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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка строк на языке Си

Студент гр. 0382 _____ Куликов М.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент : Куликов М.Д.

Группа 0382

Тема работы: Обработка строк на языке Си

Вариант 20

Исходные данные:

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

Программа должна сохранить этот текст в динамический массив строк и оперировать далее только с ним.

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. Во всем тексте удалить слово введенное пользователем. Если после удаления в предложении не останется слов, его необходимо удалить.
2. Для каждого предложения вывести все заглавные буквы в лексикографическом порядке.
3. Отсортировать предложения по среднему арифметическому чисел в предложении. Число - слово состоящее только из цифр.
4. Удалить все предложения в которых нет строчных букв.

Все сортировки должны осуществляться с использованием функции стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 02.11.2020

Дата сдачи реферата: 27.12.2020
Дата защиты реферата: 29.12.2020

Студент _____ Куликов М.Д.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

В процессе выполнения курсовой работы была написана программа, выполняющая обработку текста в зависимости от пожелания пользователя. Пользователю доступно 4 опции обработки на выбор и опция выхода из программы. Программа была реализована с помощью символьных массивов и массивов указателей, единожды используются структуры. Память для хранения и обработки данных выделяется динамически. Для удобства пользователя в консоли выводится меню с описанием возможных опций. Принекорректном выборе номера опции выводится сообщение об ошибке и предлагается заново выбрать опцию.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы с опциями обработки текста на языке Си.

Основные задачи:

1) Ввод, первоначальная обработка и хранение текста.

2)Написание функции ,выполняющей удаление введенного слова из всего текста.

3)Написание функции , создающей массив из заглавных букв предложения в лексикографическом порядке.

4)Написание функции, удаляющей предложения без строчных букв.

5)Огранизация вывода данных и освобождения памяти.

Для всех задач было использовано динамическое выделение памяти, массивы, функции обработки строк, а также функция qsort.

1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1. Считывание текста и начальная обработка текста.

Для считывания текста и его начальной обработки было реализовано несколько функций:

1) char *input_sentence() - функция, которая динамически выделяет память под предложение и считывает его в переменную sentence. В результате функции она возвращает массив char* sentence или строку «end_of_input» в случае, когда пользователь ввел два символа перевода строки.

2)char *no_spaces(char *sentence) — функция, убирающая лишние пробелы и знаки табуляции в начале каждого предложения.

3) char **input_text(int *size) — функция , которая динамически выделяет память под массив строк и записывает в него предложения, обработанные с помощью функции no_spaces до тех пор, пока функция считывания предложения не вернет строку «end_of_input». Также в этой функции присутствует важная переменная sent_count, которая отображает количество предложений в тексте в данный момент.

4)char **text_without_repeats(char **text, int *sent_count) — функция, сравнивающая все предложения между собой и удаляющая повторно встречающиеся предложения (сравнение происходит без учета регистра с помощью функции strcasecmp).

1.2 Первая опция

Первая опция обработки текста была реализована с помощью функции char **delete_word(char **text, int *sent_count), принимающей на вход изначальный текст и количество предложений в нем.

В начале с помощью алгоритма, схожим в алгоритмом считывания предложения считывается слово, которое хочет удалить пользователь (концом ввода слова является символ переноса строки).

После этого запускается для каждого предложения запускается цикл, который заканчивается, когда данного слова больше нет в предложении (это показывает функция strstr). Если данное слово есть в предложении, то начиная с адреса начала этого слова в предложении оно удаляется из предложения с помощью сдвига символов и добавления символа окончания строки в конец преобразованного предложения. В данной функции предусмотрены исключения, чтобы предложение после вывода выглядело корректно и не содержало в себе лишних символов. В конце функции удаляются пустые предложения из текста тем же методом сдвига и изменением длины массива строк. Функция возвращает измененный текст.

1.3 Вторая опция.

Для реализации второй опции была написана функция char **upper_letters_sort(char **text, int *sent_count, int *new_sent_count), принимающая на вход текст, его размер и ссылку на переменную, в которую будет записан размер нового массива.

В самой функции в начале происходит динамическое выделение памяти на массив строк, содержащих заглавные буквы и на сами эти строки.

Далее с помощью функции isupper находятся все заглавные буквы в предложениях и они записываются в массив заглавных букв с соответствующим индексом. В случае, если в предложении нет заглавных букв, то для него ничего не выводится.

