МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

Курсовая работа
по дисциплине «Программирование»
Тема: Обработка строк.

Студент гр. 0383 _____ Шквиря Е.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Шквиря Е.В.

Группа: 0383

Тема работы: обработка строк.

Исходные данные: программа должна сохранять текст, введённый пользователем, используя динамическую память и структуры. Обработка текста должна выполняться, по возможности, с помощью функций из стандартных библиотек языка Си. Программа должна быть разделена на несколько модулей и собираться при помощи Makefile.

Содержание пояснительной записки:
«Содержание», «Введение», «Задание работы», «Ход выполнения работы» «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания: 2.11.2020
Дата сдачи реферата: 25.12.2020
Дата защиты реферата: 26.12.2020

Студент гр. 0383 _____ Шквиря Е.В.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

В курсовой работе была реализована обработка текста произвольной длины, для этого были использованы динамические массивы и структуры. Также в работе, по возможности, использовались стандартные библиотеки языка Си, содержащиеся в них функции и типы данных (например, *wchar_t*). В работе программы была реализована защита от некорректных запросов или неудачного выделения или перевыделения памяти в ходе работы. О каждой из ошибок выводится соответствующее сообщение. В программе реализован элементарный интерфейс общения с пользователем и выполнение запрашиваемых им действий.

Пример работы программы приведён в приложении А.

Исходный код программы привежён в приложении Б.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: создать стабильно работающую программу, производящую выбранную пользователем обработку поданного текста. Реализация программы должна содержать работу со структурами (для хранения текста и отдельных предложений, а также дополнительных данных), работу с динамически выделенной памятью и использование стандартных библиотек, в том числе для работы с национальным алфавитом (wchar.h).

Разработка велась на базе операционной системы Linux Ubuntu в среде разработки CLion. Отладка и тестирование программы производились при помощи отладчика gdb и системы тестирования Valgrind.

В результате была разработана программа, считывающая вводимый пользователем текст, выводящая меню со списком доступных функций и выполняющая выбранные. Если пользователю нужна информация о тексте, то в случае корректного выполнения действий программа выводит результат, иначе появится сообщение об ошибке. Если пользователю нужно изменить текст, то при корректных данных и достаточном количестве памяти программа выведет результат «Выполнено!», и текст, при необходимости, можно будет вывести с помощью специального пункта в меню. Также при выходе их программы производятся действия по очистке динамически выделенной памяти и корректному завершению работы.

1. ЗАДАНИЕ

Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой) Длина текста и каждого предложения заранее не известна.

Для хранения предложения и для хранения текста требуется реализовать структуры Sentence и Text.

Программа должна сохранить (считать) текст в виде динамического массива предложений и оперировать далее только с ним. Функции обработки также должны принимать на вход либо текст (Text), либо предложение (Sentence).

Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).

Далее, программа должна запрашивать у пользователя одно из следующих доступных действий (программа должна печатать для этого подсказку. Также следует предусмотреть возможность выхода из программы):

1) Сделать сдвиг слов в предложении на положительное целое число N. Например, предложение “abc b#c ИЙ два” при N = 2 должно принять вид “ИЙ два abc b#c”.
2) Вывести все уникальные кириллические и латинские символы в тексте.
3) Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д..
4) Удалить все слова, которые заканчиваются на заглавный символ.

Все сортировки и операции со строками должны осуществляться с использованием функций стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Также, должен быть написан Makefile.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Создание структур и заголовочных файлов.

Для работы с текстом были созданы три структуры: Text, Sentense и Word. Для удобства при объявлении при помощи оператора *typedef* они были определены как новые типы данных Text, Sentence и Word соответственно. Структуры были объявлены в заголовочных файлах, которые названы соответственно. В файле, содержащем точку входа программы, была подключена препроцессорная директива *pragma once* для избежание повторного объявления. Так как Text работает с данными типа Sentence и Word, и Sentence с данными Word, то во избежании повторного включения файлов — все необходимые стандартные библиотеки языка подключались к файлу word.h, а он, в свою очередь, уже подключался к sentence.h, который, в свою очередь, подключался к text.h.

Содержание структур: в Text содержатся переменные – счётчик количества предложений, объём выделенной памяти под предложения и динамический массив указателей на структуру Sentense. В Sentence содержатся переменные — счётчик количества слов в предложении, объём выделенной памяти под хранение слов и динамический массив указателей на структуру Word. В Word содержатся переменные — счётчик количества символов в слове, объём выделенной памяти под хранение символов слова и динамический массив указателей на *wchar_t*.

2.2. Чтение текста и сохранение его в структуру Text.

