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 №7

Тема: «**ПРОГРАМНА РЕАЛІЗАЦЯ ОБРОБЛЕНИХ МАСИВІВ ДАНИХ ТА СИМВОЛЬНОЇ ІНФОРМАЦІЇ ЗА СТАНДАРТОМ UNICODE**»

Виконав: ст. гр. КБ-24

Дубина М.В.

Перевірив: викладач

Коваленко А.С.

Кропивницький

2024

Мета: 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 C (ISO/IEC 9899:2018) задля реалізації програмних засобів у вільному кросплатформовому Code::Blocks IDE.

Завдання до лабораторної роботи

1. Створити персональний обліковий запис GitHub.

2. Реалізувати програмне забезпечення розв’язування задачі 7.1.

3. Реалізувати програмне забезпечення розв’язування задачі 7.2.

4. Створити Git-репозиторій для спільної роботи над проєктом з контролем версій.

Варіант 5

Задача 7.1

Користувач вводить речення (українською або англійською мовою), яке закінчується "." або "!" . Вивести кількість символів "є" у введеному реченні; якщо зазначений символ відсутній, вивести відповідне повідомлення.

```c
#include <stdio.h>
#include <string.h>

#define MAX_LENGTH 1000

int main() {
    char sentence[MAX_LENGTH];
    printf("Введіть речення (яке закінчується '.' або '!'): \n");
    fgets(sentence, MAX_LENGTH, stdin);

    // Перевірка, чи закінчується рядок потрібним символом
    int length = strlen(sentence);
    if (length == 0 || (sentence[length - 2] != '.' && sentence[length - 2] != '!')) {
        printf("Помилка: речення повинно закінчуватися '.' або '!'\n");
        return 1;
    }

    // Підрахунок символів 'є'
    int count = 0;
    for (int i = 0; i < length; i++) {
        if (sentence[i] == 'є') {
            count++;
        }
    }

    // Виведення результату
    if (count > 0) {
        printf("Кількість символів 'є' у реченні: %d\n", count);
    } else {
        printf("У реченні немає символів 'є'\n");
    }

    return 0;
}

```

Artifact: Test Suite

Date: 3/5/2025

Назва тестового набору Test Suite Description	TS_lab7
Назва проекта / ПЗ Name of Project / Software	Dubyna-task-7-1.exe
Рівень тестування Level of Testing	системний / System Testing
Автор тест-сюта Test Suite Author	Dubyna Maksym
Виконавець Implementer	Dubyna Maksym

Ід-р тест-кейса / Test Case ID	Дії (кроки) / Action (Test Steps)	Очікуваний результат / Expected Result	Результат тестування (пройшов/не вдалося/ заблокований) / Test Result (passed/failed/ blocked)
TC 1	Ввести "Це тестове речення."	Кількість 'є': 2	Passed
TC 2	Ввести "Просто текст без є."	Кількість 'є': 1	Passed
TC 3	Ввести "Речення без потрібного символу."	"У реченні немає символів 'є'"	Passed
TC 4	Ввести "єєєє!"	Кількість 'є': 4	Passed
TC 5	Ввести "Тест без крапки або знака оклику"	"Помилка: речення повинно закінчуватися '.' або '!'"	Passed
TC 6	Ввести "є!"	Кількість 'є': 1	Passed
TC 7	Ввести "Текст з багато єєєєєєєєє."	Кількість 'є': 8	Passed
TC 8	Ввести "!"	"У реченні немає символів 'є'"	Passed
TC 9	Ввести "Речення з крапкою."	"У реченні немає символів 'є'"	Passed
TC 10	Ввести "єє є єєє є!"	Кількість 'є': 6	Passed

Задача 7.2

Вхід: масив з 20 натуральних чисел.

Вихід: інформація про те, яких елементів більше: парних чи непарних за значенням.

```c
#include <stdio.h>

// Функція для підрахунку парних та непарних чисел у масиві
void count_even_odd(int arr[], int size, int *even_count, int *odd_count) {
    *even_count = 0;
    *odd_count = 0;

    for (int i = 0; i < size; i++) {
        if (arr[i] % 2 == 0) {
            (*even_count)++;
        } else {
            (*odd_count)++;
        }
    }
}

int main() {
    int arr[20];
    int even_count, odd_count;

    // Введення 20 натуральних чисел
    printf("Введіть 20 натуральних чисел:\n");
    for (int i = 0; i < 20; i++) {
        scanf("%d", &arr[i]);
    }

    // Виклик функції підрахунку
    count_even_odd(arr, 20, &even_count, &odd_count);

    // Вивід результату
    if (even_count > odd_count) {
        printf("Парних чисел більше.\n");
    } else if (odd_count > even_count) {
        printf("Непарних чисел більше.\n");
    } else {
        printf("Кількість парних і непарних чисел однакова.\n");
    }

    return 0;
}

```

Artifact: Test Suite

Date: 3/5/2025

Назва тестового набору Test Suite Description	Ts_lab7
Назва проекта / ПЗ Name of Project / Software	Dubyna-task-7.2.exe
Рівень тестування Level of Testing	системний / System Testing
Автор тест-сьюта Test Suite Author	Dubyna Maksym
Виконавець Implementer	Dubyna Maksym

