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.
* Робота в діалоговому режимі з користувачем через консольне меню.

#### **3. Вимоги до програмного забезпечення**

##### **Функціональні вимоги:**

* Програма повинна дозволяти вводити й редагувати повні відомості про заклади освіти.
* Забезпечити збереження даних між сесіями за допомогою текстового або бінарного файлу.
* Наявність інтерфейсу для взаємодії з користувачем (меню).
* Обробка некоректного вводу з боку користувача (мінімальний захист від помилок).

**Формати вводу/виводу:**

Ввід: із клавіатури.

Вивід: у консоль або текстовий файл.

**Обраний вид динамічної структури (однозв’язний список):**

Для реалізації бази даних «Деканат: облік студентів» обрано **однозв’язний список**, оскільки він:

* дозволяє **динамічно змінювати розмір** бази без попереднього резервування пам’яті;
* забезпечує **швидке додавання, видалення та перегляд записів**;
* простий у реалізації та зручний для **лінійного пошуку за прізвищем**, що повністю відповідає вимогам завдання.

Інші структури (дерева, стек, черга) або складніші у реалізації, або обмежують доступ до даних.

**Обрані типи:**

**Обрані типи:**

* string — використовується для зберігання текстових даних:
  + код закладу (code)
  + повна та коротка назва (fullName, shortName)
  + англомовна назва (fullNameEng)
  + форма власності (ownership)
  + керівний орган (governingBody)
  + ПІБ та посада керівника (headFullName, headPosition)
  + адреса, телефон/факс, email, сайт (address, phoneFax, email, website)
  + позначки щодо окупованих територій та наявності військової кафедри (ocCrimea, ocDonbas, militaryDepartment)
* int — використовується для зберігання числового значення:

**План виконання ІТ-проєкта:**

|  |  |
| --- | --- |
| **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 |

# ПОРЯДОК ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

**Загальний опис** Функція:

1. Постановка задачі
2. Проєктування структури даних.
3. Створення заголовкового файлу
4. Розробка основних функцій програми.
5. Розробка головного меню
6. Оформлення програми

**Лістинг функції:**

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

using namespace std;

struct EducationInstitution {

string code;

string fullName;

string shortName;

string fullNameEng;

string ownership;

string governingBody;

string headPosition;

string headFullName;

string address;

string phoneFax;

string email;

string website;

string ocCrimea;

string ocDonbas;

string militaryDepartment;

int foundationYear;

};

void loadFromFile(vector<EducationInstitution>& list, const string& filename) {

ifstream file(filename);

if (!file.is\_open()) return;

EducationInstitution inst;

while (getline(file, inst.code)) {

getline(file, inst.fullName);

getline(file, inst.shortName);

getline(file, inst.fullNameEng);

getline(file, inst.ownership);

getline(file, inst.governingBody);

getline(file, inst.headPosition);

getline(file, inst.headFullName);

getline(file, inst.address);

getline(file, inst.phoneFax);

getline(file, inst.email);

getline(file, inst.website);

getline(file, inst.ocCrimea);

getline(file, inst.ocDonbas);

getline(file, inst.militaryDepartment);

file >> inst.foundationYear;

file.ignore(); // перейти на наступний рядок

list.push\_back(inst);

}

file.close();

}

void saveToFile(const vector<EducationInstitution>& list, const string& filename) {

ofstream file(filename);

for (const auto& inst : list) {

file << inst.code << "\n"

<< inst.fullName << "\n"

<< inst.shortName << "\n"

<< inst.fullNameEng << "\n"

<< inst.ownership << "\n"

<< inst.governingBody << "\n"

<< inst.headPosition << "\n"

<< inst.headFullName << "\n"

<< inst.address << "\n"

<< inst.phoneFax << "\n"

<< inst.email << "\n"

<< inst.website << "\n"

<< inst.ocCrimea << "\n"

<< inst.ocDonbas << "\n"

<< inst.militaryDepartment << "\n"

<< inst.foundationYear << "\n";

}

file.close();

}

void displayAll(const vector<EducationInstitution>& list) {

for (const auto& inst : list) {

cout << "Код: " << inst.code << "\n"

<< "Назва: " << inst.fullName << "\n"

<< "Коротка назва: " << inst.shortName << "\n"

<< "Назва англ.: " << inst.fullNameEng << "\n"

<< "Форма власності: " << inst.ownership << "\n"

<< "Орган: " << inst.governingBody << "\n"

<< "Посада керівника: " << inst.headPosition << "\n"

<< "ПІБ керівника: " << inst.headFullName << "\n"

<< "Адреса: " << inst.address << "\n"

<< "Тел./Факс: " << inst.phoneFax << "\n"

<< "Email: " << inst.email << "\n"

<< "Сайт: " << inst.website << "\n"

<< "ОЦ Крим: " << inst.ocCrimea << "\n"

<< "ОЦ Донбас: " << inst.ocDonbas << "\n"

<< "Військова кафедра: " << inst.militaryDepartment << "\n"

<< "Рік заснування: " << inst.foundationYear << "\n\n";