В файле text.c содержится функция *input_text* для считывания текста из стандартного потока ввода. Она формирует начальные инициализации для каждого предложения и для каждого слова. Функции начальной инициализации структур имеют тип возвращаемого значения *int* и возвращают значения *SOME_ERROR* в случае возникновения какой-либо ошибки, а также выводят сообщения об этом пользователю. Если функция завершилась

успешно, то возвращается значение *ALL_OK*. Считывание происходит следующим образом: для каждого предложения создаётся структура Sentence (*sntc*), а также для каждого слова создаются две структуры Word (*word* и *sepWord*). В *word* попадают слова, а в *sepWord* — разделители. После этого каждое слово и разделитель добавляются в *sntc*, которое добавляется в *text* после того, как встретится точка. В случае, когда заранее выделенной памяти не хватает, каждая функция *push_back* перевыделяет память с использованием временной переменной. Таким образом, каждое предложение добавляется в текст. Символы, являющиеся разделителями слов, хранятся в Sentence неразрывно со словами и имеют всегда нечётные позиции (если считать с 0), что позволяет явно отличить их от обычных слов. Признаком конца ввода считается два переноса строки.

2.3. Начальная обработка текста.

В файле text.c реализована функция *delete_dubl*, которая сравнивает предложения без учёта регистра и удаляет повторяющиеся. Важным замечанием является то, что она также учитывает разделители, в том числе и после точки. Для определённости, всегда остаётся первое вхождение предложения. Их удаление происходит путём последовательного сдвига в конец текста с помощью функции *swap* и дальнейшего освобождения памяти, выделенной под них. Для удобства сравнения слов и предложений друг с другом реализованы функции *is_equal*, но сами же слова сравниваются с помощью функции *wcscasecmp* из стандартной библиотеки языка. Так как все действия происходят непосредственно с текстом, то после обработки его можно вывести с помощью номера специальной команды, которая выполнит функцию *print_text*.

2.4. Функции для взаимодействия с пользователем.

Изначально при входе в программу для пользователя выводится сообщение, что нужно ввести текст и что признаком конца строки является

два переноса строки. После ввода происходит предобработка текста, во время которой удаляются повторяющиеся предложений. Если на момент выполнения программы достаточно памяти для сохранения текста и не произошло ошибок, то выполнится функция *print_interface*, которая выведет пользователю список из доступных операций. Среди них есть 4 функции, необходимых по условию, а также дополнительные для удобства использования: вывод текста на экран, очистка терминала и выход из программы. После этого вызывается функция *do_task*, которая при помощи конструкции *switch* выполняет запрашиваемую операцию или сигнализирует об ошибке. Если пользователь не выразил желания выйти из программы, то функция вернёт значение *WORKING*, в противном же случае — *EXIT*. При возвращаемом значении *WORKING* цикл *while* снова будет выводить интерфейс программы и запрашивать у пользователя дальнейшие действия, иначе цикл остановится, выделенная под текст память очистится (вызовется функция *delete_all*) и программа завершится.

2.5. Функции для выполнения указанных операций.

Для выполнения каждой подзадачи из задания были реализованы функции. Они принимают в качестве аргумента указатель на *Text* или на *Sentence*, в зависимости от того, с чем они работают, а также дополнительные данные.

2.5.1 *Сдвиг слов в предложении на положительное целое число N.*

Так как эта операция выполняется над предложением, то её реализация хранится в файле sentense.c. Функция носит название *move_word_n*. Её выполнение происходит в такой последовательности:

1. Убираются лишние круги для n, которые не повлияют на результат.
2. Заводится массив *used*, в котором будут отмечаться уже перемещённые слова.

3. Последовательно, начиная с конца, перебираются не перемещённые слова и сдвигаются на n. Изначально запоминается слово (*swapWord*), на его место перемещается другое, которое на n смещений позади него, считая по кругу. Когда мы приходим к ситуации, что нам нужно переместить слово из позиции, в которой мы начали выполнение цикла *do while*, то мы записываем первое слово (*swapWord*).
4. Таким образом, перемещаются все слова на *n* позиций вперёд, и под конец выполнения функции освобождается память, выделенная под массив *used*.
5. Если во время выполнения функции произошла ошибка, то возвращается значение *SOME_ERROR*, иначе *ALL_OK*.

2.5.2 Вывод всех уникальных кириллических и латинских символов в тексте.

