

**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:**

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 команди IT – проєкту:**

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

**Варіант №2:**

**3.1 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 оператора).

1. **Додавання нового запису** (“особова картка”).
2. **Пошук запису** за прізвищем працівника.
3. **Видалення запису** за прізвищем.
4. **Збереження бази** у файл при завершенні роботи.

**3.2 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)

**3.3 Вибір динамічної структури даних**

— **Тип:** однозв’язний список.

**— Обґрунтування:**

1. Динамічність (будь-яка кількість записів).

2. 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) {}

};

**3.5 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() |

**3.6 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 |

**Аналіз задач та вимоги до ПЗ Module\_CRUD\_Kondratenko:**

що має робити ПЗ:

Створення записів про співробітників:

* Збір особистих даних працівника (ПІБ, дата народження, адреса тощо).
* Збір інформації про трудовий стаж, освіту, громадянство.
* Визначення статі, виду роботи (основна / за сумісництвом).
* Можливість вказати дату звільнення та причину.

Збереження даних:

* Дані додаються до зв’язного списку Node.
* Нові записи зберігаються в динамічну структуру пам’яті.

Вивід даних:

* Виведення всіх записів у консоль або файл.
* Форматований друк з розміткою полів.

Валідація введення:

* Перевірка форматів дати, числових значень.
* Валідація обраних значень (наприклад, стать або вид роботи).

**Аналіз задач:**

Основні задачі:

* Додавання нового запису про працівника
* Перегляд усіх записів
* Збереження записів у файл
* Форматований вивід інформації про працівника
* Перевірка коректності введення даних

### ****Підзадачі:****

#### Задача 1: Додавання нового запису

* Введення дати заповнення
* Введення табельного номера
* Введення ІПН (ідентифікаційного номера)
* Вибір статі (ч/ж/інше)
* Вибір виду роботи (основна/за сумісництвом)
* Введення ПІБ
* Введення дати народження
* Введення громадянства
* Введення освіти
* Введення останнього місця роботи та посади
* Введення трудового стажу (дні, місяці, роки)
* Введення адреси проживання
* Введення паспортних даних
* Введення додаткових відомостей
* Якщо працівник звільнений:

Введення дати звільнення

Введення причини звільнення

* Додавання в кінець зв’язаного списку

#### Задача 2: Виведення бази даних

* Перевірка, чи є записи
* Виведення всіх записів (у консоль або у файл)
* Виведення кожного запису з форматуванням

#### Задача 3: Перевірка введення

* Контроль формату дати
* Обробка помилок введення числових значень
* Обробка пустого введення (де дозволено)
* Обмеження вибору певними значеннями (стать, вид роботи, "так/ні")

Ось два тест-сьюти для модуля Module\_CRUD\_Kondratenko, зокрема для функцій addRecord і printAllRecords. Кожен тест-сьют містить 5 тест-кейсів.





Вихідний код проєкту prj\_2\_Kondratenko:  
  
#include <iostream>

#include <limits>

#include <thread>

#include <chrono>

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

#include "Module\_CRUD\_Kondratenko.h"

#include "Module\_File\_Zaritskyi.h"

#include "Module\_SearchDel\_Kolesnyk.h"

using namespace std;

void pause() {

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

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

cin.get();

}

void showMenu() {

cout << "\n====================================\n";

cout << " Меню управління базою даних\n";

cout << "====================================\n";

cout << "1. Завантажити базу з файлу\n";

cout << "2. Додати новий запис\n";

cout << "3. Вивести всі записи\n";

cout << "4. Зберегти базу у файл\n";

cout << "5. Пошук за ПІБ\n";

cout << "6. Видалити запис за табельним номером\n";

cout << "0. Вихід\n";

cout << "====================================\n";

cout << "Ваш вибір: ";

}

int main() {

Node\* head = nullptr;

int choice;

string filename = "database.bin";

do {

showMenu();

cin >> choice;

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

switch (choice) {

case 1:

cout << "\n Завантаження бази..." << endl;

loadDatabase(head, filename);

this\_thread::sleep\_for(chrono::milliseconds(600));

pause();

break;

case 2:

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

addRecord(head);

this\_thread::sleep\_for(chrono::milliseconds(400));

pause();

break;

case 3:

cout << "\n Виведення всіх записів:\n";

printAllRecords(head);

pause();

break;

case 4:

cout << "\n Збереження бази у файл..." << endl;

saveDatabase(head, filename);

this\_thread::sleep\_for(chrono::milliseconds(500));

pause();

break;

case 5: {

string searchName;

cout << "\n Введіть ПІБ для пошуку: ";

getline(cin, searchName);

findRecordByName(head, searchName);

pause();

break;

}

case 6: {

int id;

cout << "\n Введіть табельний номер для видалення: ";

cin >> id;

