**Лабораторная работа 2**

Написать функции для работы со строками в двух вариантах: для ANSI-строк (char\*) и для строк Unicode (wchar\_t\*).

1. **Функция countwords**: подсчитывает количество слов в строке. Слово — это последовательность символов, отделенная пробелами, табуляциями или знаками препинания (например, запятыми, точками).  
   int countwords(const char\* str); / int countwords(const wchar\_t\* str);
2. **Функция reversestring**: переворачивает строку (меняет порядок символов на противоположный).  
   void reversestring(char\* str); / void reversestring(wchar\_t\* str);
3. **Функция isstringpalindrome**: проверяет, является ли строка палиндромом (читается одинаково слева направо и справа налево, игнорируя регистр и пробелы). Возвращает TRUEили FALSE.  
   BOOL isstringpalindrome(const char\* str); / BOOL isstringpalindrome(const wchar\_t\* str);

**Код программы**:

#include <windows.h>

#include <iostream>

#include <string>

using namespace std;

//Прототипы ANSI-функций

int countwords(const char\* str);

void reversestring(char\* str);

BOOL isstringpalindrome(const char\* str);

//Прототипы Unicode-функций

int countwords(const wchar\_t\* str);

void reversestring(wchar\_t\* str);

BOOL isstringpalindrome(const wchar\_t\* str);

void printError(DWORD errorCode);

//Подсчет слов

int countwords(const char\* str) {

if (str == NULL) return 0;

bool inWord = false;

int wordCount = 0;

while (\*str != '\0') {

// Если текущий символ - часть слова (буква или цифра)

if (isalnum((unsigned char)\*str)) {

if (!inWord) {

wordCount++;

inWord = true;

}

}

else {

inWord = false;

}

str++;

}

return wordCount;

}

//Переворот строки

void reversestring(char\* str) {

if (!str) {

printError(ERROR\_INVALID\_PARAMETER);

return;

};

int length = strlen(str);

for (int i = 0; i < length / 2; i++) {

swap(str[i], str[length - 1 - i]);

}

}

//Палидром

BOOL isstringpalindrome(const char\* str) {

if (!str) return FALSE;

int left = 0;

int right = strlen(str) - 1;

while (left < right) {

// Пропускаем не-буквы

while (left < right && !isalnum(str[left])) left++;

while (left < right && !isalnum(str[right])) right--;

if (tolower(str[left]) != tolower(str[right])) {

return FALSE; // Если символы не равны, строка не палиндром

}

left++;

right--;

}

return TRUE; // Если все проверки пройдены, строка палиндром

}

//Для Unicode

int countwords(const wchar\_t\* str) {

if (str == NULL) return 0;

bool inWord = false;

int wordCount = 0;

while (\*str != L'\0') {

if (iswalnum(\*str)) {

if (!inWord) {

wordCount++;

inWord = true;

}

}

else {

inWord = false;

}

str++;

}

return wordCount;

}

void reversestring(wchar\_t\* str) {

if (!str) {

printError(ERROR\_INVALID\_PARAMETER);

return;

}

int length = wcslen(str);

for (int i = 0; i < length / 2; i++) {

swap(str[i], str[length - 1 - i]);

}

}

BOOL isstringpalindrome(const wchar\_t\* str) {

if (!str) return FALSE;

int left = 0;

int right = wcslen(str) - 1;

while (left < right) {

while (left < right && !iswalnum(str[left])) left++;

while (left < right && !iswalnum(str[right])) right--;

if (towlower(str[left]) != towlower(str[right])) {

return FALSE;

}

left++;

right--;

}

return TRUE;

}

//Вывод ошибки

void printError(DWORD errorCode) {

LPSTR errorMsg = nullptr;

FormatMessageA(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL,

errorCode,

0,

(LPSTR)&errorMsg,

0,

NULL

);

cout << "Ошибка " << errorCode << ": " << errorMsg << endl;

LocalFree(errorMsg);

}

int main()

{

setlocale(LC\_ALL,"RU");

//Работа ANSI-функций

cout << "=== ANSI Functions Demo ===" << endl;

//Тестовая строка

char ansiStr[] = "а роза упала на лапу Азора";

cout << "Original ANSI string: " << ansiStr <<endl;

// Подсчет слов

cout << "Количество слов: " << countwords(ansiStr) << endl;

//Переворот строки

reversestring(ansiStr);

cout << "После переворота: " << ansiStr << endl;

//Палидром

cout << "Строка является палиндромом? "<< (isstringpalindrome(ansiStr) ? "YES" : "NO") << endl;

//Работа Unicode-функций

wcout << L"\n=== Unicode Functions Demo ===" << endl;

//Тестовая строка

wchar\_t unicodeStr[] = L"Привет, мир! Hello, world!";

wcout << L"Original UNICODE string: " << unicodeStr << endl;

// Подсчет слов

wcout << L"Количество слов: " << countwords(unicodeStr) << endl;

//Переворот строки

reversestring(unicodeStr);

wcout << L"После переворота: '" << unicodeStr << L"'" << endl;

//Палидром

wcout << "Строка является палиндромом? " << (isstringpalindrome(unicodeStr) ? "YES" : "NO") << endl;

cout << "\n=== Testing Error Handling ===" << endl;

reversestring((char\*)nullptr);

reversestring((wchar\_t\*)nullptr);

return 0;

}

**Результат:**

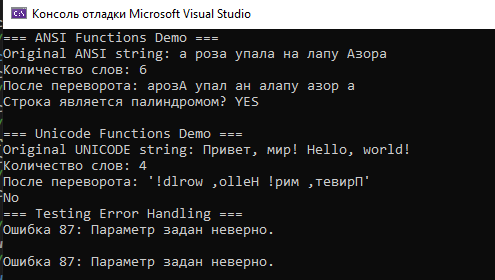


Рисунок 1 - Вывод на консоль

**Контрольные вопросы**

1. В чем принципиальное отличие кодировок ANSI и Unicode?