Так как эта операция выполняется над текстом, то её реализация хранится в файле text.c. Функция носит название *unique_symb*. Её выполнение происходит в такой последовательности:

1. Отмечаются буквы, которые встречаются в тексте, чтобы понимать, сколько из них уникальных. Так как буква в верхнем и нижнем регистре считается неразличимой, то в специальную переменную *symb* записывается её символ в нижнем регистре. Также для правильности ответа был обработан случай встречи буквы Ё в тексте.
2. После того, как был вычислен размер итогового массива, под него выделяется память с помощью функции *malloc*. В случае невозможности выделения достаточного количества памяти, возвращается значение *NULL*.
3. В массив *res* добавляются буквы в нижнем регистре. Из-за особенности реализации они идут в лексикографическом порядке, причём первыми идут кириллические символы, а после них латинские.

4. В случае отсутствия ошибок функция записывает по указателю на размер массива значение переменной *size*, которая хранит количество уникальных символов. Сама же функция возвращает указатель на массив *res*.

2.5.3 Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д..

Так как эта операция выполняется над текстом, то её реализация хранится в файле text.c. Функция носит название *unique_len_word*. Её выполнение происходит в такой последовательности:

1. Изначально вычисляется максимальная длина слова (слова-разделители не учитываются).
2. После того, как был вычислен размер итогового массива, под него выделяется память с помощью функции *calloc*. В случае невозможности выделения достаточного количества памяти, возвращается значение значение *NULL*.
3. В массиве *res* содержится количество длин каждого слова. Для экономии памяти максимальная длина слова, которая попадёт в массив, совпадает размером массива.
4. В случае отсутствия ошибок функция записывает по указателю на размер массива значение переменной *size*, которая хранит количество уникальных символов. Сама же функция возвращает указатель на массив *res*.

2.5.4 Удаление всех слов, которые заканчиваются на заглавный символ.

Так как эта операция выполняется над предложениями, то её реализация хранится в файле sentense.c. Функция носит название *rm_word_last_cptlz*. Её выполнение происходит в такой последовательности:

1. С помощью цикла *for* проверяются последние буквы всех слов. В случае, если такое слово стоит в начале предложения, то оно удаляется вместе с его последующим разделяющим словом.
2. В случае, если такое слово стоит не в начале, то изначально удаляется слово, а его следующее разделяющее слово сливается с предшествующим разделяющим.
3. Таким образом, система, где разделяющие слова всегда идут после обычных, не ломается.

2.5.5 *Промежуточный вывод текста*

В случае, если результатом выполнения функции является обработанный текст, то выводится лишь сообщение «Выполнено!». Пользователь, при желании, может вывести текст благодаря 8 пункту интерфейса. В случае, если результатом выполнения функции является информация о тексте, то она выводится на экран сразу же после выполнения функции. Если же во время выполнения функции произошла ошибка, то пользователь также будет об этом извещён.

2.6. Очистка памяти и завершение работы.

После выбора пользователем пункта, завершающего программу, происходит выход из цикла, обеспечивающего вывод интерфейса и считывания команд. В дальнейшем функция *delete_all* очищает выделенную память и на этом программа завершается. В случае, если во время считывания текста происходит ошибка, то программа также очищает выделенную память и аварийно завершается.

2.7. Создание Makefile.

После создания всех файлов с исходным кодом и заголовочных файлов, для облегчения процесса сборки проекта был создан специальный файл Makefile, в котором были прописаны зависимости компиляции и линковки

программы. Для каждого объектного файла была прописана зависимость от соответствующего исходника. В конечном итоге, выполняется сборка проекта и создание исполняемого файла. После сборки проекта объектные файлы удаляются.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы были применены различные приёмы работы со структурами, динамической памятью и стандартными функциями библиотек языка Си. Была создана стабильно работающая программа с обработкой от нестандартных ситуаций (например, ввода некорректных данных) и выполняющая функции, указанные в задании и дополнительные для удобства использования. Программа осуществляет взаимодействие с пользователем через консольный интерфейс и корректно обрабатывает различные виды некорректных входных данных (в соответствии с условием). По окончанию работы программы происходит освобождение выделенной памяти.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.

ПРИЛОЖЕНИЕ А

ПРИМЕР РАБОТЫ ПРОГРАММЫ

Пример вывода приветствия пользователя и ввод им текста:

```
Введите текст(два переноса строки являются признаком конца ввода текста)
Привет, я уже на месте. Надеюсь, что ты скоро придёшь.
Не переживай, я через 2 минуты буду.

Доступные команды:
1)Сделать сдвиг на N слов вперёд
2)Вывести все уникальные кириллические и латинские символы
3)Посчитать количество слов с длиной 1,2,3...
4)Удалить все слова, оканчивающиеся на заглавный символ
8)Вывести текст
9)Очистить терминал
0)Выйти из программы
Введите номер команды...
```

