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

Авраменко В. В.

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.

**Варіант №9**

**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.

**Склад IT-команди**

* Авраменко Владислав, КБ-24
* Ковальчук Володимир, КБ-24
* Терещенко Владислав, КН-24

### 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

### 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

### 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

};

### 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.

### 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

### 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

**1.1. Архітектурне проєктування модуля Авраменка**

Загальна архітектура системи передбачає три рівні:

1. **Рівень представлення (UI)** – взаємодія з користувачем (меню, введення/виведення даних).
2. **Рівень бізнес-логіки (Data Processing)** – реалізовані функції CRUD (тут входять addRecord, codeExists).
3. **Рівень даних (File I/O Operations)** – читання/запис бінарного файлу (тут – loadFromFile).

* **addRecord** викликається з головного меню (UI) під час команд “Додати запис”.
* **loadFromFile** викликається на старті програми (UI) для первинного наповнення однозв’язного списку.
* **codeExists** використовується у addRecord (щоб не допускати дублювання коду) й може застосовуватися й іншими модулями перед додаванням або зміненими даною функцією.

**1.2. Детальне проєктування**

**1.2.1. Структура даних**

struct EducationalInstitution {

int edeboCode;

string fullName;

string shortName;

string fullNameEng;

string ownershipForm;

string managementBody;

string headPosition;

string headFullName;

string address;

string phone;

string email;

string website;

bool crimUkraine;

bool donbasUkraine;

bool militaryDepartment;

int foundationYear;

EducationalInstitution\* next;

// Конструктор ініціалізує поля за замовчуванням

};

* Однозв’язний список із вказівником head на перший елемент.
* Порожнє поле head == nullptr означає відсутність жодного запису.

**1.2.2. Алгоритм функції addRecord**

1. Створити новий вузол типу EducationalInstitution\* newInstitution = new EducationalInstitution().
2. **Ввести** з консолі всі необхідні поля:
   * edeboCode (перевірити, що це додатнє ціле; якщо код уже існує, показати помилку й запитати інший).
   * Решта рядкових полів (fullName, shortName, …, website).
   * Три булеві поля: crimUkraine, donbasUkraine, militaryDepartment (через введення y/n).
   * foundationYear (перевірка: від 1000 до поточного року 2024).
3. Перед вставкою в список викликати codeExists(head, newInstitution->edeboCode):
   * Якщо повертає true → вивести повідомлення “Код уже існує” → попросити введення нового коду.
4. Додавання **на початок** списку:
5. newInstitution->next = head;
6. head = newInstitution;
7. Вивести повідомлення “Запис успішно додано!” і повернути оновлений head.

**1.2.3. Алгоритм функції loadFromFile**

1. Спроба відкрити файл filename у бінарному режимі (ifstream file(filename, ios::binary)).
2. Якщо !file.is\_open() → вивести “Файл не знайдено. Створюється новий реєстр.” → повернути nullptr.
3. Інакше:
   * Ініціалізувати EducationalInstitution\* head = nullptr; EducationalInstitution\* tail = nullptr;.
   * Поки не досягнуто кінця файлу (while (file.peek() != EOF)):
     1. Створити новий вузол newInstitution = new EducationalInstitution().
     2. **Читати** по черзі:
        + Ціле edeboCode (file.read(reinterpret\_cast<char\*>(&newInstitution->edeboCode), sizeof(int))).
        + Для кожного рядкового поля:
          1. Зчитати розмір size\_t length.
          2. Виділити буфер char\* buffer = new char[length+1];
          3. file.read(buffer, length); buffer[length] = '\0'; newInstitution-><поле> = string(buffer); delete[] buffer;.
        + Булеві поля (file.read(reinterpret\_cast<char\*>(&newInstitution->crimUkraine), sizeof(bool)), і т.д.).
        + foundationYear (file.read(reinterpret\_cast<char\*>(&newInstitution->foundationYear), sizeof(int))).
     3. Встановити newInstitution->next = nullptr.
     4. Додавати в кінець списку:
     5. if (head == nullptr) {
     6. head = tail = newInstitution;
     7. } else {
     8. tail->next = newInstitution;
     9. tail = newInstitution;
     10. }
   * Закрити файл.
   * Обчислити count = getRegistrySize(head).
   * Вивести “Завантажено N записів з файлу.”
   * Повернути head.

**1.2.4. Алгоритм функції codeExists**

1. Якщо head == nullptr → повернути false.
2. Ітеруватися по списку:
3. current = head;
4. while (current != nullptr) {
5. if (current->edeboCode == code) return true;
6. current = current->next;
7. }
8. return false;

**1.3. Тестування модуля Авраменка**

**Пояснення:** Тест-сьюти призначені для перевірки коректності addRecord, loadFromFile та codeExists.  
Використовуємо порожній тестовий файл, файл із кількома записами та різні сценарії додавання.

| **Test Case ID** | **Action (test steps)** | **Expected Result** | **Test Result** |
| --- | --- | --- | --- |
| A-01 | **loadFromFile:** Запустити програму з відсутністю файлу test.bin. |  |  |

1. Переконатися, що файлу немає на диску.
2. Виклик loadFromFile("test.bin"). | Виводиться: “Файл test.bin не знайдено. Створюється новий реєстр.”  
   Повертається head = nullptr. | passed |  
   | A-02 | **loadFromFile:** Створити бінарний файл test2.bin із 2 записами (попередньо застосувати saveToFile або вручну).
3. Виклик loadFromFile("test2.bin").
4. Порахувати кількість вузлів у списку (getRegistrySize). | Виводиться: “Завантажено 2 записів з файлу.”  
   getRegistrySize(head) повертає 2. | passed |  
   | A-03 | **codeExists:**
5. Виклик codeExists(head, код\_існуючого\_запису).
6. Виклик codeExists(head, код\_неіснуючого\_запису). | Перший виклик → true.  
   Другий виклик → false. | passed |  
   | A-04 | **addRecord:** Додати запис із унікальним позитивним кодом у порожній список.
7. Виклик addRecord(nullptr).
8. Ввести: edeboCode = 10, решта полів коректні. | Після введення → вивід “Запис успішно додано!”  
   Повертається новий head (edeboCode == 10, next == nullptr). | passed |  
   | A-05 | **addRecord (повторний код):**
9. Припустимо, у списку вже існує вузол із edeboCode = 20.
10. Виклик addRecord(head).
11. Ввести edeboCode = 20 (існуючий) → очікується повідомлення “Код уже існує…” → ввести edeboCode = 30.
12. Решта полів введено коректно. | Після введення 20 → вивід “Помилка! Код уже існує. Введіть інший.”  
    Після введення 30 → “Запис успішно додано!”  
    У результаті в списку маємо новий вузол із edeboCode = 30 на початку. | passed |  
    | A-06 | **addRecord (некоректні дані):**
13. Виклик addRecord(nullptr).
14. Ввести edeboCode = -5 → очікується повторний запит “Помилка! Введіть коректний код”.
15. Ввести далі решту полів, але в полі “рік заснування” вводиться 500 (менше 1000) → повторний запит.
16. Потім коректний рік, решта полів коректно. | Після введення некоректного коду/року – відповідні повідомлення про помилку;  
    Після коректних – “Запис успішно додано!”.  
    У списку створений вузол із правильними полями. | passed |