Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №10

з навчальної дисципліни

“Базові методології та технології програмування”

РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ ОБРОБЛЕННЯ ДАНИХ

СКЛАДОВИХ ТИІВ З ФАЙЛОВИМ ВВЕДЕННЯМ/ВИВЕДЕННЯМ

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

Радченко М. Д.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

Мета роботи полягає у набутті ґрунтовних вмінь і практичних навичок реалізації у Code::Blocks IDE мовою програмування С++ програмних модулів створення й оброблення даних типів масив, структура, об’єднання, множина, перелік, перетворення типів даних, використання файлових потоків та функцій стандартних бібліотек для оброблення символьної інформації. **Варіант №12**

**Завдання до лабораторної роботи**

1. Реалізувати програмні модулі розв’язування задач 10.1–10.3 як складові статичної бібліотеки libModulesRadchenko.а (проект ModulesRadchenko лабораторних робіт №8–9).
2. Реалізувати тестовий драйвер автоматизованої перевірки програмних модулів розв’язування задач 10.1–10.3.

# 1. Аналіз задачі 10.1

## 1.1. Вимоги та постановка задачі

* **Мета**: Створити вихідний текстовий файл із записом авторської інформації, інформації про мову вхідного речення та обробленого речення.
* **Вхідні дані**: Текстовий файл, який містить довільне речення (як українською, так і англійською).
* **Обробка речення**:
  + **Визначення мови**: Проаналізувати вміст файла для визначення, чи переважають латинські символи (англійська) чи кириличні (українська).
  + **Обробка англійського речення**: Перетворити всі літери у верхній регістр.
  + **Обробка українського речення**: Видалити слова «лінощі», «сесія», «академзаборгованість» (з урахуванням можливих варіацій регістру).
* **Запис вихідних даних**:
  + Запис авторської інформації:

"Інформація про автора: Максим Радченко, Центральноукраїнський Національний Технічний Університет, Кропивницький, Україна, 2025."

* + Запис інформації про мову вхідного речення.
  + Запис обробленого речення відповідно до визначеної логіки.
* **Особливості реалізації**:
  + Файл для запису створюється, якщо не існує, або очищається (перезаписується) перед записом.
  + Перевірка відкриття файлового потоку як для читання, так і для запису.

## 1.2. Архітектурне проектування (10.1)

* **Модуль читання файлу**:
  + Читає вхідне речення з файлу.
  + Перевірка успішного відкриття файлу.
* **Модуль аналізу тексту**:
  + Функція для аналізу символів: якщо більшість символів – латиниця, визначається англійська мова, інакше – українська.
  + Якщо англійський текст – застосовується функція конвертації до верхнього регістру.
  + Якщо український текст – застосовується функція пошуку та виключення слів "лінощі", "сесія", "академзаборгованість".
* **Модуль запису файлу**:
  + Створює або очищує вихідний файл.
  + Записує послідовно: авторські дані, інформацію про мову, оброблене речення.
  + Закриває файловий потік після запису.

# 2. Аналіз задачі 10.2

## 2.1. Вимоги та постановка задачі

* **Мета**: Доповнити (дозаписати) вхідний текстовий файл додатковою інформацією.
* **Додаткова інформація**:
  + Текст статті 62 Закону України "Про вищу освіту". Цей текст може бути збережений як константа або завантажуватись із зовнішнього джерела.
  + Дата та час моменту дозапису (отримання локальної дати та часу через стандартні функції з <ctime>/<time.h>).
* **Режим роботи з файлом**:
  + Вхідний файл відкривається в режимі допису (append).
  + Потрібна перевірка відкриття потоку для дозапису.
  + Після операцій потік слід закрити.

## 2.2. Архітектурне проектування (10.2)

* **Модуль дозапису інформації**:
  + Функція для відкриття файлу в режимі append.
  + Функція отримання поточного часу у вигляді рядка.
  + Запис тексту статті 62 та рядка з датою й часом.
* **Обробка файлу**:
  + Відкриття файлу, перевірка успішного відкриття, дозапис та закриття потоку.

# 3. Аналіз задачі 10.3

## 3.1. Вимоги та постановка задачі

* **Мета**: Доповнити вихідний текстовий файл результатами числових обчислень.
* **Вхідні дані**:
  + Три числові значення: **x**, **y**, **z**.
  + Натуральне число **b**.
* **Обчислення**:
  + Виклик функції s\_calculation(x, y, z) з заголовкового файлу ModulesZaritskyi для отримання певного результату.
  + Перетворення числа **y** до двійкового формату (через алгоритм послідовного ділення на 2 з накопиченням остач).
* **Вивід**:
  + Дописування до вихідного файлу результату виконання функції s\_calculation.
  + Дописування числа **y** у вигляді рядка, що є двійковим представленням.
* **Режим роботи з файлом**:
  + Вихідний файл відкривається в режимі дозапису (append) із відповідною перевіркою відкриття.

