Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 7

з навчальної дисципліни

“Базові методології та технології програмування”

ПРОГРАМНА РЕАЛІЗАЦІЯ ОБРОБЛЕННЯ МАСИВІВ ДАНИХ ТА СИМВОЛЬНОЇ ІНФОРМАЦІЇ

ЗА СТАНДАРТОМ UNICODE

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

Марченко К. О.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

Мета роботи полягає у набутті ґрунтовних вмінь і практичних навичок синтезу алгоритмів оброблення масивів даних та символьної (текстової) інформації у кодуваннях UTF-8 і CP866, їх програмної реалізації мовою програмування мовою програмування С (ISO/IEC 9899:2018) задля реалізації програмних засобів у вільному кросплатформовому Code::Blocks IDE.

1. Аналіз задачі 7.1

**Умова:**  
Користувач вводить речення (українською або англійською), яке має закінчуватися символом кінця речення: **"."**, **"!"** або **"?"**.

**Завдання:**  
Перевірити, чи містить введене речення слово **"програма"** незалежно від регістру (тобто пошук повинен бути нечутливим до регістру символів).

**Основні кроки аналізу:**

* Переконатись, що рядок завершується одним із допустимих символів.
* Перетворити введене речення до одного регістру (наприклад, до нижнього) для полегшення пошуку.
* Шукати слово «програма» із врахуванням розділових знаків та пробілів (щоб не помилково знаходити частини інших слів).
* Вивести повідомлення про наявність чи відсутність цього слова.

Алгоритм у вигляді псевдокоду до задачі 7.1:

Початок

Вивести анотацію застосунку та інформацію про розробника (з використанням псевдографіки)

Зчитати введене речення від користувача

Якщо останній символ речення не є '.', '!' або '?'

Вивести повідомлення про помилку формату

Завершити програму

Перетворити речення до нижнього регістру

Якщо слово "програма" зустрічається як окреме слово в реченні

Вивести повідомлення: "Речення містить слово 'програма'"

Інакше

Вивести повідомлення: "Речення не містить слово 'програма'"

Кінець

Лістинг проекту Marchenko-task\_7\_1:

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <wchar.h>

#include <locale.h>

#include <string.h>

// Функція виведення анотації та інформації про розробника

void show\_about() {

printf("╔════════════════════════════════════════╗\n");

printf("║ Лабораторна робота 7.1 ║\n");

printf("║ Обробка символьної інформації (UTF-8) ║\n");

printf("╠════════════════════════════════════════╣\n");

printf("║ Розробник: Марченко Кирило ║\n");

printf("║ Група: КБ-24 ║\n");

printf("║ Навчальний заклад: ЦНТУ ║\n");

printf("╚════════════════════════════════════════╝\n\n");

}

// Функція для перетворення одного символу в нижній регістр з урахуванням української мови

wchar\_t ua\_tolower(wchar\_t c) {

// Перетворення українських літер верхнього регістру в нижній

if (c >= L'А' && c <= L'Я') {

return c + (L'а' - L'А');

}

// Специфічні українські літери

if (c == L'Ї') return L'ї';

if (c == L'І') return L'і';

if (c == L'Є') return L'є';

if (c == L'Ґ') return L'ґ';

return towlower(c); // Для інших символів використовуємо стандартну функцію

}

int main() {

// Налаштування локалі та консолі для роботи з UTF-8

setlocale(LC\_ALL, "uk\_UA.UTF-8");

SetConsoleCP(65001);

SetConsoleOutputCP(65001);

// Виведення анотації

show\_about();

char input[1024];

// Зчитування речення від користувача

printf("Введіть речення (яке завершується на '.', '!' або '?'):\n");

if (fgets(input, sizeof(input), stdin) == NULL) {

fprintf(stderr, "Помилка зчитування введення.\n");

return EXIT\_FAILURE;

}

// Видалення символу нового рядка, якщо він є

size\_t len = strlen(input);

if (len > 0 && input[len - 1] == '\n') {

input[len - 1] = '\0';

len--;