Пример вывода прощания с пользователем и результата:

```
...
Введите номер команды...
8
Привет, я уже на месте. Надеюсь, что ты скоро придёшь.
Не переживай, я через 2 минуты буду.

Доступные команды:
1)Сделать сдвиг на N слов вперёд
2)Вывести все уникальные кириллические и латинские символы
3)Посчитать количество слов с длиной 1,2,3...
4)Удалить все слова, оканчивающиеся на заглавный символ
8)Вывести текст
9)Очистить терминал
0)Выйти из программы
Введите номер команды...
0
Работа завершена!
```

Пример обработки текста (все команды тестировались с исходным текстом):

Текст:

```
Петя, я тебя нигде не ВИЖУ. Я уже ТУТ. Вижу ЕЁ.
Pite, I`m here. I see him. I sEE Him. Too.
```

Обработка команд:

```
Введите номер команды...
1
Введите номер предложения (1, 2, ...) и смещение:
1 2
Выполнено!
…
8)Вывести текст
…
Введите номер команды...
8
не, ВИЖУ Петя я тебя нигде. Я уже ТУТ. Вижу ЕЁ.
Pite, I`m here. I see him. Too.
```

```
Введите номер команды...
2
Выполнено!
б в г д е ё ж и н п т у я e h i m o p r s t
```

```
Введите номер команды...
3
Выполнено!
1) 3
2) 2
3) 6
4) 6
5) 1
```

```
Введите номер команды...
4
Выполнено!
…
8)Вывести текст
…
Введите номер команды...
8
Петя, я тебя нигде не . уже . Вижу .
Pite, I`m here. see him. Too.
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: start.c

```
#pragma once

#include "text.h"
#include <stdio.h>
#include <wchar.h>
#include <locale.h>


int do_task(Text *text);
void print_interface();

//состояния работы программы
#define EXIT 0
#define WORKING 1

//возможные завершения программы
enum EXITS
{
	GOOD_EXIT,
	BAD_EXIT
};

//перечисление операций
enum TASKS
{
	MOVE_WORD = 1,
	UNIQ_SYMB,//заполняются последовательно
	CNT_WORD_WITH_LEN,
	RM_WORD_LAST_CAPIT,
	OUT_TEXT = 8,
```

```
     CLEAR_TERM = 9,
     STOP_WORK = 0,
};

//точка запуска программы
int main()
{
     setlocale(LC_ALL, "");
     Text text;
     wprintf(L"Введите текст(два переноса строки являются
признаком конца ввода текста)\n");
     if (!initial_text(&text, 2) || !input_text(&text))
     {
          delete_all(&text);
          return BAD_EXIT;
     }
     delete_dubl(&text);
     int working = 1;
     while (working)
     {
          print_interface();
          working = do_task(&text);
     }
     delete_all(&text);
     return GOOD_EXIT;
}

//выполнить опцию
int do_task(Text *text)
{
     int task = 0;
     wscanf(L"%d", &task);
     switch (task)
     {
          case MOVE_WORD:
```

```
          {
               wprintf(L"Введите номер предложения (1, 2, ...) и
смещение:\n");
               int ind, n;
               wscanf(L"%d%d", &ind, &n);
               ind--;
               if (0 <= ind && ind <= text->size - 1 && n >= 0)
               {
                    if(move_word_n(&text->sntcs[ind], n) ==
ALL_OK)
                         wprintf(L"Выполнено!\n");
                    else
                         fwprintf(stderr, L"%sОшибка памяти при
исполнении функции!!%s\n", ERROR_CLR, STD_CLR);
               }
               else
                    wprintf(L"%sНеверный индекс или смещение!%s\
n", ERROR_CLR, STD_CLR);
               break;
          }
          case UNIQ_SYMB:
          {
               wchar_t *res;
               int size = 0;
               res = unique_symb(*text, &size);
               if (res != NULL)
               {
                    wprintf(L"Выполнено!\n");
                    for (int i = 0; i < size; ++i)
                    {
                         wprintf(L"%lc ", res[i]);
                    }
                    wprintf(L"\n");
                    free(res);
               }
```

```
                else
                    fwprintf(stderr, L"%sНе получилось выделить
память!!%s\n", ERROR_CLR, STD_CLR);
                break;
            }
            case CNT_WORD_WITH_LEN:
            {
                int size = 0;
                int *len_words = unique_len_word(*text, &size);
                if (len_words != NULL)
                {
                    wprintf(L"Выполнено!\n");
                    for (int i = 0; i < size; ++i)
                        wprintf(L"%d) %d\n", i + 1,
len_words[i]);
                    free(len_words);
                }
                else
                    fwprintf(stderr, L"%sНе получилось выделить
память!!%s\n", ERROR_CLR, STD_CLR);
                break;
            }
            case RM_WORD_LAST_CAPIT:
            {
                for (int i = 0; i < text->size; ++i)
                {
                    rm_word_last_cptlz(&(text->sntcs[i]));
                }
                wprintf(L"Выполнено!\n");
                break;
            }
            case OUT_TEXT:
            {
                print_text(text);
                wprintf(L"\n");
```

