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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текстовых данных

Студент гр. 3381 _____ Иванов А.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2023

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Иванов А.А.

Группа 3381

Тема работы: Обработка текстовых данных

Исходные данные:

Вариант 5.13

Вывод программы должен быть произведен в стандартный поток вывода: ***stdout***.

Ввод данных в программе в стандартный поток ввода: ***stdin***.

В случае использования Makefile название исполняемого файла должно быть: ***cw***.

Важно: первой строкой при запуске программы нужно выводить информацию о варианте курсовой работе и об авторе программы в строго определенном формате:

Course work for option <V>, created by <Name> <Surname>.

Где V – вариант курсовой и Имя и Фамилия, как указано в репозитории группы. Данное предложение должно быть строго первым предложением в выводе программы и является отдельной строкой (заканчивается знаком ‘\n’).

Например:

Course work for option 3.2, created by Ivan Ivanov.

Ввод данных:

После вывода информацию о варианте курсовой работе программа ожидает ввода пользователем числа – номера команды:

- 0 – вывод текста после первичной обязательной обработки (если предусмотрена заданием данного уровня сложности)
- 1 – вызов функции под номером 1 из списка задания

- 2 – вызов функции под номером 2 из списка задания
- 3 – вызов функции под номером 3 из списка задания
- 4 – вызов функции под номером 4 из списка задания
- 5 – вывод справки о функциях, которые реализует программа.

Программа не должна выводить никаких строк, пока пользователь не введет число.

В случае вызова справки (опция 5) текст на вход подаваться не должен, во всех остальных случаях после выбора опции должен быть считан текст.

Признаком конца текста считается два подряд идущих символа переноса строки ‘\n’. После каждой из функций нужно вывести результат работы программы и завершить программу.

В случае **ошибки** и невозможности выполнить функцию по какой-либо причине, нужно вывести строку:

Error: <причина ошибки>

Задание

Каждое предложение должно выводиться в отдельной строке, пустых строк быть не должно. 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).

Программа должна выполнить одно из введенных пользователем

действий и завершить работу:

1. Сделать сдвиг слов во всех предложениях 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. Символы выводить через пробел. Например, для “111222333qqwwwe” должно быть выведено “1 2 3 q w e”
3. Подсчитать и вывести количество слов (плюс вывести 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.

Содержание пояснительной записки:

«Аннотация», «Введение», «Содержание», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 16.10.2023

Дата сдачи реферата: 13.12.2023

Дата защиты реферата: 15.12.2023

Студент _____ Иванов А.А.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа заключается в выполнении практической задачи с помощью написания программы на языке программирования Си. Программа принимает на вход текст неизвестной длины и обрабатывает его на основе заданной команды.

Первая строка содержит номер исполняемой команды. Если номер команды от 0 до 4 (включительно), то программа производит считывание текста и его дальнейшую обработку. Если введенная команда – 5, то происходит вывод информации об исполняемых функциях. В том случае, если номера такой команды не существует, то выводится ошибка и программа завершает свою работу.

SUMMARY

The course work consists of completing a practical task by writing a program in the C programming language. The program takes text of unknown length as input and processes it based on the given command.

The first line contains the number of the command to be executed. If the command number is from 0 to 4 (inclusive), then the program reads the text and processes it further. If the entered command is 5, then information about the functions being executed is displayed. If the number of such a command does not exist, an error is displayed and the program terminates.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель курсовой работы – освоить подходы к работе с текстовыми данными в языке Си. Достижение этой цели включает следующие задачи:

1. Разработать способ представления и хранения текста, предложений и слов в памяти.
2. Реализовать алгоритм считывания текста произвольной длины с клавиатуры и вывода его на экран.
3. Освоить способ выделения из текста отдельных его частей: подстрок, являющихся словами, частями слов, предложениями.
4. Освоить способ модификации символов текста или его частей. Удаление, добавление новых символов.
5. Выполнить задания курсовой работы на обработку текста в соответствии с условием варианта.
6. Написать *Makefile* для сборки программы.

1. ВВОД И ПЕРВИЧНАЯ ОБРАБОТКА ТЕКСТА

1.1 Подключение необходимых библиотек

Подключаем в файл программы main.c все необходимые заголовчные файлы:

Стандартная библиотека:

1. <*stdio.h*> - заголовочный файл для работы с потоками ввода-вывода.
2. <stdint.h> - заголовочный файл для организации кроссплатформенности программы.
3. <*wchar.h*> - заголовочный файл, содержащий в себе тип *wchar_t* и прототипы функций для работы с широкими символами.
4. <*locale.h*> - заголовочный файл, использующийся для корректной работы с кириллицей.

Пользовательские заголовочные файлы:

1. «structs.h» - загловочный файл с определением структур, необходимых для хранения текста.
2. «printWelcomeMessage.h» — заголовочный файл с определением функции для вывода приветственного сообщения.
3. «printManual.h» - заголовочный файл с определением функции вывода «мануала» к описанным возможностям.
4. «printText» - заголовочный файл с функцией для вывода считанного текста.
5. «printUniqSymbolsInText.h» - заголовочный файл с функцией вывода уникальных символов текста.
6. «printNumOfWordsOfACertainLen.h» - заголовочный файл с функцией для вывода длин слов и слов с такой длиной.
7. «readText.h» - заголовочный файл с функцией считывания текста.
8. «remDupFromText.h» - заголовочный файл с определением функции удаления дублирующихся предложений из текста.

9. «remWordsWithLastUppercaseLetter.h» - заголовочный файл с функцией, необходимой для удаления слов, оканчивающихся на большую букву из текста.
10. «shiftingWordsInText.h» - заголовочный файл с функцией, которая сдвигает слова в каждом предложении текста на введённое пользователем число.

1.2 Организация хранения текста в программе.

Для хранения текста в программе используются структуры Word, *Sentence* и *Text*, которые объявлены в заголовочном файле structs.h.

Структура Word имеет поля для хранения самого слова (wchar_t *word), знака препинания (wchar_t punct), стоящего после него, длины слова (uint32_t len) и количества выделенной памяти (uint32_t allocated_size).

Структура *Sentence* имеет поля для хранения массива указателей на слова предложения (struct Word **words_array), флага того (uint8_t is_last), что предложение является последним в тексте, длины предложения (uint32_t len) и количества выделенной памяти (uint32_t allocated_size).

Структура *Text* имеет поля для хранения массива указателей на предложения (struct Sentence **sentences_array), длины текста (uint32_t len) и количества выделенной памяти (uint32_t allocated_size).

1.3 Ввод и первичная обработка текста.

При запуске программы выводится приветственное сообщение указанного формата, после чего ожидается ввод команды пользователем. Если пользователь вводит число вне диапазона от 0 до 5 включительно или считывание было произведено некорректно, то программа выводит сообщение об ошибке. После чего программа завершается. Если же команда была введена верно — идём дальше. Если введённая команда равна 5, выводим «мануал» и завершаем программу. Если нет — начинаем считывание и обработку текста. Сначала выполняется считывание текста с помощью функции readText, в ней

выделяется место в динамической памяти под структуру Text, поля структуру инициализируются стандартными значениями. После чего определяется и инициализируется NULL переменная для хранения текущего считанного предложения, создаются переменные для хранения количество считанных предложений и количества единиц памяти, выделенных под предложения. Далее начинатеся бесконечный цикл, который будет выполняться пока не будет выполнено условие конца считывания. В первой строке цикла проверяем, что под массив указателей на предложения в структуре текст выделено место с помощью пользовательской функции safetyReallocMemToSentenceStructsArray, эта функция принимает первый аргументов указатель на массив указателей на предложения, а вторым указатель на количество выделенной памяти, служит для безопасного выделения и перевыделения памяти. Далее в curr_sentence сохраняется текущее предложение и проверяется, что поле is_last переменной curr_sentence равно 1(это будет означать, что считывание окончено). Елси это условие выполняется, то проверяем куда указывает words_array, если никуда не указывает, то это означает, что не было считано ни одного слова, проверяем, что количество считанных предложений больше нуля, если это так, то на самом деле предыдущее предложение является последним, после чего выходим из цикла. Иначе сохраняем считанное предложение в структуру Text, увеличивем счетчик количества считанных предложений на 1 и выходим из цикла.

