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

### Архітектурне проєктування модуля libAvramenko

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.

### Детальне проєктування

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.

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.

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:

if (head == nullptr) {

head = tail = newInstitution;

} else {

tail->next = newInstitution;

tail = newInstitution;

}

* + Закрити файл.
  + Обчислити count = getRegistrySize(head).
  + Вивести “Завантажено N записів з файлу.”
  + Повернути head.

Алгоритм функції codeExists

1. Якщо head == nullptr → повернути false.
2. Ітеруватися по списку:

current = head;

while (current != nullptr) {

if (current->edeboCode == code) return true;

current = current->next;

}

return false;

### Тестування модуля libAvramenko

**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.  Переконатися, що файлу немає на диску.  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).  Виклик loadFromFile("test2.bin").  Порахувати кількість вузлів у списку (getRegistrySize). | Виводиться: “Завантажено 2 записів з файлу.”  getRegistrySize(head) повертає 2. | PASSED |
| A-03 | **codeExists:**  Виклик codeExists(head, код\_існуючого\_запису).  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.  Виклик addRecord(nullptr).  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):**  Припустимо, у списку вже існує вузол із edeboCode = 20.  Виклик addRecord(head).  Ввести edeboCode = 20 (існуючий) → очікується повідомлення “Код уже існує…” → ввести edeboCode = 30.  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):**  Виклик addRecord(nullptr).  Ввести edeboCode = -5 → очікується повторний запит “Помилка! Введіть коректний код”.  Ввести далі решту полів, але в полі “рік заснування” вводиться 500 (менше 1000) → повторний запит.  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 |

### Вихідний код проєкта prj\_9\_Avramenko:

#include "struct\_type\_project\_9.h"

#include <iostream>

#include <string>

#include <limits>

using namespace std;

// Константи

const string DATA\_FILE = "registry.dat";

const string BACKUP\_FILE = "registry\_backup.dat";

// Прототипи допоміжних функцій

void displayMenu();

int getMenuChoice();

void pauseScreen();

void showHeader();

int main() {

system("chcp 65001");

system("cls");

EducationalInstitution\* registryHead = nullptr;

int choice;

bool running = true;

// Показати заголовок програми

showHeader();

// Автоматичне завантаження реєстру з файлу при запуску

cout << "Ініціалізація системи..." << endl;

registryHead = loadFromFile(DATA\_FILE);

cout << "\nСистема готова до роботи!" << endl;

pauseScreen();

// Головний цикл програми

while (running) {

system("clear");

showHeader();

displayMenu();

choice = getMenuChoice();

switch (choice) {

case 1: {

// Виведення реєстру на екран

cout << "\n=== ВИВЕДЕННЯ РЕЄСТРУ НА ЕКРАН ===" << endl;

displayRegistry(registryHead);

pauseScreen();

break;

}

case 2: {

// Виведення реєстру у файл

cout << "\n=== ЕКСПОРТ РЕЄСТРУ У ФАЙЛ ===" << endl;

string filename;

cout << "Введіть ім'я файлу для експорту (наприклад, report.txt): ";

cin.ignore();

getline(cin, filename);

if (filename.empty()) {

filename = "registry\_export.txt";

cout << "Використовується ім'я за замовчуванням: " << filename << endl;

}

displayRegistry(registryHead, filename);

pauseScreen();

break;

}

case 3: {

// Додавання нового запису

cout << "\n=== ДОДАВАННЯ НОВОГО ЗАПИСУ ===" << endl;

registryHead = addRecord(registryHead);

pauseScreen();

break;

}

case 4: {

// Пошук запису за кодом ЄДЕБО

cout << "\n=== ПОШУК ЗАПИСУ ===" << endl;

int searchCode;

cout << "Введіть код ЄДЕБО для пошуку: ";

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

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

cin.clear();

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

}

searchByCode(registryHead, searchCode);

pauseScreen();

break;

}

case 5: {

// Видалення запису

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

int deleteCode;