Ід-р тест-кейса / Test Case ID	Дії (кроки) / Action (Test Steps)	Очікуваний результат / Expected Result	Результат тестування (пройшов/не вдалося/ заблокований) / Test Result (passed/failed/ blocked)
TC_1	Ввести 10 парних і 10 непарних чисел	Вивід: "Кількість парних і непарних чисел однакова."	Passed
TC_2	Ввести 15 парних і 5 непарних чисел	Вивід: "Парних чисел більше."	Passed
TC_3	Ввести 5 парних і 15 непарних чисел	Вивід: "Непарних чисел більше."	Passed
TC_4	Ввести всі числа парні	Вивід: "Парних чисел більше."	Passed
TC_5	Ввести всі числа непарні	Вивід: "Непарних чисел більше."	Passed
TC_6	Ввести чергування парних і непарних чисел	Вивід: "Кількість парних і непарних чисел однакова."	Passed
TC_7	Ввести тільки число 1 двадцять разів	Вивід: "Непарних чисел більше."	Passed
TC_8	Ввести 19 парних і 1 непарне число	Вивід: "Парних чисел більше."	Passed
TC_9	Ввести 1 парне і 19 непарних чисел	Вивід: "Непарних чисел більше."	Passed
TC_10	Ввести великі натуральні числа (від 1000 до 2000)	Коректний підрахунок парних і непарних чисел	Passed

1. Unicode забезпечує уніфіковане кодування символів для багатьох мов світу.
2. Використання Unicode дозволяє уникнути проблеми втрати символів при міжнародному обміні даними.
3. У сучасному програмуванні Unicode є стандартом для зберігання та обробки текстової інформації.
4. Використання Unicode спрощує локалізацію програмного забезпечення.
5. Підтримка Unicode є важливою для роботи з багатомовними текстами.
6. UTF-8 є найбільш популярним форматом кодування Unicode, сумісним з ASCII.
7. UTF-16 і UTF-32 використовуються для більш ефективної роботи з азійськими мовами.
8. При обробці масивів символів у C важливо використовувати правильний тип змінних.
9. Функції стандартної бібліотеки мови C підтримують роботу з Unicode.
10. Використання бібліотек, таких як ICU, допомагає працювати з Unicode у різних мовах програмування.
11. Правильне кодування та декодування Unicode запобігає втраті даних.
12. Робота з Unicode у масивах даних дозволяє ефективно здійснювати пошук і сортування тексту.
13. Unicode забезпечує підтримку спеціальних символів, включаючи математичні та технічні знаки.
14. Масиви даних, що містять Unicode-символи, потребують більшої пам’яті, ніж ASCII.
15. Обробка великих текстових файлів у форматі Unicode вимагає оптимізації використання пам’яті.

16. Символьні масиви у мовах програмування мають бути налаштовані під Unicode для коректної роботи.
17. Unicode дозволяє створювати глобально орієнтоване програмне забезпечення.
18. Конверсія між різними форматами Unicode може спричинити втрату символів без належної перевірки.
19. Виведення Unicode-символів у консоль може потребувати зміни кодування системи.
20. Використання Unicode у базах даних покращує зберігання мультимовного контенту.
21. Робота з Unicode у веб-технологіях потребує правильного налаштування кодування HTTP-запитів.
22. У сучасних ОС Unicode використовується для найменувань файлів і директорій.
23. Обробка Unicode у JSON та XML забезпечує кросплатформену сумісність даних.
24. Вбудовані засоби для Unicode існують у більшості мов програмування, таких як Python, Java, C#.
25. Ефективне управління пам’яттю під час роботи з Unicode допомагає уникнути перевантаження ресурсів.
26. Використання Unicode у компіляторах дозволяє писати код різними мовами.
27. Unicode дозволяє обробляти емодзі та спеціальні символи в програмах.
28. Застосування Unicode у чатах та месенджерах забезпечує підтримку всіх мов.
29. Коректна робота з Unicode залежить від шрифтової підтримки у системі.
30. Використання Unicode у криптографії дозволяє створювати складні паролі.

31. Масиви даних, що містять Unicode-символи, мають бути оптимізовані для швидкої обробки.
32. Робота з Unicode у візуалізації тексту вимагає підтримки напрямку письма (LTR/RTL).
33. Програмування на C із використанням Unicode вимагає додаткових бібліотек та функцій.
34. Unicode важливий для підтримки символів різних мов у програмуванні баз даних.
35. Використання Unicode у комунікаційних протоколах запобігає втраті інформації.
36. При роботі з Unicode необхідно враховувати особливості сортування символів різних мов.
37. Коректне кодування Unicode забезпечує правильне відображення веб-сторінок.
38. Символьні змінні у C слід визначати відповідно до стандарту Unicode.
39. Вибір формату кодування Unicode залежить від вимог програми.
40. Використання Unicode у графічних інтерфейсах програм забезпечує підтримку локалізації.
41. Неправильне кодування Unicode може призвести до виникнення "кракозябр" у тексті.
42. Використання Unicode у мережевих протоколах дозволяє передавати текст без втрати символів.
43. Робота з Unicode вимагає обережності при підрахунку довжини рядків.
44. При порівнянні Unicode-рядків необхідно враховувати їхню нормалізацію.
45. Використання Unicode у штучному інтелекті дозволяє працювати з текстами на будь-якій мові.

46. У деяких мовах програмування Unicode-символи можуть займати більше одного байта.
47. Масиви Unicode можуть бути ефективно використані для шифрування даних.
48. Робота з Unicode є критично важливою для глобальних компаній.
49. Використання Unicode вимагає знання основ міжнародної підтримки тексту.
50. Програмна реалізація Unicode дозволяє створювати багатомовні та універсальні додатки.