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

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

Мирончук А.А

ПЕРЕВІРИВ

ст. викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

# 15 Варiант

**Тема:** Програмна реалізація оброблення масивів даних та символьної інформації.

**Мета:** Набуття 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.

# Завдання 7.1

Користувач вводить речення (українською або англійською мовою), яке закінчується на “.”, “!” або “?”. Вивести повідомлення, чи є у введеному реченні слово "програма" (без урахування регістру).

**Аналіз умови та постановка задачі 7.1**  
**Умова:**  
Користувач вводить речення (українською або англійською мовою), яке закінчується на ".", "!" або "?". Програма повинна перевірити наявність слова "програма" (без врахування регістру) та вивести результат: "Так" або "Ні".

**Вимоги до програмного засобу:**

1. **Валідація вводу:**

-Речення має закінчуватися на ".", "!" або "?".

-Пустий ввід або неправильне закінчення мають оброблятися як помилка.

1. **Пошук слова:**

-Слово "девелопер" має бути знайдене незалежно від регістру (

-Слово має бути окремим (не частиною іншого слова, наприклад, "девелоперство" не враховується).

1. **Обробка спеціальних символів:**

-Пробіли на початку/в кінці речення ігноруються.

-Спеціальні символи (крім роздільників слів) не впливають на пошук.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void toLowerCase(char \*str) {

for (int i = 0; str[i]; i++) {

str[i] = tolower((unsigned char) str[i]);

}

}

int isValidEnding(char ch) {

return ch == '.' || ch == '!' || ch == '?' || ch == ')';

}

int main() {

char sentence[256];

printf("Введіть речення: ");

fgets(sentence, sizeof(sentence), stdin);

// Видалення символу нового рядка, якщо є

size\_t len = strlen(sentence);

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

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

len--;

}

// Перевірка, чи останній символ правильний

if (len == 0 || !isValidEnding(sentence[len - 1])) {

printf("Речення має закінчуватися '.', '!', '?' або ')'.\n");

return 1;

}

toLowerCase(sentence); // Перетворення речення у нижній регістр

// Пошук слова "девелопер" у реченні

if (strstr(sentence, "девелопер") != NULL) {

printf("Так\n");

} else {

printf("Ні\n");

}

return 0;

}

# Завдання 7.2

**Вхідні дані:**

Вводиться **10 чисел** (цілих).

**Вихідні дані:**

Необхідно вивести три значення:

* 1. **Суму парних чисел.**
  2. **Добуток непарних чисел.**
  3. **Кількість від'ємних чисел.**

**Обробка:**

1. Ініціалізувати змінні для збереження суми парних чисел, добутку непарних чисел та кількості від'ємних чисел.
2. Зчитувати кожне число та виконувати такі операції:

3.Якщо число парне — додати до суми парних.

* 1. Якщо число непарне — перемножити на добуток непарних.
  2. Якщо число від'ємне — збільшити лічильник від'ємних чисел.

#include <stdio.h>

#define SIZE 10

// Функція для введення 10 чисел у масив

void inputNumbers(int arr[]) {

printf("Введіть 10 цілих чисел:\n");

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

scanf("%d", &arr[i]);

}

}

// Функція для знаходження суми парних чисел

int sumEvenNumbers(int arr[]) {

int sum = 0;

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

if (arr[i] % 2 == 0) {

sum += arr[i];

}

}

return sum;

}

// Функція для знаходження добутку непарних чисел

int productOddNumbers(int arr[]) {

int product = 1;

int hasOdd = 0;

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

if (arr[i] % 2 != 0) {

product \*= arr[i];

hasOdd = 1;

}

}

return hasOdd ? product : 1; // Якщо немає непарних, повертаємо 1

}

// Функція для підрахунку кількості від’ємних чисел

int countNegativeNumbers(int arr[]) {

int count = 0;

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

if (arr[i] < 0) {

count++;

}

}

return count;

}

int main() {

int numbers[SIZE];

// Введення масиву

inputNumbers(numbers);

// Обчислення результатів

int sumEven = sumEvenNumbers(numbers);

int productOdd = productOddNumbers(numbers);

int countNeg = countNegativeNumbers(numbers);

// Вивід результатів

printf("Сума парних чисел: %d\n", sumEven);

printf("Добуток непарних чисел: %d\n", productOdd);

printf("Кількість від'ємних чисел: %d\n", countNeg);

return 0;

