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

Заріцкий В. А.

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 №2**

**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**

* Заріцкий Віктор (<https://github.com/odorenskyi/Zaritskyi-Viktor-KB24>)
* Кондратенко Дмитро (<https://github.com/odorenskyi/Kondratenko-Dmytro-KB24>)
* Колесник Віктор(<https://github.com/odorenskyi/Kolesnyk-Viktor-KB24>)

# Аналіз задач ІТ‑проєкту та вимог до ПЗ

1. Завантаження бази з файлу при старті програми.
2. Відображення всієї бази на екрані або вивід у текстовий файл (за вибором користувача).
3. Додавання нового запису («особова картка»).
4. Пошук запису за прізвищем працівника.
5. Видалення запису за табельним номером.
6. Збереження бази у файл при завершенні роботи.

# Специфікації ПЗ, концептуальні рішення, архітектура

— Головний файл (main.cpp) — єдиний «оркестратор» з консольним меню.

— Три незалежні модулі (по 2 функції кожний), усі залежать тільки від спільного заголовка struct\_type\_project\_2.h.

Схема взаємодії:

main.cpp → Module\_File\_Zaritskyi

main.cpp → Module\_CRUD\_Kondratenko

main.cpp → Module\_SearchDel\_Kolesnyk

Module\_File\_Zaritskyi → struct\_type\_project\_2.h

Module\_CRUD\_Kondratenko → struct\_type\_project\_2.h

Module\_SearchDel\_Kolesnyk → struct\_type\_project\_2.h

Алгоритми викликів із main.cpp:

1. Завантажити: loadDatabase(head, filename)
2. Меню оператора: виклик addRecord, printAllRecords, findRecordByName або deleteRecord
3. На виході: saveDatabase(head, filename)

# Вибір динамічної структури даних

Тип: однозв’язний список.

Обґрунтування:

* Динамічність (будь-яка кількість записів).
* Простота вставки/видалення та проходження лінійно (O(n) — прийнятно для кількох сотень записів).

Вузол списку (Node):

struct Node {

EmployeeRecord record;

Node\* next;

Node(EmployeeRecord rec) : record(rec), next(nullptr) {}

};

Таблиця 1 – Розподіл підзадач (по модулях)

|  |  |  |
| --- | --- | --- |
| **Модуль** | **Студент** | **Функції** |
| Module\_File\_Zaritskyi | Заріцкий Віктор | loadDatabase(), saveDatabase() |
| Module\_CRUD\_Kondratenko | Кондратенко Дмитро | addRecord(), printAllRecords() |
| Module\_SearchDel\_Kolesnyk | Колесник Віктор | findRecordByName(), deleteRecord() |

Таблиця 2 – План робіт за ISO/IEC 12207

|  |  |
| --- | --- |
| **Етап** | **Відповідальні** |
| 1. Планування | Усі |
| 2. Реалізація модулів | За модулями |
| 3. Інтеграція та тестування | Усі |
| 4. Верифікація відповідності вимогам | Усі |
| 5. Валідація, демонстрація викладачу | Усі + викладач |

# Модуль Module\_File\_Zaritskyi

Модуль Module\_File\_Zaritskyi забезпечує функції для роботи з бінарним представленням бази даних у пам’яті та на диску:

* loadDatabase(Node\*& head, const std::string& filename) – завантажує список записів із бінарного файлу у зв’язаний список у пам’яті;
* saveDatabase(Node\* head, const std::string& filename) – зберігає поточний зв’язаний список у бінарний файл.

Обидві функції повертають bool, що сигналізує про успіх (true) чи невдачу (false) операції.

## Функція loadDatabase

**Призначення:**

* Ініціалізує порожній список (у разі наявності даних – звільняє пам’ять).
* Зчитує спочатку кількість записів (recordCount), потім кожен запис у цикл­у.
* Побудоває нові вузли Node із відновленими EmployeeRecord і ланцюжить їх у список.

**Особливості:**

* Перед зчитуванням старий список звільняється, щоб уникнути витоку пам’яті.
* Після читання кожного блоку перевіряється file.fail(), і в разі помилки відбувається очищення вже створених вузлів.

## Функція saveDatabase

**Призначення:**

* Обчислює кількість записів у списку (recordCount).
* Записує спочатку recordCount, потім послідовно бінарізує поля кожного EmployeeRecord:
  + прості типи (Date, int, short int, enum) через reinterpret\_cast,
  + строки — через запис size\_t len + c\_str() з довжиною .

