

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Работа с текстом в Си

Студентка гр. 0382

Михайлова О.Д.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Михайлова О.Д.

Группа 0382

Тема работы: работа с текстом в Си

Исходные данные:

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.

Содержание пояснительной записки: «Аннотация», «Содержание», «Задание», «Ход работы», «Заключение», «Список использованных источников», «Примеры работы программы», «Исходный код программы».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 02.11.2020

Дата сдачи реферата: 28.11.2020

Дата защиты реферата: 29.11.2020

Студентка

Михайлова О.Д.

Преподаватель

Жангиров Т.Р.

АННОТАЦИЯ

Разработана программа на языке Си, которая обрабатывает текст, введенный пользователем. Для работы с текстом используются структуры и символьные массивы, для которых выделяется динамическая память. При обработке входных данных программа удаляет все повторно встречающиеся предложения. Далее выводится меню возможных действий. При выборе несуществующего варианта, программа печатает сообщение об ошибке. При корректном выборе реализуется необходимое действие. Также производятся действия по очистке выделенной памяти.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

СОДЕРЖАНИЕ

	Введение	5
1.	Задание	6
2.	Ход работы	8
2.1.	Создание и объявление структур	8
2.2.	Функции для считывания текста	9
2.3.	Функции для обработки текста	10
2.4.	Функция main	11
2.5.	Создание Makefile	12
	Заключение	14
	Список использованных источников	15
	Приложение А. Примеры работы программы	16
	Приложение Б. Исходный код программы	18

ВВЕДЕНИЕ

Цель работы:

Написать программу, выполняющую действия по обработке текста, выбранные пользователем. Реализация программы происходит с помощью использования структур (для хранения текста, предложений и некоторых данных о них), стандартных библиотек (в том числе `wchar.h` и `wctype.h` для работы с кириллическими буквами) и выделения динамической памяти. Сборка программы происходит с помощью `Makefile` и утилиты `make`.

Для достижения поставленной цели требуется решить следующие задачи: изучить синтаксис языка программирования C, разработать код программы, написать `Makefile`, собрать проект, протестировать работоспособность программы.

Программа разработана для операционных систем на базе Linux.

Разработка производилась на операционной системе Ubuntu Linux в IDE Clion и редакторе Vim.

1. ЗАДАНИЕ

Вариант 10

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) Для каждого предложения вывести “Палиндромы на месте!” если все слова являются палиндромами, “Кто-то потерял все палиндромы.” если ни одного слово не является палиндромом, в остальных случаях “Чего-то не хватает.”.

2) Заменить все специальные символы (включая .,:;) в тексте на подстроку “>special symbol<”.

3) Отсортировать предложения по уменьшению количества латинских букв.

4) Удалить все предложения в которых в одном слове встречаются латинские и кириллические буквы.

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 РАБОТЫ

2.1. Создание и объявление структур

Для работы с текстом созданы структуры Sentence и Text:

```
struct Sentence {
    wchar_t* sentence;
    wchar_t** words;
    int count_w;
    int* len_w;
    int len_s;
    int size_sent;
    int final;
};

struct Text {
    struct Sentence* sentences;
    int len_t;
    int size_text;
};
```

Переменные структуры Sentence:

- wchar_t* sentence – указатель на массив с предложением;
- wchar_t** words – массив указателей на слова в предложении;
- int count_w – переменная, в которую записано количество слов в предложении;
- int* len_w – указатель на массив, в котором записаны длины каждого слова предложения;
- int len_s – переменная, в которую записана длина предложения;
- int size_sent – переменная, в которую записывается размер памяти, выделяемой в динамической памяти для хранения предложения;
- int final – переменная, которая необходима для определения конца ввода;

Переменные структуры Text:

- struct Sentence* sentences – указатель на массив предложений и их данных;

- `int len_t` – переменная, в которую записано количество предложений в тексте;
- `int size_text` - переменная, в которую записывается размер памяти, выделяемой в динамической памяти для хранения текста;

Структуры объявлены в заголовочном файле `structures.h`.