}

}

void addInstitution(vector<EducationInstitution>& list) {

EducationInstitution inst;

cout << "\nВведіть дані нового закладу:\n";

cout << "Код: "; getline(cin, inst.code);

cout << "Повна назва: "; getline(cin, inst.fullName);

cout << "Коротка назва: "; getline(cin, inst.shortName);

cout << "Назва англ.: "; getline(cin, inst.fullNameEng);

cout << "Форма власності: "; getline(cin, inst.ownership);

cout << "Орган: "; getline(cin, inst.governingBody);

cout << "Посада керівника: "; getline(cin, inst.headPosition);

cout << "ПІБ керівника: "; getline(cin, inst.headFullName);

cout << "Адреса: "; getline(cin, inst.address);

cout << "Тел./Факс: "; getline(cin, inst.phoneFax);

cout << "Email: "; getline(cin, inst.email);

cout << "Сайт: "; getline(cin, inst.website);

cout << "ОЦ Крим: "; getline(cin, inst.ocCrimea);

cout << "ОЦ Донбас: "; getline(cin, inst.ocDonbas);

cout << "Військова кафедра: "; getline(cin, inst.militaryDepartment);

cout << "Рік заснування: "; cin >> inst.foundationYear;

cin.ignore();

list.push\_back(inst);

}

void searchByCode(const vector<EducationInstitution>& list, const string& code) {

for (const auto& inst : list) {

if (inst.code == code) {

cout << "Знайдено запис:\n";

cout << "Назва: " << inst.fullName << "\n";

return;

}

}

cout << "Запис з кодом " << code << " не знайдено.\n";

}

void deleteByCode(vector<EducationInstitution>& list, const string& code) {

for (auto it = list.begin(); it != list.end(); ++it) {

if (it->code == code) {

list.erase(it);

cout << "Запис видалено.\n";

return;

}

}

cout << "Запис не знайдено.\n";

}

int main() {

vector<EducationInstitution> registry;

string filename = "registry.txt";

loadFromFile(registry, filename);

int choice;

do {

cout << "\n1. Вивести реєстр\n2. Додати запис\n3. Пошук за кодом\n4. Видалити запис\n5. Завершити\nВаш вибір: ";

cin >> choice;

cin.ignore();

if (choice == 1) displayAll(registry);

else if (choice == 2) addInstitution(registry);

else if (choice == 3) {

string code;

cout << "Введіть код: "; getline(cin, code);

searchByCode(registry, code);

}

else if (choice == 4) {

string code;

cout << "Введіть код для видалення: "; getline(cin, code);

deleteByCode(registry, code);

}

} while (choice != 5);

saveToFile(registry, filename);

cout << "Роботу завершено. Реєстр збережено.\n";

return 0;

}

Принцип роботи

**1)Завантажує дані** з файлу у вектор структур EducationInstitution.

2) **Показує меню** користувачу для вибору дії.

3) **Виводить усі записи** у зручному форматі.

4) **Додає новий запис**, вводячи всі поля з клавіатури.

5) **Шукає запис за кодом** і виводить назву, якщо знайдено.

6) **Видаляє запис за кодом** або повідомляє, якщо запис не знайдено.

7)Виводить повідомлення про результат.

**Лістинг функції:**

int main() {

vector<EducationInstitution> registry;

string filename = "registry.txt";

loadFromFile(registry, filename);

int choice;

do {

cout << "\n1. Вивести реєстр\n"

<< "2. Додати запис\n"

<< "3. Пошук за кодом\n"

<< "4. Видалити запис\n"

<< "5. Завершити\n"

<< "Ваш вибір: ";

cin >> choice;

cin.ignore();

if (choice == 1) {

displayAll(registry); // Вивід усіх записів

}

else if (choice == 2) {

addInstitution(registry);

}

else if (choice == 3) {

string code;

cout << "Введіть код: ";

getline(cin, code);

searchByCode(registry, code);

}

else if (choice == 4) {

string code;

cout << "Введіть код для видалення: ";

getline(cin, code);

deleteByCode(registry, code);

}

} while (choice != 5);

saveToFile(registry, filename);

cout << "Роботу завершено. Реєстр збережено.\n";

return 0;

}

**Лістинг .h файлу визначення структур:**

#ifndef EDUCATION\_INSTITUTION\_H

#define EDUCATION\_INSTITUTION\_H

#include <string>

struct EducationInstitution {

std::string code; // Код закладу

std::string fullName; // Повна назва

std::string shortName; // Коротка назва

std::string fullNameEng; // Назва англійською

std::string ownership; // Форма власності

std::string governingBody; // Орган управління

std::string headPosition; // Посада керівника

std::string headFullName; // ПІБ керівника

std::string address; // Адреса

std::string phoneFax; // Телефон / факс

std::string email; // Електронна пошта

std::string website; // Сайт

std::string ocCrimea; // Освітній центр Крим

std::string ocDonbas; // Освітній центр Донбас

std::string militaryDepartment; // Наявність військової кафедри

int foundationYear; // Рік заснування

};

#endif

**Лістинг main.cpp файлу:**

#include "interface.h" int

main() {

system("chcp 65001 > nul");

handleUserChoice(); // Основна функція для вибору операцій

return 0;

}

**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.