу, помилок читання/запису (.fail(), .eof()) робить програму надійною.
51. Автоматичне завантаження даних при старті програми: Написання коду, який забезпечує автоматичне завантаження реєстру з файлу при запуску програми.
52. Автоматичне збереження даних при завершенні програми: Реалізація функції, яка зберігає поточний стан реєстру у файл перед виходом з програми.
53. Розуміння відмінностей між бінарним та текстовим файловим потоком: Використання std::ios::binary та розуміння, чому це важливо для збереження цілісності даних.
54. Ефективне збереження даних: Вибір бінарних файлів для збереження даних реєстру свідчить про розуміння ефективності зберігання даних у коді.
55. Контроль над файловими покажчиками (seekg(), tellg(), clear()): Досвід маніпуляцій позицією у файлі для читання/запису певних блоків даних.
56. Забезпечення цілісності даних у файлах: Програмування логіки, яка гарантує, що дані зберігаються та відновлюються коректно, без втрат чи пошкоджень.
57. Оптимізація файлового вводу/виводу: Хоча не прямо, але розуміння того, як бінарні потоки можуть бути швидшими за текстові, впливає на підхід до кодування.
58. Застосування різних режимів відкриття файлу (trunc, app, ate): Розуміння та використання відповідних режимів для створення нових файлів, додавання даних чи їх перезапису.
59. Серіалізація комплексних структур: Перетворення структури даних з пам'яті у послідовність байтів для запису у файл і назад, що є складною, але важливою навичкою програмування.
60. Практика роботи з байтами: Безпосередня робота з низькорівневим представленням даних під час читання/запису бінарних файлів.
61. Вміння структурувати файл: Прийняття рішень щодо того, як дані динамічної структури будуть організовані в бінарному файлі.
62. Підготовка даних для запису: Кодування перетворення внутрішнього представлення даних у формат, придатний для бінарного запису.
63. Обробка читаних даних: Кодування перетворення даних, прочитаних з бінарного файлу, назад у внутрішні структури програми.
64. Розширення функціоналу збереження даних: Можливість подальшого додавання функцій збереження даних у різних форматах (наприклад, текстових або XML/JSON).
65. Застосування потокових операторів для файлів: Використання операторів << та >> для зручного читання/запису (хоча менш поширено для бінарних файлів, ніж read/write, але розуміння їхньої роботи важливе).
66. Практичне застосування теоретичних знань C++: Лабораторна робота є чудовим прикладом того, як теоретичні знання з синтаксису C++, покажчиків, структур, класів та файлового вводу/виводу трансформуються в робочий код.
67. Розвиток навичок самостійної роботи з кодом: Значна частина завдання виконується індивідуально, що розвиває навички самостійного написання та налагодження коду.
68. Здатність до планування реалізації ПЗ: Розроблення та затвердження плану робіт, що включає етапи кодування, демонструє вміння планувати процес написання коду.
69. Розуміння життєвого циклу розробки програмного забезпечення (SDLC) на практиці: Проходження всіх етапів від аналізу до тестування та інтеграції через призму написання коду.
70. Навички створення консольних додатків: Створення проекту консольного додатка (prj\_1\_Sereda) у Code::Blocks IDE є базовою, але важливою навичкою.
71. Дотримання вимог технічного завдання при кодуванні: Забезпечення того, що функціонал реєстру (виведення, додавання, пошук, видалення, збереження/завантаження) повністю реалізований у коді.
72. Формування навичок рефакторингу коду: У процесі розробки та інтеграції може виникнути необхідність покращити структуру або читабельність наявного коду.
73. Розвиток навичок тестування власного коду: Створення набору тестів (TestSuite) до модулів свідчить про розуміння важливості перевірки написаного коду.
74. Вміння адаптуватись до середовища розробки (Code::Blocks): Ефективне використання IDE для написання, компіляції та налагодження коду.
75. Здобуття впевненості у написанні складних програмних рішень: Успішне завершення такого комплексного завдання значно підвищує впевненість у власних здібностях як програміста.

