МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

Лабораторна робота №1

з дисципліни

«Операційні системи»

Тема: «РОЗРОБКА УНІВЕРСАЛЬНИХ ДОДАТКІВ ДЛЯ РІЗНИХ ТИПІВ КОДУВАНЬ СИМВОЛЬНОЇ ІНФОРМАЦІЇ»

Виконав:
ст. гр. ПЗПІ-23-5
Ткач Леонід Ярославович

Перевірила
доц. каф. ПІ
Мельнікова Роксана Валеріївна

Харків 2025

Мета роботи

Метою даної лабораторної роботи є навчитися опрацьовувати тексти для типів кодування ASCII та UNICODE, при чому сама програма неповинна залежати від обраного способу.

Хід роботи

1. Повний опис усіх типів і функцій, які використовуються для забезпеченості універсальності кодування:

– SetConsoleOutputCP(1251) та SetConsoleCP(1251) потрібні для коректного відображення кирилиці в консолі.

– TCHAR - тип даних який дозволяє працювати з рядками в незалежності від типу їхнього кодування (Unicode, ASCII тощо).

2. Повний опис усіх типів і функцій, які використовуються для виведення інформації в консольному режимі на російській (українській) мові:

– _wcout дозволяє виводити в консоль інформацію формату Unicode.

– _tprintf дозволяє виводи в консоль інформацію формату Unicode або ASCII, що залежить від налаштувань проєкту.