}

// Перевірка коректності закінчення речення

if (len == 0 || (input[len - 1] != '.' && input[len - 1] != '!' && input[len - 1] != '?')) {

printf("Помилка: речення повинно завершуватися на '.', '!' або '?'.\n");

return EXIT\_FAILURE;

}

// Перетворення UTF-8 рядка у широкий рядок

int wlen = MultiByteToWideChar(CP\_UTF8, 0, input, -1, NULL, 0);

if (wlen <= 0 || wlen > 1024) {

fprintf(stderr, "Помилка перетворення кодування.\n");

return EXIT\_FAILURE;

}

wchar\_t winput[1024];

MultiByteToWideChar(CP\_UTF8, 0, input, -1, winput, wlen);

// Створення копії введеного тексту в нижньому регістрі

wchar\_t winput\_lower[1024];

wcscpy(winput\_lower, winput);

// Приведення широкого рядка до нижнього регістру для нечутливості до регістру

// з використанням власної функції

for (wchar\_t \*p = winput\_lower; \*p != L'\0'; p++) {

\*p = ua\_tolower(\*p);

}

// Також приведемо пошукове слово до нижнього регістру

wchar\_t searchWord[] = L"програма";

// Пошук слова "програма" у широкому рядку (приведеному до нижнього регістру)

wchar\_t \*found = wcsstr(winput\_lower, searchWord);

if (found != NULL) {

printf("Речення містить слово \"програма\".\n");

} else {

printf("Речення не містить слово \"програма\".\n");

}

return EXIT\_SUCCESS;

}

Аналіз умови задачі 7.2

**Умова:**  
На вхід подається 14 натуральних чисел.  
**Завдання:**  
Обчислити та вивести кількість чисел, що рівні одному з трьох заданих значень: **150**, **1000** або **10000**.

**Основні кроки аналізу:**

* **Перевірка кількості чисел:** Переконатись, що користувач вводить рівно 14 натуральних чисел.
* **Обхід масиву чисел:** Переглянути кожне число.
* **Порівняння:** Якщо число дорівнює 150, 1000 або 10000, збільшити лічильник.
* **Вивід результату:** Вивести кількість чисел, що відповідають заданим умовам.

Алгоритм у вигляді псевдокоду до задачі 7.2:

Початок

Вивести анотацію застосунку та інформацію про розробника (з використанням псевдографіки)

Зчитати 14 натуральних чисел (масив)

Якщо кількість введених чисел не дорівнює 14

Вивести повідомлення про помилку

Завершити програму

Ініціалізувати лічильник = 0

Для кожного числа в масиві:

Якщо число дорівнює 150 або 1000 або 10000

Збільшити лічильник на 1

Вивести значення лічильника

Кінець

Лістинг проекту Marchenko-task\_7\_2:

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <locale.h>

// Функція для виведення анотації та інформації про розробника

void show\_about() {

printf("╔════════════════════════════════════════╗\n");

printf("║ Лабораторна робота 7.2 ║\n");

printf("║ Обчислення кількості заданих чисел ║\n");

printf("╠════════════════════════════════════════╣\n");

printf("║ Розробник: Марченко Кирило ║\n");

printf("║ Група: КБ-24 ║\n");

printf("║ Навчальний заклад: ЦНТУ ║\n");

printf("╚════════════════════════════════════════╝\n\n");

}

int main(void) {

// Налаштування локалі та консолі для роботи з UTF-8

setlocale(LC\_ALL, "uk\_UA.UTF-8");

SetConsoleCP(65001);

SetConsoleOutputCP(65001);

// Виведення анотації

show\_about();

int numbers[14];

int total = 0;

printf("Введіть 14 натуральних чисел:\n");

for (int i = 0; i < 14; i++) {

if (scanf("%d", &numbers[i]) != 1) {

fprintf(stderr, "Помилка зчитування числа.\n");

return EXIT\_FAILURE;

}

}

// Підрахунок чисел, рівних 150, 1000 або 10000

for (int i = 0; i < 14; i++) {

if (numbers[i] == 150 || numbers[i] == 1000 || numbers[i] == 10000) {

total++;