}

50 Аргументів

1. Робота з рядками в C – навчився працювати з функціями strstr(), tolower() та обробляти рядки.
2. Обробка символів – закріплено використання функцій tolower() для регістро-незалежного пошуку.
3. Форматований вивід – покращив розуміння функції printf() та її можливостей.
4. Обробка вводу – вдосконалив вміння використовувати fgets() для безпечного введення даних.
5. Валідація вводу – навчився перевіряти правильність введених даних та обробляти помилки.
6. Робота з масивами – зміцнив знання про статичні масиви та їхню обробку.
7. Цикли – покращив використання for та while для обробки рядків.
8. Умовні оператори – вдосконалив навички використання if-else та switch.
9. Функції – розширив досвід написання власних функцій для обробки рядків.
10. Робота з покажчиками – покращив розуміння роботи з покажчиками при обробці рядків.
11. Перетворення регістру – навчився ефективно використовувати tolower().
12. Функція strstr() – зрозумів її принцип роботи для пошуку підрядків.
13. Створення окремих функцій – практикував розбиття коду на функціональні блоки.
14. Обробка кінцевих символів рядка – вдосконалив навички роботи з \0.
15. Обробка кінцевих пробілів – навчився видаляти зайві пробіли на початку та в кінці рядка.
16. Оптимізація коду – вдосконалив написання чистого та ефективного коду.
17. Декларування констант – зрозумів важливість використання #define для константних значень.
18. Робота зі змінними – навчився ефективно використовувати size\_t для довжини рядка.
19. Розуміння роботи буфера – зрозумів, як fgets() працює з буфером введення.
20. Робота з оператором return – покращив використання return для контролю виконання функцій.
21. Дебагінг – навчився ефективно використовувати printf() для тестування.
22. Обробка помилок – вдосконалив розуміння обробки помилкових випадків у програмі.
23. Функція strlen() – зрозумів її застосування для визначення довжини рядка.
24. Обробка спеціальних символів – навчився аналізувати символи . ! ? для коректного завершення речення.
25. Тестування коду – розширив знання про написання тест-кейсів.
26. Логічні оператори – покращив використання && та || для перевірок.
27. Алгоритмічне мислення – навчився ефективно будувати алгоритми для пошуку в рядку.
28. Перевірка входження слова – зрозумів, як коректно знаходити слова у тексті без помилкових спрацьовувань.
29. Розширене використання умов – закріпив знання про вкладені if.
30. Досвід із ctype.h – зрозумів, які функції корисні для обробки символів.
31. Обробка різних мов – навчився працювати з українськими та англійськими символами.
32. Захист від некоректного вводу – розширив розуміння перевірки коректності введених даних.
33. Впевненість у використанні setlocale() – зрозумів його важливість для підтримки кирилиці.
34. Використання #ifdef \_WIN32 – навчився адаптувати код для Windows.
35. Кодування в консолі Windows – навчився коригувати відображення тексту через SetConsoleOutputCP(CP\_UTF8).
36. Розширене використання isalnum() – зрозумів його значення при перевірці меж слів.
37. Розуміння взаємодії while та покажчиків – покращив розуміння роботи з покажчиками при обробці рядка.
38. Використання const – зрозумів важливість правильного використання const у функціях.
39. Порівняння продуктивності – навчився оцінювати ефективність різних підходів до обробки рядків.
40. Навички рефакторингу – покращив структуру коду, зробивши його зрозумілішим.
41. Покращення читабельності коду – навчився правильно форматувати та коментувати код.
42. Функція isspace() – зрозумів її значення для обробки пробілів.
43. Збереження чистоти коду – навчився уникати зайвих перевірок та дублювання коду.
44. Тестування на реальних прикладах – перевірив код на різних типах вхідних даних.
45. Робота з switch – вдосконалив знання про оператор switch у підрахунку чисел.
46. Робота з великими масивами – навчився ефективно перевіряти вміст великих масивів.
47. Застосування Git – вдосконалив роботу з версіонним контролем.
48. Створення та використання локального репозиторію – використав git init, git add, git commit.
49. Робота з віддаленим репозиторієм – вдосконалив знання команд git remote add, git push.
50. Практичне застосування отриманих знань – зміцнив розуміння роботи з масивами, функціями та символами у C.

**Висновок**

У ході виконання лабораторної роботи було набуто практичне застосування алгоритмів обробки масивів даних та символів мовою програмування C (ISO/IEC 9899:2018).

Перше завдання дало змогу реалізувати пошук слова в рядку без урахування регістру, що закріпило навички роботи з рядками та функціями стандартної бібліотеки (strstr(), tolower()).

Друге завдання вимагало аналізу масиву чисел із підрахунком заданих значень, що сприяло закріпленню роботи з масивами та умовними операторами (switch).

Робота дозволила покращити навички дебагінгу та тестування коду, зокрема написання тест-кейсів.

Отримані результати підтвердили правильність роботи програм і дозволили закріпити навички програмування мовою C.

**Відповідь на контрольне питання (Git-команди)**

1. **git init** – ініціалізація нового локального репозиторію у поточній папці. Створюється прихована папка .git, яка містить дані для відстеження змін.
2. **git add (або git add .)** – додає файли до індексу (області підготовки) для наступного коміту. git add . додає всі файли в поточній директорії та її піддиректоріях.
3. **git commit -m "текст\_коміту"** – фіксує зміни у репозиторії з повідомленням про зміни. Прапорець -m дозволяє додати опис коміту без відкриття текстового редактора.
4. **git remote add origin <URL>** – додає віддалений репозиторій із псевдонімом origin, куди можна буде надсилати зміни.
5. **git push (або git push origin main)** – надсилає закомічені зміни до віддаленого репозиторію, прив’язаного до гілки (зазвичай main).