**Особливості:**

* Прямий бінарний дамп структур, що дає високу швидкість.
* Після успіху виводить кількість збережених записів — зручно для відлагодження.

## Обробка помилок

* Обидві функції повертають false і виводять діагностичне повідомлення через std::cout, якщо не вдається відкрити файл чи виникають помилки введення/виведення.
* При критичній помилці в loadDatabase модуль очищує вже зчитані дані, щоб не залишати неповний список.

# Вихідний код проєкту prj\_2\_Zaritskyi

## main.cpp

#include <iostream>

#include <string>

#include <limits>

#include "struct\_type\_project\_2.h"

#include "Module\_File\_Zaritskyi.h"

#include "Module\_CRUD\_Kondratenko.h"

#include "Module\_SearchDel\_Kolesnyk.h"

const std::string DATABASE\_FILENAME = "employee\_database.bin";

void clearScreen() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

void pauseExecution() {

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

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

}

void displayMenu() {

clearScreen();

std::cout << "=== Відділ кадрів: особова картка працівника ===\n" << std::endl;

std::cout << "1. Переглянути всю базу даних на екрані" << std::endl;

std::cout << "2. Вивести базу даних у текстовий файл" << std::endl;

std::cout << "3. Додати новий запис" << std::endl;

std::cout << "4. Пошук запису за прізвищем" << std::endl;

std::cout << "5. Видалити запис за табельним номером" << std::endl;

std::cout << "0. Вихід" << std::endl;

std::cout << "\nВиберіть опцію: ";

}

int main() {

#ifdef \_WIN32

system("chcp 65001");

system("cls");

#endif

Node\* head = nullptr;

loadDatabase(head, DATABASE\_FILENAME);

int choice;

do {

displayMenu();

if (!(std::cin >> choice)) {

std::cin.clear();

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

std::cout << "Некоректне введення. Спробуйте ще раз." << std::endl;

pauseExecution();

continue;

}

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

switch (choice) {

case 1: // Переглянути всю базу

{

clearScreen();

std::cout << "=== Перегляд бази даних ===" << std::endl;

printAllRecords(head);

pauseExecution();

break;

}

case 2: // Вивести базу в текстовий файл

{

clearScreen();

std::cout << "=== Виведення бази даних у файл ===" << std::endl;

std::string filename;

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

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

if (filename.empty()) {

filename = "output.txt";

}

printAllRecords(head, true, filename);

pauseExecution();

break;

}

case 3: // Додати новий запис

{

clearScreen();

addRecord(head);

pauseExecution();

break;

}

case 4: // Пошук за прізвищем

{

clearScreen();

std::cout << "=== Пошук за прізвищем ===" << std::endl;

std::string name;

std::cout << "Введіть прізвище або частину ПІБ для пошуку: ";

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

if (!name.empty()) {

findRecordByName(head, name);

} else {

std::cout << "Пошуковий запит не може бути порожнім." << std::endl;

}

pauseExecution();

break;

}

case 5: // Видалити запис

{

clearScreen();

std::cout << "=== Видалення запису ===" << std::endl;

int id;

std::cout << "Введіть табельний номер працівника для видалення: ";

if (std::cin >> id) {

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

deleteRecord(head, id);

} else {

std::cin.clear();

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

std::cout << "Некоректний табельний номер." << std::endl;

}

pauseExecution();

break;

}

case 0: // Вихід

{

std::cout << "Збереження бази даних перед виходом..." << std::endl;

saveDatabase(head, DATABASE\_FILENAME);

std::cout << "Дякуємо за використання програми!" << std::endl;

break;

}

default:

{

std::cout << "Невідома опція. Спробуйте ще раз." << std::endl;

pauseExecution();

break;

}

}

} while (choice != 0);

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

## struct\_type\_project\_2.h

#ifndef STRUCT\_TYPE\_PROJECT\_2\_H\_INCLUDED

#define STRUCT\_TYPE\_PROJECT\_2\_H\_INCLUDED

#include <string>

using namespace std;

struct Date {

int day;

int month;

int year;

Date() : day(0), month(0), year(0) {}

Date(int d, int m, int y) : day(d), month(m), year(y) {}

bool isEmpty() const {

return day == 0 && month == 0 && year == 0;

}

string toString() const {

if (isEmpty()) return "-";

return to\_string(day) + "." + to\_string(month) + "." + to\_string(year);