```
                break;
            }
            case CLEAR_TERM:
            {
                system("clear");
                break;
            }
            case STOP_WORK:
            {
                wprintf(L"Работа завершена!\n");
                return EXIT;
            }
            default:
                wprintf(L"%sКоманда не распознана!%s\n",
ERROR_CLR, STD_CLR);
        }
        return WORKING;
}

//вывести интерфейс
void print_interface()
{
        wprintf(L"Доступные команды:\n"
                L"1)Сделать сдвиг на N слов вперёд\n"
                L"2)Вывести все уникальные кириллические и
латинские символы\n"
                L"3)Посчитать количество слов с длиной 1,2,3...\n"
                L"4)Удалить все слова, оканчивающиеся на заглавный
символ\n"
                L"8)Вывести текст\n"
                L"9)Очистить терминал\n"
                L"0)Выйти из программы\n"
                L"Введите номер команды...\n");
}
```

Название файла: word.h

```
#include <wchar.h>
#include <wctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
#include <string.h>
#define ERROR_CLR   "\033[0;31m"
#define STD_CLR   "\033[0m"
#define MAX(a,b) (a > b ? a : b)

#define INCREASE 1.6 //коэффициент для увеличения памяти
#define WORD_START_SIZE 4 //начальный размер памяти для слова

//коды ошибок
#define SOME_ERROR 0
#define ALL_OK 1

//варианты равенства
#define NOT_EQUAL 0
#define EQUAL 1

struct Word
{
     wchar_t *word;
     int size;
     int realSize;
};



typedef struct Word Word;

int initial_word(Word* new_word, int start_size);
int push_back_word(Word* word, wchar_t c);
```

```
int is_equal_word(Word* word1, Word* word2);
void swap_word(Word* word1, Word* word2);
int is_sep_symb(wchar_t c);
```

Название файла: word.c

```
#include "word.h"

//начальная инициализация слова
int initial_word(Word *new_word, int start_size)
{
     if (start_size < 0)
     {
          fwprintf(stderr, L"%sОшибка, неверный размер для нового
слова!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     new_word->word = (wchar_t *) malloc(new_word->realSize *
sizeof(wchar_t));
     if (new_word->word == NULL)
     {
          fwprintf(stderr, L"%sНе получилось выделить память для
слова!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     new_word->size = 0;
     new_word->realSize = MAX(start_size + 1, WORD_START_SIZE);//
+1 за счёт \0
     return ALL_OK;
}

//положить назад новое слово
int push_back_word(Word *word, wchar_t c)
{
     //если есть место
     if (word->size + 1 < word->realSize) //str + '\0' < str_max
```

```
    {
        word->word[word->size] = c;
        word->word[++word->size] = L'\0';
        return ALL_OK;
    }

    //если нет места
    wchar_t *tempWord = (wchar_t *) realloc(word->word, (size_t)
(word->realSize * INCREASE) * sizeof(wchar_t));
    if (tempWord == NULL)
    {
        fwprintf(stderr, L"%sНе получилось выделить память для
слова!!%s\n", ERROR_CLR, STD_CLR);
        return SOME_ERROR;
    }
    word->word = tempWord;
    word->realSize = (size_t) (word->realSize * INCREASE);
    word->word[word->size] = c;
    word->word[++word->size] = L'\0';
    return ALL_OK;
}

//проверка на равенство слов
int is_equal_word(Word *word1, Word *word2)
{
    if(word1->size != word2->size || wcscasecmp(word1->word,
word2->word) != 0)
        return NOT_EQUAL;
    else
        return EQUAL;
}

//обмен значениями двух слов
void swap_word(Word *word1, Word *word2)
{
```

```
    Word tempWord = *word1;
    *word1 = *word2;
    *word2 = tempWord;
}

//сравнение с разделяющими символами
int is_sep_symb(wchar_t c)
{
    return c == L'.' || c == L',' || c == L' ';
}
```

Название файла: sentence.h

```
#include "word.h"
#define SNTC_START_SIZE 2 //начальный размер памяти для
предложения


struct Sentence
{
    Word *words;
    int size;
    int realSize;
};

typedef struct Sentence Sentence;

int initial_sntc(Sentence *new_sntc, int start_size);
int push_back_sntc(Sentence *sntc, Word *word);
int is_equal_sntc(Sentence *sntc1, Sentence *sntc2);
void remove_word(Sentence *sntc, int ind);
void swap_sntc(Sentence *sntc1, Sentence *sntc2);
int move_word_n(Sentence *sntc, int n);
void rm_word_last_cptlz(Sentence *sntc);
```

Название файла: sentence.c