}

}

printf("\nКількість чисел, рівних 150, 1000 або 10000: %d\n", total);

return EXIT\_SUCCESS;

}

**Аргументи досягнення мети лабораторної роботи:**

1. **Чітке розуміння постановки задач:** Аналіз умов завдань 7.1 і 7.2 дозволив правильно сформулювати вимоги до ПЗ.
2. **Універсальність завдань:** Робота з різними типами даних (текст і числа) показала здатність застосовувати алгоритмічне мислення.
3. **Використання стандарту Unicode:** Забезпечення роботи з UTF-8 дає можливість обробляти багатомовну інформацію.
4. **Адаптивність до різних кодових сторінок:** Налаштування консолі (SetConsoleCP/SetConsoleOutputCP) гарантує коректне відображення символів.
5. **Застосування локалі:** Використання setlocale(LC\_ALL, "uk\_UA.UTF-8") сприяє правильній обробці кирилиці.
6. **Перетворення рядків у широкий формат:** MultiByteToWideChar дозволяє коректно працювати з Unicode, що є критичним для завдання.
7. **Приведення тексту до нижнього регістру:** Забезпечує нечутливість до регістру при пошуку ключових слів.
8. **Ретельна перевірка вводу:** Контроль правильного завершення речення дозволяє уникнути помилок на початковому етапі.
9. **Створення тест-сьютів:** Розробка не менше ніж 10 тест-кейсів для кожної задачі сприяє системному тестуванню ПЗ.
10. **Системне тестування:** Проведення тестування забезпечує стабільність роботи застосунку.
11. **Реалізація псевдографіки:** Використання рамок та символів покращує інтерфейс і робить застосунок більш інформативним.
12. **Документування коду:** Чітке коментування коду сприяє його зрозумілості та подальшому супроводу.
13. **Алгоритмічна розбивка завдання:** Поділ задач на підзадачі дозволив краще структуризувати рішення.
14. **Використання циклів для обробки даних:** Перебір масивів і рядків засвідчує знання основ програмування.
15. **Робота з умовними операторами:** Використання if/else гарантує коректну логіку перевірок.
16. **Обробка помилкових даних:** Перевірки на помилки вводу забезпечують стабільність застосунку.
17. **Підрахунок чисел за умовою:** Алгоритм коректно визначає числа, що відповідають заданим значенням (150, 1000, 10000).
18. **Гнучкість реалізації:** Програми легко модифікувати для розширення функціоналу.
19. **Покращення навичок роботи з IDE:** Використання Code::Blocks сприяє засвоєнню принципів кросплатформенної розробки.
20. **Практичний досвід роботи з Git:** Створення та управління Git-репозиторієм зміцнює навички командної роботи і контролю версій.
21. **Фокус на стандарті С (С18):** Дотримання стандартів забезпечує сумісність і надійність коду.
22. **Обробка текстових даних:** Алгоритми для роботи з рядками розвивають аналітичні здібності.
23. **Застосування функцій Windows API:** Використання SetConsoleCP та SetConsoleOutputCP покращує роботу з кодуванням у Windows.
24. **Розуміння особливостей Unicode:** Практична робота дає глибше розуміння стандартів кодування символів.
25. **Модульність рішення:** Виділення окремих функцій (наприклад, для виведення анотації) покращує структуру програми.
26. **Використання широких рядків (wchar\_t):** Забезпечує коректну обробку символів, що виходять за межі ASCII.
27. **Підвищення якості ПЗ:** Системне тестування допомагає знаходити та виправляти помилки на ранніх етапах розробки.
28. **Критичне мислення:** Аналіз помилок і їх вирішення розвиває навички проблемного мислення.
29. **Інтерактивність застосунку:** Застосунок реагує на введення користувача, що підвищує користувацький досвід.
30. **Обробка натуральних чисел:** Завдання 7.2 дозволяє ефективно працювати з масивами числових даних.
31. **Оптимізація алгоритмів:** Ретельне тестування дозволило оптимізувати логіку обробки даних.
32. **Практичне застосування теорії:** Лабораторна робота демонструє практичне застосування знань з теорії програмування.
33. **Розвиток навичок алгоритмізації:** Створення псевдоалгоритмів сприяє глибшому розумінню логіки вирішення задач.
34. **Системний підхід до тестування:** Розробка тест-сьютів сприяє системному підходу до перевірки якості ПЗ.
35. **Уважність до деталей:** Детальна перевірка вводу та виводу підвищує надійність програми.
36. **Практична орієнтованість:** Завдання спрямоване на набуття практичних навичок, що важливо для майбутньої кар’єри.
37. **Використання сучасних технологій:** Інтеграція з GitHub та Code::Blocks показує сучасний підхід до розробки ПЗ.
38. **Розвиток логічного мислення:** Алгоритмічна реалізація задач стимулює розвиток логічного мислення.
39. **Застосування циклічних структур:** Використання циклів для обробки даних дозволило чітко структурувати процес розв’язання.
40. **Глибоке занурення в мову С:** Реалізація задач сприяє кращому розумінню особливостей мови програмування С.
41. **Практичні знання з обробки даних:** Робота з масивами та рядками дає практичний досвід у обробці даних.
42. **Розуміння важливості кодування:** Використання UTF-8 підкреслює значення кодування для коректного відображення даних.
43. **Самостійне вирішення проблем:** Під час виконання лабораторної роботи виникали ситуації, що вимагали самостійного пошуку рішень.
44. **Покращення навичок налагодження:** Виявлення і усунення помилок у коді сприяє розвитку умінь налагодження ПЗ.
45. **Особиста відповідальність:** Аналіз помилок і їх виправлення підвищують відповідальність за кінцевий результат.
46. **Орієнтація на досягнення мети:** Кожен етап роботи спрямований на досягнення конкретної мети, що підвищує ефективність процесу.
47. **Застосування модульного тестування:** Розробка окремих тест-кейсів дозволила ізолювати та перевірити кожну частину алгоритму.
48. **Конструктивна критика власної роботи:** Аналіз помилок і їх усунення сприяє постійному вдосконаленню навичок.
49. **Практична корисність здобутих знань:** Отримані навички з обробки масивів і текстової інформації є безпосередньо застосовними у розробці ПЗ.
50. **Особисті враження та мотивація:** Виконання лабораторної роботи принесло задоволення від усвідомлення власного професійного зростання, розширило технічний кругозір та надало впевненості у власних силах для подальших завдань.

