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
Радомська Д. М
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 ІТпроєкта 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

Загальний опис

Функція:

1. Збирає дані про студента з консолі (з перевіркою правильності введення через getValidatedInput()).
2. Створює новий об'єкт Student.
3. Обгортає його в новий вузол Node.
4. Додає вузол до кінця зв'язаного списку.
5. Виводить повідомлення про успішне додавання.

Лістинг функції:

```
// Функція для додавання запису
void addNewRecord() {
    Student newStudent;


    cout << "\n• Введення персональних даних студента •\n";
    getValidatedInput(newStudent.name.lastName,   "  Введіть
прізвище: ");
```

```
    getValidatedInput(newStudent.name.firstName,  "  Введіть ім'я:
");
    getValidatedInput(newStudent.name.middleName, "  Введіть по
батькові: ");


    cout << "\n• Введення дати народження •\n";
    getValidatedInput(newStudent.birthDate.day,   "  День: ");
    getValidatedInput(newStudent.birthDate.month, "  Місяць: ");
    getValidatedInput(newStudent.birthDate.year,  "  Рік: ");


    cout << "\n• Введення місця народження •\n";
    getValidatedInput(newStudent.birthPlace.postalCode, "  Поштовий
індекс: ");
    getValidatedInput(newStudent.birthPlace.region,     "  Область:
");
    getValidatedInput(newStudent.birthPlace.district,   "  Район: ");
    getValidatedInput(newStudent.birthPlace.locality,   "  Населений
пункт: ");


    cout << "\n• Інші персональні дані •\n";
    getValidatedInput(newStudent.citizenship,    "  Громадянство: ");
    getValidatedInput(newStudent.graduatedFrom,  "  Навчальний
заклад, що закінчив: ");
    getValidatedInput(newStudent.graduationYear, "  Рік закінчення:
");
    getValidatedInput(newStudent.familyStatus,   "  Сімейний стан:
");


    cout << "\n• Введення адреси проживання •\n";
    getValidatedInput(newStudent.address.postalCode, "  Поштовий
індекс: ");
    getValidatedInput(newStudent.address.region,     "  Область: ");
    getValidatedInput(newStudent.address.district,   "  Район: ");
    getValidatedInput(newStudent.address.locality,   "  Населений
пункт: ");


    cout << "\n• Введення інформації про освіту у ЗВО •\n";
    getValidatedInput(newStudent.education.institutionName,       "
Назва закладу: ");
    getValidatedInput(newStudent.education.department.institute,  "
Інститут: ");
```

```
    getValidatedInput(newStudent.education.department.faculty,    "
Факультет: ");

    getValidatedInput(newStudent.education.department.department,
"  Відділення: ");

    getValidatedInput(newStudent.education.educationLevel,        "
Рівень освіти: ");


    cout << "\n• Напрям підготовки •\n";

    getValidatedInput(newStudent.education.trainingDirection.code,
"  Шифр напряму: ");

    getValidatedInput(newStudent.education.trainingDirection.name,
"  Назва напряму: ");


    cout << "\n• Спеціальність •\n";

    getValidatedInput(newStudent.education.specialty.code, "  Шифр
спеціальності: ");

    getValidatedInput(newStudent.education.specialty.name, "  Назва
спеціальності: ");


    cout << "\n• Спеціалізація •\n";

    getValidatedInput(newStudent.education.specialization.code,
"  Шифр спеціалізації: ");

    getValidatedInput(newStudent.education.specialization.name,
"  Назва спеціалізації: ");


    // Додавання нового вузла

    Node* newNode = new Node(newStudent);

    if (head == nullptr) head = newNode;

    else {

        Node* temp = head;

        while (temp->next != nullptr) temp = temp->next;

        temp->next = newNode;

    }


    cout << "\n+-----------------------------------------------------
-------+\n";

    cout <<   "|                    НОВИЙ ЗАПИС УСПІШНО
ДОДАНО               |\n";

    cout <<   "+-----------------------------------------------------
-------+\n";

    cout <<   " Студента з прізвищем " << newStudent.name.lastName <<
" успішно додано в систему!\n";
```

```
}
```

Принцип роботи функції addNewRecord()

Функція addNewRecord() відповідає за додавання нового студента до кінця списку (ймовірно, однонаправленого зв'язаного списку).

