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

ВИКОНАЛА

студентка академічної групи КН-24

Ярослава Дмитренко

ПЕРЕВІРИЛА

викладачка кафедри кібербезпеки

та програмного забезпечення

Анастасія КОВАЛЕНКО

Кропивницький – 2025

**Тема:** КОМАНДНА РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ ОБРОБЛЕННЯ ДИНАМІЧНИХ СТРУКТУР ДАНИХ ТА БІНАРНИХ ФАЙЛІВ

**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 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.

**Модуль ModulesDmytrenko**

Модуль ModulesDmytrenko відповідає за базові операції з даними у реєстрі автомобілів: створення нових записів, виведення реєстру та збереження у текстовий файл.

Основні функції:

* addNewCarRecord(List\* list) – додавання нового запису до реєстру.
* printRegister(const List\* list) – виведення всього реєстру на екран.
* printRegistryToFile(const List\* list, const std::string& filename) – виведення всього реєстру в заданий текстовий файл.
* deleteRegistry(List\* list) – звільнення пам’яті.

**Функція addNewCarRecord**

**Призначення:**

* Додає новий запис про автомобіль у кінець двозв'язного списку list.

**Опис дій:**

* Інтерактивно запитує у користувача всі поля структури CarRegister.
* Перевіряє, щоб введені значення не були порожніми, а рік — коректним.
* Створює новий вузол (Node), записує в нього дані.
* Додає новий вузол у список: або як перший елемент, або в кінець.

**Функція printRegister(const List\* list)**

**Призначення:**

* Виводить усі записи з реєстру (двозв’язного списку list) на екран.

**Опис дій:**

* Перебирає список від head до tail.
* Для кожного запису (CarRegister) виводить усю інформацію у форматованому вигляді.
* Якщо список порожній — виводить відповідне повідомлення.

**Функція printRegistryToFile(const List\* list, const std::string& filename)**

**Призначення:**

* Зберігає повний реєстр у вказаний текстовий файл.

**Опис дій:**

* Відкриває файл filename на запис.
* Якщо файл не відкрився — виводить повідомлення про помилку.
* Проходить по всіх вузлах списку та записує їх у файл у форматі, подібному до printRegister.
* По завершенню закриває файл та виводить повідомлення про успішне збереження.

**Функція deleteRegistry(List\* list)**

**Призначення:**

* Очищає список — видаляє всі вузли з пам’яті.

**Опис дій:**

* Проходить по всьому списку від head до tail.
* Для кожного вузла виконує delete.
* Після очищення встановлює head і tail в nullptr, щоб уникнути висячих вказівників.

**Лістинг ModulesDmytrenko**

#include <iostream>

#include <fstream>

#include <string>

#include <locale>

#include <windows.h>

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

void addNewCarRecord(List\* list) {

Node\* newNode = new Node;

newNode->previous = nullptr;

newNode->next = nullptr;

CarRegister& car = newNode->data;

std::cin.ignore();

do {

std::cout << "Введіть прізвище: ";

std::getline(std::cin, car.last\_name);

if (car.last\_name.empty()) {

std::cout << "Помилка! Прізвище не може бути порожнім.\n";

}

} while (car.last\_name.empty());

do {

std::cout << "Введіть ім'я: ";

std::getline(std::cin, car.first\_name);

if (car.first\_name.empty()) {

std::cout << "Помилка! Ім'я не може бути порожнім.\n";

}

} while (car.first\_name.empty());

do {

std::cout << "Введіть по батькові: ";

std::getline(std::cin, car.middle\_name);

if (car.middle\_name.empty()) {

std::cout << "Помилка! По батькові не може бути порожнім.\n";

}

} while (car.middle\_name.empty());

do {

std::cout << "Введіть марку автомобіля: ";

std::getline(std::cin, car.car\_brand);

if (car.car\_brand.empty()) {

std::cout << "Помилка! Марка авто не може бути порожньою.\n";

}

} while (car.car\_brand.empty());

do {

std::cout << "Введіть рік випуску: ";

std::string input;

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

try {

car.year\_of\_manufacture = std::stoi(input);

if (car.year\_of\_manufacture < 1886 || car.year\_of\_manufacture > 2025) {

std::cout << "Помилка! Некоректний рік випуску.\n";

car.year\_of\_manufacture = 0;

}

} catch (...) {

std::cout << "Помилка! Рік має бути числом.\n";

car.year\_of\_manufacture = 0;

}

} while (car.year\_of\_manufacture == 0);

do {

std::cout << "Введіть дату реєстрації: ";

std::getline(std::cin, car.date\_of\_registration);

if (car.date\_of\_registration.empty()) {

std::cout << "Помилка! Дата не може бути порожньою.\n";