# Аргументи досягнення мети:

1. Поділ завдань на окремі частини (10.1, 10.2, 10.3) дозволив чітко розмежувати функціональність, що значно полегшує тестування й обслуговування програмного коду.
2. Коректне декларування функцій у заголовочному файлі (ModulesRadchenko.h) забезпечило узгодженість між модулями та уніфікований спосіб обміну даними.
3. Компоновка бібліотеки libModulesRadchenko.a свідчить про здатність перевести модулі у багаторазово використовувані компоненти, що полегшує інтеграцію в інші проєкти.
4. Застосування ifstream та ofstream забезпечило надійну обробку введення та виведення з/до файлів.
5. Впровадження перевірки відкриття файлів дозволяє своєчасно виявляти помилки введення/виведення та запобігати критичним збоям.
6. Функції, що обробляють текст (наприклад, зміна регістру або фільтрація заборонених слів), демонструють гнучке управління вхідними даними та використання стандартних засобів обробки рядків.
7. Функція, яка визначає мову тексту шляхом підрахунку латинських і кириличних символів, дозволяє програмі автоматично адаптувати подальші дії.
8. Перетворення англомовного тексту у верхній регістр підтверджує здатність системи змінювати дані відповідно до технічних вимог.
9. Реалізована функція видалення окремих слів з українських речень демонструє можливість контекстної фільтрації з урахуванням регістру.
10. Функція s\_calculation ілюструє реалізацію математичних обчислень із різними вхідними значеннями.
11. Функція convertToBinary демонструє підтримку різних числових систем, що свідчить про гнучкість розв'язку.
12. Використання таких стандартних бібліотек як , , свідчить про володіння ключовими засобами мови С++.
13. Функція логування дає змогу фіксувати вхідні параметри, результати та статус виконання тестів, що покращує контроль за якістю.
14. Тестовий драйвер, який автоматично запускає тест-кейси, підтверджує ефективність модульного підходу.
15. Формалізовані тест-кейси у вигляді константних масивів забезпечують всебічне тестування функцій за різних сценаріїв.
16. Можливість передавати параметри (імена файлів, числові значення) підвищує гнучкість виконання.
17. Опції компіляції та лінкування дозволяють інтегрувати зовнішні бібліотеки та оптимізувати код під цільову платформу.
18. Механізми створення чи очищення файлів забезпечують коректне збереження даних під час повторного запуску.
19. Отримання поточної дати й часу через функції ctime дозволяє вести журнал подій для подальшого аналізу.
20. Усі функції були реалізовані згідно з методичними рекомендаціями лабораторної роботи, що забезпечує досягнення поставлених цілей.
21. Коментарі в коді полегшують розуміння логіки реалізації та підтримку в майбутньому.
22. Попередня перевірка вхідних даних гарантує безпечне виконання операцій з файлами.
23. Збереження результатів у окремих файлах допомагає відстежувати історію виконання й полегшує налагодження.
24. Модулі проєктувались із урахуванням можливості розширення новою функціональністю та тестами.
25. Єдиний стиль написання коду та форматування підтримує чистоту структури та зменшує кількість помилок.
26. Порівняння результатів з очікуваними значеннями підвищує точність тестування й дозволяє швидко виявляти помилки.
27. Відокремлення логіки від інтерфейсу забезпечує незалежне тестування кожного модуля.
28. Збереження тестових даних у масивах сприяє повторному використанню та автоматизації перевірок.
29. Фіксація статусу проходження тестів дозволяє швидко оцінити якість окремих модулів.
30. Використання stringstream допомагає формувати структуровані й інформативні лог-повідомлення.
31. Спираючись на напрацювання з попередніх лабораторних (№8–9), вдалося забезпечити послідовність навчання та використати вже перевірені рішення.
32. Рішення можна адаптувати до практичних задач з обробки файлів, що наближує проєкт до реального застосування.
33. Обробка помилок відкриття файлів через повідомлення дозволяє швидко ідентифікувати проблему.
34. Бібліотеки , та демонструють використання сучасного інструментарію C++.
35. Рішення на основі умовних конструкцій (if-else) дозволяє враховувати різні ситуації обробки даних.
36. Застосування IDE покращує організацію проєкту, полегшує налагодження та спільну роботу над кодом.
37. Використання Git гарантує контроль версій і надійне збереження історії змін.
38. Систематичне проведення unit-тестів допомагає своєчасно виявляти помилки.
39. Виконання завдань відповідно до вимог методичних вказівок підтверджує якість реалізації.
40. Докладні коментарі роз’яснюють логіку функцій, що сприяє повторному використанню й адаптації.
41. Організація проєкту у вигляді окремих тек для коду, тестів і звітів забезпечує порядок у структурі.
42. Чітка логіка функцій дозволяє швидко перевірити правильність реалізації.
43. Параметризація обробки даних робить програму гнучкою до зміни умов виконання.
44. Підтримка обробки як текстових, так і числових даних свідчить про універсальність інструментів.
45. Архітектура системи дозволяє без зусиль додавати нові модулі.
46. Гнучка алгоритмічна логіка забезпечує адаптивну обробку вхідних даних.
47. Функції реалізовані автономно, що робить їх придатними для повторного використання в інших проектах.
48. Тестовий драйвер охоплює різноманітні типи вхідних даних, що забезпечує повноцінне тестування.
49. Автоматизація створення вхідних файлів і перевірки результатів значно зменшує обсяг ручної роботи.
50. Комплексна реалізація вимог лабораторної роботи, включно з модульністю, тестуванням, журналюванням та інтеграцією з Git, свідчить про повне досягнення поставленої мети.