Відповіді на контрольні запитання:

1. git init – Ініціалізує новий локальний Git-репозиторій у поточній теці.
   * Після виконання цієї команди в поточній директорії створюється прихована папка .git, яка містить всі необхідні файли для керування версіями.
2. git add – Додає файли до індексу (стейджинг-області) для підготовки до коміту.
   * git add <ім'я\_файлу> – додає конкретний файл.
   * git add . – додає всі змінені та нові файли в поточній директорії до індексу.
3. git commit -m "текст\_коміту" – Фіксує зміни в репозиторії.
   * -m "текст\_коміту" – дозволяє вказати коментар до коміту без відкриття редактора.
   * Коміт зберігає зміни локально в історії Git, але ще не відправляє їх у віддалений репозиторій.
4. git remote add origin git@github.com:username/BMTP-LAB7-прізвище.git – Додає віддалений репозиторій (GitHub) для збереження проєкту.
   * origin – стандартне ім'я для віддаленого репозиторію.
   * Після цієї команди локальний репозиторій знає, куди відправляти зміни.
5. git push – Відправляє зміни з локального репозиторію у віддалений репозиторій на GitHub.
   * git push origin main – відправляє зміни у гілку main віддаленого репозиторію.
   * Якщо репозиторій тільки створено, може знадобитися виконати git push -u origin main, щоб встановити гілку main як основну для подальших оновлень.

Додаток А – тест-сьют до Задачі 7.1



Додаток Б – тест-сьют до Задачі 7.2