}

} while (car.date\_of\_registration.empty());

do {

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

std::getline(std::cin, car.state\_number);

if (car.state\_number.empty()) {

std::cout << "Помилка! Номер не може бути порожнім.\n";

}

} while (car.state\_number.empty());

do {

std::cout << "Введіть примітки: ";

std::getline(std::cin, car.notes);

if (car.notes.empty()) {

std::cout << "Помилка! Примітки не можуть бути порожніми (введіть хоча б '-')\n";

}

} while (car.notes.empty());

// Додавання у список

if (list->head == nullptr) {

list->head = list->tail = newNode;

} else {

newNode->previous = list->tail;

list->tail->next = newNode;

list->tail = newNode;

}

std::cout << "Новий запис успішно додано!\n";

}

void printRegister(const List\* list) // Виведення реєстру на екран

{

Node\* current = list->head;

if (!current)

{

std::cout << "Список порожній.\n";

return;

}

while (current)

{

const CarRegister& car = current->data;

std::cout << "---------------------------\n";

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

std::cout << "Ім'я: " << car.first\_name << "\n";

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

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

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

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

std::cout << "Державний номер: " << car.state\_number << "\n";

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

current = current->next;

}

}

void printRegistryToFile(const List\* list, const std::string& filename) //Виведення всього реєстру у заданий текстовий файл

{

std::ofstream outFile(filename);

if (!outFile)

{

std::cerr << "Помилка відкриття файлу для запису.\n";

return;

}

Node\* current = list->head;

while (current)

{

const CarRegister& car = current->data;

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

outFile << "Прізвище: " << car.last\_name << "\n";

outFile << "Ім'я: " << car.first\_name << "\n";

outFile << "По батькові: " << car.middle\_name << "\n";

outFile << "Марка авто: " << car.car\_brand << "\n";

outFile << "Рік випуску: " << car.year\_of\_manufacture << "\n";

outFile << "Дата реєстрації: " << car.date\_of\_registration << "\n";

outFile << "Державний номер: " << car.state\_number << "\n";

outFile << "Примітки: " << car.notes << "\n";

current = current->next;

}

outFile.close();

std::cout << "Реєстр збережено в файл: " << filename << "\n";

}

// Звільнення пам'яті

void deleteRegistry(List\* list)

{

Node\* current = list->head;

while (current)

{

Node\* next = current->next;

delete current;

current = next;

}

list->head = list->tail = nullptr;

}

**Лістинг struct\_type\_project\_1**

#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

**Лістинг ModulesDmytrenko.h**

#ifndef MODULESDMYTRENKO\_H\_INCLUDED

#define MODULESDMYTRENKO\_H\_INCLUDED

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

void addNewCarRecord(List\* list);

void printRegister(const List\* list);

void printRegistryToFile(const List\* list, const std::string& filename);

void deleteRegistry(List\* list);

#endif // MODULESDMYTRENKO\_H\_INCLUDED

**Лістинг програми prj\_1\_Dmytrenko**

#include <iostream>

#include <fstream>

#include <string>

#include <locale>

#include <windows.h>

#include "ModulesDmytrenko.h"

#include "ModulesBezkrovna.h"

#include "ModulesSereda.h"

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

int main() {

SetConsoleCP(CP\_UTF8);

SetConsoleOutputCP(CP\_UTF8);

setlocale(LC\_ALL, "Ukrainian");

std::string filename;

std::cout << "Введіть ім'я файлу для зчитування: ";

std::cin >> filename;

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

if (list == nullptr)

{

list = new List;

list->head = nullptr;

list->tail = nullptr;

}

int choice;

do {

std::cout << "\nМеню:\n";

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

std::cout << "2. Вивести всі записи\n";

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

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

std::cout << "5. Видалення запису з реєстра за державним номером\n";

std::cout << "6. Вихід\n";

std::cout << "Ваш вибір: ";

std::cin >> choice;

std::cin.ignore();

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

if (list == nullptr)

{

list = new List;

list->head = nullptr;

list->tail = nullptr;

}

switch (choice) {

case 1:

addNewCarRecord(list);

break;

case 2:

printRegister(list);

break;

case 3: {

std::string filename;

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

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

printRegistryToFile(list, filename);

break;

}

case 4: {

std::string state\_number;

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

std::cin >> state\_number;

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

break;

}

case 5: {

std::string stt\_number;

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

std::cin >> stt\_number;

if (removeByStateNumber(\*list, stt\_number)) {

std::cout << "Запис видалено.\n";

} else {

std::cout << "Запис не знайдено.\n";

}

break;

}

case 6: {

std::string filename;

std::cout << "Введіть ім'я бінарного файлу для збереження: ";

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

save\_to\_file(list, filename);

deleteRegistry(list);

std::cout << "Завершення роботи.\n";

break;

}

default:

std::cout << "Невірний вибір!\n";

}

} while (choice != 6);

return 0;

}

**Висновок:** У результаті виконання лабораторної роботи я набула цінного досвіду командної реалізації програмного забезпечення, що працює з динамічними структурами даних та бінарними файлами. В результаті командної роботи було розроблено власні програмні модулі, які забезпечують ефективне створення, оброблення та збереження інформації у вигляді двозв’язного списку, а також реалізували можливість взаємодії з файлами у текстовому та бінарному форматах. Ознайомилася з особливостями керування динамічною пам’яттю в мові C++.