```
#include "sentence.h"

//начальная инициализация предложения
int initial_sntc(Sentence *new_sntc, int start_size)
{
     if (start_size < 0)
     {
          fwprintf(stderr, L"%sОшибка, неверный размер для нового
предложения!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     new_sntc->words = (Word *) malloc(new_sntc->realSize *
sizeof(Word));
     if (new_sntc->words == NULL)
     {
          fwprintf(stderr, L"%sНе получилось выделить память для
предложения!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     new_sntc->size = 0;
     new_sntc->realSize = MAX(start_size, SNTC_START_SIZE);
     return ALL_OK;
}

//положить назад новое слово
int push_back_sntc(Sentence *sntc, Word *word)
{
     //если есть место
     if (sntc->size < sntc->realSize)
     {
          sntc->words[sntc->size++] = *word;
          return ALL_OK;
     }
```

```
     //если нет места
     Word *tempSntc = (Word *) realloc(sntc->words, (size_t)
(sntc->realSize * INCREASE) * sizeof(Word));
     if (tempSntc == NULL)
     {
          fwprintf(stderr, L"%sНе получилось выделить память для
предложения!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     sntc->words = tempSntc;
     sntc->realSize = (size_t) (sntc->realSize * INCREASE);
     sntc->words[sntc->size++] = *word;
     return ALL_OK;
}

//проверка предложений на равенство, не учитывая регистр
int is_equal_sntc(Sentence *sntc1, Sentence *sntc2)
{
     if (sntc1->size != sntc2->size)
          return NOT_EQUAL;
     for (int i = 0; i < sntc1->size; ++i)
     {
          if (!is_equal_word(&sntc1->words[i], &sntc2->words[i]))
               return NOT_EQUAL;
     }
     return EQUAL;
}

//удалить слово из предложения
void remove_word(Sentence *sntc, int ind)
{
     //смещаем все слова после того, которое мы удалили
     for (int i = ind; i < sntc->realSize - 1; ++i)
     {
          swap_word(&(sntc->words[i]), &(sntc->words[i+1]));
```

```
     }
     //освобождаем память за последним
     free(sntc->words[sntc->realSize - 1].word);
     sntc->size--;
     sntc->realSize--;
}

//обмен значениями двух предложений
void swap_sntc(Sentence *sntc1, Sentence *sntc2)
{
     Sentence tempSntc = *sntc1;
     *sntc1 = *sntc2;
     *sntc2 = tempSntc;
}

//сместить слова на n позиций вперёд
int move_word_n(Sentence *sntc, int n)
{
     n %= (sntc->size / 2);
     //запоминаем, какие слова мы уже переместили
     int *used = (int *) calloc(sntc->size, sizeof(int));
     if (!used)
          return SOME_ERROR;
     Word swapWord;
     for (int i = sntc->size - 2; i >= 0; i -= 2)
     {
          if (!used[i])//если до этого не были здесь
          {
               int ind = i;
               swapWord = sntc->words[ind]; // запоминаем
последнее слово
               do
               {
                    //меняем слова местами
                    used[ind] = 1;
```

```
                    sntc->words[ind] = sntc->words[(sntc->size +
ind - 2 * n) % sntc->size];
                    ind = (sntc->size + ind - 2 * n) % sntc-
>size;
               } while ((sntc->size + ind - 2 * n) % sntc->size !
= i);
               used[ind] = 1;
               sntc->words[ind] = swapWord;
          }
     }
     free(used);
     return ALL_OK;
}


//удалить слова с последней заглавной буквой
void rm_word_last_cptlz(Sentence *sntc)
{//удаляем с конца для оптимизации (двигаем только те, которые
сохранены)
     for (int i = sntc->size - 2; i >= 0; i -= 2)
     {
          if (iswupper(sntc->words[i].word[sntc->words[i].size -
1]))
          {//удаляем слово
               remove_word(sntc, i);
               if (i == 0)//если слово было первым, то удаляем
его дальнейшие разделители
                    remove_word(sntc, i);
               else
               {//если было не первым, то склеиваем разделители и
удаляем следующий
                    for (int j = 0; j < sntc->words[i].size; ++j)
                         push_back_word(&sntc->words[i - 1],
sntc->words[i].word[j]);
                    remove_word(sntc, i);
```

