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
Булюкін В. Ю.
Перевірив: викладач
Коваленко А.С.

Кропивницький 2025

Варіант - 4

Мета роботи - полягає у набутті ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів C++ 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 ІТ-проєкта модулів.

СКЛАД КОМАНДИ ІТ-ПРОЄКТА

Група: КН-24

1. Куріщенко Павло;

Підзадачі:

1) Виведення всієї бази на екран або у текстовий файл (на вибір користувача);

2) Пошук запису за введеним диспетчером прізвищем студента.

2. Булюкін Володимир;

Підзадачі:

1) Завантаження бази з текстового файлу;

2) Завершення роботи програми з автоматичним записом бази у файл.

3. Радомська Діана.

Підзадачі:

1) Додавання нового запису в базу

2) Видалення заданого оператором запису з бази

Аналіз задач ІТ-проєкта та вимог до ПЗ:

Функціональні вимоги:

1. **Виведення всієї бази** (Виводити на екран або зберігати у текстовий файл).
2. **Додавання записів** (Інтерактивне введення нових студентів у базу).
3. **Пошук:** (Пошук записів за прізвищем).
4. **Видалення** (Видалення обраного запису оператором).
5. **Автоматичне збереження** (Збереження бази у файл при завершенні роботи).
6. **Автоматичне завантаження** (Читання бази з файлу при старті програми).

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:

- string — для зберігання текстових даних (ПІБ, громадянство, адреса тощо);

- Date — уніфікований тип для дат (дата народження, дата заповнення, звільнення).

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

Опис модуля(Modules_Buliukin_LoadAndSaveDb):

Завантаження та збереження бази даних:

Модуль містить функції для завантаження бази даних студентів з текстового файлу при запуску програми та автоматичного збереження при завершенні роботи.

Основні функції:

1) loadDatabaseFromFile()

Призначення:

Завантажує базу даних студентів з текстового файлу у динамічну структуру даних (однозв'язний список).

Основні кроки виконання:

1. Відкриває файл для читання у текстовому режимі.
2. Перевіряє успішність відкриття файлу.
3. Читає дані студентів рядок за рядком у циклі до кінця файлу:
    - Зчитує ПІБ студента
    - Зчитує дату народження
    - Зчитує місце народження
    - Зчитує громадянство та освітню інформацію

- Зчитує адресу та додаткові дані

4. Для кожного студента створює новий вузол списку.
5. Додає вузол до кінця однозв'язного списку.
6. Закриває файл після завершення читання.

Особливості реалізації:

- Використовується функція getline() для зчитування рядків з пробілами.
- Після зчитування числових значень викликається ignore() для пропуску символу нового рядка.
- Перевіряється порожність першого рядка для визначення кінця файлу.
- Новий вузол додається до кінця списку, зберігаючи порядок записів.

Лістинг функції:

```
void loadDatabaseFromFile(const string& filename) {
    ifstream in(filename);
    if (!in.is_open()) {
        cerr << "Не вдалося відкрити файл: " << filename << '\n';
        return;
    }

    while (!in.eof()) {
        Student s;

        getline(in, s.name.lastName);
        if (s.name.lastName.empty()) break; // Якщо порожній рядок — кінець

        getline(in, s.name.firstName);
        getline(in, s.name.middleName);
        in >> s.birthDate.day >> s.birthDate.month >> s.birthDate.year;
        in.ignore();
        getline(in, s.birthPlace.postalCode);
        getline(in, s.birthPlace.region);
        getline(in, s.birthPlace.district);
        getline(in, s.birthPlace.locality);
        getline(in, s.citizenship);
        getline(in, s.graduatedFrom);
        in >> s.graduationYear;
        in.ignore();
        getline(in, s.familyStatus);
        getline(in, s.address.postalCode);
        getline(in, s.address.region);
        getline(in, s.address.district);
        getline(in, s.address.locality);
```

```
            getline(in, s.education.institutionName);
            getline(in, s.education.department.institute);
            getline(in, s.education.department.faculty);
            getline(in, s.education.department.department);
            getline(in, s.education.educationLevel);
            getline(in, s.education.trainingDirection.code);
            getline(in, s.education.trainingDirection.name);
            getline(in, s.education.specialty.code);
            getline(in, s.education.specialty.name);
            getline(in, s.education.specialization.code);
            getline(in, s.education.specialization.name);

            Node* newNode = new Node(s);
            if (!head) head = newNode;
            else {
                Node* temp = head;
                while (temp->next) temp = temp->next;
                temp->next = newNode;
            }
        }

        in.close();
}
```

2) saveDatabaseToFile()

Призначення:

Зберігає всю базу даних студентів з пам'яті у текстовий файл.

Основні кроки виконання:

7. Відкриває файл для запису у текстовому режимі.
8. Перевіряє успішність відкриття файлу.
9. Проходить по всьому однозв'язному списку head.
10. Для кожного студента записує всі поля у файл:
    a. ПІБ (кожне поле в окремому рядку)
    b. Дату народження (день, місяць, рік через пробіл)
    c. Всі текстові поля (по одному в рядку)
11. Закриває файл після завершення запису.

Особливості реалізації:

- Використовується той же формат, що і при зчитуванні для забезпечення сумісності.
- Числові значення дати записуються через пробіл в одному рядку.

- Всі текстові поля записуються у окремих рядках.
- Файл перезаписується повністю при кожному збереженні.