}

};

struct EmployeeRecord {

Date date\_filled; // Дата заповнення

int employee\_id; // Табельний номер

string tax\_id; // Ідентифікаційний номер

enum class Gender { M, F, Other } gender; // Стать

string work\_type; // Вид роботи (основна, за сумісництвом)

string full\_name; // ПІБ працівника

Date birth\_date; // Дата народження

string citizenship; // Громадянство

string education; // Освіта

string last\_job; // Останнє місце роботи

string last\_position; // Остання посада

short int exp\_days, exp\_months, exp\_years; // Стаж роботи (днів/місяців/років)

string address; // Місце проживання

string passport; // Паспортні дані

string notes; // Додаткові відомості

Date dismissal\_date; // Дата звільнення

string dismissal\_reason; // Причина звільнення

EmployeeRecord() :

employee\_id(0),

gender(Gender::Other),

exp\_days(0),

exp\_months(0),

exp\_years(0) {}

};

struct Node {

EmployeeRecord record;

Node\* next;

Node(EmployeeRecord rec) : record(rec), next(nullptr) {}

};

inline string genderToString(EmployeeRecord::Gender gender) { //вбудована функція, яка не визначає ту саму функцію більше одного разу

switch (gender) {

case EmployeeRecord::Gender::M: return "Чоловіча";

case EmployeeRecord::Gender::F: return "Жіноча";

default: return "Інше";

}

}

#endif // STRUCT\_TYPE\_PROJECT\_2\_H\_INCLUDED

# Вихідний код модулю Module\_File\_Zaritskyi

## Module\_File\_Zaritskyi.cpp

#include "Module\_File\_Zaritskyi.h"

bool loadDatabase(Node\*& head, const std::string& filename) {

std::ifstream file(filename, std::ios::binary);

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

std::cout << "Не вдалося відкрити файл для читання: " << filename << std::endl;

return false;

}

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

head = nullptr;

size\_t recordCount = 0;

file.read(reinterpret\_cast<char\*>(&recordCount), sizeof(recordCount));

if (file.fail()) {

std::cout << "Помилка читання кількості записів" << std::endl;

file.close();

return false;

}

Node\* current = nullptr;

for (size\_t i = 0; i < recordCount; ++i) {

EmployeeRecord record;

file.read(reinterpret\_cast<char\*>(&record.date\_filled), sizeof(Date));

file.read(reinterpret\_cast<char\*>(&record.employee\_id), sizeof(int));

size\_t len;

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.tax\_id.resize(len);

file.read(&record.tax\_id[0], len);

file.read(reinterpret\_cast<char\*>(&record.gender), sizeof(EmployeeRecord::Gender));

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.work\_type.resize(len);

file.read(&record.work\_type[0], len);

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.full\_name.resize(len);

file.read(&record.full\_name[0], len);

file.read(reinterpret\_cast<char\*>(&record.birth\_date), sizeof(Date));

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.citizenship.resize(len);

file.read(&record.citizenship[0], len);

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.education.resize(len);

file.read(&record.education[0], len);

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.last\_job.resize(len);

file.read(&record.last\_job[0], len);

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.last\_position.resize(len);

file.read(&record.last\_position[0], len);

file.read(reinterpret\_cast<char\*>(&record.exp\_days), sizeof(short int));

file.read(reinterpret\_cast<char\*>(&record.exp\_months), sizeof(short int));

file.read(reinterpret\_cast<char\*>(&record.exp\_years), sizeof(short int));

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.address.resize(len);

file.read(&record.address[0], len);

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.passport.resize(len);

file.read(&record.passport[0], len);

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.notes.resize(len);

file.read(&record.notes[0], len);

file.read(reinterpret\_cast<char\*>(&record.dismissal\_date), sizeof(Date));

file.read(reinterpret\_cast<char\*>(&len), sizeof(size\_t));

record.dismissal\_reason.resize(len);

file.read(&record.dismissal\_reason[0], len);

if (file.fail()) {

std::cout << "Помилка читання запису #" << i + 1 << std::endl;

file.close();

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return false;

}

Node\* newNode = new Node(record);

if (head == nullptr) {

head = newNode;

current = head;

} else {

current->next = newNode;

current = newNode;

}

}

file.close();

std::cout << "База даних успішно завантажена з файлу: " << filename << std::endl;

std::cout << "Кількість завантажених записів: " << recordCount << std::endl;

return true;

}

bool saveDatabase(Node\* head, const std::string& filename) {

std::ofstream file(filename, std::ios::binary);

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

std::cout << "Не вдалося відкрити файл для запису: " << filename << std::endl;

return false;