2.2. Функции для считывания текста

Считывания текста реализуется с помощью функций `read_sentence()` и `read_text()`.

Функция `read_sentence()` ничего не принимает на вход и возвращает структуру `Sentence`. Считывание предложения происходит посимвольно с помощью функции `fgetc` в цикле `do while` до тех пор, пока введенный символ не равен точке. Перед этим выделяется блок памяти для хранения предложения с помощью функции `malloc` и затем, после считывания символа, при необходимости блок памяти расширяется с помощью функции `realloc`. Если введенный символ – пробел или запятая, то переменная `count_w` увеличивается на 1. Если введенный символ – `'\n'`, то переменной `final` присваивается значение 1 и дальнейшего считывания символа не происходит. После цикла в конец предложения записывается символ `'\0'`. Далее выделяется динамическая память для массива указателей на слова в предложении и массива, в котором записаны длины каждого слова предложения. После этого с помощью цикла `for` выделяется динамическая память для массивов слов предложения и происходит заполнение этих массивов.

Функция `read_text()` ничего не принимает на вход и возвращает структуру `Text`. Сначала выделяется блок памяти для хранения текста с помощью функции `malloc`. В цикле `while` вызывается функция `read_sentence()` и происходит проверка, есть ли уже в тексте такое предложение. Если предложение не встречалось ранее в тексте и значение переменной `final` не

равно 1, то предложение добавляется в текст и увеличивается переменная `len_t`. При необходимости блок памяти для хранения текста увеличивается с помощью функции `realloc`.

2.3. Функции для обработки текста

- Для реализации первого действия создана функция ***is_palindrom***, которая принимает на вход объект структуры `Sentence`. С помощью цикла `for` посимвольно проверяется каждое слово. Когда слово является палиндромом, то счетчик `count2` увеличивается на 1. Если `count2` равен 0, то функция возвращает 0. Если `count2` равен количеству слов в предложении, то функция возвращает 1. При других значениях `count2` функция возвращает 2.

- Для реализации второго действия создана функция ***special_sym***, которая принимает на вход объект структуры `Sentence`. Сначала создается массив, в котором хранится строка `">special simbol<"`. В цикле `for` проверяется, хватит ли памяти для добавления в предложение 15 символов. Если нет, с помощью функции `realloc` выделяется дополнительный блок памяти. Далее происходит поиск специального символа в предложении. Если такой символ найден, то все символы предложения с конца до специального символа перезаписываются в ячейки, индексы которых на 15 больше. На их места, начиная с ячейки, где хранится специальный символ, записывается строка `">special simbol<"`. Длина предложения при этом увеличивается на 15. Если специальный символ – точка, то строка `">special simbol<"` записывается в конец предложения начиная с ячейки, в которой хранится точка, и затем добавляется символ `'\0'`. Функция возвращает измененный объект структуры `Sentence`.

- Для реализации третьего действия используется функция `qsort` стандартной библиотеки `stdlib.h`, для которой создана функция-компаратор ***sort_sent***. Она принимает на вход два аргумента `const void* arg1` и `const void* arg2`, которые затем приводятся к типу `struct Sentence*`. Далее функция считает

в обоих предложениях количество латинских символов и возвращает результат сравнения.

- Для реализации четвертого действия создана функция `delete_sent`, которая получает на вход объект структуры `Sentence`. В цикле `for` происходит проверка каждого слова предложения на наличие одновременно и кириллического и латинского символов. Если такое слово в предложении есть, то функция возвращает 1, если нет – 0.

2.4. Функция `main`

Для корректной работы с кириллическими символами используется функция `setlocale`.

Далее на экран выводит подсказка для пользователя о том, что нужно ввести текст. Ввод заканчивается символом переноса строки. Затем вызывается функция `read_text()` и введенный текст записывается в переменную `text` типа `struct Text`.