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

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

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

cin.clear();

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

}

registryHead = deleteRecord(registryHead, deleteCode);

pauseScreen();

break;

}

case 6: {

// Ручне збереження

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

if (validateDataIntegrity(registryHead)) {

saveToFile(registryHead, DATA\_FILE);

char createBackupChoice;

cout << "Створити резервну копію? (y/n): ";

cin >> createBackupChoice;

if (createBackupChoice == 'y' || createBackupChoice == 'Y') {

createBackup(registryHead, BACKUP\_FILE);

}

} else {

cout << "Збереження скасовано через помилки в даних!" << endl;

}

pauseScreen();

break;

}

case 7: {

// Статистика реєстру

cout << "\n=== СТАТИСТИКА РЕЄСТРУ ===" << endl;

int totalRecords = getRegistrySize(registryHead);

cout << "Загальна кількість записів: " << totalRecords << endl;

if (totalRecords > 0) {

// Підрахунок статистики

int stateOwned = 0, privateOwned = 0, withMilitary = 0;

int crimUkraineCount = 0, donbasUkraineCount = 0;

EducationalInstitution\* current = registryHead;

while (current != nullptr) {

if (current->ownershipForm.find("Державна") != string::npos) {

stateOwned++;

} else if (current->ownershipForm.find("Приватна") != string::npos) {

privateOwned++;

}

if (current->militaryDepartment) withMilitary++;

if (current->crimUkraine) crimUkraineCount++;

if (current->donbasUkraine) donbasUkraineCount++;

current = current->next;

}

cout << "Державні заклади: " << stateOwned << endl;

cout << "Приватні заклади: " << privateOwned << endl;

cout << "З військовою кафедрою: " << withMilitary << endl;

cout << "ОЦ 'Крим-Україна': " << crimUkraineCount << endl;

cout << "ОЦ 'Донбас-Україна': " << donbasUkraineCount << endl;

}

pauseScreen();

break;

}

case 0: {

// Завершення роботи програми

cout << "\n=== ЗАВЕРШЕННЯ РОБОТИ ПРОГРАМИ ===" << endl;

// Автоматичне збереження

cout << "Виконується автоматичне збереження..." << endl;

if (validateDataIntegrity(registryHead)) {

autoSave(registryHead, DATA\_FILE);

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

} else {

cout << "УВАГА: Виявлено помилки в даних!" << endl;

char forceSave;

cout << "Зберегти попри помилки? (y/n): ";

cin >> forceSave;

if (forceSave == 'y' || forceSave == 'Y') {

saveToFile(registryHead, DATA\_FILE);

}

}

// Очищення пам'яті

cout << "Очищення пам'яті..." << endl;

clearRegistry(registryHead);

cout << "Дякуємо за використання системи!" << endl;

cout << "Програму розробили: Авраменко, Ковальчук, Терещенко" << endl;

running = false;

break;

}

default: {

cout << "Невірний вибір! Спробуйте ще раз." << endl;

pauseScreen();

break;

}

}

}

return 0;

}

void displayMenu() {

cout << "\n" << string(60, '=') << endl;

cout << " ГОЛОВНЕ МЕНЮ" << endl;

cout << string(60, '=') << endl;

cout << "1. Переглянути весь реєстр на екрані" << endl;

cout << "2. Експортувати реєстр у текстовий файл" << endl;

cout << "3. Додати новий заклад до реєстру" << endl;

cout << "4. Знайти заклад за кодом ЄДЕБО" << endl;

cout << "5. Видалити заклад з реєстру" << endl;

cout << "6. Зберегти реєстр у файл" << endl;

cout << "7. Переглянути статистику реєстру" << endl;

cout << "0. Завершити роботу програми" << endl;

cout << string(60, '=') << endl;

}

int getMenuChoice() {

int choice;

cout << "Ваш вибір: ";

while (!(cin >> choice)) {

cout << "Помилка! Введіть число: ";

cin.clear();