Если изначально условие if не выполнилось и не был произведён выход из цикла, сохраняем предложение, увеличивем счетчик и считываем дальше. После окончания считывания записываем в поле для хранения длины получившееся количество предложений в тексте, после чего возвращаем структуру Text, хранящую считанный текст. После чего вызываем пользовательскую функцию remDupFromText, принимающую в качестве аргумента указатель на структуру Text. А данной функции происходит сравнение каждого предложения с последующими, если находится совпадающее предложение, то оно удаляется вызовом функции remSentence, принимающей указатель на структуру Text и индекс удаляемого предложения.

Внутри функции remSentence происходит сдвиг массива влево, вместе с удалением нужного предложения.

Первая обработка завершена, а значит пришло время продолжить выполнение программы. Для этого используется оператор множественного выбора switch. При считывании команды с номером:

0. Больше не произоводится никаких манипуляций над текстом, он просто выводится
1. Считывается количество сдвигов, на которые нужно сдвинуть слова в предложениях текста, после чего вызывается пользовательская функция shiftingWordsInText, принимающая в качетсве аргументов указатель на текст и количество сдвигов. Внутри функии shiftingWordsInText первой строкой производится проверка того, что количество сдвигов больше или равно одному и количество слов в предложении больше или равно двум, при невыполнении этого условия обработка не требуется, поэтому программа сразу же завершается. Иначе начинается цикл, который проходит по каждому предложению в тексте. Создаётся указатель на текущее предложение текста, убирается точка после последнего слова в предложении, выделяется место под массив слов new_words_array такого же размера, как и массив слов структуры Sentence. Далее попадаем в цикл, проходящий по каждому слову в предложении, с помощью операции взятия остатка зацикливаем индексы массива предложений и деляем сдвиг. После цикла присываиваем массиву слов структуры полученный массив слов, ставим точку после последнего слова и очищаем память, выделенную под массив указателей на слова.
2. Попадаем в функцию printUniqSymbolsInText, которая выводит количество уникальных символов в тексте и принимает в качестве аргумента указатель на структуру считанный текст. Внутри файла, содержащего данную функцию также есть функция «компаратор», которая необходима для использования библиотечной функции быстрого поиска qsort. В функции printUniqSymbolsInText в первой строке

объявляются и инициализируются переменные для хранения количества считанных символов и количества памяти, выделенного под эти символы. Далее заводим переменную, которая будет хранить весь текст единой строкой, для получения такой строки используется пользовательская функция textStructToString, принимающая на вход указатель на текст. Внутри данной функции происходит посимвольный перебор каждого символа в тексте и его запись в данную переменную. Далее выделяется место под строку, которая будет хранить уникальный символы текста и сохраняется длина текста в переменную string_len. После чего строка текста сортируется по возрастанию, это нужно, чтобы найти уникальные символ в данной строке путём сравнивания символов с текущим и предыдущим. В конце присутсвует проверка, которая выведет предупреждение о том, что в тексте нет уникальных символов. Если уникальные символы всё же есть, то выводится строка с этими симсволами, а в конце выделенная под строки uniq_symbols_string и text_string память очищается.

3. Данная подзадача должна вывести длины слов текста и данные слова в скобках. Для этого используется пользовательская функция printNumOfWordsOfACertainLen, принмающая на вход указатель на структуру, хранящую считанный текст. В первыой строке заводится переменная для хранения количества слов текста count_of_read_words, значение которой получается в цикле, идущем после объявления данной переменной. Далее выделяется память под массив указателей на слова текста, который далее заполняется в цикле. После чего происходит сортировка слов в данном массиве по длине слов с помощью функции qsort стандартной библиотеки. Далее происходит вывод длин слов и слов соответсвующей длинны в необходимом формате. В конце очищается память, выделенная под массив указателей на слова текста.
4. В данной подзадаче необходимо удалить слова все слова в тексте, заканчивающиеся на заглавные буквы. Для выполнения данной задачи

используется пользовательская функция remWordsWithLastUppercaseLetter, принимающая на вход указатель на текст. В первой строке заводятся индексы для обхода массива, далее начинается перебор текста по словам. Если встреченной слово последнее в предложении, но перемещаем точку в конец предыдущего слова и выполяем дальнейшую проверку. Если слово последнее не последнее в предложении, то удаляем слово и продолжаем цикл, иначе уменьшаем длину предложения на один и выходим из цикла. Далее проверяем: если длина предложения стала нулевой, это значит что все слова в предложении должны быть удалены, удаляем всё предложение из текста, иначе прибавляем индкс j. После выполнения всех циклов если какое-то предложение стало нулевой длины также его удаляем. Далее прибавляем индекс i, а j приравниваем к нулю. После выполнения функции выводим полученный текст.

Когда выбранная пользователем подзадача была выполнена, очищаем выделенное под текст место и завершаем программу.

Результаты тестирования см. в приложении А.

Разработанный код см. в приложении Б.

2. СБОРКА ПРОГРАММЫ

Makefile.

Главная цель сборки – *переменная TARGET=cw*

Цель $(*TARGET)* требует объектные файлы:

- *main.o*
- *readSentence.o*
- *readText.o*
- *printWelcomeMessage.o*
- *printManual.o*
- *printUniqSymbolsInText.o*
- *printNumOfWordsOfACertainLen.o*
- *printText.o*
- *safetyReallocMemToWStr.o*
- *safetyReallocMemToWordStructsArray.o*
- *safetyReallocMemToSentenceStructsArray.o*
- *remSentence.o*
- *remDupFromText.o*
- *remWord.o*
- *remWordsWithLastUppercaseLetter.o*
- *shiftingWordsInText.o*
- *textStructToString.o*
- *freeText.o*
- *freeSentence.o*

После чего происходит компиляция с помощью компилятора, хранящегося в переменной CC с флагами LINK__FLAGS, имя файла — значение переменной TARGET. После линковки объктных файлов происходит очистка рабочей папки от объектных файлов

Дальнейшие цели создают объектные файлы описанные выше с флагами компиляции $(COMP_FLAGS) и компилятором $(CC),

Цель clean необходима для очистки папки с Makefile от исполяемого файла и от объяектных файлов.

Цель rebuild используется для пересборки проекта, сначала вызывает цель clean, полностью очищая рабочую директорию, после чего вызывает цель, с именем $(TARGET).

ЗАКЛЮЧЕНИЕ

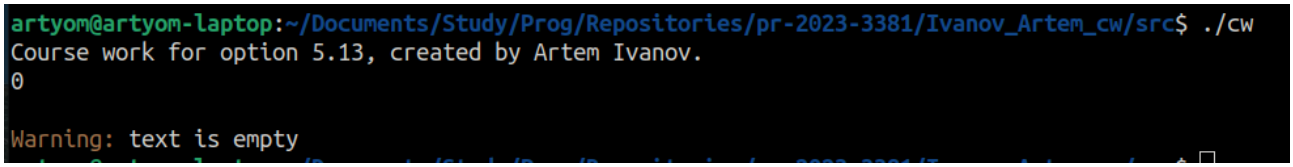
В результате выполнения курсовой работы были реализованы функции для считывания команды и текста неизвестной длины. Для сохранения исходного текста были реализованы структуры Word, *Text* и *Sentence*. Программа обрабатывает текст согласно введенной пользователем команде. Программа производит первоначальную обработку текста при условии, что не была вызвана справка – удаляет табуляции и пробелы в начале предложений, выводит каждое предложение с новой строки, удаляет повторно встретившиеся предложения. В процессе создания программы был написан *Makefile*, совершающий компиляцию и последующую линковку исходных файлов согласно прописанным целям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