Далее на экран выводит меню действий для обработки текст, одно из которых может выбрать пользователь. Считывание команды, выбранной пользователем, осуществляется с помощью функции `wscanf`. Пользователь может осуществлять выбор действий до тех пор, пока не выберет команду “0”, которая означает выход из меню. Это реализовано с помощью цикла `while`.

С помощью оператора `switch` определяется, какую задачу необходимо выполнить.

Если пользователь выбрал первое действие, то для каждого предложения реализуется функция `is_palindrom`. Значение, которое она вернула присваивается переменной `res1`. В зависимости от полученного значения на экран выводится номер предложения и одно из сообщений: “Кто-то потерял все палиндромы.”, “Палиндромы на месте!”, “Чего-то не хватает.”.

Если пользователь выбрал второе действие, то для каждого предложения реализуется функция `special_sym`.

Если пользователь выбрал третье действие, то реализуется функция `qsort`, которая сортирует все предложения текста по уменьшению количества латинских с помощью функции-компаратора `sort_sentence`.

Если пользователь выбрал четвертое действие, то для каждого предложения реализуется функция `delete_sent`. Значение, которое вернула функция записывается в переменную `res4`. Если это значение “1”, т.е. в предложении есть слово, состоящее из кириллических и латинских символов, то происходит удаление этого предложения из текста:

```
for (j = i; j < text.len_t; j++) {
    text.sentences[j].sentence = text.sentences[j+1].sentence;
    text.sentences[j].len_s = text.sentences[j+1].len_s;
    text.sentences[j].count_w = text.sentences[j+1].count_w;
    text.sentences[j].len_w = text.sentences[j+1].len_w;
    text.sentences[j].words = text.sentences[j+1].words;
}
free(text.sentences[text.len_t].sentence);
for (j = 0; j < text.sentences[text.len_t].count_w; j++) {
    free(text.sentences[text.len_t].words[j]);
}
free(text.sentences[text.len_t].words);
text.len_t -= 1;
```

В любой момент времени пользователь может вывести измененный текст, выбрав команду “5”. Если пользователь выбирает несуществующую команду, то на экран выводит сообщение “Данные некорректны.”.

В конце осуществляется очистка памяти:

```
for (i=0; i<text.len_t; i++){
    for (j=0; j<text.sentences[i].count_w; j++){
        free(text.sentences[i].words[j]);
    }
    free(text.sentences[i].sentence);
}
free(text.sentences);
```

2.5. Создание Makefile

Программа была разделена на следующие файлы:

- Structures.h – заголовочный файл, в котором содержится объявление структур;
- Input_text.h – заголовочный файл, который содержит объявление функций read_sentence и read_text для ввода текста, также в него включены стандартные библиотеки и файл structures.h;
- Input_text.c – файл, который содержит определения функций из файла input_text.h. также в него включены стандартные библиотеки и файл structures.h;
- Funcs_for_tasks.h – файл, который содержит объявление функций обработки текста, а именно is_palindrom, special_sym, delete_sent;
- Funcs_for_tasks.c – файл, который содержит определения функций из файла funcs_for_tasks.h, также в него включены стандартные библиотеки и файл structures.h;
- Other_funcs.h – файл, в котором содержится объявление функции-компаратора sort_sent;
- Other_funcs.c – файл, в котором содержится определение функции sort_sent, также в него включены стандартные библиотеки и файл structures.h;
- Main.c – основной файл, в котором содержится функция main. В него включены все заголовочные файлы.

В каждый заголовочный файл включена директива #pragma once для защиты от повторного подключения заголовочного файла.

Сборка программы осуществляется с помощью Makefile, в котором прописаны все необходимые цели и зависимости, и утилиты make.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была создана программа для обработки текста. Для хранения текста были реализованы структуры Text и Sentence. Программа обрабатывает введенный текст в соответствии с выбором пользователя. Программа может выполнять следующие действия:

1) Для каждого предложения вывести “Палиндромы на месте!” если все слова являются палиндромами, “Кто-то потерял все палиндромы.” если ни одного слово не является палиндромом, в остальных случаях “Чего-то не хватает.”;

2) Заменить все специальные символы (включая .,:;) в тексте на подстроку “>special symbol<”;

3) Отсортировать предложения по уменьшению количества латинских букв;

4) Удалить все предложения, в которых в одном слове встречаются латинские и кириллические буквы;

5) Вывести измененный текст на экран;

6) Выйти из меню действий.