В конце функции используется функция qsort в которую передаются соответствующие аргументы и функия int letters_comparator(const void *a, const void *b) в которой сравниваются коды букв по таблице ASCII , что соответствует лексикографическому порядку.

Функция возвращает массив с заглавными буквами соответствующих предложений.

1.4 Третья опция

Для реализации третьей опции была написана функция void sort_by_numbers(char **text, int sent_count) и реализована структура numbers_with_sentences.

В начале функции был создан массив для среднимих арифметических чисел каждого предложения. Далее с помощью функций isdigit , isalpha и флагов были считаны и сложены числа из предложений и записаны в массим средних арифметических. Если в предложении нет чисел, то в массива средних арифметических по соответствующему индексу записывается отрицательное значение, которое в последствии увеличивается, что позволяет вывести все предложения без чисел в начале в порядке их ввода пользователем.

Далее был создан массив структур final_array и в его поля number и sentence по соответствующим индексам были записаны предложения и их средние арифметические.

После этого с помощью функции qsort и передаваемой в нее функции int numbers_comparator(const void *a, const void *b) были отсортированы структуры по возрастанию среднего арифметического.

В конце функции производится вывод полей sentence (предложений) всех структур из массива.

1.5 Четвертая опция

Для реализации четвертой опции была написана функция void delete_without_lowers(char **text, int *sent_len).

В ней с помощью функции islower и флагов были и удалены предложения, в которых отсутствуют строчные буквы. Размер массива строк также изменяется в соответствии с количеством удаленных предложений.

1.6 Вспомогательные функции вывода и освобождения памяти.

Для читаемости кода программы были написаны вспомогательные функции вывода и освобождения памяти.

1)void print_text(char **text, int sent_count) — функция, печатающая текст.

2)void free_text(char **text, int sent_count) — функция, освобождающая память, выделенную под текст.

3)void print_and_free_letters_and_text — функция, которая печатает массив строк с заглавными буквами предложений, освобождает память выделенную под него и освобождает память, выделенную под текст.

1.7 Функция main.

В начале главной функции выводится подсказка для пользователя, связанная с форматом ввода текста и используются функции input_text и text_without_repeats, с помощью чего получается готовый для обработки с помощью опций текст.

После этого печатается подсказка с описанием опций, которые может выбрать пользователь.

Если пользователь вводит некорректный номер опции, то выбор опции повторяется до тех пор, пока не исполнится одна из опций.

Обработка выбора опции с помощью ввода цифры реализована с помощью конструкции switch — case — default.

Для того, чтобы убрать лишний символ перевода строки после ввода номера опции используется функция fgetc.

В зависимости от выбранной опции выполняются соответствующие функции, после чего выполняются вспомогательные функции вывода и освобождения памяти.

Статус выполненности обработки отслеживается с помощью флага prog_done.

2. ОПИСАНИЕ ТРЕБОВАНИЙ К ВХОДНЫМ ДАННЫМ И ОПИСАНИЕ ВЫХОДНЫХ ДАННЫХ.

На вход программе подается текст, состоящий из предложений, разделенных точкой. Слова в тексте могут быть разделены запятой или пробелом. Предложения могут вводиться с новой строки. Для окончания ввода текста требуется ввести два символа перевода строки подряд (два раза нажать Enter).

Если во второй опции в каком-либо предложении не оказывается заглавных букв, то вместо него ничего не выводится, а остальные строки с буквами выводятся в порядке ввода соответствующих предложений.

Если в третьей опции в каких-либо предложенях не оказалось чисел, то эти предложения выводятся перед предложениями с числами в порядке их ввода.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы был получен опыт работы с массивами и указателями, динамическим выделением памяти и его освобождением, а также функциями стандартных библиотек. Была освоена и использована функция qsort.