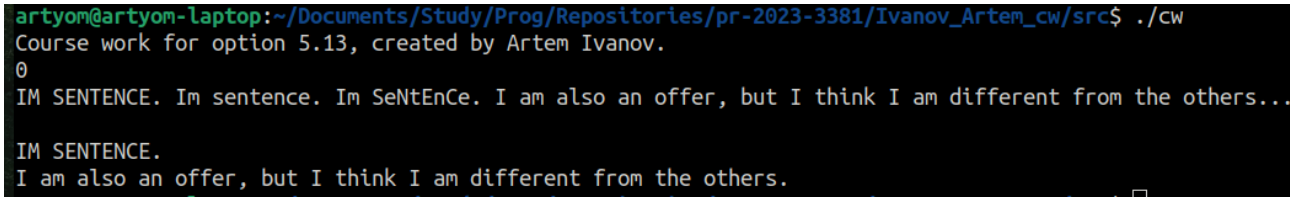
1. Онлайн-курс «Программирование на Си. Практические задания. Первый семестр» // moevm. URL: https://e.moevm.info/course/view.php?id=8 (дата обращения: 04.12.2023).
2. Керниган Б. Ритчи Д. Язык программирования Си, Пер. с англ., 3-е изд., испр. — СПб.: "Невский Диалект", 2001. - 352 с.
3. Поиск ошибок работы с памятью в C/C++ при помощи Valgrind // URL: https://eax.me/valgrind/
4. Интернет-ресурс Metanit // URL: https://metanit.com/c/
5. Интернет-ресурс C Reference // URL: https://en.cppreference.com/w/c

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

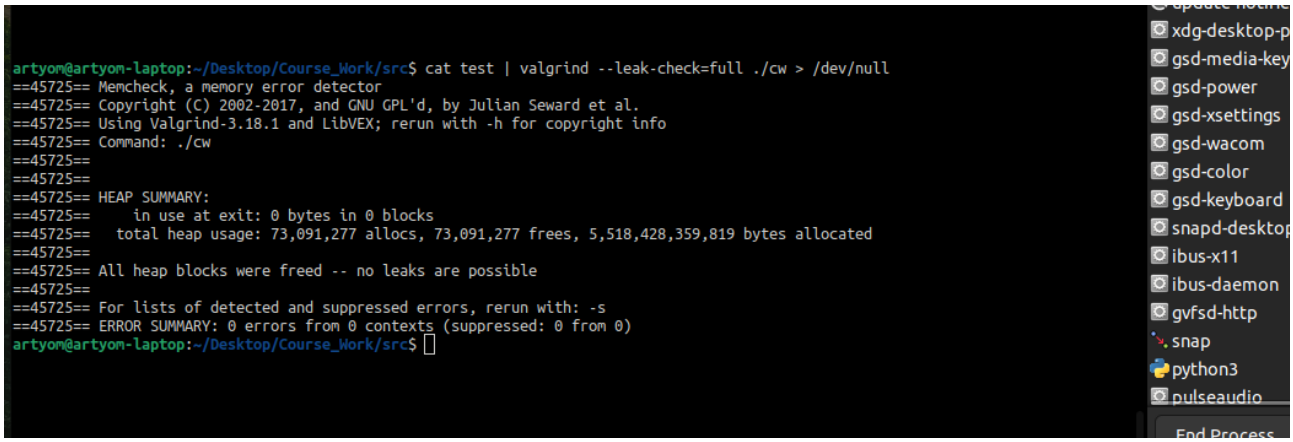
Пример 1. Проверка на пустом тексте



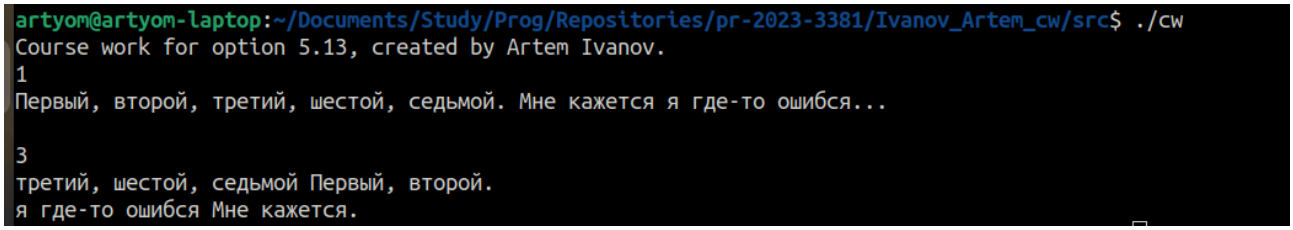
Пример 2. Проверка с несколькими предложениями



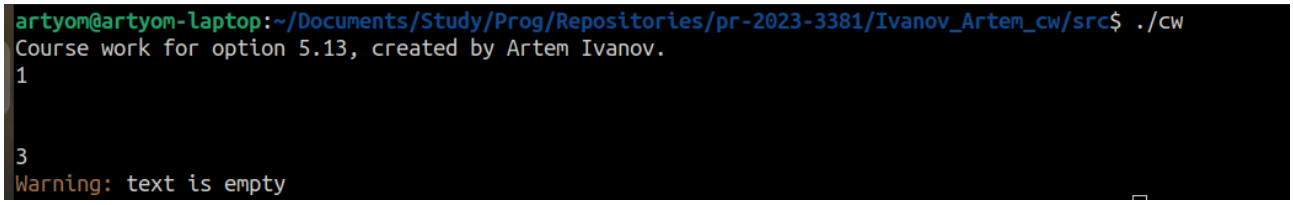
Пример 3. Тексту на вход был подан файл, содержащий 1001 dmesg (время выполнения составило от 12 до 18 часов).



Пример 4. На вход подано два предложения



Пример 5. На вход подан пустой текст



Пример 6. Проверка утечек памяти

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ valgrind --leak-check=full ./cw
==27597== Memcheck, a memory error detector
==27597== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27597== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==27597== Command: ./cw
==27597==
Course work for option 5.13, created by Artem Ivanov.
1
Hello world and all people. I'm my leather man's coursework

3
and all people Hello world.
leather man's coursework I'm my.
==27597==
==27597== HEAP SUMMARY:
==27597==     in use at exit: 0 bytes in 0 blocks
==27597==   total heap usage: 102 allocs, 102 frees, 20,447 bytes allocated
==27597==
==27597== All heap blocks were freed -- no leaks are possible
==27597==
==27597== For lists of detected and suppressed errors, rerun with: -s
==27597== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Пример 7. Проверка на тексте

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ ./cw
Course work for option 5.13, created by Artem Ivanov.
2
Я неуникален... Очень жаль, а ведь хочется быть... Вот например как он -> @

я ы х у с п м ж д в б Я О В
```

Пример 8. Проверка на пустом тексте

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ ./cw
Course work for option 5.13, created by Artem Ivanov.
2

Warning: text is empty
```

Пример 9. Проверка утечек памяти

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ valgrind --leak-check=full ./cw
==28146== Memcheck, a memory error detector
==28146== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28146== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==28146== Command: ./cw
==28146==
Course work for option 5.13, created by Artem Ivanov.
2
Надеюсь память никуда магическим образом не утечёт. Ведь память не птичка, утечёт - не найдёшь...

ю ш р й з г б Н В
==28146==
==28146== HEAP SUMMARY:
==28146==     in use at exit: 0 bytes in 0 blocks
==28146==   total heap usage: 138 allocs, 138 frees, 24,991 bytes allocated
==28146==
==28146== All heap blocks were freed -- no leaks are possible
==28146==
==28146== For lists of detected and suppressed errors, rerun with: -s
==28146== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Пример 10. Проверка на простом тексте

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ ./cw
Course work for option 5.13, created by Artem Ivanov.
3
Ну ка слова, встаньте в шеренгу, на первый-последний расчитайсь!