**Додатки**

|  |  |
| --- | --- |
| Назва тестового набору  Test Suite Description | TS\_11\_1 |
| Назва проекта / ПЗ  Name of Project / Software | ModulesSereda |
| Рівень тестування  Level of Testing | модульне / Unit Testing |
| Автор тест-сьюта  Test Suite Author | Середа Марина |
| Виконавець  Implementer | Середа Марина |

|  |  |  |  |
| --- | --- | --- | --- |
| Ід-р тест-кейса / Test Case ID | Дії (кроки) /  Action  (Test Steps) | Очікуваний  результат /  Expected Result | Результат тестування (пройшов/не вдалося/ заблокований) /  Test Result (passed/failed/ blocked) |
| 1 | 1. Запуск програми. | Введіть державний номер для пошуку: | passed |
| 2 | 1. Запуск програми. 2. Увести AA2326. | Введіть державний номер для пошуку: AA2326  Знайдено запис:  Прізвище: Середа  Ім’я: Марина  По батькові: Олександрівна  Марка авто: Mersedes  Рік випуску: 2024  Дата реєстрації: 12.02.2024  Держ. номер: AA2326  Примітки: - | passed |
| 3 | 1. Запуск програми. 2. Увести AG2536. | Введіть державний номер для пошуку: AG2536  Запис із державним номером 'AG2536' не знайдено. | passed |

|  |  |
| --- | --- |
| Назва тестового набору  Test Suite Description | TS\_11\_2 |
| Назва проекта / ПЗ  Name of Project / Software | ModulesSereda |
| Рівень тестування  Level of Testing | модульне / Unit Testing |
| Автор тест-сьюта  Test Suite Author | Середа Марина |
| Виконавець  Implementer | Середа Марина |

|  |  |  |  |
| --- | --- | --- | --- |
| Ід-р тест-кейса / Test Case ID | Дії (кроки) /  Action  (Test Steps) | Очікуваний  результат /  Expected Result | Результат тестування (пройшов/не вдалося/ заблокований) /  Test Result (passed/failed/ blocked) |
| 1 | 1. Запуск програми. | Введіть державний номер для видалення: | passed |
| 2 | 1. Запуск програми. 2. Увести AA2326. | Введіть державний номер для видалення: AA2326  Запис видалено. | passed |
| 3 | 1. Запуск програми. 2. Увести AA2326. | Введіть державний номер для видалення: AA2326  Запис не знайдено. | passed |

**Лістинги**

**ModulesSereda.cpp**

#include "ModulesSereda.h"

#include "struct\_type\_project\_1.h"

#include <iostream>

#include <fstream>

void search\_by\_state\_number(const std::string& filename, const std::string& search\_number) {

std::ifstream file(filename, std::ios::binary);

if (!file) {

std::cerr << "Не вдалося відкрити файл: " << filename << std::endl;

return;

}

auto read\_string = [&](std::ifstream& f) {

size\_t size;

f.read(reinterpret\_cast<char\*>(&size), sizeof(size));

std::string str(size, '\0');

f.read(&str[0], size);

return str;

};

bool found = false;

while (file.peek() != EOF) {

CarRegister record;

record.last\_name = read\_string(file);

record.first\_name = read\_string(file);

record.middle\_name = read\_string(file);

record.car\_brand = read\_string(file);

file.read(reinterpret\_cast<char\*>(&record.year\_of\_manufacture), sizeof(int));

record.date\_of\_registration = read\_string(file);

record.state\_number = read\_string(file);

record.notes = read\_string(file);

if (record.state\_number == search\_number) {

std::cout << "Знайдено запис:\n";

std::cout << "Прізвище: " << record.last\_name << "\n";

std::cout << "Ім’я: " << record.first\_name << "\n";

std::cout << "По батькові: " << record.middle\_name << "\n";

std::cout << "Марка авто: " << record.car\_brand << "\n";

std::cout << "Рік випуску: " << record.year\_of\_manufacture << "\n";

std::cout << "Дата реєстрації: " << record.date\_of\_registration << "\n";

std::cout << "Держ. номер: " << record.state\_number << "\n";

std::cout << "Примітки: " << record.notes << "\n";

found = true;

break;

}

}

if (!found) {

std::cout << "Запис із державним номером '" << search\_number << "' не знайдено.\n";

}

file.close();

