**Пояснения к лабораторной 2**

**Общие пояснения для всех вариантов**

Тип проекта для **всех вариантов** — **"Консольное приложение"** в Visual Studio. Весь ввод и вывод осуществляется через std::cout, std::wcout, std::cerr.

**Цель:** Научиться работать с разными кодировками (ANSI и Unicode) в Windows, используя WinAPI.

**Суть:** Для каждой функции нужно написать **две версии**:

* **ANSI-версия (суффикс** \_a**):** Работает с типом char\*.
* **Unicode-версия (суффикс** \_w**):** Работает с типом wchar\_t\*.

**Ключевые моменты:**

* Использование TCHAR, LPCTSTR и макроса TEXT() **не требуется** в этом задании. Вы пишете две явные реализации.
* **Обязательный пункт:** Демонстрация преобразования кодировок (например, с помощью WideCharToMultiByte) и обработки ошибок WinAPI через GetLastError() и FormatMessage().
* Для вывода Unicode-строк (wchar\_t\*) в консоли используйте std::wcout.
* Все функции должны работать с **C-строками** (массивы символов, заканчивающиеся на \0), а не с классом std::string.

**Пояснения по конкретным вариантам**

**Вариант 1: Базовые строковые операции (Исходный)**

Самый простой вариант, фокусируется на основах манипуляции с памятью и символами.

1. addslashes: Нужно не просто вывести строку со слэшами, а **модифицировать исходную строку**. Это требует динамического сдвига символов вправо для вставки \. **Алгоритм:**
   * Пройти по строке, подсчитайте, сколько кавычек нужно экранировать.
   * Рассчитать новую длину.
   * Пройти с конца строки, сдвигая символы вправо и вставляя \ перед каждой кавычкой.
2. ltrim: Не создавать новую строку, а **сдвигать содержимое исходного буфера**. Найти первый непробельный символ и переместить его и все последующие (включая \0) в начало буфера.
3. comparestrings: Просто сравнить длины строк с помощью strlen (для ANSI) и wcslen (для Unicode). Не путать с лексикографическим сравнением (strcmp).

**Вариант 2: Базовые строковые операции**

Похож на Вариант 1, но с другими функциями.

1. mystrlwr: Реализовать аналог стандартной функции strlwr. Пройти по каждому символу строки и применить к нему tolower() (для ANSI) или towlower() (для Unicode).
2. mystrrchr: Реализовать аналог стандартной функции strrchr. Пройти по строке, запоминая указатель на последнее совпадение с искомым символом.
3. isstringsame: Аналог stricmp. Сравнивать строки посимвольно, приводя каждый символ к нижнему регистру. Если длины строк разные, можно сразу вернуть FALSE.

**Вариант 3: Обработка и анализ строк**

Более сложный вариант, требующий умения работать с логикой состояния (подсчет слов) и алгоритмами (палиндром).

1. countwords: **Основная сложность** — определение границ слов. Используйте флаг-индикатор того, находитесь ли вы внутри слова. Инкрементируйте счетчик, когда переходите от не-буквы к букве.
2. reversestring: Классический алгоритм. Используйте два указателя (или индекса): один движется с начала, другой с конца строки, и меняйте местами символы, пока они не встретятся в середине.
3. isstringpalindrome: **Алгоритм:**
   * Создать копию строки, приведенную к одному регистру (mystrlwr) и с удаленными пробелами (нужно написать вспомогательную функцию или модифицировать на месте).
   * Применить к этой копии алгоритм reversestring.
   * Сравнить исходную "очищенную" строку с перевернутой. Если они равны — это палиндром.

**Вариант 4: Строки и системные вызовы**

**Ключевой акцент — на обработке ошибок WinAPI.** Функции не изобретаются, а являются обертками вокруг системных вызовов.

1. getdrivesinfo**,** getusername**,** getsystemdirectory: Структура у всех функций одинаковая:
   * Вызвать соответствующую функцию WinAPI (GetLogicalDriveStringsA/W, GetUserNameA/W, GetSystemDirectoryA/W).
   * **Проверить ее возвращаемое значение.** Если функция вернула FALSE (или 0) — вызвать GetLastError.
   * Получить текст ошибки через FormatMessage и вывести его.
   * Если вызов успешен — скопировать результат в переданный буфер outputBuffer.
   * **Внимание:** Функция GetLogicalDriveStrings возвращает все диски в виде одной длинной строки, где имена дисков разделены \0, а вся строка завершается двойным \0. Ее нужно красиво распарсить и сформировать итоговую строку с пробелами.

**Вариант 5: "Умные" строковые операции (повышенная сложность)**

Самый сложный и интересный вариант, требует работы с динамической памятью или сложными алгоритмами.

1. strreplace: **Главная challenge** этой лабораторной.
   * Нужно учесть, что новая подстрока может быть длиннее или короче старой.
   * **Алгоритм:** Найти все вхождения oldSubstr, подсчитать итоговую длину новой строки. Если она не помещается в bufferSize — вернуть ошибку.
   * Создать временный буфер для новой строки.
   * Пройти по исходной строке, копируя символы в новый буфер. При нахождении oldSubstr скопировать в буфер newSubstr.
   * Скопировать результат из временного буфера в исходный str.
2. parsecsventry: Требуется парсинг с учетом состояния (находится ли парсер внутри кавычек или нет).
   * Реализуйте простой конечный автомат.
   * Состояние 1: Вне кавычек. Запятая означает конец поля.
   * Состояние 2: Внутри кавычек. Запятая игнорируется, кавычка может быть escape-символом или закрывающей.
3. getstringencoding: **Эвристический анализ.**
   * **Проверка на UTF-16LE (Unicode):** Если первые два байта — 0xFF 0xFE (BOM), или если строка wchar\_t\* (здесь нужно проанализировать сам массив байт).
   * **Проверка на UTF-8:** Можно проверить на наличие BOM (0xEF 0xBB 0xBF) или искать валидные UTF-8 последовательности (байты в диапазонах 0xC0-0xDF, за которыми следует байт 0x80-0xBF и т.д.).
   * **По умолчанию:** Если ничего не подошло, вернуть ANSI.