}

size\_t recordCount = 0;

Node\* current = head;

while (current != nullptr) {

recordCount++;

current = current->next;

}

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

current = head;

while (current != nullptr) {

EmployeeRecord record = current->record;

file.write(reinterpret\_cast<const char\*>(&record.date\_filled), sizeof(Date));

file.write(reinterpret\_cast<const char\*>(&record.employee\_id), sizeof(int));

size\_t len = record.tax\_id.length();

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

file.write(record.tax\_id.c\_str(), len);

file.write(reinterpret\_cast<const char\*>(&record.gender), sizeof(EmployeeRecord::Gender));

len = record.work\_type.length();

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

file.write(record.work\_type.c\_str(), len);

len = record.full\_name.length();

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

file.write(record.full\_name.c\_str(), len);

file.write(reinterpret\_cast<const char\*>(&record.birth\_date), sizeof(Date));

len = record.citizenship.length();

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

file.write(record.citizenship.c\_str(), len);

len = record.education.length();

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

file.write(record.education.c\_str(), len);

len = record.last\_job.length();

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

file.write(record.last\_job.c\_str(), len);

len = record.last\_position.length();

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

file.write(record.last\_position.c\_str(), len);

file.write(reinterpret\_cast<const char\*>(&record.exp\_days), sizeof(short int));

file.write(reinterpret\_cast<const char\*>(&record.exp\_months), sizeof(short int));

file.write(reinterpret\_cast<const char\*>(&record.exp\_years), sizeof(short int));

len = record.address.length();

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

file.write(record.address.c\_str(), len);

len = record.passport.length();

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

file.write(record.passport.c\_str(), len);

len = record.notes.length();

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

file.write(record.notes.c\_str(), len);

file.write(reinterpret\_cast<const char\*>(&record.dismissal\_date), sizeof(Date));

len = record.dismissal\_reason.length();

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

file.write(record.dismissal\_reason.c\_str(), len);

current = current->next;

}

if (file.fail()) {

std::cout << "Помилка запису даних у файл" << std::endl;

file.close();

return false;

}

file.close();

std::cout << "База даних успішно збережена у файл: " << filename << std::endl;

std::cout << "Кількість збережених записів: " << recordCount << std::endl;

return true;

}

## Module\_File\_Zaritskyi.h

#ifndef MODULE\_FILE\_ZARITSKYI\_H

#define MODULE\_FILE\_ZARITSKYI\_H

#include "struct\_type\_project\_2.h"

#include <string>

#include <fstream>

#include <iostream>

bool loadDatabase(Node\*& head, const std::string& filename);

bool saveDatabase(Node\* head, const std::string& filename);

#endif // MODULE\_FILE\_ZARITSKYI\_H

# Вихідний код модулю Module\_CRUD\_Kondratenko

## Module\_CRUD\_Kondratenko.cpp

#include "Module\_CRUD\_Kondratenko.h"

#include <iostream>

#include <fstream>

#include <iomanip>

#include <limits>

using namespace std;

Date inputDate(const string& prompt, bool allowEmpty = false) {

Date date;

string input;

while (true) {

cout << prompt;

getline(cin, input);

if (allowEmpty && input.empty()) {

return Date();

}

if (sscanf(input.c\_str(), "%d.%d.%d", &date.day, &date.month, &date.year) == 3) {

if (date.day >= 1 && date.day <= 31 && date.month >= 1 && date.month <= 12 && date.year >= 1900) {

return date;

}

}

cout << "Некоректний формат дати. Використовуйте формат ДД.ММ.РРРР" << endl;

}

}

template<typename T>

T getInput(const string& prompt) {

T value;

while (true) {

cout << prompt;

if (cin >> value) {

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

return value;

} else {

cin.clear();

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

cout << "Некоректне введення. Спробуйте ще раз." << endl;

}

}

}

string getStringInput(const string& prompt) {

string value;

cout << prompt;

getline(cin, value);

return value;

}

bool isEmployeeIdUnique(Node\* head, int employee\_id) {

Node\* current = head;

while (current != nullptr) {

if (current->record.employee\_id == employee\_id) {

return false;

}

current = current->next;

}

return true;

}

bool addRecord(Node\*& head) {

cout << "\n=== Додавання нового запису ===\n" << endl;

EmployeeRecord newRecord;

newRecord.date\_filled = inputDate("Дата заповнення (ДД.ММ.РРРР): ");

while (true) {

int id = getInput<int>("Табельний номер: ");

if (isEmployeeIdUnique(head, id)) {

newRecord.employee\_id = id;

break;