2) removeRecordByFullName()

Принцип роботи

1. Введення ПІБ студента для пошуку.
2. Перевірка першого елемента списку:
   - Якщо він відповідає ПІБ — видаляється одразу.
3. Пошук у середині списку:
   - Якщо знайдено вузол із відповідним ПІБ — видаляється.
4. Якщо запис не знайдено, виводиться повідомлення про це.

Особливості

- Працює з однонаправленим зв’язаним списком.
- Очищає пам’ять (delete) після видалення.
- Виводить повідомлення про результат.

Лістинг функції:

```
/ Функція для видалення запису
void removeRecordByFullName() {
    string lastName, firstName, middleName;

    cout << "\n=== Вилучення запису студента ===\n";
    getValidatedInput(lastName, "Введіть прізвище: ");
    getValidatedInput(firstName, "Введіть ім'я: ");
    getValidatedInput(middleName, "Введіть по батькові: ");

    // Перевірка, чи перший елемент — це той, кого потрібно видалити
```

```
    while (head != nullptr &&
           head->data.name.lastName == lastName &&
           head->data.name.firstName == firstName &&
           head->data.name.middleName == middleName) {

        Node* temp = head;
        head = head->next;
        delete temp;

        cout << "\n+-----------------------------------------------
---+\n";
        cout <<   "|                ЗАПИС УСПІШНО
ВИДАЛЕНО              |\n";
        cout <<   "+-----------------------------------------------
---+\n";
        cout << " " << lastName << " " << firstName << " " <<
middleName << " був(ла) видалений(а).\n";
        return;
    }

    // Проходимо по списку, шукаючи вузол, що йде ПІСЛЯ потрібного
    Node* current = head;
    while (current != nullptr && current->next != nullptr) {
        Student& s = current->next->data;

        if (s.name.lastName == lastName &&
            s.name.firstName == firstName &&
            s.name.middleName == middleName) {

            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;

            cout << "\n+-----------------------------------------------
-------+\n";
            cout <<   "|                ЗАПИС УСПІШНО
ВИДАЛЕНО              |\n";
            cout <<   "+-----------------------------------------------
-------+\n";
            cout << " " << lastName << " " << firstName << " " <<
middleName << " був(ла) видалений(а).\n";
            return;
        }

        current = current->next;
    }

    cerr << "\nСтудента з таким ПІБ не знайдено.\n";
```

```
}
```

Лістинг .h файлу визначення структур:

```
#ifndef STRUCT_TYPE_PROJECT_4_H #define STRUCT_TYPE_PROJECT_4_H
#include  using
namespace std; struct
DepartmentInfo {
string institute;
string faculty;
string department;
};
struct CodeName {
string code;
string name;
};
struct EducationInfo {  string
institutionName;
DepartmentInfo department;
string educationLevel;
CodeName trainingDirection;
CodeName specialty;
CodeName specialization;
};
struct FullName {
string lastName;  string
firstName;  string
middleName;
}; struct Date {
int day;  int
month;  int
year;  };
struct Address
{  string
postalCode;
string region;
string
```

```
district;
string
locality;
};
struct Student {  FullName
name;
Date birthDate;  Address
birthPlace;  string
citizenship;  string
graduatedFrom;  string
graduationYear;  string
familyStatus;  Address
address;
EducationInfo education;
}; struct
Node {
Student
data;
Node* next;
Node(const Student& studentData) : data(studentData), next(nullptr)
{}
};
#endif // STRUCT_TYPE_PROJECT_4_H
```

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.