1: (в)
2: (Ну ка на)
5: (слова)
7: (шеренгу)
8: (встаньте)
11: (расчитайсь!)
16: (первый-последний)
```

Пример 11. Проверка утечек

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ valgrind --leak-check=full ./cw
==28754== Memcheck, a memory error detector
==28754== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28754== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==28754== Command: ./cw
==28754==
Course work for option 5.13, created by Artem Ivanov.
3
Ловись утечка большая и маленькая!

1: (и)
6: (Ловись утечка)
7: (большая)
10: (маленькая!)
==28754==
==28754== HEAP SUMMARY:
==28754==     in use at exit: 0 bytes in 0 blocks
==28754==   total heap usage: 81 allocs, 81 frees, 19,763 bytes allocated
==28754==
==28754== All heap blocks were freed -- no leaks are possible
==28754==
==28754== For lists of detected and suppressed errors, rerun with: -s
==28754== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Пример 12.

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ ./cw
Course work for option 5.13, created by Artem Ivanov.
4
УдаляемсЯ друзьЯ!ю

друзьЯ!ю.
```

Пример 13.

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ valgrind --leak-check=full ./cw
==29210== Memcheck, a memory error detector
==29210== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==29210== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==29210== Command: ./cw
==29210==
Course work for option 5.13, created by Artem Ivanov.
4
Утечёт или нет, вот в чём вопрос... Надеюсь, что нет. А то опять править придётся...

Утечёт или нет, вот в чём вопрос.
Надеюсь, что нет.
А то опять править
==29210==
==29210== HEAP SUMMARY:
==29210==     in use at exit: 0 bytes in 0 blocks
==29210==   total heap usage: 118 allocs, 118 frees, 20,695 bytes allocated
==29210==
==29210== All heap blocks were freed -- no leaks are possible
==29210==
==29210== For lists of detected and suppressed errors, rerun with: -s
==29210== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Пример 14. Вывод справки

```
artyom@artyom-laptop:~/Documents/Study/Prog/Repositories/pr-2023-3381/Ivanov_Artem_cw/src$ ./cw
Course work for option 5.13, created by Artem Ivanov.
5
╔═══╦═══════════════════════════════════════════╣ MANUAL ╠═══════════════════════════════════════════╗
| 1 | Сделть 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 вывести слова в скобках) длина которых равна 1, 2, 3, и т.д. |
╠═══╬═══════════════════════════════════════════════════════════════════════════════════════════════╣
| 4 | Удалить все слова которые заканчиваются на заглавный символ.                                      |
╚═══╩═══════════════════════════════════════════════════════════════════════════════════════════════╝
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла:

main.c

```
#include <stdio.h>
#include <stdint.h>
#include <wchar.h>
#include <locale.h>

#include "./lib/structs.h"
#include "./lib/print_functions/printWelcomeMessage.h"
#include "./lib/print_functions/printManual.h"
#include "./lib/print_functions/printText.h"
#include "./lib/print_functions/printUniqSymbolsInText.h"
#include "./lib/print_functions/printNumOfWordsOfACertainLen.h"
#include "./lib/read_functions/readText.h"
#include "./lib/remove_functions/remDupFromText.h"
#include "./lib/remove_functions/remWordsWithLastUppercaseLetter.h"
#include "./lib/shifting_functions/shiftingWordsInText.h"
#include "./lib/free_functions/freeText.h"

int main()
{
	setlocale(LC_ALL, "");

	// печатаем приветственное сообщение
	printWelcomeMessage();

	// считываем номер команды
	uint8_t command;
	//wprintf(L"\nEnter a number (a number from 0 to 5): ");
	uint32_t success = wscanf(L"%u", &command);

	// проверяем команду на корректность
	if (success != 0 && (command > 5 || command < 0))
	{
		fwprintf(stderr, L"\033[31mError:\033[0m wrong option\n");
	}
	else
	{
		if (command == 5) { printManual(); }
		else
```

```
        {
            //wprintf(L"\nEnter the text below:\n");

            struct Text* text = readText();
            remDupFromText(&text);

            uint32_t number_of_shifts = 0;

            // выполняем подзадачи
            switch(command)
            {
                case 0:
                    //wprintf(L"Text without repeating
sentences:\n");
                    printText(&text);
                    break;

                case 1:
                    //wprintf(L"\nEnter the number by which you
want to shift the words in text: ");
                    wscanf(L"%u", &number_of_shifts);

                    shiftingWordsInText(&text, number_of_shifts);

                    //wprintf(L"\nText with sentences shifted by
[%u]:\n", number_of_shifts);
                    printText(&text);
                    break;

                case 2:
                    printUniqSymbolsInText(&text);
                    break;

                case 3:
                    printNumOfWordsOfACertainLen(&text);
                    break;

                case 4:
                    remWordsWithLastUppercaseLetter(&text);

                    //wprintf(L"Text without words with the last
letter in uppercase:\n");
                    printText(&text);
                    break;
```

```
                }
                freeText(&text);
            }
        }

        return 0;
}
```

<u>Папка lib с исходными файлами программы:</u>

Название файла:

print_functions/printWelcomeMessage.c

```
 #include "./printWelcomeMessage.h"

void printWelcomeMessage()
{
    wprintf(L"Course work for option 5.13, created by Artem Ivanov.\n");
}
Название файла: print_functions/printManual.h
#include "./printManual.h"

void printManual()
{
    wprintf(L"╔═══╦═══════════════════════════════════════════════════╣ MANUAL ╠═══════════════════════════════════════════════════╗\n");
    wprintf(L"| 1 | Сделть сдвиг слов в предложении на положительное целое число N.                                             |\n");
    wprintf(L"|   | Например, предложение “abc b#c ИЙ два” при N = 2 должно принять вид “ИЙ два abc b#c”.                        |\n");

wprintf(L"╠═══╬═══════════════════════════════════════════════════════════════════════════════════════════════════════════╣\n");
    wprintf(L"| 2 | Вывести все уникальные кириллические и латинские символы в тексте.                                          |\n");

wprintf(L"╠═══╬═══════════════════════════════════════════════════════════════════════════════════════════════════════════╣\n");
    wprintf(L"| 3 | Подсчитать и вывести количество слов (плюс вывести слова в скобках) длина которых равна 1, 2, 3, и т.д.  |\n");

wprintf(L"╠═══╬═══════════════════════════════════════════════════════════════════════════════════════════════════════════╣\n");
    wprintf(L"| 4 | Удалить все слова которые заканчиваются на заглавный символ.                                                |\n");
```

```
wprintf(L"╚═══╝═══════════════════════════════════════════════════════
═══════════════════════════════════════════════════╝\n");
}
```

Название файла:

print_functions/printText.c

```
#include "./printText.h"

void printText(struct Text **text)
{
	struct Sentence** sentences_array = (*text)->sentences_array;

	if ((*text)->len == 0)
	{
		wprintf(L"\033[33mWarning:\033[0m text is empty\n");
	}
	// перебор предложений текста
	for (uint32_t i = 0; i < (*text)->len; i++)
	{
		struct Sentence* sentence = sentences_array[i];

		// перебор слов текста
		for (uint32_t j = 0; j < sentence->len; j++)
		{
			if ((sentence->words_array)[j]->punct == L'.')
{ wprintf(L"%ls%lc", sentence->words_array[j]->word, sentence-
>words_array[j]->punct); }

			else if ((sentence->words_array)[j]->punct != L'\0')
{ wprintf(L"%ls%lc ", sentence->words_array[j]->word, sentence-
>words_array[j]->punct); }

			else { wprintf(L"%ls ", (sentence->words_array)[j]-
>word); }
		}
		wprintf(L"\n");
	}
}
```

Название файла:

print_functions/printNumOfWordsOfACertainLen.c

```
#include "./printNumOfWordsOfACertainLen.h"
```

```
// comparator to quick sort Word structs array
int32_t words_len_cmp(const void *w1, const void *w2)
{
	const uint32_t len1 = (*(*((struct Word **)w1))).len;
	const uint32_t len2 = (*(*((struct Word **)w2))).len;

	if (len1 > len2) { return 1; }
	else if (len1 == len2) { return 0; }
	else { return -1; }
}

void printNumOfWordsOfACertainLen(struct Text **const text)
{
	uint32_t count_of_read_words = 0;

	// total word count
	for (uint32_t i = 0; i < (*text)->len; i++) {
		count_of_read_words += (*text)->sentences_array[i]->len;
	}


	// create Word struct array to contain all words in text
	struct Word **text_words_array = (struct
Word**)malloc(count_of_read_words * sizeof(struct Word*));