Программа была разбита на файлы, и написан Makefile.

Программа была успешно протестирована на работоспособность.

Примеры тестирования смотреть в приложении А.


СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи Д. Язык программирования Си М.: Вильямс, 1978
288 с.
2. Cplusplus URL: <http://cplusplus.com> (дата обращения: 24.12.2020).

ПРИЛОЖЕНИЕ А

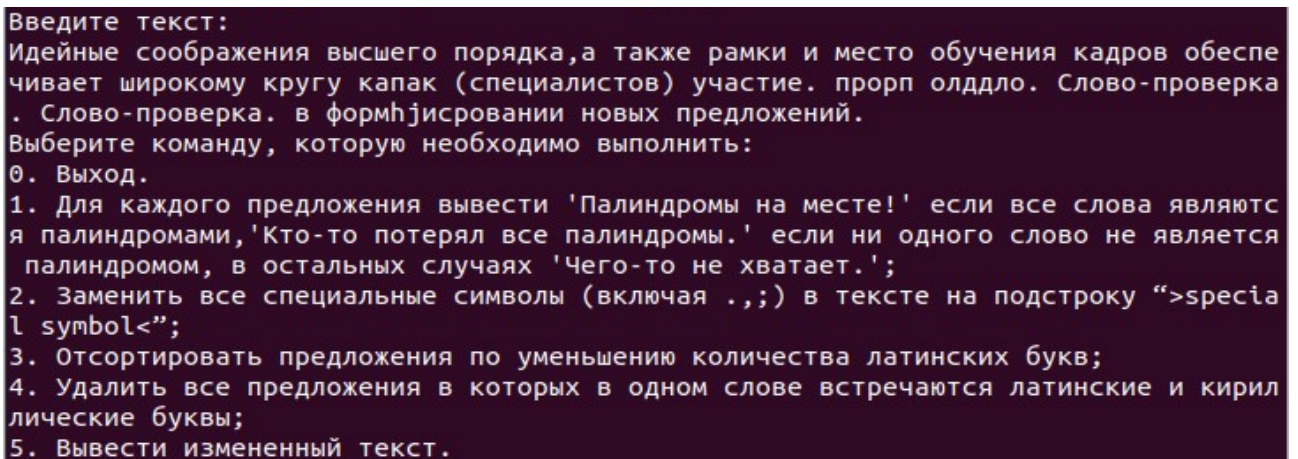
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Вывод подсказки для пользователя после запуска исполняемого файла:



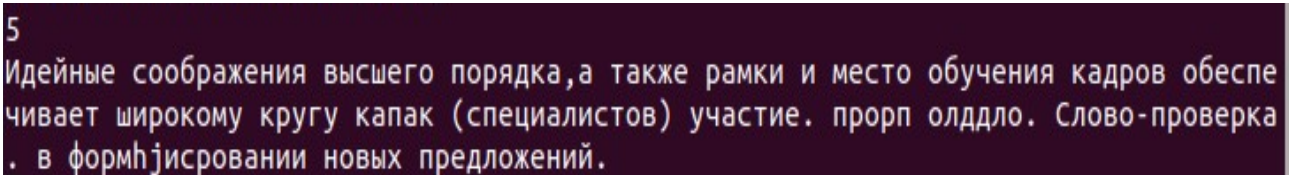
```
Введите текст:
```

Пример введенного текста и вывод меню для пользователя:



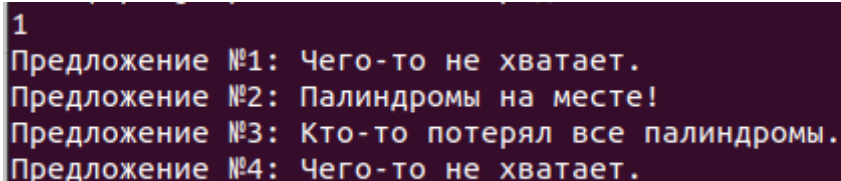
```
Введите текст:
Идейные соображения высшего порядка, а также рамки и место обучения кадров обеспе
чивает широкому кругу капаков (специалистов) участие. прорп олддло. Слово-проверка
. Слово-проверка. в формhjировании новых предложений.
Выберите команду, которую необходимо выполнить:
0. Выход.
1. Для каждого предложения вывести 'Палиндромы на месте!' если все слова являютс
я палиндромами, 'Кто-то потерял все палиндромы.' если ни одного слово не является
палиндромом, в остальных случаях 'Чего-то не хватает.';
2. Заменить все специальные символы (включая .,;) в тексте на подстроку ">specia
l symbol<";
3. Отсортировать предложения по уменьшению количества латинских букв;
4. Удалить все предложения в которых в одном слове встречаются латинские и кирил
лические буквы;
5. Вывести измененный текст.
```

Программа удалила все повторно встречающиеся предложения:



```
5
Идейные соображения высшего порядка, а также рамки и место обучения кадров обеспе
чивает широкому кругу капаков (специалистов) участие. прорп олддло. Слово-проверка
. в формhjировании новых предложений.
```

Выполнение первой задачи:



```
1
Предложение №1: Чего-то не хватает.
Предложение №2: Палиндромы на месте!
Предложение №3: Кто-то потерял все палиндромы.
Предложение №4: Чего-то не хватает.
```

Выполнение третьей задачи:


```
3
5
в формировании новых предложений. Идеиные соображения высшего порядка, а также
рамки и место обучения кадров обеспечивает широкому кругу капаков (специалистов)
участие. проп олдло. Слово-проверка.
```

Выполнение четвертой задачи:

```
4
5
Идеиные соображения высшего порядка, а также рамки и место обучения кадров обеспе-
чивает широкому кругу капаков (специалистов) участие. проп олдло.
```

Выполнение второй задачи:

```
2
5
Идеиные>special symbol<соображения>special symbol<высшего>special symbol<порядка>
>special symbol<а>special symbol<также>special symbol<рамки>special symbol<и>spe-
cial symbol<место>special symbol<обучения>special symbol<кадров>special symbol<о-
беспечивает>special symbol<широкому>special symbol<кругу>special symbol<капак>sp-
ecial symbol<>special symbol<специалистов>special symbol<>special symbol<участие>
>special symbol< проп>special symbol<олдло>special symbol< Слово>special symbol<
l<проверка>special symbol<
```

Вывод сообщения об ошибке при вводе несуществующей команды:

```
7
Данные некорректны.
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл structures.h

```
#pragma once

struct Sentence {
    wchar_t* sentence;
    wchar_t** words;
    int count_w;
    int* len_w;
    int len_s;
    int size_sent;
    int final;
};

struct Text {
    struct Sentence* sentences;
    int len_t;
    int size_text;
};
```

Файл input_text.h

```
#pragma once
#include "structures.h"