3. Текст розробленої програми з коментарями:

```
#include <iostream>
#include <string>
#include <windows.h>
#include <tchar.h>

#include <vector>
#include <algorithm>
#include <fstream>
```

```
enum class FileEncoding {
    UTF8,
    UTF16_LE,
    UTF16_BE
};

class FileProcessor {
private:
    std::string inputFilePath;
    std::string outputFilePath;
    FileEncoding encoding;

    // For UTF-8
    std::vector<unsigned char> utf8Bytes;
    std::vector<std::vector<unsigned char>> utf8Lines;

    // For UTF-16
    std::vector<char16_t> utf16Bytes;
    std::vector<std::u16string> utf16Lines;
    char16_t bomMarkerLE = 0xFEFF; // BOM marker for UTF-16 Little Endian
    char16_t bomMarkerBE = 0xFFFE; // BOM marker for UTF-16 Big Endian

    // Detect file encoding
    FileEncoding detectEncoding() {
        std::ifstream inputFile(inputFilePath, std::ios::binary);
        if (!inputFile) {
            throw std::runtime_error("Failed to open input file");
        }

        // Check for UTF-16 BOM
        unsigned char bom[2];
        inputFile.read(reinterpret_cast<char*>(bom), 2);
```

```
    if (bom[0] == 0xFF && bom[1] == 0xFE) {
        return FileEncoding::UTF16_LE;
    }
    else if (bom[0] == 0xFE && bom[1] == 0xFF) {
        return FileEncoding::UTF16_BE;
    }

    return FileEncoding::UTF8;
}

// Read UTF-8 file
bool readUTF8File() {
    std::ifstream inputFile(inputFilePath, std::ios::binary);
    if (!inputFile) {
        std::cerr << "Failed to open input file" << std::endl;
        return false;
    }

    utf8Bytes = std::vector<unsigned char>(
        (std::istreambuf_iterator<char>(inputFile)),
        std::istreambuf_iterator<char>()
    );
    inputFile.close();
    return true;
}

// Split UTF-8 file into lines
void splitUTF8IntoLines() {
    std::vector<unsigned char> currentLine;
    for (unsigned char byte : utf8Bytes) {
        currentLine.push_back(byte);

        // Check line end (0D 0A or 0A)
        if (byte == 0x0A ||
            (currentLine.size() >= 2 &&
                currentLine[currentLine.size() - 2] == 0x0D &&
                currentLine[currentLine.size() - 1] == 0x0A)) {
```

```
            utf8Lines.push_back(currentLine);
            currentLine.clear();
        }
    }

    // Add last line if not empty
    if (!currentLine.empty()) {
        utf8Lines.push_back(currentLine);
    }
}

// Reverse UTF-8 lines
void reverseUTF8Lines() {
    for (auto& line : utf8Lines) {
        // Find line end position
        size_t endingPos = line.size();

        // Check for 0D 0A
        if (line.size() >= 2 &&
            line[line.size() - 2] == 0x0D &&
            line[line.size() - 1] == 0x0A) {
            endingPos -= 2;
        }
        // Check for 0A
        else if (line.back() == 0x0A) {
            endingPos -= 1;
        }

        // Reverse characters before line end
        std::reverse(line.begin(), line.begin() + endingPos);
    }
}

// Write UTF-8 to file
bool writeUTF8ToFile() {
    std::ofstream outputFile(outputFilePath, std::ios::binary);
    if (!outputFile) {
```

```
        std::cerr << "Failed to open output file" << std::endl;
        return false;
    }

    for (const auto& line : utf8Lines) {
        outputFile.write(reinterpret_cast<const char*>(line.data()), line.size());
    }

    outputFile.close();
    return true;
}

// Read UTF-16 file (updated to support Little and Big Endian)
bool readUTF16File() {
    std::ifstream inputFile(inputFilePath, std::ios::binary | std::ios::ate);
    if (!inputFile) {
        std::cerr << "Failed to open input file" << std::endl;
        return false;
    }

    // Get file size
    std::streampos fileSize = inputFile.tellg();
    inputFile.seekg(0, std::ios::beg);

    // Read full file contents
    std::vector<char> rawBytes(fileSize);
    if (!inputFile.read(rawBytes.data(), fileSize)) {
        std::cerr << "Failed to read file contents" << std::endl;
        return false;
    }

    // Convert raw bytes to UTF-16
    utf16Bytes.clear();
    for (size_t i = 2; i + 1 < rawBytes.size(); i += 2) {
        char16_t ch;
        if (encoding == FileEncoding::UTF16_LE) {
            // Little Endian: low byte first
```

```
            ch = (rawBytes[i + 1] << 8) | rawBytes[i];
        }
        else {
            // Big Endian: high byte first
            ch = (rawBytes[i] << 8) | rawBytes[i + 1];
        }
        utf16Bytes.push_back(ch);
    }

    return true;
}

// Split UTF-16 into lines
void splitUTF16IntoLines() {
    std::u16string currentLine;
    for (size_t i = 0; i < utf16Bytes.size(); ++i) {
        char16_t ch = utf16Bytes[i];

        currentLine += ch;

        // Check line end (0D 0A or 0A or 0D)
        if (ch == 0x000D || ch == 0x000A) {
            // Special handling for CRLF
            if (ch == 0x000D && i + 1 < utf16Bytes.size() && utf16Bytes[i + 1] == 0x000A) {
                currentLine += utf16Bytes[++i];
            }

            utf16Lines.push_back(currentLine);
            currentLine.clear();
        }
    }

    // Add last line if not empty
    if (!currentLine.empty()) {
        utf16Lines.push_back(currentLine);
    }
}
```

```
// Reverse UTF-16 lines
void reverseUTF16Lines() {
    for (auto& line : utf16Lines) {
        // Find line end position
        size_t endingPos = line.length();
        while (endingPos > 0 && (line[endingPos - 1] == 0x000D || line[endingPos - 1] == 0x000A)) {
            endingPos--;
        }

        // Reverse characters before line end
        std::reverse(line.begin(), line.begin() + endingPos);
    }
}

// Write UTF-16 to file (updated to support Little and Big Endian)
bool writeUTF16ToFile() {
    std::ofstream outputFile(outputFilePath, std::ios::binary);
    if (!outputFile) {
        std::cerr << "Failed to open output file" << std::endl;
        return false;
    }

    // Write appropriate BOM marker
    if (encoding == FileEncoding::UTF16_LE) {
        outputFile.put(bomMarkerLE & 0xFF);
        outputFile.put((bomMarkerLE >> 8) & 0xFF);
    }
    else {
        outputFile.put((bomMarkerBE >> 8) & 0xFF);
        outputFile.put(bomMarkerBE & 0xFF);
    }

    // Write each line
    for (const auto& line : utf16Lines) {
        for (char16_t ch : line) {
            if (encoding == FileEncoding::UTF16_LE) {
```

```
                // Little Endian: low byte first
                outputFile.put(ch & 0xFF);
                outputFile.put((ch >> 8) & 0xFF);