	// fill words array
	uint32_t k = 0;
	for (uint32_t i = 0; i < (*text)->len; i++)
	{
		for (uint32_t j = 0; j < (*text)->sentences_array[i]->len; j+
+)
		{
			text_words_array[k++] = (*text)->sentences_array[i]-
>words_array[j];
		}
	}

	// сортируем массив слов по возрастанию длины слова
	qsort(text_words_array, count_of_read_words, sizeof(struct Word *),
words_len_cmp);

	// выводим длину слова, и слова этой длины
	uint32_t curr_len = 0;
```

```
    for (uint32_t k = 0; k < count_of_read_words; k++)
    {
        if (curr_len != text_words_array[k]->len)
        {
            if (curr_len != 0) { wprintf(L")\n"); }
            wprintf(L"%ld:      (%ls",      text_words_array[k]->len,
text_words_array[k]->word);
            curr_len = text_words_array[k]->len;
            if (k + 1 == count_of_read_words) { wprintf(L")\n"); }
            continue;
        }

        wprintf(L" %ls", text_words_array[k]->word);
        if (k + 1 == count_of_read_words) { wprintf(L")\n"); }
    }

    free(text_words_array);
}
```

Название файла: print_functions/printNumOfWordsOfACertainLen.h

```
#ifndef PRINT_NUM_OF_WORDS_OF_A_CERTAIN_LEN_H
#define PRINT_NUM_OF_WORDS_OF_A_CERTAIN_LEN_H

#include <wchar.h>
#include <stdlib.h>

#include "../structs.h"
#include
"../safety_realloc_functions/safetyReallocMemToWordStructsArray.h"


void printNumOfWordsOfACertainLen(struct Text **const text);

#endif
```

Название файла: print_functions/printUniqSymbolsInText.c

```
#include "./printUniqSymbolsInText.h"

int32_t chars_cmp(const void *x, const void *y)
{
    wchar_t f = *((wchar_t *)x);
    wchar_t s = *((wchar_t *)y);
```

```
      if (f > s) { return -1; }
      else if (f == s) { return 0; }
      else { return 1; }
}

void printUniqSymbolsInText(struct Text **text)
{
      uint32_t count_of_read_chars = 0;
      uint32_t count_of_allocated_chars = 0;

      wchar_t *text_string = textStructToString(text);

      wchar_t *uniq_symbols_string  = NULL;
      safetyReallocMemToWStr(&uniq_symbols_string,
&count_of_allocated_chars);

      uint32_t string_len = wcslen(text_string);

      // сортируем строку
      qsort(text_string, string_len, sizeof(wchar_t), chars_cmp);

      for (uint32_t i = 0; i < string_len - 1; i++)
      {
            if (iswalpha(text_string[i]))
            {
                  if (i == 0 && text_string[i] != text_string[i + 1])
                  {
                        if              (count_of_read_chars              ==
count_of_allocated_chars)
                        {
                              safetyReallocMemToWStr(&uniq_symbols_string,
&count_of_allocated_chars);
                        }

                        uniq_symbols_string[count_of_read_chars++]         =
text_string[i];
                  }
                  else if (i >= 1 && text_string[i - 1] != text_string[i]
&& text_string[i] != text_string[i + 1])
                  {
                        if              (count_of_read_chars              ==
count_of_allocated_chars)
                        {
```

```
                              safetyReallocMemToWStr(&uniq_symbols_string,
&count_of_allocated_chars);
                        }

                        uniq_symbols_string[count_of_read_chars++]        =
text_string[i];
                  }
            }
      }
      if (count_of_read_chars >= count_of_allocated_chars)
      {
            safetyReallocMemToWStr(&uniq_symbols_string,
&count_of_allocated_chars);
      }
      uniq_symbols_string[count_of_read_chars] = L'\0';

      //wprintf(L"Unique Latin and Cyrillic characters:\n");

      if (count_of_read_chars == 0)
      {
            wprintf(L"\033[33mWarning:\033[0m there are no unique Cyrillic
or Latin characters in the text\n");
            count_of_allocated_chars = 0;
      }
      else
      {
            for (uint32_t i = 0; i < count_of_read_chars; i++)
            {
                  if (i == 0) { wprintf(L"%lc", uniq_symbols_string[i]); }
                  else { wprintf(L" %lc", uniq_symbols_string[i]); }
            }
            wprintf(L"\n");
      }

      free(uniq_symbols_string);
      free(text_string);
}
```

Название файла:
print_functions/printUniqSymbolsInText.h

```
#ifndef PRINT_UNIQ_SYMBOLS_IN_TEXT_H
#define PRINT_UNIQ_SYMBOLS_IN_TEXT_H

#include <stdint.h>
```

```
#include <wchar.h>
#include <stdlib.h>
#include <wctype.h>

#include "../structs.h"
#include "../textStructToString.h"
#include "../safety_realloc_functions/safetyReallocMemToWStr.h"

void printUniqSymbolsInText(struct Text **text);

#endif
```

Название файла:
read_functions/readSentence.c

```
#include "./readSentence.h"

struct Sentence* readSentence()
{
    struct Sentence *sentence = (struct Sentence*)malloc(sizeof(struct
Sentence));
    sentence->is_last = 0;

    uint32_t count_of_read_chars = 0;                    // хранит
количество считанных символов
    uint32_t count_of_allocated_chars = 0;               // хранит
количество символов, под которые выделена память

    uint32_t count_of_read_words = 0;                    // хранит
количество считанных слов
    uint32_t count_of_allocated_words = 0;               // хранит
количество слов, под которые выделена память

    wchar_t c = getwchar();                          // хранит текущий
символ
    wchar_t prev_c = '\0';                               // хранит
предыдущий символ

    uint8_t in_word = 0;                             // флаг нахождения
внутри слова

    // определяем массив указателей на структуры Word и выделяем под
него память
    struct Word **words_array = NULL;
    //wprintf(L"DEBUG_BEGIN!!");
```

```
     safetyReallocMemToWordStructsArray(&words_array,
&count_of_allocated_words);

     while (1)
     {
          // если последний знак препинания равен точке, то это значит
что предложение закончилось
          if (c == L'.' && count_of_read_words >= 1)
          {
               if (words_array[count_of_read_words - 1]->punct == L'.')
{ break; }
          }

          // проверяем условие начала слова
          if (!in_word && !iswspace(c) && c != L',' && c != L'.')
          {

     safetyReallocMemToWStr(&(words_array[count_of_read_words]->word),
&count_of_allocated_chars);
               words_array[count_of_read_words]-
>word[count_of_read_chars++] = c;
               in_word = 1;

          }
          // проверяем какие знаки стоят после слова и записываем
          else if (count_of_read_words >= 1 && !in_word)
          {
               // если точка, то дальше проверять смысла нет,
присваивает точку структуре и выходим из цикла
               if (c == L'.' && words_array[count_of_read_words - 1]-
>punct != L'.')
               {
                    words_array[count_of_read_words - 1]->punct = L'.';
                    break;
               }

               // если в промежутке между словами появилась запятая
               else if (c == ',' && words_array[count_of_read_words -
1]->punct == L'\0')
               {
                    words_array[count_of_read_words - 1]->punct = L',';
               }
          }
          else
```

```
		{
			// проверяем, что мы попали в слово
			if (in_word)
			{
				// пока в слове выполняем
				while(in_word)
				{
					// если символ допустимый - записываем его в
слово
					if (!iswspace(c) && c != L',' && c != L'.')
					{
						// проверяем, что памяти ещё хватает,
если нет - выделяем ещё
						if (count_of_read_chars + 1 >=
count_of_allocated_chars)
						{

	safetyReallocMemToWStr(&words_array[count_of_read_words]->word,

&count_of_allocated_chars);
						}

						words_array[count_of_read_words]-
>word[count_of_read_chars++] = c;

						prev_c = c;
						c = getwchar();
						continue;
					}
					else
					{
						if (count_of_read_words + 1 >=
count_of_allocated_words)
						{

	safetyReallocMemToWordStructsArray(&words_array,
&count_of_allocated_words);
						}

						in_word = 0;
						words_array[count_of_read_words]-
>word[count_of_read_chars] = L'\0';

