Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

Дисципліна: Базові методології та технології програмування

**Лабораторна робота №11** **Тема: «КОМАНДНА РЕАЛІЗАЦІЯ**

**ПРОГРАМНИХ ЗАСОБІВ**

**ОБРОБЛЕННЯ ДИНАМІЧНИХ СТРУКТУР ДАНИХ ТА БІНАРНИХ**

# ФАЙЛІВ»

Виконав: ст. гр. КН-24

Терещенко В.О

Перевірив: викладач

Коваленко А.С.

Кропивницький 2025

# Варіант - 9

*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 ІТпроєкта модулів.

**Аналіз задач ІТ-проєкта та вимог до ПЗ:**

#### **1. Загальна характеристика задачі**

Метою ІТ-проєкту є створення програмного засобу для ведення реєстру закладів освіти з можливістю зберігання, редагування, пошуку та видалення інформації. Програма має забезпечувати зручний інтерфейс для роботи з динамічною структурою даних та збереження інформації у файл для подальшого використання.

#### **2. Основні функціональні задачі**

* Зберігання структурованої інформації про освітні установи.
* Завантаження та збереження даних з/у файл.
* Додавання нових записів у динамічний список.
* Відображення повної інформації про всі заклади.
* Пошук запису за унікальним ідентифікатором (кодом).
* Видалення запису з реєстру.

**Склад команди IT-команди**

* 1.Авраменко-Владислав KБ-24
* 2.Ковальчук-Володимир KБ-24
* 3.Терещенко-Владислав KN-24

|  |  |
| --- | --- |
| **Етап** | **Хто виконує** |
| Підготовка і узгодження ідеї | Вся команда |
| Написання своїх частин коду | Кожен за своїми підзадачами |
| Збирання всього в одне ціле | Разом |
| Перевірка: чи все працює як треба | Вся команда |
| Презентація проєкта викладачу | Вся команда та викладач |

# ПЛАН РЕАЛІЗАЦІЇ IT-ПРОЄКТУ

## Електронний реєстр суб'єктів освітньої діяльності

### 3.1. АНАЛІЗ ЗАДАЧ ІТ-ПРОЄКТА ТА ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**Основна мета проєкту:** Створити електронний реєстр закладів вищої освіти з можливостями управління даними.

**Функціональні вимоги:**

* Автоматичне завантаження реєстру з файлу при запуску програми
* Виведення повного реєстру на екран або у текстовий файл
* Додавання нових записів до реєстру
* Пошук записів за кодом ЗВО в ЄДЕБО
* Вилучення записів з реєстру
* Автоматичне збереження реєстру у файл при завершенні роботи

**Нефункціональні вимоги:**

* Надійність збереження даних
* Зручний користувацький інтерфейс
* Валідація введених даних
* Обробка помилок

### 3.2. СПЕЦИФІКАЦІЇ ПЗ ТА АРХІТЕКТУРА ПРОГРАМНОГО ЗАСОБУ

**Архітектура системи:**

* Модульна архітектура з розділенням відповідальності
* Рівень представлення (User Interface)
* Рівень бізнес-логіки (Data Processing)
* Рівень даних (File I/O Operations)

**Основні модулі:**

1. **Модуль інтерфейсу користувача** - меню та взаємодія
2. **Модуль управління даними** - операції CRUD
3. **Модуль файлових операцій** - завантаження/збереження
4. **Модуль валідації** - перевірка даних

### 3.3. ОБҐРУНТУВАННЯ ДИНАМІЧНОЇ СТРУКТУРИ ДАНИХ

**Обрана структура:** Однозв'язний список

**Обґрунтування вибору:**

* Динамічне додавання/видалення записів без обмеження розміру
* Ефективне використання пам'яті
* Простота реалізації операцій вставки та видалення
* Послідовний доступ до даних відповідає логіці реєстру

**Структура елемента даних:**

struct EducationalInstitution {

int edeboCode; // Код закладу в ЄДЕБО

string fullName; // Повне найменування

string shortName; // Коротка назва

string fullNameEng; // Повне найменування (англ.)