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

deleteRecord(head, id);

pause();

break;

}

case 0:

cout << "\n Вихід з програми. До побачення!" << endl;

break;

default:

cout << "\n Невідомий вибір. Спробуйте ще раз." << endl;

pause();

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\_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\_File\_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\_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\_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 на користь досягнення мети ЛР11**

1. Вивчив основи роботи з динамічними структурами даних у C++.
2. Самостійно реалізував дві функції обробки динамічної структури.
3. Розібрався в структурі списку як динамічної структури даних.
4. Розробив заголовковий файл із описом структури відповідно до варіанта.
5. Забезпечив коректну ініціалізацію та звільнення динамічної пам’яті.
6. Навчився використовувати вказівники для побудови динамічних списків.
7. Створив статичну бібліотеку з власним прізвищем у назві, як вимагалось.
8. Вивчив синтаксис оголошення шаблонів структур.
9. Застосував Git для командної розробки і синхронізації модулів.
10. Навчився створювати проєкти в Code::Blocks.
11. Синхронізував власні модулі з бібліотеками команди.
12. Створив консольний додаток відповідно до стандарту ISO/IEC 12207.
13. Реалізував зручний інтерфейс користувача для тестування структури.
14. Забезпечив збереження даних у бінарному файлі.
15. Реалізував функції зчитування структури з файлу.
16. Опрацював відмінності між бінарним і текстовим файловим потоком.
17. Використав fstream для обробки файлів.
18. Перевірив коректність читання і запису у файл.
19. Розібрався в операціях над вказівниками та посиланнями.
20. Порівняв ефективність використання вказівника й посилання.
21. Вивчив способи перевірки нульових вказівників для безпеки.
22. Впровадив валідацію введення користувача.
23. Реалізував функцію додавання елементів до структури.
24. Реалізував функцію видалення елементів зі структури.
25. Забезпечив підтримку множинного пошуку за критеріями.
26. Оптимізував алгоритми обробки для великих обсягів даних.
27. Дотримався вимог до оформлення заголовкових файлів.
28. Використовував інкапсуляцію для обмеження доступу до полів структури.
29. Навчився структурувати код за принципами модульності.
30. Дотримався вимог до проєктування відповідно до ISO/IEC 12207.
31. Вивчив теоретичну базу щодо об'єктного програмування.
32. Навчився використовувати препроцесорні директиви.
33. Працював із макросами для спрощення коду.
34. Використав класи для логічного поділу функцій і структур.
35. Навчився оголошувати й використовувати спеціальні функції-члени.
36. Розробив архітектуру модулів у команді.
37. Сформував та узгодив структуру даних з іншими учасниками.
38. Брав участь у мітингах для обговорення технічних рішень.
39. Розподілив підзадачі в команді за компетенціями.
40. Реалізував свою частину проєкту в узгоджений строк.
41. Протестував функції свого модуля на коректність.
42. Створив unit-тести для перевірки окремих функцій.
43. Навчився використовувати assert для автоматизованого тестування.
44. Оформлював звіт згідно з вимогами ДСТУ 3008:2015.
45. Включив усі необхідні розділи до звіту.
46. Усі артефакти проєктування й тестування додано до звіту.
47. Забезпечив зрозуміле і повне коментування коду.
48. Вивчив документацію до fstream для роботи з файлами.
49. Ознайомився з особливостями консольного введення/виведення.
50. Реалізував меню управління програмою.
51. Використовував цикл while для повторного виконання операцій.
52. Передбачив обробку некоректного введення.
53. Продумав логіку завершення програми й очищення пам’яті.
54. Підключив бібліотеки команди без конфліктів.
55. Не допустив витоків пам’яті при видаленні структури.
56. Ретельно перевірив програму на помилки компіляції.
57. Проаналізував результати тестування з командою.
58. Вивчив особливості зберігання структур у бінарному форматі.
59. Реалізував сортування елементів структури за полем.
60. Створив функцію пошуку за кількома параметрами.
61. Вивчив принципи створення бібліотек у C++.
62. Визначив основні переваги командної розробки.
63. Навчився використовувати простори імен (namespace).
64. Вивчив специфіку доступу до членів структури.
65. Використав структуру з вкладеними полями.
66. Використав шаблонну функцію для універсальності.
67. Передбачив функцію експорту даних у текстовий формат.
68. Реалізував логування дій користувача.
69. Створив зрозумілу документацію до функцій.
70. Узгодив назви функцій і структур з командою.
71. Навчився працювати зі стороннім кодом (команди).
72. Поглибив знання в об'єктно-орієнтованому програмуванні.
73. Отримав досвід структурованої командної взаємодії.
74. Дотримався усіх вимог методичних вказівок.
75. У результаті виконання ЛР11 здобув практичні навички, що відповідають сучасним вимогам до розробників ПЗ.