} else {

cout << "Такий табельний номер вже існує. Введіть інший." << endl;

}

}

newRecord.tax\_id = getStringInput("Ідентифікаційний номер: ");

string gender\_input;

while (true) {

cout << "Стать (ч/ж/інше): ";

getline(cin, gender\_input);

if (gender\_input == "ч" || gender\_input == "Ч") {

newRecord.gender = EmployeeRecord::Gender::M;

break;

} else if (gender\_input == "ж" || gender\_input == "Ж") {

newRecord.gender = EmployeeRecord::Gender::F;

break;

} else if (gender\_input == "інше" || gender\_input == "Інше") {

newRecord.gender = EmployeeRecord::Gender::Other;

break;

} else {

cout << "Некоректне введення. Використовуйте 'ч', 'ж' або 'інше'." << endl;

}

}

string work\_type\_input;

while (true) {

cout << "Вид роботи (основна/за сумісництвом): ";

getline(cin, work\_type\_input);

if (work\_type\_input == "основна" || work\_type\_input == "за сумісництвом") {

newRecord.work\_type = work\_type\_input;

break;

} else {

cout << "Некоректне введення. Використовуйте 'основна' або 'за сумісництвом'." << endl;

}

}

newRecord.full\_name = getStringInput("ПІБ працівника: ");

newRecord.birth\_date = inputDate("Дата народження (ДД.ММ.РРРР): ");

newRecord.citizenship = getStringInput("Громадянство: ");

newRecord.education = getStringInput("Освіта: ");

newRecord.last\_job = getStringInput("Останнє місце роботи: ");

newRecord.last\_position = getStringInput("Остання посада: ");

newRecord.exp\_days = getInput<short int>("Стаж роботи (днів): ");

newRecord.exp\_months = getInput<short int>("Стаж роботи (місяців): ");

newRecord.exp\_years = getInput<short int>("Стаж роботи (років): ");

newRecord.address = getStringInput("Місце проживання: ");

newRecord.passport = getStringInput("Паспортні дані: ");

newRecord.notes = getStringInput("Додаткові відомості: ");

while (true) {

string dismissal\_input;

cout << "Чи звільнений працівник? (так/ні): ";

getline(cin, dismissal\_input);

if (dismissal\_input == "так" || dismissal\_input == "Так") {

newRecord.dismissal\_date = inputDate("Дата звільнення (ДД.ММ.РРРР): ");

newRecord.dismissal\_reason = getStringInput("Причина звільнення: ");

break;

} else if (dismissal\_input == "ні" || dismissal\_input == "Ні") {

break;

} else {

cout << "Некоректне введення. Використовуйте 'так' або 'ні'." << endl;

}

}

Node\* newNode = new Node(newRecord);

if (head == nullptr) {

head = newNode;

} else {

Node\* current = head;

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

}

cout << "\nЗапис успішно додано до бази даних.\n" << endl;

return true;

}

void printRecord(ostream& out, const EmployeeRecord& record, int index) {

out << "--- Запис #" << index << " ---" << endl;

out << "Табельний номер: " << record.employee\_id << endl;

out << "Дата заповнення: " << record.date\_filled.toString() << endl;

out << "ПІБ: " << record.full\_name << endl;

out << "Ідентифікаційний номер: " << record.tax\_id << endl;

out << "Стать: " << genderToString(record.gender) << endl;

out << "Вид роботи: " << record.work\_type << endl;

out << "Дата народження: " << record.birth\_date.toString() << endl;

out << "Громадянство: " << record.citizenship << endl;

out << "Освіта: " << record.education << endl;

out << "Останнє місце роботи: " << record.last\_job << endl;

out << "Остання посада: " << record.last\_position << endl;

out << "Стаж роботи: " << record.exp\_years << " років, "

<< record.exp\_months << " місяців, "

<< record.exp\_days << " днів" << endl;

out << "Місце проживання: " << record.address << endl;

out << "Паспортні дані: " << record.passport << endl;

out << "Додаткові відомості: " << record.notes << endl;

if (!record.dismissal\_date.isEmpty()) {

out << "Дата звільнення: " << record.dismissal\_date.toString() << endl;

out << "Причина звільнення: " << record.dismissal\_reason << endl;

} else {

out << "Статус: Працює" << endl;

}

out << "------------------------" << endl;

}

bool printAllRecords(Node\* head, bool toFile, const string& filename) {

if (head == nullptr) {

cout << "База даних порожня." << endl;

return false;