В ходе выполнения работы была достигнута цель — была написана программа, обрабатывающая текст, введенный пользователем и соответствующая условию задания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт http://cppstudio.com

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1 — Выполнение первой опции (удаление слова snow).

```
Введите текст для обработки. Вы можете вводить предложения с новой строки. Для окончания ввода нажмите 2 раза клавишу Enter.

winter snow.
snow snow.
autumn winter snow.
winter.
winter snow winter.

Выберите желаемую опцию из перечисленных
1.Во всем тексте удалить слово введенное пользователем. Если после удаления в предложении не останется слов, его необходимо удалить.
2.Для каждого предложения вывести все заглавные буквы в лексикографическом порядке.
3.Отсортировать предложения по среднему арифметическому чисел в предложении. Число - слово состоящее только из цифр.
4.Удалить все предложения в которых нет строчных букв.
5.Выйти из программы

1
Введите слово, которое вы хотите удалить из текста:
snow

winter.
autumn winter.
winter.
winter winter.
```

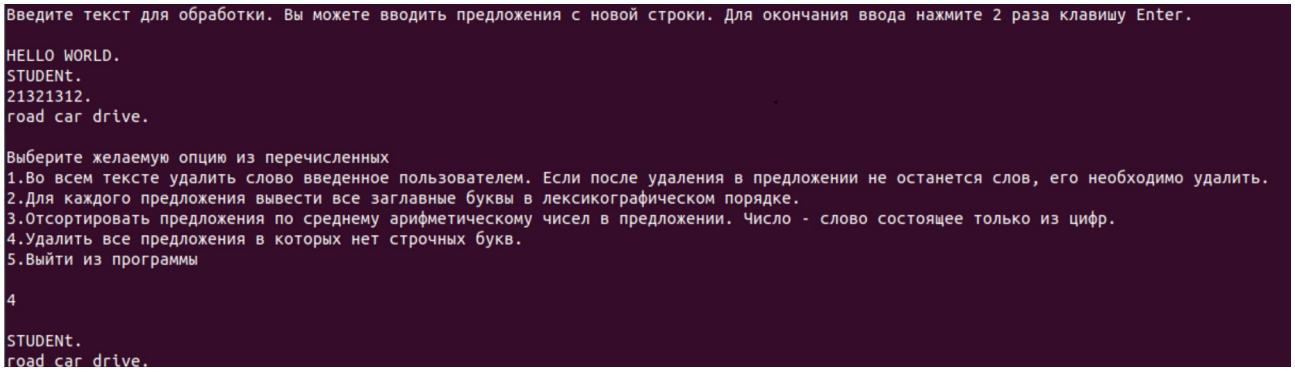
Пример 2 — Выполнение второй опции (вывод заглавных букв предложений).

```
Введите текст для обработки. Вы можете вводить предложения с новой строки. Для окончания ввода нажмите 2 раза клавишу Enter.

HellO,my name Is Maksim KulIkov.
text text text 123.
I AM 18 yEarS old.

Выберите желаемую опцию из перечисленных
1.Во всем тексте удалить слово введенное пользователем. Если после удаления в предложении не останется слов, его необходимо удалить.
2.Для каждого предложения вывести все заглавные буквы в лексикографическом порядке.
3.Отсортировать предложения по среднему арифметическому чисел в предложении. Число - слово состоящее только из цифр.
4.Удалить все предложения в которых нет строчных букв.
5.Выйти из программы

2
HIIKMO
AEIMS
```

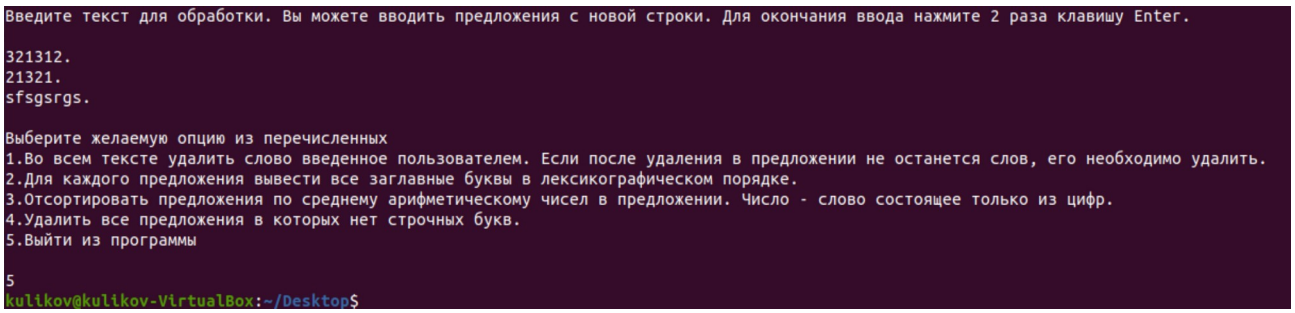
Пример 3 — Выполнение третьей опции (вывод предложений по возрастанию среднего арифметического чисел).

```
Введите текст для обработки. Вы можете вводить предложения с новой строки. Для окончания ввода нажмите 2 раза клавишу Enter.

ttt 123 tt11144 123.
ttt lll bbb nnn.
345 345 34fg56.
tyhnjg.

Выберите желаемую опцию из перечисленных
1.Во всем тексте удалить слово введенное пользователем. Если после удаления в предложении не останется слов, его необходимо удалить.
2.Для каждого предложения вывести все заглавные буквы в лексикографическом порядке.
3.Отсортировать предложения по среднему арифметическому чисел в предложении. Число - слово состоящее только из цифр.
4.Удалить все предложения в которых нет строчных букв.
5.Выйти из программы

3
ttt lll bbb nnn.
tyhnjg.
ttt 123 tt11144 123.
345 345 34fg56.
```