```
                }
            }
        }
}
```

Название файла: text.h

```
#include "sentence.h"
#define TEXT_START_SIZE 2 //начальный размер памяти для текста

struct Text
{
    Sentence *sntcs;
    int size;
    int realSize;
};

typedef struct Text Text;

int initial_text(Text* new_text, int start_size);
int push_back_text(Text* text, Sentence* sntc);
void remove_sent(Text* text, int ind);
void delete_dubl(Text* text);
void delete_all(Text *text);
int input_text(Text* text);
void print_text(Text *text);
wchar_t* unique_symb(Text text, int *n);
int *unique_len_word(Text text, int *size);
```

Название файла: text.c

```
#include "text.h"

//начальная инициализация текста
int initial_text(Text *new_text, int start_size)
{
    if (start_size < 0)
```

```
     {
          fwprintf(stderr, L"%sОшибка, неверный размер для нового
текста!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     new_text->sntcs = (Sentence *) malloc(new_text->realSize *
sizeof(Sentence));
     if (new_text->sntcs == NULL)
     {
          fwprintf(stderr, L"%sНе получилось выделить память для
текста!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     new_text->size = 0;
     new_text->realSize = MAX(start_size, TEXT_START_SIZE);
     return ALL_OK;
}

//положить назад новое предложение
int push_back_text(Text *text, Sentence *sntc)
{
     //если есть место
     if (text->size < text->realSize)
     {
          text->sntcs[text->size++] = *sntc;
          return ALL_OK;
     }
     //если нет места
     Sentence *tempText = (Sentence *) realloc(text->sntcs,
(size_t) (text->realSize * INCREASE) *

        sizeof(Sentence));
     if (tempText == NULL)
     {
```

```
          fwprintf(stderr, L"%sНе получилось выделить память для
текста!!%s\n", ERROR_CLR, STD_CLR);
          return SOME_ERROR;
     }
     text->sntcs = tempText;
     text->realSize = (size_t) (text->realSize * INCREASE);
     text->sntcs[text->size++] = *sntc;
     return ALL_OK;
}

//удалить предложение из текста
void remove_sent(Text *text, int ind)
{
     //смещаем все предложения после того, которое мы удалили
     for (int i = ind; i < text->realSize - 1; ++i)
     {
          swap_sntc(&(text->sntcs[i]), &(text->sntcs[i+1]));
     }
     //освобождаем память за последним
     free(text->sntcs[text->realSize - 1].words);
     text->size--;
     text->realSize--;
}

//удалить повторяющиеся предложения
void delete_dubl(Text *text)
{//удаляем с конца для оптимизации (двигаем только те, которые
сохранены)
     for (int i = text->size - 1; i >= 0; --i)
     {
          for (int j = i - 1; j >= 0; --j)
          {
               if (is_equal_sntc(&text->sntcs[i], &text-
>sntcs[j]))
```

```
                {//удаляем предложение, которое имеет больший
индекс, и выходим
                    remove_sent(text, i);
                    break;
                }
            }
        }
}

//освободить всю память
void delete_all(Text *text)
{
    for (int i = 0; i < text->size; ++i)
    {//удаляем до size, тк выделяли память через initial только
для [size] элементов
        for (int j = 0; j < text->sntcs[i].size; ++j)
        {
            free(text->sntcs[i].words[j].word);
        }
        free(text->sntcs[i].words);
    }
    free(text->sntcs);
}

//заполнить текст из потока ввода
int input_text(Text *text)
{
    wchar_t c = L'$';
    wchar_t cPrev = L'$';
    int flag_end_sntc;
    while ((c = getwchar()) != L'\n')
    {
        //флаг для признака конца строки
        flag_end_sntc = 0;
        Sentence sntc;
```

```
          if (!initial_sntc(&sntc, SNTC_START_SIZE))
               return SOME_ERROR;

          while (!flag_end_sntc)
          {
               Word word;
               if (!initial_word(&word, WORD_START_SIZE))
                    return SOME_ERROR;

               //считываем слово до разделителей
               while (!is_sep_symb(c) && c != '\n')
               {
                    if (!push_back_word(&word, c))
                         return SOME_ERROR;
                    c = getwchar();
               }
               //добавляем слово в предложение
               if (!push_back_sntc(&sntc, &word))
                    return SOME_ERROR;

               //специальное слово с символами разделителями
               Word sepWord;
               if (!initial_word(&sepWord, WORD_START_SIZE))
                    return SOME_ERROR;

               //считываем специальное слово
               while (is_sep_symb(c) || (c == L'\n' && cPrev !=
L'\n'))
               {
                    cPrev = c;//запоминаем прошлый разделительный
знак

                    if (c == L'.') //признак конца строки
                         flag_end_sntc = 1;
                    if (!push_back_word(&sepWord, c))
```

