2024.04.18

# Требования и рекомендации по заданию 3 (1 курс, 2 семестр) Обработка текстов

Смысл задания — анализ и обработка текстового файла. Как правило, это означает получить на входе файл, в которам записан некоторый текст, и на выходе выдать новый файл, который будет содержать новый, преобразованный текст.

Можно выделить два основных подхода к преобразованию текста — посимвольный и построчный.

Посимвольный подход означает, что мы последовательно читаем по одному символу из входного файла, анализируем этот символ, и выдаем на выход символы, соответствующие смыслу решаемой задачи. Достоинство подхода — не нужно заботиться о размерах текста, фактически это обработка последовательности символов (как ранее рассматривались алгоритмы обработки последовательности чисел). Недостаток — не все задачи можно решить при таком подходе.

Построчный подход означает, что программа читает из файла целую строку, получает тем самым массив символов, преобразует нужным образом этот массив символов, и выдает на выход новую преобразованную строку текста. В данном случае необходимо размещать прочитанную строку в памяти и нужно как-то разумно выделять для этого место. Зато, получив массив символов, мы можем многократно анализировать и преобразовывать этот массив, пока не дойдем до требуемого состояния.

Отдельной проблемой является кодировка символов. Имеется мнодество кодировок, среди которых наиболее распространены cp1251 (Windows), cp866 (DOS), utf-8 (Linux, Windows) и другие. В простейшем случае один символ кодируется одним байтом (cp1251, cp866). Таким образом мы имеем однозначное соответствие порядкового номера символа в строке и индекса байта в массиве. В кодировке utf-8 используется различное количество байт на символ, например, для латинских букв, цифр, знаков используется 1 байт, а для русских букв — 2 байта. Этот факт уже усложняет обработку текстов с наличием русских букв. Для простоты мы пока не будем разбираться с различным числом байтов на символ, т.е. можно считать, что в файле нет русских букв, и соответственно все упомянутые кодировки дают 1 байт на символ.

Для обработки текстов в C программах можно использовать функции стандартной библиотеки clib. Эти функции объявлены в заголовочном файле string.h. Все эти функции предполагают, что при хранении в памяти строка текста должна оканчиваться на нулевой байт (байт со значением 0). Этот нулевой символ является признаком конца строки и должен всегда присутствовать в байтовм массиве. Таким образом, для хранения строки из $k$ символов в памяти нужен байтовый массив не короче $k + 1$ байт (длиннее можно). Если нулевого завершающего байта не будет, то функции, проходя по массиву от начала строки к ее концу, не остановятся, и будут обращаться к последующим байтам памяти пока не дойдут до 0. Но это в явном виде есть выход за границы массива со всеми сопутствующими последствиями (ошибки, падение программы и т.п.). Библиотечные функции записывают этот 0 сами, но если вы самостоятельно формируете строку из символов, но этот 0 тоже надо записывать самому.

Настоятельно советую познакомиться с библиотечными функциями из string.h по справочной литературе. Многие из них оказываются очень полезными при решении ваших задач.

В языке C++ для работы со строками вводится специальный класс string. Использование этого класса в упрощает работу в том плане, что нам не нужно уже заботиться о выделении памяти для размещения строки, эта работа выполняется автоматически. Но автоматизация в размещении памяти выливается в замежлении работы программы. На наших примерах этого заметно не будет, но в общем случае это следует иметь в виду. В других аспектах работа со string во многом аналогична работе с байтовым массивом и функциями string.h. Аналогично, стоит познакомиться с возможностями класса string по справойной литературе.

Файл read-text.zip содержит примеры для решения простейшей задачи в стиле C и стиле

C++ и с использованием построчного и посимвольного подходов. Рекомендуется посмотреть эти примеры и понять как они работают.

## Комментарии к отдельным задачам

**1. Перекодировка.** Посимвольный подход. Заранее соствляются таблицы кодов, и прочитанный код заменяется на код из таблицы.

**2. Сократить одинаковые символы.** Посимвольный подход. Запоминаем предыдущий символ и сравниваем его с текущим. Если одинаковые — не печатаем.

**3. Замена слов.** Построчный подход. В строке ищется образец. Далее выводится начальная часть строки, потом слово-замена, и точно так же далее обрабатывается оставшаяся часть строки.

**4. "Указаталь"** Построчный подход. Читаем строки и считаем их количество. Ищем слово в строке. Печатаем номер строки и номер найденной позиции. Не забыть, что слово может встретиться в строке несколько раз.