**Аргументи на користь досягнення мети лабораторної роботи**

1. Створено окремі модулі для кожного учасника команди, що дозволяє розділити відповідальність.
2. Реалізовано функції додавання, видалення, пошуку та виведення інформації, що охоплює всі основні операції над динамічними структурами.
3. Усі функції оформлено з дотриманням принципів читабельності та повторного використання.
4. Використано структури даних (struct) для зручного зберігання інформації про об’єкти.
5. Кожен модуль відповідає за конкретну частину функціональності, що дозволяє масштабувати систему.
6. Функції реалізовано відповідно до методичних вказівок, що забезпечило правильність підходів.
7. Функції використовують захист від некоректного введення, що підвищує надійність ПЗ.
8. Забезпечено перевірку на порожність списку перед обробкою, що запобігає помилкам виконання.
9. Реалізовано логіку з використанням умовних конструкцій та циклів, що забезпечує контроль над виконанням.
10. Передбачено перевірку на дублі в певних функціях, що робить систему більш стабільною.
11. Використано двозв’язний список, який дозволяє ефективно реалізовувати як вставку, так і видалення.
12. Кожен вузол містить попередній і наступний покажчики, що полегшує навігацію в обох напрямках.
13. Список коректно ініціалізується при порожньому файлі, що дозволяє уникнути краху програми.
14. Реалізовано видалення з будь-якого місця списку, що демонструє розуміння роботи з покажчиками.
15. При додаванні нового запису дані валідуються, що гарантує коректність інформації.
16. Створення вузлів реалізовано з урахуванням динамічного виділення пам’яті (оператор new).
17. Коректно звільняється пам’ять при видаленні вузлів, що запобігає витоку пам’яті.
18. Очищення списку реалізовано перед виходом з програми, що є ознакою грамотного керування ресурсами.
19. Список можна зберегти в бінарному або текстовому форматі — це демонструє гнучкість у представленні даних.
20. Дані в списку можна легко модифікувати або розширити, що є важливою перевагою динамічної структури.
21. Реалізовано зчитування з бінарного файлу, що дозволяє швидко завантажити структуру.
22. Збереження у бінарний файл зберігає структуру в компактному вигляді.
23. Створено окрему функцію для експорту даних у текстовий файл, що полегшує читання користувачем.
24. При відкритті файлу передбачено перевірку на помилки, що захищає програму від падіння.
25. Ім’я файлу вводиться користувачем, що робить програму більш гнучкою.
26. Формат збереження забезпечує повну відповідність між структурою в пам’яті та у файлі.
27. Реалізовано функції серіалізації та десеріалізації, що відповідає практиці реальних ІТ-проєктів.
28. Обробка файлів відбувається за допомогою стандартної бібліотеки C++, що підсилює практичні навички.
29. Кожен бінарний файл може містити повну копію реєстру, що полегшує архівацію.
30. Створено резервну копію під час збереження — приклад дотримання принципів надійності.
31. Програма структурована за модулями, що забезпечує легкість підтримки та розширення.
32. Чітке розділення функціональності дозволяє незалежно оновлювати модулі.
33. Усі функції згруповані логічно — за функціональним призначенням.
34. Архітектура дозволяє масштабування: легко додати новий модуль чи функцію.
35. Принцип «розділення обов’язків» реалізовано через заголовкові файли.
36. Ієрархія залежностей між модулями зрозуміла та контрольована.
37. Використання include-охоронників захищає від багаторазового включення заголовків.
38. Проєкт підтримує принцип відкритості/закритості — можна розширювати без зміни існуючого коду.
39. Головна функція main() координує роботу модулів — вона логічно відокремлена від логіки.
40. Кожен модуль не залежить від конкретного способу збереження даних, що робить їх універсальними.
41. Кожен учасник реалізував окрему частину проєкту — розподіл праці.
42. Взаємодія між модулями демонструє ефективність спільної розробки.
43. Спільне тестування допомогло виявити і виправити помилки.
44. Учасники узгодили структуру програми перед реалізацією, що свідчить про командне планування.
45. Реалізація програмного засобу була злагодженою — частини гармонійно інтегруються.
46. Усі учасники дотримувались єдиних стилістичних іменувань змінних та функцій.
47. Командна реалізація допомогла швидше впоратися з великим обсягом завдань.
48. Комунікація всередині команди забезпечила своєчасну корекцію архітектурних рішень.
49. Кожен учасник навчився працювати з чужим кодом — це важливе вміння в індустрії.
50. Робота в команді розвинула навички відповідальності та взаємодопомоги.
51. Для перевірки функцій створено набір тестових сценаріїв — тест-кейси.
52. Тест-кейси охоплюють як типові, так і граничні випадки.
53. Тестування виявило помилки у валідації вводу — ці помилки були виправлені.
54. Створено тест-сьюти для автоматизованої перевірки функціоналу.
55. Після кожного оновлення коду проводилось повторне тестування, що зменшувало кількість багів.
56. Тестування функцій у відриві від UI дозволило локалізувати логічні помилки.
57. Тестування роботи з файлами перевірялося окремо — це гарантує надійність збереження даних.
58. Функція пошуку була протестована на різних комбінаціях даних — доведено її коректність.
59. Усі помилки тестування було задокументовано, що підвищило контроль якості.
60. Тестування підтвердило, що всі гілки умовного коду покрито тестами.
61. Методичні вказівки допомогли структуризувати підхід до реалізації.
62. Покрокове виконання завдань за методичкою дозволило уникнути хаотичної розробки.
63. Приклади з методичних вказівок слугували шаблонами для створення власних функцій.
64. Завдяки чіткому формулюванню вимог вказівок команда могла працювати скоординовано.
65. Методичні рекомендації дали змогу уникнути грубих помилок в роботі з пам’яттю.
66. Завдання розділено логічно, що полегшило планування командної роботи.
67. Методика надала орієнтири для перевірки проміжних результатів.
68. Пояснення форматів вхідних і вихідних даних допомогли уникнути несумісностей між модулями.
69. Вказівки змусили продумати валідацію, що підвищило якість вводу.
70. Виконання роботи за методикою допомогло сформувати звичку працювати структуровано.
71. Використання операторів new і delete забезпечило контроль над динамічною пам’яттю, що є критичним при роботі з динамічними структурами, запобігаючи витокам пам’яті.
72. Попереднє проєктування структури програми до її реалізації дозволило уникнути зайвої переробки коду, забезпечило узгодженість між модулями та спростило розробку.
73. Активне використання вказівників у реалізації структури дозволило повністю контролювати зв’язки між елементами, що є основою ефективного управління динамічними структурами в C++.
74. Інтеграція модулів, реалізованих різними членами команди, довела ефективність командної розробки та підтвердила важливість узгоджених стандартів кодування.
75. Дотримання циклу розробки — від планування до тестування — сприяло логічній побудові проєкту, дозволило вчасно виявити помилки та покращити якість програмного засобу.