```
                    return SOME_ERROR;
               c = getwchar();
          }
          if (!push_back_sntc(&sntc, &sepWord))
               return SOME_ERROR;

          //если достигли конца предложения
          if (flag_end_sntc)
          {
               //возвращаем лишний символ в поток ввод
               ungetwc(c, stdin);
               if (!push_back_text(text, &sntc))
                    return SOME_ERROR;
          }
     }
     return ALL_OK;
}

//вывести текст
void print_text(Text *text)
{
     for (int i = 0; i < text->size; ++i)
     {
          for (int j = 0; j < text->sntcs[i].size; ++j)
          {
               fputws(text->sntcs[i].words[j].word, stdout);
          }
     }
}

//получить массив уникальных символов(без знаков препинания)
wchar_t *unique_symb(Text text, int *n)
{
```

```
     int tableCntOfENSymb[L'z' - L'a' + 1] = {0}; //массив
количеств латинских букв
     int tableCntOfRUSymb[L'я' - L'a' + 1 + 1] = {0}; //массив
количеств кириллических букв (+1 за счёт 'ё')
     int size = 0;
     //заполняем массивы и считаем количество различных букв
     for (int i = 0; i < text.size; ++i)
     {
          for (int j = 0; j < text.sntcs[i].size; ++j)
          {
               for (int k = 0; k < text.sntcs[i].words[j].size; +
+k)
               {
                    wchar_t symb =
towlower(text.sntcs[i].words[j].word[k]);
                    if (L'a' <= symb && symb <= L'z')
                    {
                         tableCntOfENSymb[symb - L'a']++;
                         if (tableCntOfENSymb[symb - L'a'] == 1)
                              size++;
                    }
                    else if (L'a' <= symb && symb <= L'я')
                    {
                         tableCntOfRUSymb[symb - L'a']++;
                         if (tableCntOfRUSymb[symb - L'a'] == 1)
                              size++;
                    }
                    else if (symb == L'ё')
                    {
                         tableCntOfRUSymb[L'я' - L'a' + 1]++;
                         if(tableCntOfRUSymb[L'я' - L'a' + 1] ==
1)
                              size++;
                    }
               }
```

```
            }
        }
        //массив для ответа
        wchar_t *res = (wchar_t *) malloc(size * sizeof(wchar_t));
        if (!res)
            return NULL;

        int ind = 0;
        for (int i = 0; i < L'я' - L'a' + 1; ++i)
        {
            if(L'e' - L'a' + 1 == i && tableCntOfRUSymb[L'я' - L'a'
+ 1] > 0)
                res[ind++] = L'ё';
            if (tableCntOfRUSymb[i] > 0)
                res[ind++] = L'a' + i;
        }
        for (int i = 0; i < L'z' - L'a' + 1; ++i)
        {
            if (tableCntOfENSymb[i] > 0)
                res[ind++] = L'a' + i;
        }
        //задаём размер массива для ответа
        *n = size;
        return res;
}

//получить массив с количеством слов, длина которых 1, 2, ...
int *unique_len_word(Text text, int *size)
{
        //ищем максимальную длину слова
        int maxSize = 0;
        for (int i = 0; i < text.size; ++i)
        {
            for (int j = 0; j < text.sntcs[i].size; ++j)
            {
```

```
                if (!is_sep_symb(text.sntcs[i].words[j].word[0])
&& text.sntcs[i].words[j].word[0] != L'\n')
                {
                     maxSize = MAX(maxSize,
text.sntcs[i].words[j].size);
                }
          }
     }
     //массив для ответа
     int *res = (int *) calloc(maxSize, sizeof(int));
     if(!res)
          return NULL;

     for (int i = 0; i < text.size; ++i)
     {
          for (int j = 0; j < text.sntcs[i].size; ++j)
          {
                if (!is_sep_symb(text.sntcs[i].words[j].word[0])
&& text.sntcs[i].words[j].word[0] != L'\n')
                {
                     res[text.sntcs[i].words[j].size - 1]++;
                }
          }
     }
     //задаём размер массива для ответа
     *size = maxSize;
     return res;
}
```

Название файла: Makefile

```
CC=gcc
FLAGS=-c

build_proj: build_all clean
```

```
build_all: start.o sentence.o text.o word.o
     $(CC) start.o sentence.o text.o word.o -o START

start.o: start.c
     $(CC) $(FLAGS) start.c

sentence.o: sentence.c sentence.h
     $(CC) $(FLAGS) sentence.c

text.o: text.c text.h
     $(CC) $(FLAGS) text.c

word.o: word.c word.h
     $(CC) $(FLAGS) word.c

clean:
     rm -f *.o
```