string ownershipForm; // Форма власності

string managementBody; // Орган управління

string headPosition; // Посада керівника

string headFullName; // ПІБ керівника

string address; // Юридична адреса

string phone; // Телефон/факс

string email; // Електронна пошта

string website; // Веб-сайт

bool crimUkraine; // ОЦ "Крим-Україна"

bool donbasUkraine; // ОЦ "Донбас-Україна"

bool militaryDepartment; // Військова кафедра

int foundationYear; // Рік заснування

EducationalInstitution\* next; // Вказівник на наступний елемент

};

### 3.4. СТВОРЕННЯ ЗАГОЛОВКОВОГО ФАЙЛУ

**Файл:** \Lab11\prj\struct\_type\_project\_9.h

Містить оголошення структури EducationalInstitution та прототипи всіх функцій для роботи з динамічною структурою даних.

### 3.5. РОЗПОДІЛ ПІДЗАДАЧ МІЖ УЧАСНИКАМИ КОМАНДИ

**Авраменко**

1. **Функція додавання запису** (addRecord) - додавання нового закладу до списку
2. **Функція завантаження з файлу** (loadFromFile) - читання даних з бінарного файлу при запуску

**Ковальчук**

1. **Функція пошуку запису** (searchByCode) - пошук закладу за кодом ЄДЕБО
2. **Функція виведення реєстру** (displayRegistry) - виведення всіх записів на екран або у файл

**Терещенко**

1. **Функція видалення запису** (deleteRecord) - вилучення запису зі списку
2. **Функція збереження у файл** (saveToFile) - запис даних у бінарний файл

### 3.6. ПЛАН РОБІТ ЗГІДНО ISO/IEC 12207

**Етап 1: Планування та аналіз**

* Аналіз вимог та специфікацій
* Проєктування архітектури системи
* Створення технічного завдання
* Розподіл ролей та відповідальності

**Етап 2: Проєктування**

* Детальне проєктування структур даних
* Створення заголовкового файлу struct\_type\_project\_9.h
* Проєктування інтерфейсів модулів
* Визначення форматів файлів та протоколів взаємодії

**Етап 3: Реалізація**

**Паралельна робота учасників:**

**Авраменко:**

* Реалізація функції addRecord()
* Реалізація функції loadFromFile()

**Ковальчук:**

* Реалізація функції searchByCode()
* Реалізація функції displayRegistry()

**Терещенко:**

* Реалізація функції deleteRecord()
* Реалізація функції saveToFile()

**Етап 4: Інтеграція та тестування**

* Інтеграція всіх модулів
* Створення головної програми з меню
* Виправлення помилок та оптимізація

**Етап 5: Валідація та документація**

* Валідація відповідності вимогам
* Створення документації користувача
* Підготовка звітів

### РИЗИКИ ТА ЇХ МІТИГАЦІЯ

**Технічні ризики:**

* Складність роботи з бінарними файлами → Детальне вивчення документації
* Проблеми інтеграції модулів → Чітке визначення інтерфейсів

**Організаційні ризики:**

* Несвоєчасне виконання завдань → Регулярні зустрічі команди
* Конфлікти версій коду → Використання системи контролю версій

### КРИТЕРІЇ ЯКОСТІ

* Коректність роботи всіх функцій
* Стабільність при некоректному вводі
* Відповідність вимогам завдання
* Якість коду та документації
* Ефективність алгоритмів

**Модулі:**

* deleteRecord (видалення запису зі списку)
* saveToFile (збереження у бінарний файл)
* interactiveDelete (інтерактивна обгортка навколо deleteRecord)
* autoSave (автоматичне збереження, викликає saveToFile)
* createBackup (створює копію реєстру у вказаний бінарний файл)
* validateDataIntegrity (перевірка коректності даних у списку)

**3.1. Архітектурне проєктування модуля Терещенка**

Функції Терещенка розташовані у такій ієрархії системи:

* **deleteRecord** – логіка пошуку й видалення вузла за кодом.
* **interactiveDelete** – циклова обгортка навколо deleteRecord, щоб можна було видаляти кілька записів підряд.
* **validateDataIntegrity** – перевіряє кожен вузол на коректність полів.
* **autoSave** – викликає saveToFile з повідомленням “=== АВТОМАТИЧНЕ ЗБЕРЕЖЕННЯ ===”.
* **createBackup** – робить резервну копію через saveToFile в інший файл.
* **saveToFile** – сам запис даних у бінарний файл.