**5. Статистика текста.** Посимвольный подход. Заводим массив длиной 256 — количество появлений каждого символа, инициализируем нулями. При чтении символа, увеличиваем соответствующий элемент массива. Для длины слов отслеживаем переход последовательности с пробельных символов на непробельные и наборот (т.е. начало и конец слова) и считаем количество символов между этими переходами.

**6. Словарь.** Библиотечные функции позволяют прочитать одно слово — группу непробельных символов. Прочитанные слова можно сохранить в массиве. (Т.е. реализовать массив срок или массив байтовых массивов). Нвдо разумно задать длину этого массива (и контролировать возможное переполнение). Далее можно отсортировать этот массив любой понятной библиотечной функцией и при печати не печатать повторяющиеся слова.

**7. Различие файлов.** Правильное решение — алгоритм нахождения наибольшей общей подпоследовательности (LCS). Это немного сложновато. Так как у нас предполагается, что измениний немного, то считываем построчно оба файла в память и ишем наиболее длинный совпадающий кусок. Потом от этого куска проверям строки назад и вперед п файлу, отмечая какие появились, а какие были удаленыю

**8. Ввод из таблицы.** Прежде всего надо понять какие слова и в каких позициях записаны в рамках строки. Далее можно определить какие позиции соответствуют данным какого столбца. После этого можно прочитать содержимое строк и соотнести его с коэффициентыми элемента в матричном представлении. Также надо уметь преобразовывать запись числа в виде набора символов в арифметическое значение этого числа.

**9. Подрезка строк.** Заводим буфер заданной длины. Считываем в этот буфер символы из файла, пока хватает места. Далее ищем в буфере пробельные символы с конца буфера и выводим текст до этой позиции. Оставшуюся часть буфера сдвигаем к началу и продолжаем его заполнять из файла. Надо понимать, что в окрестности позиции разделения строки может быть несколько пробельных символов, и их не надо переносить в выводимый файл. Надо принять волевое редение как быть, если в буфере нет вообще пробельных символов.

**10. Удаление комментариев.** Построчный подход. Ищем в строке начальную "скобку". Если не нашли — печатаем строку. Если нашли, выводим начальную часть строки и устанавливаем режим "комментария". ищем далее закрывающую скобку, но ничего не печатаем. Если нашли закрывающую скобку, то отключаем "режим комментария" и начинаем искать открывающую и печататьвсе, что перед ней. Особый случай, если открывающая встретилась, а закрывающей для нее нет.

**12. Реализовать include.** Считаем, что запись include заниает целиком одну строку. Построчный подход. Читаем очередную строку и, если она не содержат include, выводи ее в выходной файл. Если встречается include, то извлекаем из этой строки имя файла, открываем этот файл и рекурсивно вызываем для него ту же самую функцию обработки. Есть проблема зацикливания, когда последующие файлы ссылабтся на уже открытые предыдущие. Нужно

вести список имен открытых в данный момент файлов. При открытиии нового файла его имя добавляется в этот список, а при окончании обработки файла — удаляется. Если include содердит имя файла, уже зарегистрированного в списке, то это ошибка зацикливания, можно выдать сообщение об ошибке и не включать этот файл в обработку.

**12. Реализовать define и т.д.** Поддерживаем множество имя-замена, определяемых строками define и undef. Меняем это мноежство по мере появления инструкций define и undef. В текущей строке ищем имена из данного множества и выполняем замену.

**13. Реализовать ifdef и т.д.** Читаем по строкам и сохраняем имена, которые встречаются в этих конструкциях. Каждая прочитанная строка либо содержит определение или отмену некоторого имени, либо условие для печати последующих строки, либо содержателную строку, котору. придется либо напечатать, либо пропустить. Таким образом, надо поддерживать два можества: 1) какие имена нужны для печати текущей строки, 2) какием имена определны в данный момент. Можество 1 обнodляется при появлении конструкций ifdef, else, endif, множество 2 обнвляется при появлкнии констукций define, undef. При считавании очереной содержательной строки алализируются множества 1 и 2 и принмается рещение печатать строку или нет.

**14. Форматирование абзаца** Аналогично задаче 9, но пробелы между словами сначала сокращаюся до одного символа. Если выводимая строка оказывается короче, чем требуемая длина абзацаЮ то пробелы между словами равномерно увеличиваются до выравнивания длины строки по заданного размера. К последней строке абзаца это не применяется, она созраняет одиночные пробелы между словами. В исходном файле должно быть как-то помечено когда начаинать и когда заканчивать собирать исходнве строки в абзац. Например, признаком конца абзаца может служить пустая строка.