* **ANSI** — это один язык для одного символа. Каждый символ (буква, цифра) кодируется одним байтом. Из-за этого в ANSI можно использовать только 256 разных символов. Этого хватает для английского и некоторых европейских языков, но невозможно одновременно написать, например, по-русски и по-арабски.
* **Unicode** — это все языки мира для одного символа. Каждый символ кодируется двумя или более байтами. Это позволяет иметь десятки тысяч уникальных символов: кириллица, иероглифы, эмодзи, математические символы — все что угодно в одной строке.

2. Для чего нужны макросы TCHAR, LPCTSTR, TEXT()?

**Их главная цель —** упростить жизнь программисту. Они позволяют написать один исходный код, который можно без изменений скомпилировать как в версию под Unicode, так и в версию под ANSI.

**Как они работают:**

* **TCHAR** — это "умный" символ. Если в настройках проекта указано "Use Unicode", то TCHAR превращается в wchar\_t (широкий символ, 2 байта). Если нет — в обычный char (1 байт).
* **LPCTSTR** — это "умная" строка. Указатель на строку из константных (неизменяемых) символов TCHAR.
* **TEXT()** — это "умный" строковой литерал. Он превращает текст в кавычках, например, TEXT("Привет"), либо в Unicode-строку, либо в ANSI-строку, в зависимости от настроек компиляции.

**TCHAR**: Условный тип символа.

Если в настройках проекта определена директива UNICODE (или \_UNICODE), TCHAR раскрывается в wchar\_t (широкий символ, 2 байта). Если директива не определена, TCHAR раскрывается в char (ANSI-символ, 1 байт).

Для чего: Объявлять символьные буферы и строки, которые автоматически станут правильного типа.

**LPSTR / LPCSTR:** Условный тип указателя на ANSI-строку (char\*/const char\*).

**TEXT()** используется для условных строковых литералов.

Превращает строку в кавычках в нужный формат.  
TEXT("Hello") в Unicode-сборке становится L"Hello", а в ANSI-сборке — просто "Hello".

Для чего: Чтобы строковые литералы в коде были совместимы с выбранным режимом сборки.

3. Какая функция WinAPI используется для преобразования Unicode-строки в ANSI-строку? Опишите ее основные параметры.

WideCharToMultiByte: Преобразует Unicode-строку в ANSI-строку (или в строку в любой другой кодовой странице, например, UTF-8).

Параметры:

1. UINT CodePage [Входной]: Кодовая страница, в которую нужно преобразовать.
2. DWORD dwFlags [Входной]: Флаги, указывающие тип преобразования. Функция выполняется быстрее, если ни один из этих флагов не задан. Обычно 0.
3. LPCWSTR lpWideCharStr [Входной]: Указатель на исходную Unicode-строку (wchar\_t\*), которую нужно преобразовать.
4. int cchWideChar [Входной]: Длина исходной строки в символах. Если передать -1, функция сама вычислит длину (строка должна быть нуль-терминированной).
5. LPSTR lpMultiByteStr [Выходной]: Указатель на буфер, который примет результат (ANSI-строку).
6. int cbMultiByte [Входной]: Размер выходного буфера (lpMultiByteStr) в байтах.
7. LPCCH lpDefaultChar [Входной]: Указатель на символ-заменитель. Если символ нельзя представить в выбранной кодовой странице, он будет заменен на этот. Если NULL, используется системный умолчальный символ (обычно '?').
8. LPBOOL lpUsedDefaultChar [Выходной]: Флаг, который функция устанавливает в TRUE, если при преобразовании хотя бы один символ был заменен на умолчальный. Может быть NULL.

4. Как в программе на Win32 получить текстовое описание последней произошедшей ошибки?

Функция GetLastError(): Возвращает код последней ошибки, возникшей при вызове API-функции.

Функция FormatMessage(): Позволяет преобразовать код ошибки, полученный от GetLastError(), в читаемое текстовое сообщение. Это основной инструмент для диагностики проблем.

5. Почему для современных приложений рекомендуется использовать Unicode?

Unicode — это единый стандарт, который поддерживает все языки мира одновременно, устраняет проблемы с несовместимыми кодовыми страницами, обеспечивает лучшую производительность в современных ОС (которые внутри используют Unicode) и является обязательным требованием для глобализации приложений.

1. Глобализация (Globalization): Unicode поддерживает все языки мира. Ваше приложение сможет корректно работать с текстом на английском, китайском, арабском, русском и любом другом языке одновременно, без переключения кодых страниц.
2. Единая кодовая база в ОС: Современные версии Windows (начиная с Windows 2000) являются Unicode-системами изнутри. Все внутренние API используют UTF-16. Когда вы вызываете ANSI-версию функции (например, CreateWindowA), система сначала преобразует ваши строки в Unicode, вызывает внутреннюю Unicode-версию (CreateWindowW), а затем при необходимости преобразует результаты обратно. Это лишние накладные расходы.
3. Надежность: Исключаются ошибки, связанные с несовпадением кодых страниц (так называемый "кракозябры"). Символ с определенным кодом в Unicode всегда означает один и тот же символ, независимо от локали системы.
4. Упрощение разработки: Нет необходимости беспокоиться о переключении кодых страниц и использовать условные макросы вроде TCHAR. Код становится чище и проще.
5. Совместимость: Это стандарт де-факто для веба (UTF-8), мобильных ОС (Android, iOS), macOS и Linux. Использование Unicode обеспечивает лучшую совместимость вашего приложения с другими платформами и технологиями.