}

ofstream file;

if (toFile) {

file.open(filename);

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

cout << "Не вдалося відкрити файл для запису: " << filename << endl;

return false;

}

}

ostream& out = toFile ? file : cout;

out << "\n=== База даних співробітників ===\n" << endl;

Node\* current = head;

int index = 1;

while (current != nullptr) {

printRecord(out, current->record, index++);

current = current->next;

}

if (toFile) {

file.close();

cout << "Дані успішно виведені у файл: " << filename << endl;

}

return true;

}

## Module\_CRUD\_Kondratenko.h

#ifndef MODULE\_CRUD\_KONDRATENKO\_H

#define MODULE\_CRUD\_KONDRATENKO\_H

#include "struct\_type\_project\_2.h"

#include <string>

using namespace std;

bool addRecord(Node\*& head);

bool printAllRecords(Node\* head, bool toFile = false, const string& filename = "");

#endif // MODULE\_CRUD\_KONDRATENKO\_H

# Вихідний код модулю Module\_SearchDel\_Kolesnyk

## Module\_SearchDel\_Kolesnyk.cpp

#include <iostream>

#include <cctype>

#include "Module\_SearchDel\_Kolesnyk.h"

using namespace std;

string toLower(const string& str) {

string result;

for (size\_t i = 0; i < str.size(); ++i) {

unsigned char c = str[i];

if ((c & 0xF0) == 0xD0 || (c & 0xF0) == 0xD1) {

if (i + 1 < str.size()) {

unsigned char next = str[i + 1];

if (c == 0xD0 && next >= 0x80 && next <= 0x8F) {

result += c;

result += next + 0x20;

} else if (c == 0xD0 && next >= 0x90 && next <= 0xAF) {

result += c;

result += next + 0x20;

} else if (c == 0xD1 && next >= 0x80 && next <= 0x8F) {

result += c;

result += next + 0x20;

} else {

result += c;

result += next;

}

++i;

}

} else {

if (isupper(c)) {

result += tolower(c);

} else {

result += c;

}

}

}

return result;

}

void printFoundRecord(const EmployeeRecord& record, int index) {

cout << "\n--- Знайдений запис #" << index << " ---" << endl;

cout << "Табельний номер: " << record.employee\_id << endl;

cout << "ПІБ: " << record.full\_name << endl;

cout << "Ідентифікаційний номер: " << record.tax\_id << endl;

cout << "Дата заповнення: " << record.date\_filled.toString() << endl;

cout << "Стать: " << genderToString(record.gender) << endl;

cout << "Вид роботи: " << record.work\_type << endl;

cout << "Дата народження: " << record.birth\_date.toString() << endl;

cout << "Громадянство: " << record.citizenship << endl;

cout << "Освіта: " << record.education << endl;

cout << "Останнє місце роботи: " << record.last\_job << endl;

cout << "Остання посада: " << record.last\_position << endl;

cout << "Стаж роботи: " << record.exp\_years << " років, "

<< record.exp\_months << " місяців, "

<< record.exp\_days << " днів" << endl;

cout << "Місце проживання: " << record.address << endl;

cout << "Паспортні дані: " << record.passport << endl;

cout << "Додаткові відомості: " << record.notes << endl;

if (!record.dismissal\_date.isEmpty()) {

cout << "Дата звільнення: " << record.dismissal\_date.toString() << endl;

cout << "Причина звільнення: " << record.dismissal\_reason << endl;

} else {

cout << "Статус: Працює" << endl;

}

cout << "------------------------" << endl;

}

bool findRecordByName(Node\* head, const string& name) {

if (head == nullptr) {

cout << "База даних порожня." << endl;

return false;

}

string searchName = toLower(name);

Node\* current = head;

int count = 0;

int index = 1;

cout << "\n=== Результати пошуку за '" << name << "' ===" << endl;

while (current != nullptr) {

string currentName = toLower(current->record.full\_name);

if (currentName.find(searchName) != string::npos) {

printFoundRecord(current->record, index);

count++;

}

current = current->next;

index++;

}

if (count == 0) {

cout << "Записів з прізвищем '" << name << "' не знайдено." << endl;

return false;

} else {

cout << "Знайдено записів: " << count << endl;

return true;

}

}

bool deleteRecord(Node\*& head, int employee\_id) {

if (head == nullptr) {

cout << "База даних порожня." << endl;

return false;