						// сохраняем длину слова
```

```
                                    words_array[count_of_read_words]->len   =
count_of_read_chars;

                                    // сохраняем знак после слова
                                    if   (c   ==   L','   ||   c   ==   L'.')
{ words_array[count_of_read_words]->punct = c; }
                                    else { words_array[count_of_read_words]-
>punct = '\0'; }

                                    count_of_read_chars = 0;
                                    count_of_allocated_chars = 0;

                                    count_of_read_words++;
                              }
                        }
                        continue;
                  }
            }

            if (c == L'\n' && prev_c == L'\n')
                  {
                        if (count_of_read_words >= 1)
                        {
                              words_array[count_of_read_words - 1]->punct =
L'.';
                        }
                        sentence->is_last = 1;
                        break;
                  }

            prev_c = c;
            c = getwchar();
      }

      // если не было считано ни одного слова, то
      if (count_of_read_words == 0)
      {
            for (uint32_t i = 0; i < count_of_allocated_words; ++i)
            {
                  free(words_array[i]);
            }
            free(words_array);
            sentence->words_array = NULL;
            sentence->len = 0;
```

```
            sentence->allocated_size = 0;
      }
      else
      {
            sentence->words_array = words_array;
            sentence->len = count_of_read_words;
            sentence->allocated_size = count_of_allocated_words;
      }
      //wprintf(L"DEBUG_END!!");
      return sentence;
}
```

Название файла:
read_functions/readSentence.h

```
#ifndef READ_SENTENCE_H
#define READ_SENTENCE_H

#include <wchar.h>
#include <stdlib.h>
#include <stdint.h>
#include <wctype.h>

#include "../structs.h"
#include "../safety_realloc_functions/safetyReallocMemToWStr.h"
#include
"../safety_realloc_functions/safetyReallocMemToWordStructsArray.h"
#include
"../safety_realloc_functions/safetyReallocMemToSentenceStructsArray.h"

struct Sentence* readSentence();

#endif
```

Название файла:
read_functions/readText.c

```
#include "./readText.h"

struct Text* readText()
{
      // объявление результирующей структуры Text и инициализиация её
значений
      struct Text *text = (struct Text *)malloc(sizeof(struct Text));
      text->sentences_array = NULL;
      text->len = 0;
```

```
    struct Sentence* curr_sentence = NULL;          // хранит считанное
предложение

    uint32_t  count_of_read_sentences  =  0;                  //  хранит
количество считанных предложений
    uint32_t  count_of_allocated_sentences  =  0;             //  хранит
количество предложений, под которые выделеан память

    while (1)
    {
        //  проверяем,  что  всё  ещё  хватает  памяти  для  сохранения
предложений
        if (count_of_read_sentences >= count_of_allocated_sentences) {
            safetyReallocMemToSentenceStructsArray(&(text-
>sentences_array),

&count_of_allocated_sentences);
        }

        curr_sentence = readSentence();

        // обрабатываем случай последнего предложения
        if (curr_sentence->is_last)
        {
            // если функция чтения предложения вернула пустой массив
words_array, делаем break
            if (curr_sentence->words_array == NULL)
            {
                freeSentence(&curr_sentence);
                if (count_of_read_sentences >= 1)
                {
                    text->sentences_array[count_of_read_sentences
- 1]->is_last = 1;
                }
                break;
            }
            else
            {
                freeSentence(&(text-
>sentences_array[count_of_read_sentences]));
                text->sentences_array[count_of_read_sentences]    =
curr_sentence;
                count_of_read_sentences++;
```

```
                    break;
                }
            }
            freeSentence(&(text-
>sentences_array[count_of_read_sentences]));
            text->sentences_array[count_of_read_sentences]                =
curr_sentence;
            count_of_read_sentences++;
      }

      text->len = count_of_read_sentences;
      text->allocated_size = count_of_allocated_sentences;
      //wprintf(L"D: a - %d, s - %d\n", text->allocated_size, text->len);

      return text;
}
```

Название файла:
read_functions/readText.h

```
#ifndef READ_TEXT_H
#define READ_TEXT_H

#include <wchar.h>
#include <stdlib.h>

#include "../structs.h"
#include "../read_functions/readSentence.h"
#include
"../safety_realloc_functions/safetyReallocMemToSentenceStructsArray.h"
#include "../free_functions/freeSentence.h"

struct Text* readText();

#endif
```

Название файла:
remove_functions/remDupFromText.c

```
#include "./remDupFromText.h"

void remDupFromText(struct Text** text)
{
      // выполням функцию только если в ней есть хотя бы два предложения,
иначе это просто не имеет смысла
      //<REVIEW> if-while
      if ((*text)->len >= 2)
```

```
    {
        // флаг равенства предложений
        uint8_t flag = 0;

        // индекс фиксрованного предложения
        uint32_t i = 0;

        while(1)
        {
            struct Sentence *sentence1;

            // фиксируем предложение
            if(i <= (*text)->len - 1) { sentence1 = (*text)-
>sentences_array[i]; }
            else { break; }

            // проверяем, что в тексте есть следующиее за
зафиксированным предложение
            if (!sentence1->is_last)
            {
                uint32_t j = i + 1;

                // проходимся по всем предложениям, начиная от
следующего после зафиксированного
                while(1)
                {
                    struct Sentence *sentence2;

                    // проверяем, что j ещё не вышел за пределы
допустимых индексов
                    if (j <= (*text)->len - 1) { sentence2 =
(*text)->sentences_array[j]; }
                    else { break; }

                    // если количество слов в предложении разное,
то дальше проверять смысла нет
                    if (sentence1->len != sentence2->len)
                    {
                        j++;
                        continue;
                    }
                    else
                    {
```

```
                                    // сраниваем каждое слово и его знак
препинания
                                    for (uint32_t k = 0; k < sentence1->len;
k++)
                                    {
                                          if          (!wcscasecmp(sentence1-
>words_array[k]->word,
                                                                     sentence2-
>words_array[k]->word) &&
                                                                  sentence1-
>words_array[k]->punct == sentence2->words_array[k]->punct)
                                          {
                                                flag = 1;
                                          }
                                          else
                                          {
                                                flag = 0;
                                                break;
                                          }
                                    }
                              }
                              if (flag)
                              {
                                    remSentence(&(*text), j);
                                    flag = 0;
                              }
                              else{ j++; }
                        }
                  }
                  // если за зафиксированным больше нет предложений -
выходим из цикла
                  else
                  {
                        break;
                  }
                  i++;
            }
      }
}
```

Название файла:
remove_functions/remDupFromText.h

```
#ifndef REM_DUP_FROM_TEXT_H
#define REM_DUP_FROM_TEXT_H
```

```
#include <wchar.h>
#include <stdlib.h>

#include "../structs.h"
#include "./remSentence.h"

void remDupFromText(struct Text** text);

#endif
```

Название файла:
remove_functions/remSentence.c

```
#include "./remSentence.h"

void remSentence(struct Text **text, const uint32_t index_of_sentence)
{
	freeSentence(&((*text)->sentences_array[index_of_sentence]));

	for(uint32_t i = index_of_sentence; i < (*text)->len-1; i++) {
		(*text)->sentences_array[i] = (*text)->sentences_array[i+1];
	}
	(*text)->sentences_array[(*text)->len-1] = NULL;
	(*text)->len--;
}
```

Название файла:
remove_functions/remSentence.h

```
#ifndef REM_SENTENCE_H
#define REM_SENTENCE_H

#include <wchar.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#include "../structs.h"
#include "../free_functions/freeSentence.h"

void remSentence(struct Text **text, uint32_t index_of_sentence);