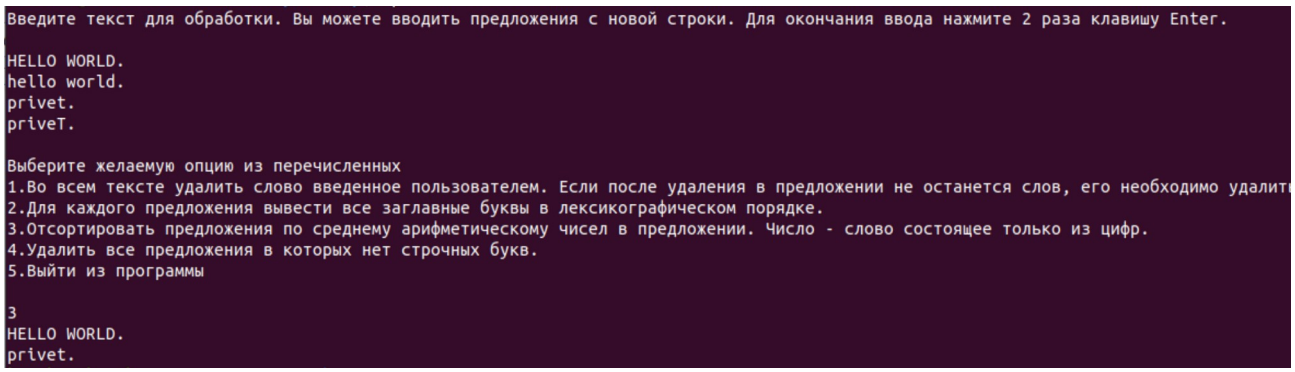
Пример 4 — Выполнение четвертой опции (удаление предложений без строчных букв).



Пример 5 — Выход из программы.



Пример 6 — Удаление повторяющихся предложений. (опция 3 ничего не изменяет в данном примере)



ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>


struct numbers_with_sentences {
    char *sentence;
    int number;
};

char **delete_word(char **text, int *sent_count) {
    char *word = malloc(20);
    int word_size = 20;
    int word_len = 0;
    char c = ' ';
    while (c != '\n') {
        c = (char) getchar();
        word[word_len] = c;
        if (word[word_len] == '\n') {
            word[word_len] = '\0';
            break;
        }
        word_len++;
        if (word_len == word_size) {
            word_size += 20;
            word = realloc(word, word_size);
        }
    }
    char *substr_pointer;
    int k;
    int j;
    for (int i = 0; i < *sent_count; i++) {
        while ((substr_pointer = strstr(text[i], word)) != NULL) {
            if (*(substr_pointer + strlen(word)) == '.' && (substr_pointer - text[i]) != 0) {
                for (j = 0; j < strlen(word) + 1; j++) {
                    for (k = substr_pointer - text[i] - 1; k < strlen(text[i]); k++) {
                        text[i][k] = text[i][k + 1];
                    }
```

```
            }
        } else {
            for (j = 0; j < strlen(word) + 1; j++) {
                for (k = substr_pointer - text[i]; k < strlen(text[i]); k++) {
                    text[i][k] = text[i][k + 1];
                }
            }

        }
        text[i][k] = '\0';
    }
    if (text[i][0] == '\0') {
        for (int n = i; n < *sent_count - 1; n++) {
            text[n] = text[n + 1];
        }
        i -= 1;
        *sent_count = *sent_count - 1;
        text = realloc(text, *sent_count * sizeof(char *));