}

if (head->record.employee\_id == employee\_id) {

Node\* temp = head;

head = head->next;

delete temp;

cout << "Запис з табельним номером " << employee\_id << " успішно видалено." << endl;

return true;

}

Node\* current = head;

while (current->next != nullptr && current->next->record.employee\_id != employee\_id) {

current = current->next;

}

if (current->next != nullptr) {

Node\* temp = current->next;

current->next = temp->next;

delete temp;

cout << "Запис з табельним номером " << employee\_id << " успішно видалено." << endl;

return true;

} else {

cout << "Запис з табельним номером " << employee\_id << " не знайдено." << endl;

return false;

}

}

## Module\_SearchDel\_Kolesnyk.h

#ifndef MODULE\_SEARCHDEL\_KOLESNYK\_H

#define MODULE\_SEARCHDEL\_KOLESNYK\_H

#include <string>

#include "struct\_type\_project\_2.h"

using namespace std;

bool findRecordByName(Node\*, const string&);

bool deleteRecord(Node\*&, int);

#endif // MODULE\_SEARCHDEL\_KOLESNYK\_H

# Аргументи досягнення мети

1. **Спільний аналіз вимог**: робота в команді змушує виробити вміння обговорювати та узгоджувати вимоги ПЗ.
2. **Проєктні мітинги**: регулярні зустрічі тренують координацію дій та навички фасилітації дискусій.
3. **Розподіл підзадач**: кожен учасник відповідає за конкретний модуль, що формує відповідальність і самостійність.
4. **Планування за ISO/IEC 12207**: складання й затвердження плану робіт знайомить зі стандартом життєвого циклу ПЗ.
5. **Версіонування в Git**: робота з репозиторієм тренує навички контролю версій та командної роботи з кодом.
6. **Координація через Git-issues**: оформлення завдань у системі трекінгу сприяє прозорості процесу.
7. **Колективні code reviews**: взаємне рецензування коду підвищує якість та обмін знаннями.
8. **Синхронізація змін**: практика merge-конфліктів у Git вчить узгоджувати різні версії.
9. **Документування рішень**: опис архітектури в заголовкових файлах формує навичку підтримуваної документації.
10. **Захист проєкту**: підготовка до захисту тренує презентаційні здібності.
11. **Вибір ADT**: обґрунтування списку, стеку чи черги навчає аналізувати вимоги до ефективності.
12. **Опис структури в** .h: робота з заголовковим файлом загартовує в умінні формалізувати інтерфейс.
13. **Абстрактні типи даних**: практична реалізація ADT закріплює теоретичні знання з методології.
14. **Інкапсуляція даних**: заховання внутрішньої представи структури підвищує надійність модулів.
15. **Модульність**: розбиття на OKP-модулі (операції над структурою) сприяє пере­використанню коду.
16. **Тестування CRUD-функцій**: окремі модулі пошуку, додавання й видалення тренують юніт-тестування.
17. **Перевірка коректності**: написання негативних тестів (наприклад, видалення неіснуючого елементу) міцно закріплює валідацію вхідних даних.
18. **Використання шаблонного середовища**: Code::Blocks навчає працювати у кросплат­форменому IDE.
19. **Розширення простору імен**: практика namespace в C++ покращує організацію коду.
20. **Шаблони за замовчуванням**: параметри функцій за замовчуванням роблять API гнучкішим.
21. new/delete: реалізація вставки та видалення вузлів тренує безпечне використання динамічної пам’яті.
22. **Запобігання memory leaks**: вивільнення списку в кінці роботи закріплює культуру роботи з ресурсами.
23. **Перевантаження операторів**: за бажанням, опрацьовує перевантаження new/delete чи операторів присвоєння.
24. **Smart pointers (за розширеним завданням)**: знайомство зі std::unique\_ptr/ shared\_ptr підвищує безпеку.
25. **Ручне управління vs. автоматичне**: порівняльний аналіз показує переваги RAII.
26. **Відлагодження пам’яті**: практика debug-інструментів (Valgrind, AddressSanitizer) покращує якість.
27. **Graceful failure**: обробка помилок new (std::bad\_alloc) тренує надійність.
28. **Перевірка null-посилань**: захист від nullptr у функціях модулів закріплює defensive programming.
29. **Структури із self‑referencing**: опис Node\* next формує розуміння рекурсивних типів.
30. **Оптимізація роботи зі стеком**: аналіз стекових викликів під час рекурсії.
31. fstream::write/read: практика бінарного вводу/виводу закріплює знання функцій fstream.