}

bool removeByStateNumber(List& list, const std::string& state\_number) {

Node\* current = list.head;

while (current != nullptr) {

if (current->data.state\_number == state\_number) {

if (current->previous)

current->previous->next = current->next;

else

list.head = current->next;

if (current->next)

current->next->previous = current->previous;

else

list.tail = current->previous;

delete current;

return true; // Успішне видалення

}

current = current->next;

}

return false; // Запис з таким номером не знайдено

}

void clearList(List& list) {

Node\* current = list.head;

while (current != nullptr) {

Node\* toDelete = current;

current = current->next;

delete toDelete;

}

list.head = list.tail = nullptr;

}

**ModulesSereda.h**

#ifndef MODULESSEREDA\_H\_INCLUDED

#define MODULESSEREDA\_H\_INCLUDED

#include <string>

#include "struct\_type\_project\_1.h"

void search\_by\_state\_number(const std::string& filename, const std::string& state\_number);

bool removeByStateNumber(List& list, const std::string& state\_number);

void clearList(List& list);

#endif // MODULESSEREDA\_H\_INCLUDED

**struct\_type\_project\_1.h**

#ifndef STRUCT\_TYPE\_PROJECT\_1\_H\_INCLUDED

#define STRUCT\_TYPE\_PROJECT\_1\_H\_INCLUDED

#include <string>

struct CarRegister {

std::string last\_name; //Прізвище

std::string first\_name; //Ім'я

std::string middle\_name; //По батькові

std::string car\_brand; // Марка автомобіля

int year\_of\_manufacture; //Рік випуску

std::string date\_of\_registration; //Дата реєстрації

std::string state\_number; // Виданий державний номер

std::string notes; //Примітки

};

// Вузол списку

struct Node {

CarRegister data;

Node\* previous;

Node\* next;

};

// Двобічний список

struct List {

Node\* head = nullptr;

Node\* tail = nullptr;

};

#endif // STRUCT\_TYPE\_PROJECT\_1\_H\_INCLUDED

**prj\_1\_Sereda.cpp**

#include <iostream>

#include <locale>

#include <windows.h>

#include "ModulesSereda.h"

#include "ModulesDmytrenko.h"

#include "ModulesBezkrovna.h"

#include "struct\_type\_project\_1.h"

int main() {

SetConsoleCP(65001); // Дозволяє вводити українською

SetConsoleOutputCP(65001); // Дозволяє виводити українською

List carRegistry;

std::string filename;

std::cout << "Введіть ім'я файлу: ";

std::cin >> filename;

bool running = true;

// Завантаження існуючих даних із файлу

List\* loadedList = load\_register(filename);

if (loadedList != nullptr) {

carRegistry = \*loadedList;

delete loadedList;

}

List \*list = load\_register(filename);

if (list == nullptr){

list = new List;

list->head = nullptr;

list->tail = nullptr;

}

while (running) {

std::cout << "\n=== МЕНЮ ===\n";

std::cout << "1. Додати новий запис\n";

std::cout << "2. Вивести реєстр на екран\n";

std::cout << "3. Вивести реєстр у текстовий файл\n";

std::cout << "4. Пошук за державним номером\n";

std::cout << "5. Видалити запис за номером\n";

std::cout << "6. Вихід\n";

std::cout << "Ваш вибір: ";

int choice;

std::cin >> choice;

switch (choice) {

case 1:

addNewCarRecord(&carRegistry);

save\_to\_file(list, filename);

break;

case 2:

printRegister(&carRegistry);

break;

case 3: {

std::string textFilename;

std::cout << "Введіть ім'я текстового файлу: ";

std::cin >> textFilename;

printRegistryToFile(&carRegistry, textFilename);

break;

}

case 4: {

std::string searchNumber;

std::cout << "Введіть державний номер для пошуку: ";

std::cin >> searchNumber;

search\_by\_state\_number(filename, searchNumber);

break;

}

case 5: {tf

std::string removeNumber;

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

std::cin >> removeNumber;

if (removeByStateNumber(carRegistry, removeNumber)) {

std::cout << "Запис видалено.\n";

} else {

std::cout << "Запис не знайдено.\n";

}

break;

}

case 6:

running = false;

save\_to\_file(&carRegistry, filename);

deleteRegistry(&carRegistry);

break;

default:

std::cout << "Невірний вибір. Спробуйте ще раз.\n";

}

}

delete list;

return 0;

}