    }

  }
  return text;
}


int letters_comparator(const void *a, const void *b) {
  char *arr1 = (char *) a;
  char *arr2 = (char *) b;

  return *arr1 - *arr2;
}


char **upper_letters_sort(char **text, int *sent_count, int *new_sent_count) {
  int arr_size = 30;
  char **upper_letters = malloc(arr_size);
  for (int i = 0; i < *sent_count; i++) {
    int letter_counter = 0;
    char *sentence_letters = malloc(20);
    int sentence_letters_size = 20;
    for (int j = 0; j < strlen(text[i]) - 1; j++) {
      if (isupper(text[i][j])) {
        sentence_letters[letter_counter] = text[i][j];
```

```
            letter_counter++;
            if (sentence_letters_size == letter_counter) {
                sentence_letters_size += 20;
                sentence_letters = realloc(sentence_letters, sentence_letters_size);
            }
        }
    }

    sentence_letters[letter_counter] = '\0';
    if (i == arr_size) {
        arr_size += 30;
        upper_letters = realloc(upper_letters, arr_size);
    }
    if (sentence_letters[0] != '\0') {
        upper_letters[*new_sent_count] = sentence_letters;
                    qsort(upper_letters[*new_sent_count], letter_counter, sizeof(char),
letters_comparator);
        *new_sent_count += 1;
    }
  }
  return upper_letters;
}


char *input_sentence() {
  int sentence_size = 100, sym_count = 0;
  char sym = 0;
  int n_counter = 0;
  char *sentence = malloc(sentence_size * sizeof(char));
  while (sym != '.') {
    sym = (char) getchar();
    if (sym == '\n') {
      n_counter++;
      if (n_counter == 2) {
        free(sentence);
        return "end_of_input";
      }
    } else {
      sentence[sym_count] = sym;
      sym_count += 1;
      n_counter = 0;
      if (sym_count == sentence_size) {
        sentence_size = sentence_size + 100 * (int) sizeof(char);
        sentence = realloc(sentence, sentence_size);
```

```
            }
        }
    }
    sentence[sym_count + 1] = '\0';
    return sentence;

}


char **text_without_repeats(char **text, int *sent_count) {
    for (int i = 0; i < *sent_count - 1; i++) {
        for (int j = i + 1; j < *sent_count; j++) {
            if (!strcasecmp(text[i], text[j])) {
                for (int n = j; n < *sent_count - 1; n++)
                    text[n] = text[n + 1];
                *sent_count -= 1;
                j -= 1;
            }
        }
    }
    return text;
}


char *no_spaces(char *sentence) {
    if (strcmp(sentence, "end_of_input") != 0) {
        while (sentence[0] == ' ' || sentence[0] == '\t') {
            int i;
            for (i = 0; i < strlen(sentence) - 1; i++) {
                sentence[i] = sentence[i + 1];
            }
            sentence[i] = '\0';
        }
        sentence = realloc(sentence, strlen(sentence));
    }
    return sentence;
}


void print_menu() {
    printf("Выберите желаемую опцию из перечисленных\n"
           "1.Во всем тексте удалить слово введенное пользователем. Если после удаления в предложении не останется слов, его необходимо удалить.\n"
```

```
                "2.Для каждого предложения вывести все заглавные буквы в лексикографическом порядке.\n"
                "3.Отсортировать предложения по среднему арифметическому чисел в предложении. Число - слово состоящее только из цифр.\n"
        "4.Удалить все предложения в которых нет строчных букв.\n"
        "5.Выйти из программы\n\n");
}


char **input_text(int *size) {
    int text_size = 10;
    int sent_count = 0;
    char **text = malloc(text_size * sizeof(char *));
    while (1) {
        text[sent_count] = input_sentence();
        text[sent_count] = no_spaces(text[sent_count]);
        if (!strcmp(text[sent_count], "end_of_input"))
            break;
        sent_count += 1;
        if (sent_count == text_size) {
            text_size = text_size + 10 * (int) sizeof(char *);
            text = realloc(text, text_size);
        }
    }
    *size = sent_count;
    return text;
}


void print_text(char **text, int sent_count) {
          printf("\n");
    for (int i = 0; i < sent_count; i++)
        printf("%s\n", text[i]);
}

void free_text(char **text, int sent_count) {
    for (int i = 0; i < sent_count; i++)
        free(text[i]);
    free(text);
}


void print_and_free_letters_and_text(char **upper_letters, int new_sent_count, char **text, int sent_count) {
```