#endif
```

Название файла:
remove_functions/remWord.c

```
#include "./remWord.h"
```

```
void remWord(struct Sentence **sentence, const uint32_t index_of_word)
{
	free((*sentence)->words_array[index_of_word]->word);
	free((*sentence)->words_array[index_of_word]);

	for(uint32_t i = index_of_word; i < (*sentence)->len-1; i++) {
		(*sentence)->words_array[i] = (*sentence)->words_array[i+1];
	}
	(*sentence)->words_array[(*sentence)->len-1] = NULL;
	(*sentence)->len--;
}
```

Название файла:
remove_functions/remWord.h

```
#ifndef REM_WORD_H
#define REM_WORD_H

#include <string.h>
#include <stdint.h>
#include <stdlib.h>

#include "../structs.h"

void remWord(struct Sentence **sentence, const uint32_t index_of_word);

#endif
```

Название файла:
remove_functions/remWordsWithLastUppercaseLetter.c

```
#include "./remWordsWithLastUppercaseLetter.h"

void remWordsWithLastUppercaseLetter(struct Text **text)
{
	//<REVIEW> if-while
	uint32_t i = 0, j = 0;

	// перебор по предложениям текста
	while(i < (*text)->len)
	{
		// перебор по словам в i-м предложении
		while(j < (*text)->sentences_array[i]->len)
		{
			if ((*text)->sentences_array[i]->len != 0)
			{
```

```
                    // проверяем, что последний символ в верхнем
регистре
                    if     ((*text)->sentences_array[i]->words_array[j]-
>len >= 1 &&
                          iswupper(((*text)->sentences_array[i]-
>words_array[j]->word)[(*text)->sentences_array[i]->words_array[j]->len -
1]))
                    {
                          // если слово последнее в предложении и длина
предложения не равна 1,
                          // то ставим после прошлого слова точку
                          if (j == (*text)->sentences_array[i]->len - 1
&& (*text)->sentences_array[i]->len != 1)
                          {
                                (*text)->sentences_array[i]-
>words_array[j - 1]->punct = L'.';
                          }

                          // проверяем, что слово не является последним
в предложении
                          if               ((*text)->sentences_array[i]-
>words_array[j]->len - 1 != j)
                          {
                                remWord(&((*text)->sentences_array[i]),
j);
                                continue;
                          }
                          else
                          {
                                (*(*text)->sentences_array[i]).len--;
                                break;
                          }
                    }
                    // проверяем случай, когда после удаления слова
предложение стало пустым
                    if ((*text)->sentences_array[i]->len == 0)
                    {
                          remSentence(text, i);
                    }
                    else { j++; }
              }
        }
        if ((*text)->sentences_array[i]->len == 0)
        {
```

```
                remSentence(text, i);
            }
            i++;
            j = 0;
        }
}
```

Название файла:
remove_functions/remWordsWithLastUppercaseLetter.h

```
#ifndef REM_WORDS_WITH_LAST_UPPERCASE_LETTER_H
#define REM_WORDS_WITH_LAST_UPPERCASE_LETTER_H

#include <wchar.h>
#include <stdlib.h>
#include <wctype.h>

#include "../structs.h"
#include "./remWord.h"
#include "./remSentence.h"

void remWordsWithLastUppercaseLetter(struct Text **text);

#endif
```

Название файла:
safety_realloc_functions/safetyReallocMemToWstr.c

```
#include "./safetyReallocMemToWStr.h"

void safetyReallocMemToWStr(wchar_t** wstring, int32_t*
count_of_allocated_chars)
{
    // сохраняем область памяти, на которую указывает buffer изначально
    // пробуем выделить память
    wchar_t* new_wstring = (wchar_t*)realloc(*wstring,
(*count_of_allocated_chars + BLOCK_SIZE)*sizeof(wchar_t));

    // если указатель на buffer == NULL, значит произошла ошибка
    if (new_wstring == NULL)
    {
        // если память изначально указывала на NULL, это значит что мы
её выделяли впервые
        // выводим соответствующее сообщение об ошибке
        if (*wstring == NULL)
        {
            fwprintf(stderr, L"Error: failed to allocate memory\n");
```

```
                exit(EXIT_FAILURE);
            }
            // иначе была ошибка при перевыделении
            else
            {
                fwprintf(stderr,   L"Error:   failed   to   re-allocate
memory\n");
                exit(EXIT_FAILURE);
            }
        }
        else
        {
            *wstring = new_wstring;
            *count_of_allocated_chars += BLOCK_SIZE;
        }
}
```

Название файла:
safety_realloc_functions/safetyReallocMemToWstr.h

```
#ifndef SAFETY_REALLOC_MEM_TO_WSTR_H
#define SAFETY_REALLOC_MEM_TO_WSTR_H

#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "../macro.h"

void       safetyReallocMemToWStr(wchar_t**       wstring,       int32_t*
count_of_allocated_chars);

#endif
```

Название файла:
safety_realloc_functions/safetyReallocMemToWstr.c

```
#include "./safetyReallocMemToWStr.h"

void safetyReallocMemToWStr(wchar_t** wstring, int32_t* count_of_allocated_chars)
{
    // сохраняем область памяти, на которую указывает buffer изначально
    // пробуем выделить память
    wchar_t* new_wstring = (wchar_t*)realloc(*wstring, (*count_of_allocated_chars
+ BLOCK_SIZE)*sizeof(wchar_t));
```

```
	// если указатель на buffer == NULL, значит произошла ошибка
	if (new_wstring == NULL)
	{
		// если память изначально указывала на NULL, это значит что мы её выделяли впервые
		// выводим соответствующее сообщение об ошибке
		if (*wstring == NULL)
		{
			fwprintf(stderr, L"Error: failed to allocate memory\n");
			exit(EXIT_FAILURE);
		}
		// иначе была ошибка при перевыделении
		else
		{
			fwprintf(stderr, L"Error: failed to re-allocate memory\n");
			exit(EXIT_FAILURE);
		}
	}
	else
	{
		*wstring = new_wstring;
		*count_of_allocated_chars += BLOCK_SIZE;
	}
}
```

Название файла: safety_realloc_functions/safetyReallocMemToWstr.h

```
#ifndef SAFETY_REALLOC_MEM_TO_WSTR_H
#define SAFETY_REALLOC_MEM_TO_WSTR_H

#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "../macro.h"

void	safetyReallocMemToWStr(wchar_t**	wstring,	int32_t*
count_of_allocated_chars);

#endif
```

Название файла: safety_realloc_functions/safetyReallocMemToWordStructsArray.c

```
#include "./safetyReallocMemToWordStructsArray.h"

void safetyReallocMemToWordStructsArray(struct Word*** words_array,
int32_t* count_of_allocated_words)
{
	// пробуем выделить память
	struct Word** new_words_array = (struct
Word**)realloc(*words_array, (*count_of_allocated_words +
BLOCK_SIZE)*sizeof(struct Word*));

	// если указатель на buffer == NULL, значит произошла ошибка
	if (new_words_array == NULL)
	{
		// если память изначально указывала на NULL, это значит что мы
её выделяли впервые
		// выводим соответствующее сообщение об ошибке
		if (*words_array == NULL)
		{
			fwprintf(stderr, L"Error: failed to allocate memory\n");
			exit(EXIT_FAILURE);
		}
		// иначе была ошибка при перевыделении
		else
		{
			fwprintf(stderr, L"Error: failed to re-allocate
memory\n");
			exit(EXIT_FAILURE);
		}
	}
	else
	{
		for (uint32_t i = *count_of_allocated_words; i <
*count_of_allocated_words + BLOCK_SIZE; i++)
		{
			new_words_array[i] = (struct Word*)malloc(sizeof(struct
Word));
			new_words_array[i]->word = NULL;
			new_words_array[i]->punct = L'\0';
			new_words_array[i]->len = 0;
			new_words_array[i]->allocated_size = 0;
			//wprintf(L"Allocate addr: %p\n", new_words_array[i]);


			if (new_words_array[i] == NULL)
```

```
                {
                        fwprintf(stderr, L"Error: failed to allocate memory
to struct Word\n");
                        exit(EXIT_FAILURE);
                }
        }

        *words_array = new_words_array;
        *count_of_allocated_words += BLOCK_SIZE;
    }
}
```

Название файла:

safety_realloc_functions/safetyReallocMemToWordStructsArray.h