Лістинг функції:

```
void saveDatabaseToFile(const string& filename) {
    ofstream out(filename);
    if (!out.is_open()) {
        cerr << "Не вдалося зберегти базу даних у файл: " << filename
<< '\n';
        return;
    }

    Node* current = head;
    while (current) {
        const Student& s = current->data;
        out << s.name.lastName << '\n'
            << s.name.firstName << '\n'
            << s.name.middleName << '\n'
            << s.birthDate.day << ' ' << s.birthDate.month << ' ' <<
s.birthDate.year << '\n'
            << s.birthPlace.postalCode << '\n'
            << s.birthPlace.region << '\n'
            << s.birthPlace.district << '\n'
            << s.birthPlace.locality << '\n'
            << s.citizenship << '\n'
            << s.graduatedFrom << '\n'
            << s.graduationYear << '\n'
            << s.familyStatus << '\n'
            << s.address.postalCode << '\n'
            << s.address.region << '\n'
            << s.address.district << '\n'
            << s.address.locality << '\n'
            << s.education.institutionName << '\n'
            << s.education.department.institute << '\n'
            << s.education.department.faculty << '\n'
            << s.education.department.department << '\n'
            << s.education.educationLevel << '\n'
            << s.education.trainingDirection.code << '\n'
            << s.education.trainingDirection.name << '\n'
            << s.education.specialty.code << '\n'
            << s.education.specialty.name << '\n'
            << s.education.specialization.code << '\n'
            << s.education.specialization.name << '\n';
        current = current->next;
    }

    out.close();
}
```

3) **saveDatabaseToFile()**

Призначення:

Забезпечує коректне завершення програми з автоматичним збереженням бази даних.

Основні кроки виконання:

1. Викликає функцію saveDatabaseToFile() для збереження поточної бази.

2. Виводить повідомлення про успішне збереження даних.
3. Повертає управління для завершення програми.

Особливості реалізації:

- Гарантує, що дані не будуть втрачені при закритті програми.
- Використовує стандартне ім'я файлу "students_database.txt".
- Надає зворотний зв'язок користувачу про статус збереження.

Лістинг функції:

```
void exitProgram() {
    saveDatabaseToFile("students_database.txt");
    cout << "Програма завершена. Дані збережено у файл
'students_database.txt'.\n";
}
```

Лістинг .h файлу модуля:

```
#ifndef MODULES_BULIUKIN_LOADANDSAVEDB_H
#define MODULES_BULIUKIN_LOADANDSAVEDB_H

#include <string>
#include <fstream>
#include <iostream>
#include <limits>
#include "../struct_type_project_4.h"

using namespace std;

void loadDatabaseFromFile(const std::string& filename);
void exitProgram();

#endif // MODULES_BULIUKIN_LOADANDSAVEDB_H
```

Лістинг main.cpp файлу:

```
#include "interface.h"


int main() {
    system("chcp 65001 > nul");
    handleUserChoice(); // Основна функція для вибору операцій
    return 0;
}
```

Висновок

У процесі виконання лабораторної роботи №11 я здобув важливий практичний досвід у командній розробці програмного забезпечення з

використанням динамічних структур даних та файлових операцій у середовищі C++. Моя частина проєкту була критично важливою для функціонування всієї системи, оскільки забезпечувала збереження та відновлення даних. Нижче наведено перелік знань і навичок, яких я набув:

1. Командна робота в складі IT-проєкту
2. Участь у технічних мітингах та плануванні
3. Координація роботи з іншими членами команди
4. Розподіл відповідальності та дотримання термінів
5. Робота з текстовими файлами у C++
6. Використання ifstream та ofstream
7. Розуміння різниці між текстовими та бінарними потоками
8. Обробка помилок при роботі з файлами
9. Правильне закриття файлових потоків
10. Використання функцій getline() та ignore()
11. Поглиблене розуміння однозв'язних списків
12. Створення та додавання вузлів до списку
13. Керування динамічною пам'яттю
14. Алгоритми обходу списків
15. Робота з вказівниками та посиланнями
16. Серіалізація складних структур у текстовий формат
17. Створення сумісних форматів читання/запису
18. Збереження цілісності даних при перетвореннях
19. Відновлення структур з файлу
20. Збереження ієрархічних структур даних
21. Перевірка успішності відкриття файлів
22. Обробка помилок введення/виведення
23. Валідація даних при читанні
24. Механізми відновлення після помилок
25. Створення зрозумілих повідомлень про помилки
26. Розробка модулів з чітким інтерфейсом
27. Розділення функціональності між файлами
28. Принципи інкапсуляції та приховування деталей

29. Створення повторно використовуваних функцій
30. Документування власного модуля
31. Робота з Qt Creator та .pro файлами
32. Налаштування статичних бібліотек
33. Компіляція та лінкування модулів
34. Налаштування залежностей між модулями
35. Тестування функцій на різних даних
36. Робота з порожніми файлами
37. Відлагодження проблем форматування
38. Інтеграційне тестування з модулями колег
39. Дотримання стандартів C++17
40. Правильне іменування функцій та змінних
41. Додавання коментарів та документації
42. Структурування коду для читабельності
43. Використання Git для командної роботи
44. Створення та злиття гілок
45. Координація змін з розробниками
46. Вирішення конфліктів при злитті коду
47. Аналіз ефективності алгоритмів
48. Оптимізація швидкості завантаження даних
49. Мінімізація використання пам'яті
50. Забезпечення стабільної роботи з файлами різного розміру