```
    for (int i = 0; i < new_sent_count; i++)
        printf("%s\n", upper_letters[i]);
    for (int i = 0; i < new_sent_count; i++)
        free(upper_letters[i]);
    free(upper_letters);
    for (int i = 0; i < sent_count; i++)
        free(text[i]);
    free(text);
}

int numbers_comparator(const void *a, const void *b) {
    struct numbers_with_sentences *a1 = (struct numbers_with_sentences *) a;
    struct numbers_with_sentences *a2 = (struct numbers_with_sentences *) b;

    return a1->number - a2->number;
}

void sort_by_numbers(char **text, int sent_count) {
    float avg_numbers_array[sent_count];
    float no_numbers = 0;
    float num_counter = 0, num = 0;
    float sum = 0;
    int flag;
    for (int i = 0; i < sent_count; i++) {
        flag = 0;
        for (int j = 0; j < strlen(text[i]); j++) {
            if (isalpha(text[i][j]))
                flag = 0;
            if (isdigit(text[i][j]) && (flag == 1 || j == 0)) {
                num = num * 10 + text[i][j] - 48;
                flag = 1;
            }
            if (text[i][j] == ' ' || text[i][j] == ',' || text[i][j] == '.') {
                if (flag == 1) {
                    sum += num;
                    num_counter += 1;
                    num = 0;
                } else
                    flag = 1;
            }
        }
        if (num_counter != 0) {
            avg_numbers_array[i] = sum / num_counter;
            sum = 0;
```

```
            num_counter = 0;
        } else {
            avg_numbers_array[i] = -(sent_count) + no_numbers;
            no_numbers += 1;
        }
    }

    struct numbers_with_sentences final_array[sent_count];
    for (int i = 0; i < sent_count; i++) {
        final_array[i].sentence = text[i];
        final_array[i].number = avg_numbers_array[i];
    }
                qsort(final_array, sent_count, sizeof(struct numbers_with_sentences),
numbers_comparator);
    for (int i = 0; i < sent_count; i++) {
        printf("%s\n", final_array[i].sentence);
    }

}

void delete_without_lowers(char **text, int *sent_len) {
    int i, j;
    for (i = 0; i < *sent_len; i++) {
        int flag = 0;
        for (j = 0; j < strlen(text[i]); j++) {
            if (islower(text[i][j]))
                flag = 1;
        }
        if (flag == 0) {
            for (int n = i; n < *sent_len - 1; n++) {
                text[n] = text[n + 1];
            }
            i -= 1;
            *sent_len -= 1;
            text = realloc(text, *sent_len * sizeof(char *));
        }


    }


}
```

```
int main() {
    char **text;
    char **upper_letters;
    int sent_count, prog_done = 0, choice, new_sent_count;
     printf("Введите текст для обработки. Вы можете вводить предложения с новой строки. Для окончания ввода нажмите 2 раза клавишу Enter.\n\n");
    text = input_text(&sent_count);
    text = text_without_repeats(text, &sent_count);
    text = realloc(text, sent_count * sizeof(char *));
    print_menu();
    scanf("%d", &choice);
    fgetc(stdin);

    while (prog_done != 1) {

        switch (choice) {
            case 1:
                printf("Введите слово, которое вы хотите удалить из текста:\n");
                text = delete_word(text, &sent_count);
                print_text(text, sent_count);
                free_text(text, sent_count);
                prog_done = 1;
                break;

            case 2:
                upper_letters = upper_letters_sort(text, &sent_count, &new_sent_count);
                        print_and_free_letters_and_text(upper_letters, new_sent_count, text, sent_count);
                prog_done = 1;
                break;
            case 3:
                sort_by_numbers(text, sent_count);
                free_text(text, sent_count);
                prog_done = 1;
                break;
            case 4:
                delete_without_lowers(text, &sent_count);
                print_text(text, sent_count);
                free_text(text, sent_count);
                prog_done = 1;
                break;
            case 5:
                prog_done = 1;
                break;
```

```
            default:
                printf("Некорректный номер опции,попробуйте снова\n");
                scanf("%d", &choice);
                fgetc(stdin);
        }

    }

    return 0;
}
```