**3.2. Детальне проєктування**

**3.2.1. Алгоритм функції deleteRecord**

EducationalInstitution\* deleteRecord(EducationalInstitution\* head, int code) {

if (head == nullptr) {

cout << "Реєстр порожній! Нема що видаляти." << endl;

return nullptr;

}

EducationalInstitution\* current = head;

EducationalInstitution\* previous = nullptr;

// Шукаємо вузол із заданим кодом

while (current != nullptr && current->edeboCode != code) {

previous = current;

current = current->next;

}

// Якщо не знайдено

if (current == nullptr) {

cout << "Запис з кодом ЄДЕБО " << code << " не знайдено в реєстрі." << endl;

return head;

}

// Виводимо коротку інформацію про запис

cout << "\n=== ЗНАЙДЕНО ЗАПИС ДЛЯ ВИДАЛЕННЯ ===" << endl;

cout << "Код ЄДЕБО: " << current->edeboCode << endl;

cout << "Назва: " << current->fullName << endl;

cout << "Коротка назва: " << current->shortName << endl;

// Запит підтвердження

char confirmation;

cout << "\nВи дійсно хочете видалити цей запис? (y/n): ";

cin >> confirmation;

if (confirmation != 'y' && confirmation != 'Y') {

cout << "Видалення скасовано." << endl;

return head;

}

// Видаляємо вузол

if (previous == nullptr) {

// Видаляємо перший елемент

head = current->next;

} else {

// Видаляємо внутрішній чи останній елемент

previous->next = current->next;

}

delete current;

cout << "Запис успішно видалено з реєстру!" << endl;

return head;

}

**3.2.2. Алгоритм функції saveToFile**

void saveToFile(EducationalInstitution\* head, const string& filename) {

ofstream file(filename, ios::binary | ios::trunc);

if (!file.is\_open()) {

cout << "Помилка створення файлу " << filename << " для збереження!" << endl;

return;

}

if (head == nullptr) {

cout << "Реєстр порожній. Створено порожній файл " << filename << endl;

file.close();

return;

}

cout << "Збереження реєстру у файл " << filename << "..." << endl;

EducationalInstitution\* current = head;

int count = 0;

auto writeString = [&file](const string& str) {

size\_t length = str.length();

file.write(reinterpret\_cast<const char\*>(&length), sizeof(size\_t));

file.write(str.c\_str(), length);

};

while (current != nullptr) {

// Записуємо код

file.write(reinterpret\_cast<const char\*>(&current->edeboCode), sizeof(int));

// Запис усіх рядкових полів

writeString(current->fullName);

writeString(current->shortName);

writeString(current->fullNameEng);

writeString(current->ownershipForm);

writeString(current->managementBody);

writeString(current->headPosition);

writeString(current->headFullName);

writeString(current->address);

writeString(current->phone);

writeString(current->email);

writeString(current->website);

// Запис булевих полів

file.write(reinterpret\_cast<const char\*>(&current->crimUkraine), sizeof(bool));

file.write(reinterpret\_cast<const char\*>(&current->donbasUkraine), sizeof(bool));

file.write(reinterpret\_cast<const char\*>(&current->militaryDepartment), sizeof(bool));

// Запис року заснування

file.write(reinterpret\_cast<const char\*>(&current->foundationYear), sizeof(int));

current = current->next;

count++;

}

file.close();

if (file.fail()) {

cout << "Помилка при записі у файл!" << endl;

} else {

cout << "Реєстр успішно збережено!" << endl;

cout << "Збережено " << count << " записів у файл " << filename << endl;

}

}

**3.2.3. Алгоритм функції interactiveDelete**

EducationalInstitution\* interactiveDelete(EducationalInstitution\* head) {

int code;

char choice;

do {

if (head == nullptr) {

cout << "Реєстр порожній! Нема що видаляти." << endl;

break;