32. **Різниця text vs. binary**: аргументуйте переваги компактного та швидкого доступу до даних.
33. **Запис лічильника записів**: перший запис sizeof(size\_t) дозволяє одразу дізнатися обсяг.
34. **Відновлення структури з файлу**: реалізація loadDatabase тренує перевірку коректності даних.
35. **Обробка помилок I/O**: практика перевірки .is\_open() та перевантаженого count дає надійність.
36. **Пакування даних**: обґрунтування необхідності write(reinterpret\_cast…) пояснює низькорівневі механізми.
37. **Endian‑незалежність (вдосконалення)**: розробка перетворення для різних архітектур.
38. **Переносимість файлів**: розуміння alignment та padding у структурах.
39. **Контроль версій структури**: додавання заголовка версії файлу для майбутніх змін.
40. **Тестування save/load циклів**: перевірка round‑trip дає гарантію збереження цілісності.
41. **Класи як ADT**: перехід від struct до класів закріплює інкапсуляцію.
42. **Спеціальні функції‑члени**: написання конструкторів/деструкторів поглиблює розуміння життєвого циклу об’єктів.
43. **Перевантаження функцій**: створення кількох версій printRecord розвиває поліморфізм.
44. **Шаблони**: за опцією – оголошення шаблонного Node розвиває generic‑програмування.
45. **Композитний підхід**: поєднання Date і EmployeeRecord демонструє композицію.
46. **Інтерфейси модулів**: чіткий API функцій CRUD формує контракт між виконавцями.
47. **Наслідування (за розширеним завданням)**: можливість створити базовий Record та похідні.
48. **Абстракція**: реалізація інтерфейсу IDataStorage із методами load/save.
49. **Поліморфізм через віртуальні методи**: застосування для різних форматів файла (JSON, XML, bin).
50. **Препроцесорні директиви**: умовна компіляція (#ifdef \_WIN32) підвищує переносимість.
51. **Оформлення звіту за ДСТУ**: практика стандартизованого оформлення розвиває академічну культуру.
52. **Зміст звіту**: чітка структура (мета, варіант, завдання, результати) підвищує читабельність.
53. **Додатки (код, тестові набори)**: оформлення артефактів у звіті формує навичку повноти документації.
54. **Таблиці тест-кейсів**: включення тест-сьютів демонструє важливість системного тестування.
55. **Аргументовані висновки**: написання обґрунтованих висновків навчає аналітичному стилю.
56. **Стислість і точність**: робота з методичними рекомендаціями виробляє лаконічність викладу.
57. **Джерела й посилання**: вставка літератури (McConnell, Booch) знайомить із бібліографічними стандартами.
58. **Git-звітність**: вказівка на URL репозиторію демонструє відкритість роботи.
59. **Етапи виконання**: розбивка процесу на підпункти тренує вміння структурувати проектний звіт.
60. **Підготовка до захисту**: репетиція доповіді за матеріалами звіту покращує риторичні навички.
61. Artifact\_TEST\_SUITE\_lab.doc: заповнення шаблону навчить стандартизованому тестуванню .
62. **Позитивні тести**: перевірка правильного збереження/зчитування даних підтверджує коректність.
63. **Негативні тести**: обробка корупції файлу чи некоректного вводу затверджує robustness.
64. **Автоматизація тестів**: можливість підключити unit‑тест фреймворки (Google Test) підвищує ефективність.
65. **Регресійне тестування**: повторний прогін тестів після змін гарантує збереження функціоналу.
66. **Логування**: виведення діагностичних повідомлень при load/save формує практику трасування.
67. **Перевірка граничних умов**: тести для порожньої бази та великого обсягу даних закріплюють уважність.
68. **Моніторинг пам’яті**: перевірка через інструменти убезпечує від витоків.
69. **Код-рев’ю тестів**: взаємна перевірка тест-кейсів підвищує якість процедури.
70. **Звітування про баги**: оформлення issue в Git навчає журналістиці програмних помилок.
71. **Методологія об’єктного аналізу**: практична реалізація формує глибоке розуміння аналізу .
72. **Перевантаження макросів**: робота з директивами #define розвиває низькорівневі навички .
73. **Директиви препроцесора**: умовна компіляція й інклюди підвищують гнучкість коду .
74. **Параметри функцій за замовчуванням**: полегшують підтримку та розширення API .
75. **Підготовка до наступних курсів**: набутий досвід з динамічними структурами та файлами стане фундаментом для СУБД, мережевого програмування, тощо.

* 