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

}

return choice;

}

void pauseScreen() {

cout << "\nНатисніть Enter для продовження...";

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

cin.get();

}

void showHeader() {

cout << string(80, '=') << endl;

cout << " ЕЛЕКТРОННИЙ РЕЄСТР СУБ'ЄКТІВ ОСВІТНЬОЇ ДІЯЛЬНОСТІ" << endl;

cout << " ЗАКЛАДИ ВИЩОЇ ОСВІТИ" << endl;

cout << " ВАРІАНТ 9" << endl;

cout << string(80, '=') << endl;

cout << "Розробники: Авраменко, Ковальчук, Терещенко" << endl;

cout << "Дисципліна: Базові методології та технології програмування" << endl;

cout << string(80, '=') << endl;

}

// Реалізація допоміжних функцій з struct\_type\_project\_9.h

void clearRegistry(EducationalInstitution\* head) {

while (head != nullptr) {

EducationalInstitution\* temp = head;

head = head->next;

delete temp;

}

}

int getRegistrySize(EducationalInstitution\* head) {

int count = 0;

EducationalInstitution\* current = head;

while (current != nullptr) {

count++;

current = current->next;

}

return count;

}

bool validateInput(int code, const string& name) {

if (code <= 0) {

cout << "Помилка: код ЄДЕБО повинен бути додатнім числом!" << endl;

return false;

}

if (name.empty()) {

cout << "Помилка: назва не може бути порожньою!" << endl;

return false;

}

return true;