}

cout << "\n=== ВИДАЛЕННЯ ЗАПИСУ З РЕЄСТРУ ===" << endl;

cout << "Введіть код ЄДЕБО запису для видалення: ";

while (!(cin >> code) || code <= 0) {

cout << "Помилка! Введіть коректний код (ціле додатне число): ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

head = deleteRecord(head, code);

if (head != nullptr) {

cout << "\nВидалити ще один запис? (y/n): ";

cin >> choice;

} else {

cout << "Реєстр тепер порожній." << endl;

break;

}

} while (choice == 'y' || choice == 'Y');

return head;

}

**3.2.4. Алгоритм функції autoSave**

void autoSave(EducationalInstitution\* head, const string& filename) {

cout << "\n=== АВТОМАТИЧНЕ ЗБЕРЕЖЕННЯ ===" << endl;

saveToFile(head, filename);

}

**3.2.5. Алгоритм функції createBackup**

void createBackup(EducationalInstitution\* head, const string& backupFilename) {

cout << "Створення резервної копії..." << endl;

saveToFile(head, backupFilename);

cout << "Резервну копію створено: " << backupFilename << endl;

}

**3.2.6. Алгоритм функції validateDataIntegrity**

bool validateDataIntegrity(EducationalInstitution\* head) {

if (head == nullptr) {

return true; // Порожній список – коректний

}

EducationalInstitution\* current = head;

int count = 0;

while (current != nullptr) {

// Перевірка обов'язкових полів

if (current->edeboCode <= 0) {

cout << "Помилка: некоректний код ЄДЕБО у записі №" << (count + 1) << endl;

return false;

}

if (current->fullName.empty()) {

cout << "Помилка: відсутня назва у записі №" << (count + 1) << endl;

return false;

}

if (current->foundationYear < 1000 || current->foundationYear > 2024) {

cout << "Помилка: некоректний рік заснування у записі №" << (count + 1) << endl;

return false;

}

current = current->next;

count++;

}

cout << "Перевірка цілісності пройшла успішно. Записів: " << count << endl;

return true;

}

**3.3. Тестування модуля Терещенка**

| **Test Case ID** | **Action (test steps)** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| T-01 | **deleteRecord** на порожньому списку: |  |  |

1. Передати head = nullptr, code = 50.
2. Виклик deleteRecord(nullptr, 50). | Виводиться “Реєстр порожній! Нема що видаляти.”  
   Повертається nullptr. | passed |  
   | T-02 | **deleteRecord** – видалення першого елементу:
3. Створити список: вузол A (код=10) → вузол B (код=20).
4. Виклик deleteRecord(head, 10).
5. Після підтвердження (y). | Виводиться “=== ЗНАЙДЕНО ЗАПИС ДЛЯ ВИДАЛЕННЯ ===” + коротка інформація про вузол із кодом 10.  
   Після введення y → “Запис успішно видалено з реєстру!”  
   Повертається новий head – вказівник на вузол B (код=20). | passed |  
   | T-03 | **deleteRecord** – відміна видалення:
6. Той же список A(10)→B(20).
7. Виклик deleteRecord(head, 10).
8. На підтвердженні введено n. | Виводиться інформація про запис із кодом 10.  
   При введенні n → “Видалення скасовано.”  
   head лишається незмінним (A→B). | passed |  
   | T-04 | **deleteRecord** – код відсутній:
9. Список із кодами [5, 15, 25].
10. Виклик deleteRecord(head, 99). | Виводиться “Запис з кодом ЄДЕБО 99 не знайдено в реєстрі.”  
    head не змінюється (залишається той самий список). | passed |  
    | T-05 | **interactiveDelete** – серія операцій:
11. Список із кодами [100, 200, 300].
12. Виклик interactiveDelete(head).
13. Ввести “200” → підтвердити (y).
14. Ввести “y” (ще одне видалення) → ввести “999” → вихід (n). | Перший крок: “=== ЗНАЙДЕНО ЗАПИС ДЛЯ ВИДАЛЕННЯ ===” + показ інформації про код=200 → після y “Запис успішно видалено…”  
    → Після цього (ще один цикл) ввести 999 → “Запис ... не знайдено.” → слід n → вихід із циклу.  
    Після повернення head – видалено код 200, лишилися [100 → 300]. | passed |  
    | T-06 | **saveToFile** – порожній список:
15. head = nullptr, filename = "empty\_save.bin".
16. Виклик saveToFile(nullptr, "empty\_save.bin"). | Виводиться “Реєстр порожній. Створено порожній файл empty\_save.bin”.  
    Справді створюється файл розміром 0 байт (або мінімального розміру без запису даних). | passed |  
    | T-07 | **saveToFile** – запис у файл:
17. Список із 2 вузлів: A (код=11, …), B (код=22, …).
18. Виклик saveToFile(head, "test\_save.bin").
19. Відкрити “test\_save.bin” у режимі бінарного читання та перевірити, що перший запис починається з коду 11 (зчитуванням int). | Консоль:  
    “Збереження реєстру у файл test\_save.bin…”  
    “Реєстр успішно збережено!”  
    “Збережено 2 записів у файл test\_save.bin”.