            }
            else {
                // Big Endian: high byte first
                outputFile.put((ch >> 8) & 0xFF);
                outputFile.put(ch & 0xFF);
            }
        }
    }

    outputFile.close();
    return true;
}

public:
    FileProcessor(const std::string& input, const std::string& output)
        : inputFilePath(input), outputFilePath(output) {
        encoding = detectEncoding();
    }

    // Process file
    bool processFile() {
        switch (encoding) {
        case FileEncoding::UTF8:
            if (!readUTF8File()) return false;
            splitUTF8IntoLines();
            reverseUTF8Lines();
            return writeUTF8ToFile();

        case FileEncoding::UTF16_LE:
        case FileEncoding::UTF16_BE:
            if (!readUTF16File()) return false;
            splitUTF16IntoLines();
            reverseUTF16Lines();
            return writeUTF16ToFile();
```

```
        default:
            std::cerr << "Unsupported file encoding" << std::endl;
            return false;
        }
    }
};

int main() {
    // 2. Складіть оператори для програмної перевірки типу кодування
    // та визначте довжину в байтах типу TCHAR:
    std::cout << "Size of TCHAR: " << sizeof(TCHAR) << " bytes." << std::endl;
    std::cout << "------------------------------" << std::endl;
    // 3. Визначення типу кодування по макросах у командному рядку
    // Перевірка, чи визначено макрос UNICODE
#ifdef UNICODE
    std::wcout << L"Using UNICODE encoding" << std::endl;
#else
    std::cout << "Using ASCII encoding" << std::endl;
#endif
    std::cout << "------------------------------" << std::endl;

    // 6. Задайте ПІБ членів своєї сім'ї в ASCII та виведіть задані значення
    // Приклад: ПІБ членів сім'ї в ASCII
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "ukrainian");  // Для української локалі
    char family_name_ascii[][90] = { {"Ткач Леонід Ярославович"},{"Ткач Михайло Ярославович"},{"Андрієнко Андрій Андрійович"} };
    std::cout << "Family name in ASCII: " << family_name_ascii[0] << std::endl;
    std::cout << "------------------------------" << std::endl;
    // 7. Переведіть рядки в UNICODE за допомогою MultiByteToWideChar
    TCHAR family_name_unicode[sizeof(family_name_ascii) / sizeof(family_name_ascii[0])][90];
    MultiByteToWideChar(CP_ACP, 0, family_name_ascii[0], sizeof(family_name_ascii[0]), family_name_unicode[0], sizeof(family_name_unicode[0]));
    MultiByteToWideChar(CP_ACP, 0, family_name_ascii[1], sizeof(family_name_ascii[1]), family_name_unicode[1], sizeof(family_name_unicode[1]));
    MultiByteToWideChar(CP_ACP, 0, family_name_ascii[2], sizeof(family_name_ascii[2]), family_name_unicode[2], sizeof(family_name_unicode[2]));
```

```
    // 8. Виведення отриманого масиву за допомогою трьох методів
    // 8.1 Використання _tprintf для виведення в залежності від налаштувань кодування
    _tprintf(_T("Family name in UNICODE using _tprintf: %s\n"), family_name_unicode[0]);

    // 8.2 Використання wcout для виведення в UNICODE
    std::wcout << L"Family name in UNICODE using wcout: " << family_name_unicode[0] << std::endl;

    // 8.3 Використання MessageBox для виведення в UNICODE
    MessageBox(0, family_name_unicode[0], _T("Family Name"), MB_OK);
    std::cout << "-----------------------------" << std::endl;
    // 9. Упорядкування масиву рядків, заданих в UNICODE
    // Створюємо копію масиву для сортування
    wchar_t family_name_unicode_copy[sizeof(family_name_unicode) / sizeof(family_name_unicode[0])][90];
    memcpy(family_name_unicode_copy, family_name_unicode, sizeof(family_name_unicode));

    std::wcout << L"Before sorting:" << std::endl;
    for (size_t i = 0; i < sizeof(family_name_unicode_copy) / sizeof(family_name_unicode_copy[0]); i++) {
        std::wcout << family_name_unicode_copy[i] << std::endl;
    }

    // Використання qsort для сортування масиву рядків
    qsort(family_name_unicode_copy, sizeof(family_name_unicode_copy) / sizeof(family_name_unicode_copy[0]),
sizeof(family_name_unicode_copy[0]),
        [](const void* a, const void* b) -> int {
            return wcscmp(static_cast<const wchar_t*>(a), static_cast<const wchar_t*>(b));
        });
    std::cout << "-----------------------------" << std::endl;
    // 9.1 Виведення відсортованого масиву після qsort
    std::wcout << L"After sorting (qsort):" << std::endl;
    for (size_t i = 0; i < sizeof(family_name_unicode_copy) / sizeof(family_name_unicode_copy[0]); i++) {
        std::wcout << family_name_unicode_copy[i] << std::endl;
    }

    // Створюємо копію масиву для std::sort
    std::vector<std::wstring> family_names;
    for (size_t i = 0; i < sizeof(family_name_unicode) / sizeof(family_name_unicode[0]); i++) {
        family_names.emplace_back(family_name_unicode[i]);
    }
```

```
// Використання std::sort для сортування
std::sort(family_names.begin(), family_names.end());
std::cout << "-----------------------------" << std::endl;
// 9.2 Виведення відсортованого масиву після std::sort
std::wcout << L"After sorting (std::sort):" << std::endl;
for (const auto& name : family_names) {
    std::wcout << name << std::endl;
}
std::cout << "-----------------------------" << std::endl;
// 10. Зворотне перетворення масиву з Unicode в ASCII
char family_name_ascii_reversed[3][100];
for (int i = 0; i < 3; i++) {
    WideCharToMultiByte(CP_ACP, 0, family_name_unicode[i], -1,
        family_name_ascii_reversed[i], 100, NULL, NULL);
}

// 11. Виведення результату зворотного перетворення
std::cout << "Reversed ASCII names: "<<std::endl;
for (int i = 0; i < 3; i++) {
    std::cout << family_name_ascii_reversed[i] << std::endl;
}
std::cout << "-----------------------------" << std::endl;
// 12. Обробка текстового файлу (перестановка символів у зворотному порядку)
try {
    std::string inputPath = "C:\\Users\\user\\source\\repos\\NURE_OS_lab_1\\NURE_OS_lab_1\\inputO.txt";
    std::string outputPath = "C:\\Users\\user\\source\\repos\\NURE_OS_lab_1\\NURE_OS_lab_1\\output.txt";

    // Создаем объект FileProcessor для обработки файла
    FileProcessor processor(inputPath, outputPath);

    if (processor.processFile()) {
        std::wcout << L"Successful file processing." << std::endl;
    }
    else {
        std::wcerr << L"File processing error." << std::endl;
    }
```

```
    }
    catch (const std::exception& e) {
        std::cerr << "Unexpected error: " << e.what() << std::endl;
    }

    return 0;
}
```