**Додатки**

|  |  |
| --- | --- |
| Назва тестового набору  Test Suite Description | TestSuite\_11 |
| Назва проекта / ПЗ  Name of Project / Software | TestDriver.ехе |
| Рівень тестування  Level of Testing | модульне / ModulesTesting |
| Автор тест-сьюта  Test Suite Author | Ярослава Дмитренко  телеграм: t.me/y\_aradm  e-пошта: dmytrenkovika4@gmail.com |
| Виконавець  Implementer | Ярослава Дмитренко |

Artifact: Test Suite

Date: 5/8/2025

|  |  |  |  |
| --- | --- | --- | --- |
| Ід-р тест-кейса / Test Case ID | Дії (кроки) /  Action (Test Steps) | Очікуваний  результат /  Expected Result | Результат тестування /  Test Result |
| TC-1 | 1. Викликати функцію додання нового нового запису в реєстрі. 2. Ввести коректні значення для всіх полів (наприклад, прізвище, ім'я, марка авто, рік випуску). | Новий запис додається до реєстру. Результати коректно зберігаються в списку. | passed |
| TC-2 | 1. Викликати функцію виведення всього реєстру на екран. 2. Перевірити, чи виводиться інформація про реєстрацію в консоль. | Реєстр виводиться на екран, відображаються всі записи. | passed |
| TC-3 | 1. Викликати функцію printRegistryToFile. 2. Ввести ім'я файлу для збереження (наприклад, "test\_registry.txt"). | Файл зберігається з коректною інформацією про всі записи реєстру. | passed |
| TC-4 | 1. Викликати функцію deleteRegistry. 2. Перевірити, чи список після видалення стає порожнім. | Список очищається, всі записи видаляються, список порожній. | passed |