```
#ifndef SAFETY_REALLOC_MEM_TO_WORD_STRUCTS_ARRAY_H
#define SAFETY_REALLOC_MEM_TO_WORD_STRUCTS_ARRAY_H

#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "../structs.h"
#include "../macro.h"

void safetyReallocMemToWordStructsArray(struct Word*** words_array,
int32_t* count_of_allocated_words);

#endif
```

Название файла:

```
safety_realloc_functions/safetyReallocMemToSentenceStructsArray.c
#include "./safetyReallocMemToSentenceStructsArray.h"

void safetyReallocMemToSentenceStructsArray(struct Sentence***
sentences_array, int32_t* count_of_allocated_sentences)
{
    // пробуем выделить память
    struct Sentence** new_sentences_array = (struct
Sentence**)realloc(*sentences_array, (*count_of_allocated_sentences +
BLOCK_SIZE)*sizeof(struct Sentence*));

    // если указатель на buffer == NULL, значит произошла ошибка
    if (new_sentences_array == NULL)
    {
```

```
            // если память изначально указывала на NULL, это значит что мы
её выделяли впервые
            // выводим соответствующее сообщение об ошибке
            if (*sentences_array == NULL)
            {
                  fwprintf(stderr, L"Error: failed to allocate memory\n");
                  exit(EXIT_FAILURE);
            }
            // иначе была ошибка при перевыделении
            else
            {
                  fwprintf(stderr, L"Error: failed to re-allocate memory,
the part of sequence that has already been read will be returned\n");
                  exit(EXIT_FAILURE);
            }
      }
      else
      {
            for (uint32_t i = *count_of_allocated_sentences; i <
*count_of_allocated_sentences + BLOCK_SIZE; i++)
            {
                  new_sentences_array[i] = (struct
Sentence*)malloc(sizeof(struct Sentence));
                  new_sentences_array[i]->words_array = NULL;
                  new_sentences_array[i]->is_last = 0;
                  new_sentences_array[i]->len = 0;
                  new_sentences_array[i]->allocated_size = 0;
                  //wprintf(L"Allocate addr: %p\n",
new_sentences_array[i]);


                  if (new_sentences_array[i] == NULL)
                  {
                        fwprintf(stderr, L"Error: failed to allocate memory
to struct Sentence\n");
                        exit(EXIT_FAILURE);
                  }
            }
            *sentences_array = new_sentences_array;
            *count_of_allocated_sentences += BLOCK_SIZE;
      }
}
```

Название файла:
safety_realloc_functions/safetyReallocMemToSentenceStructsArray.h

```
#ifndef SAFETY_REALLOC_MEM_TO_SENTENCE_STRUCTS_ARRAY_H
#define SAFETY_REALLOC_MEM_TO_SENTENCE_STRUCTS_ARRAY_H

#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "../structs.h"
#include "../macro.h"

void safetyReallocMemToSentenceStructsArray(struct Sentence***
sentences_array, int32_t* count_of_allocated_sentences);

#endif
```

Название файла:

```
shifting_functions/shiftingWordsInText.c
#include "./shiftingWordsInText.h"

void shiftingWordsInText(struct Text **text, const uint32_t
number_of_shifts)
{
    // если подано количество сдвигов, равное 0, то обрабатывать далее
смысла нет
    if (number_of_shifts > 0 && (*text)->len >= 2)
    {
        for (uint32_t i = 0; i < (*text)->len; i++)
        {
            struct Sentence *sentence = (*text)->sentences_array[i];

            // убираем точку у последнего слова, чтобы в конце
поставить её после нового последнего слова
            sentence->words_array[sentence->len - 1]->punct = '\0';
            struct Word** new_words_array = (struct
Word**)malloc(sentence->allocated_size*sizeof(struct Word*));

            uint32_t new_ind;
            for(uint32_t j = 0; j < sentence->len; j++) {
                new_ind = (j + number_of_shifts)%(sentence->len);
                new_words_array[new_ind] = sentence-
>words_array[j];
            }

            for(uint32_t j = sentence->len; j < sentence-
>allocated_size; j++) {
```

```
                    new_words_array[j] = sentence->words_array[j];
                }
                free(sentence->words_array);
                sentence->words_array = new_words_array;
            }
        }
}
```

Название файла:
shifting_functions/shiftingWordsInText.h

```
#ifndef SHIFTING_WORDS_IN_TEXT_H
#define SHIFTING_WORDS_IN_TEXT_H

#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "../structs.h"

void shiftingWordsInText(struct Text **text, const uint32_t
number_of_shifts);

#endif
```

Название файла:
macro.h

```
#ifndef MACRO_H
#define MACRO_H

#define BLOCK_SIZE 5

#endif
```

Название файла:
structs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H

#include <wchar.h>
#include <stdint.h>

struct Word
{
```

```
      wchar_t* word;
      wchar_t punct;
     uint32_t len;
     uint32_t allocated_size;
};

struct Sentence
{
      struct Word** words_array;
      uint8_t is_last;
      uint32_t len;
      uint32_t allocated_size;
};

struct Text
{
      struct Sentence** sentences_array;
      uint32_t len;
      uint32_t allocated_size;
};

#endif
```

Название файла:
textStructToString.c

```
#include "./textStructToString.h"

wchar_t *textStructToString(struct Text **text)
{
      uint32_t count_of_read_chars = 0;                  // хранит
количество считанных символов
      uint32_t count_of_allocated_chars = 0;

      if ((*text)->len == 0) {
            wprintf(L"\033[33mWarning:\033[0m text is empty\n");
            freeText(text);
            exit(EXIT_SUCCESS);
      }

      // объявляем и выделяем память под строку, которая будет хранить
текст единой строкой
      wchar_t *string_text = NULL;
      safetyReallocMemToWStr(&string_text, &count_of_allocated_chars);
```

```
	// итерируемся по предложениям текста
	for (uint32_t i = 0; i < (*text)->len; i++)
	{
		// итерирование по словам
		for (uint32_t j = 0; j < (*text)->sentences_array[i]->len; j+
+)
		{
			// итерирование по символам в слове
			for (uint32_t k = 0; k < (*text)->sentences_array[i]-
>words_array[j]->len; k++)
			{
				// проверяем, что памяти ещё хватает
				if (count_of_read_chars >=
count_of_allocated_chars)
				{
					safetyReallocMemToWStr(&string_text,
&count_of_allocated_chars);
				}
				// записываем очередной символ
				string_text[count_of_read_chars++] = (*text)-
>sentences_array[i]->words_array[j]->word[k];
			}

			// если знак препинания у слова есть
			if ((*text)->sentences_array[i]->words_array[j]->punct !
= L'\0')
			{
				// проверяем, что памяти ещё хватает
				if (count_of_read_chars ==
count_of_allocated_chars)
				{
					safetyReallocMemToWStr(&string_text,
&count_of_allocated_chars);
				}
				// записываем знак препинания в строку
				string_text[count_of_read_chars++] = (*text)-
>sentences_array[i]->words_array[j]->punct;
			}

			// проверяем, что памяти ещё хватает
			if (count_of_read_chars == count_of_allocated_chars)
			{
				safetyReallocMemToWStr(&string_text,
&count_of_allocated_chars);
```

```
                }

                // ставит пробел между словами
                string_text[count_of_read_chars++] = L' ';
            }
        }

        // снова проверяем, что памяти ещё хватает
        if (count_of_read_chars == count_of_allocated_chars)
        {
            safetyReallocMemToWStr(&string_text,
&count_of_allocated_chars);
        }

        // переходим на новую строку и ставим знак конца строки
        string_text[count_of_read_chars - 1] = '\n';
        string_text[count_of_read_chars] = L'\0';

        return string_text;
}
```

Название файла:
textStructToString.h

```
#ifndef TEXT_STRUCT_TO_STRING_H
#define TEXT_STRUCT_TO_STRING_H

#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#include "./structs.h"
#include "./safety_realloc_functions/safetyReallocMemToWStr.h"
#include "./free_functions/freeText.h"

wchar_t *textStructToString(struct Text **text);

#endif
```