4. Скриншоти з роботи програми:

```
Size of TCHAR: 2 bytes.
----------------------------
Using UNICODE encoding
----------------------------
Family name in ASCII: Ткач Леонід Ярославович
----------------------------
Family name in UNICODE using _tprintf: Ткач Леонід Ярославович
Family name in UNICODE using wcout: Ткач Леонід Ярославович
----------------------------
Before sorting:
Ткач Леонід Ярославович
Ткач Михайло Ярославович
Андрієнко Андрій Андрійович
----------------------------
After sorting (qsort):
Андрієнко Андрій Андрійович
Ткач Леонід Ярославович
Ткач Михайло Ярославович
----------------------------
After sorting (std::sort):
Андрієнко Андрій Андрійович
Ткач Леонід Ярославович
Ткач Михайло Ярославович
----------------------------
Reversed ASCII names:
Ткач Леонід Ярославович
Ткач Михайло Ярославович
Андрієнко Андрій Андрійович
----------------------------
Successful file processing.
```

Рис. 1 – результат виконання програми

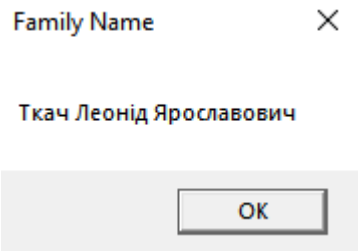

Рис. 2 – відображення елемента MessageBox із інформацією про члена родини

Висновок

У ході виконання лабораторної роботи було вивчено та практично застосовано методи роботи з кодуваннями символів у C++. Досліджено способи

визначення типу кодування символів за замовчуванням, а також через макроси командного рядка. Було розглянуто процес перемикання між ASCII та UNICODE у середовищі розробки.

Вивчено особливості обробки текстових даних у різних кодуваннях, а також методи коректного виведення кириличних символів у консольному режимі за допомогою _tprintf, wcout та MessageBox. Опановано функцію MultiByteToWideChar для перетворення рядків з ASCII у UNICODE та зворотне перетворення.

Досліджено сортування масивів UNICODE-рядків, використовуючи стандартні функції qsort та sort. Також було реалізовано алгоритм обробки файлів, який дозволяє незалежно від способу кодування символів у файлі виконувати зчитування, обробку та запис результату у вихідний файл.

У результаті лабораторної роботи було здобуто досвід роботи з кодуваннями символів, файлами, стандартними засобами C++ та функціями WinAPI, що дозволяють ефективно обробляти текстові дані у різних форматах.