Файл зчитується так:

* перші 4 байти = 11 (int),
* далі – довжина рядка fullName (size\_t), потім символи, …
* після другого запису досягнуто кінця файлу. | passed |  
  | T-08 | **autoSave** (виклик saveToFile):

1. Будь-який непорожній список (наприклад, із кодами [7, 14]).
2. Виклик autoSave(head, "auto.bin"). | Виводиться “=== АВТОМАТИЧНЕ ЗБЕРЕЖЕННЯ ===”  
   Далі викликається saveToFile → “Збереження реєстру у файл auto.bin…” → “Реєстр успішно збережено! Збережено 2 записів у файл auto.bin”.  
   Створений файл “auto.bin” аналогічний результату звичайного збереження. | passed |  
   | T-09 | **createBackup**:
3. Список із кодами [33, 44].
4. Виклик createBackup(head, "backup.bin"). | Виводиться “Створення резервної копії...”  
   Далі викликається saveToFile → “Збереження реєстру у файл backup.bin…” → “Реєстр успішно збережено! ...”  
   Після – “Резервну копію створено: backup.bin”.  
   Файл backup.bin містить ті самі дані, що й при звичайному збереженні (2 записи). | passed |  
   | T-10 | **validateDataIntegrity** – коректний список:
5. Список із 3 вузлів, у яких edeboCode > 0, fullName != "", 1000 ≤ foundationYear ≤ 2024.
6. Виклик validateDataIntegrity(head). | Виводиться “Перевірка цілісності пройшла успішно. Записів: 3”  
   Повертається true. | passed |  
   | T-11 | **validateDataIntegrity** – некоректний код:
7. Список: вузол A (edeboCode = -1, fullName="X", foundationYear=2000), B (коректний).
8. Виклик validateDataIntegrity(head). | Виводить “Помилка: некоректний код ЄДЕБО у записі №1”  
   Повертає false. | passed |  
   | T-12 | **validateDataIntegrity** – відсутня назва:
9. Список: вузол A (edeboCode=5, fullName="", foundationYear=2000).
10. Виклик validateDataIntegrity(head). | Виводить “Помилка: відсутня назва у записі №1”  
    Повертає false. | passed |  
    | T-13 | **validateDataIntegrity** – некоректний рік:
11. Список: вузол A (edeboCode=5, fullName="Test", foundationYear=999).
12. Виклик validateDataIntegrity(head). | Виводить “Помилка: некоректний рік заснування у записі №1”  
    Повертає false. | passed |

**Висновок**

У результаті виконання командної роботи на тему **"Командна реалізація програмних засобів оброблення динамічних структур даних та бінарних файлів"** було досягнуто поставленої мети – здобуто практичні навички спільної розробки програмного забезпечення, а також закріплено знання з роботи з динамічною пам’яттю та файловими потоками у мові програмування C++.