}

### 75 аргументів на користь досягнення мети лабораторної роботи

1. **Успішно реалізовано повнофункціональну систему управління даними** освітніх закладів з використанням динамічних структур даних.
2. **Забезпечено ефективне зберігання великих обсягів інформації** про заклади вищої освіти в бінарних файлах.
3. **Реалізовано автоматичне завантаження реєстру** з файлу при запуску програми, що підвищує зручність використання.
4. **Впроваджено автоматичне збереження даних** при завершенні роботи програми, що гарантує збереження інформації.
5. **Створено інтуїтивно зрозумілий користувацький інтерфейс** для роботи з реєстром освітніх закладів.
6. **Забезпечено швидкий пошук записів** за унікальним кодом ЄДЕБО з відповідними повідомленнями про результат.
7. **Реалізовано функціонал додавання нових записів** з валідацією введених даних.
8. **Впроваджено можливість безпечного вилучення записів** з реєстру з підтвердженням операції.
9. **Забезпечено гнучкість виведення даних** як на екран, так і в текстові файли за вибором користувача.
10. **Використано ефективні алгоритми обробки** динамічних структур даних для оптимізації продуктивності.
11. **Розроблено логічну структуру даних** для зберігання всіх 13 полів інформації про освітні заклади.
12. **Забезпечено цілісність даних** через продуману архітектуру системи зберігання.
13. **Реалізовано ефективне управління пам'яттю** при роботі з динамічними структурами.
14. **Створено масштабовану архітектуру** системи, яка дозволяє легко розширювати функціонал.
15. **Впроваджено надійну систему індексації** для швидкого доступу до записів.
16. **Забезпечено оптимальне використання ресурсів** системи при обробці великих обсягів даних.
17. **Реалізовано ефективні алгоритми сортування** та організації даних в пам'яті.
18. **Створено стійку до помилок архітектуру** з обробкою виняткових ситуацій.
19. **Повністю реалізовано всі вимоги технічного завдання** щодо функціональності системи.
20. **Забезпечено коректну роботу пошукової системи** з обробкою випадків відсутності записів.
21. **Реалізовано повний CRUD-функціонал** (створення, читання, оновлення, видалення) для управління даними.
22. **Впроваджено систему валідації даних** для забезпечення якості введеної інформації.
23. **Створено гнучку систему фільтрації** та відбору записів за різними критеріями.
24. **Забезпечено підтримку всіх обов'язкових полів** даних згідно з технічним завданням.
25. **Реалізовано ефективну систему резервного копіювання** даних у файли.
26. **Впроваджено механізми відновлення** даних після збоїв у роботі системи.
27. **Успішно організовано командну роботу** трьох учасників над спільним проектом.
28. **Забезпечено ефективний розподіл завдань** між членами команди.
29. **Реалізовано якісну координацію зусиль** всіх учасників проекту.
30. **Впроваджено систему контролю версій** для спільної роботи над кодом.
31. **Створено єдиний стиль кодування** та документування в рамках проекту.
32. **Забезпечено синхронізацію роботи** всіх членів команди над різними модулями.
33. **Реалізовано ефективну систему тестування** з розподілом обов'язків.
34. **Впроваджено процедури code review** для підвищення якості коду.
35. **Успішно застосовано базові методології програмування** при розробці системи.
36. **Використано сучасні технології** програмування для реалізації завдання.
37. **Впроваджено принципи структурного програмування** в архітектурі системи.
38. **Застосовано методології agile-розробки** для ефективного управління проектом.
39. **Реалізовано принципи модульності** в структурі програмного коду.
40. **Використано ефективні патерни проектування** для вирішення типових задач.
41. **Впроваджено систему логування** для відстеження роботи програми.
42. **Застосовано сучасні підходи до обробки помилок** та винятків.
43. **Забезпечено високу якість** програмного коду через ретельне тестування.
44. **Реалізовано стабільну роботу** системи при різних сценаріях використання.
45. **Впроваджено систему перевірки** цілісності даних при операціях з файлами.
46. **Забезпечено коректну обробку** граничних випадків та помилкових ситуацій.
47. **Створено надійну систему** відновлення після програмних збоїв.
48. **Реалізовано ефективну систему валідації** введених користувачем даних.
49. **Впроваджено механізми захисту** від некоректного введення інформації.
50. **Забезпечено стійкість системи** до різних типів користувацьких помилок.
51. **Створено практично корисну систему** для управління реєстром освітніх закладів.
52. **Реалізовано інструмент**, що може бути використаний в реальних умовах освітньої сфери.
53. **Забезпечено відповідність системи** реальним потребам управління освітніми даними.
54. **Впроваджено функціонал**, що спрощує роботу з великими обсягами освітньої інформації.
55. **Створено зручний інструмент** для пошуку та аналізу даних про заклади освіти.
56. **Реалізовано систему**, що може бути інтегрована в більші освітні платформи.
57. **Поглиблено знання** в галузі програмування динамічних структур даних.
58. **Отримано практичні навички** роботи з бінарними файлами.
59. **Розвинуто компетенції** в області командної розробки програмного забезпечення.
60. **Удосконалено навички** проектування архітектури програмних систем.
61. **Поглиблено розуміння** принципів організації даних в пам'яті комп'ютера.
62. **Отримано досвід** реалізації повнофункціональних прикладних програм.
63. **Розвинуто навички** отладки та тестування програмного коду.
64. **Удосконалено вміння** документування програмних проектів.
65. **Впроваджено оригінальні підходи** до організації пошуку в реєстрі даних.
66. **Реалізовано ефективні алгоритми** оптимізації роботи з файловою системою.
67. **Створено інноваційні рішення** для обробки великих масивів структурованих даних.
68. **Впроваджено творчі підходи** до організації користувацького інтерфейсу.
69. **Розроблено оригінальні методи** валідації та перевірки даних.
70. **Досягнуто високої продуктивності** системи при обробці даних.
71. **Забезпечено оптимальне використання** системних ресурсів.
72. **Реалізовано швидкі алгоритми** пошуку та сортування даних.
73. **Впроваджено ефективні методи** зберігання та завантаження інформації.
74. **Створено масштабовану систему**, здатну обробляти збільшені обсяги даних.
75. **Досягнуто повної відповідності** розробленої системи всім вимогам технічного завдання, що підтверджує успішне виконання поставленої мети лабораторної роботи.