**Аргументи до лабораторної роботи**

1. Динамічні структури дозволяють ефективно використовувати оперативну пам’ять.
2. Зв’язані списки забезпечують гнучкість у керуванні обсягами даних.
3. Динамічна пам’ять дозволяє уникнути жорстких обмежень на розмір структури.
4. Знання роботи з new та delete є критично важливими для C++програміста.
5. Бінарні файли дозволяють працювати з даними у більш стислій формі.
6. Бінарне збереження структур гарантує точність та цілісність даних.
7. Опрацювання динамічної пам’яті формує розуміння роботи низькорівневих механізмів.
8. Покажчики – це основа управління динамікою в C++.
9. Створення зв’язаного списку тренує уважність до логіки переходу між вузлами.
10. Робота зі структурами з вкладеними підструктурами ускладнює, але водночас поглиблює знання.
11. Допомагає засвоїти абстрактні концепції через практичну реалізацію.
12. Вчить ділити складну задачу на простіші частини.
13. Закріплює тему динамічної пам’яті в C++.
14. Тренує вміння працювати з файлами (читання та запис).
15. Дає досвід читання та налагодження чужого коду.
16. Розвиває алгоритмічне мислення та логіку.
17. Формує структурованість під час програмування.
18. Навчає системному тестуванню та пошуку помилок.
19. Пояснює вплив пам’яті на швидкодію програм.
20. Дає реальне уявлення про зберігання складних типів даних.
21. Навчає працювати в команді над одним кодом.
22. Дає досвід розподілу ролей між учасниками.
23. Демонструє важливість узгоджених інтерфейсів між модулями.
24. Формує здатність домовлятися та вирішувати конфлікти.
25. Вчить шукати компроміси між різними варіантами реалізації.
26. Досвід спільного користування Git або GitHub.
27. Практика злиття гілок у репозиторії.
28. Покращення технічної комунікації.
29. Підготовка до участі у реальних розробках.
30. Підвищення розуміння важливості коментарів та документації.
31. Реалізація повного циклу обробки даних: додавання, перегляд, редагування, видалення.
32. Робота з персональними та освітніми даними — тренування структуризації.
33. Функція пошуку в списку вимагає точного порівняння полів.
34. Видалення вузла — важливий приклад роботи з покажчиками.
35. Запис у бінарний файл — тренування роботи зі структурованими даними.
36. Зчитування з бінарного файлу — вивчення послідовності байтів.
37. Побудова зв’язаного списку — база для більш складних структур (дерева, графи).
38. Можливість повторного використання функцій для інших типів даних.
39. Підтримка інтерактивного введення користувачем — тренування інтерфейсу.
40. Відпрацювання валідації введених даних.
41. Формує навички для участі у професійних проєктах.
42. Створює базу для реалізації баз даних у простому вигляді.
43. Вчить працювати з архівуванням даних (через бінарні потоки).
44. Дає змогу створити програму для зберігання анкет, реєстрів, каталогів.
45. Є основою для реалізації простих CRM-систем.
46. Закладає принципи побудови модульного коду.
47. Можливість переходу до графічного інтерфейсу (на основі цієї логіки).
48. Підвищення впевненості у власних програмістських навичках.
49. Відпрацювання реальних задач на низькому рівні.
50. Готує до роботи з системами, які потребують ефективної пам’яті.
51. Поглиблене використання структур struct.
52. Вкладені структури як приклад об’єктного мислення.
53. Тренування роботи з посиланнями.
54. Робота з файлами через fstream.
55. Застосування reinterpret\_cast або аналогічних методів (при потребі).
56. Організація циклічної обробки через меню.
57. Практика з ручним керуванням ресурсами.
58. Усвідомлення важливості обробки помилок при файлах.
59. Розуміння роботи списку без STL (без std::list).
60. Осмислення, чому динамічні структури — гнучкіші за масиви.
61. Формування структурованої логіки побудови програм.
62. Оптимізація операцій додавання/видалення (через пряме керування).
63. Мінімізація дублювання коду.
64. Вивчення особливостей зберігання об'єктів у пам’яті.
65. Аналіз ефективності коду.
66. Вироблення інтуїції щодо складності алгоритмів.
67. Робота з відлагодженням та трасуванням.
68. Практика оформлення коду та дотримання стилю.
69. Вивчення способів уникнення витоків пам’яті.
70. Підготовка до більш складних тем: дерева, хеш-таблиці, графи.
71. Формує практичні навички роботи з вкладеними структурами.
72. Дає змогу закріпити логіку сортування й фільтрації даних.
73. Поглиблює розуміння принципів роботи алгоритмів пошуку.
74. Навчає створювати ефективні алгоритми видалення елементів із динамічних структур.
75. Сприяє розвитку командних soft skills через колективне налагодження коду.