struct Sentence read_sentence();
struct Text read_text();
```

Файл input_text.c

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <wctype.h>
#include <wchar.h>
#include <locale.h>

#include "structures.h"
#include "input_text.h"

#define INIT_SIZE 50

struct Sentence read_sentence() {
    struct Sentence Sent;
    Sent.size_sent = INIT_SIZE;
    Sent.len_s = 0;
    Sent.final = 0;
    Sent.count_w = 0;
    int i;
    int j=0;
```

```

int k=0;

Sent.sentence = malloc(Sent.size_sent * sizeof(wchar_t));

wchar_t sym;
sym = (wchar_t) fgetwc(stdin);
if (sym == L'\n') {
    Sent.final = 1;
}
else {
    while (sym == L' ') {
        sym = (wchar_t) fgetwc(stdin);
    }
    Sent.sentence[Sent.len_s++] = sym;
    Sent.count_w = 1;
    do {
        sym = (wchar_t) fgetwc(stdin);
        if ((sym == L' ') || (sym == L',')){
            Sent.count_w += 1;
        }
        Sent.sentence[Sent.len_s++] = sym;
        if (Sent.len_s == Sent.size_sent - 2) {
            Sent.size_sent += INIT_SIZE;
            Sent.sentence = realloc(Sent.sentence,
Sent.size_sent * (sizeof(wchar_t)));
        }
    } while (sym != L'.');
}
if (Sent.len_s > 0) {
    Sent.sentence[Sent.len_s] = L'\0';
}

Sent.words = malloc(Sent.size_sent * sizeof(wchar_t*));
Sent.len_w = malloc(Sent.count_w * sizeof(int));
for (i = 0; i<Sent.count_w; i++){
    Sent.words[i] = malloc(Sent.size_sent * sizeof(wchar_t));
}

for (i = 0; i<(Sent.len_s-1); i++){
    if ((Sent.sentence[i] == L' ') || (Sent.sentence[i] ==
L',')){
        Sent.words[k][j] = L'\0';
        Sent.len_w[k] = j;
        k += 1;
        j = 0;
    }
    else{
        Sent.words[k][j++] = Sent.sentence[i];
    }
    Sent.words[k][j] = L'\0';
    Sent.len_w[k] = j;
}

return Sent;
}

struct Text read_text(){
    int i, j;

```

```

        int not_same = 0;
        struct Text text;
        text.len_t = 0;
        text.size_text = INIT_SIZE;

        text.sentences = malloc(text.size_text * (sizeof(struct
Sentence*)));
        while(1){
            struct Sentence Sent = read_sentence();

            for (i = 0; i<text.len_t; i++) {
                if (wcscasecmp(text.sentences[i].sentence,
Sent.sentence)) {
                    not_same += 1;
                }
            }
            if (Sent.final) {
                break;
            }
            if (not_same == text.len_t) {
                text.sentences[text.len_t++] = Sent;
                if (text.len_t == text.size_text - 2) {
                    text.size_text += INIT_SIZE;
                    text.sentences = realloc(text.sentences,
text.size_text * (sizeof(struct Sentence *)));
                }
            }
            not_same = 0;
        }
        return text;
    }
}

```

Файл funcs_for_tasks.h

```

#pragma once
#include "structures.h"

int is_palindrom(struct Sentence Sent);
struct Sentence special_sym(struct Sentence Sent);
int delete_sent(struct Sentence Sent);

```

Файл funcs_for_tasks.c

```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <wctype.h>
#include <wchar.h>
#include <locale.h>

#include "structures.h"
#include "funcs_for_tasks.h"

#define INIT_SIZE 50

```

```

int is_palindrom(struct Sentence Sent){
    int i, j, count1;
    int count2 = 0;
    for (j=0; j<Sent.count_w; j++) {
        count1 = 1;
        for (i=0; i<(Sent.len_w[j]/2); i++){
            if (Sent.words[j][i] == Sent.words[j][(Sent.len_w[j]-1-
i))){
                count1 *= 1;
            }
            else{
                count1 *= 0;
            }
        }
        if (count1){
            count2 += 1;
        }
    }
    if (count2 == Sent.count_w){
        return 1;
    }
    if (count2 == 0){
        return 0;
    }
    if ((count2 < Sent.count_w) && (count2 != 0)){
        return 2;
    }
}

```

```

struct Sentence special_sym(struct Sentence Sent){
    int i,j;
    wchar_t* s = L">special symbol<";
    for (i=0; i< Sent.len_s;){
        if (Sent.len_s > Sent.size_sent - 16){
            Sent.size_sent += INIT_SIZE;
            Sent.sentence = realloc(Sent.sentence, Sent.size_sent *
(sizeof(wchar_t)));
        }
        if ((iswalpha(Sent.sentence[i]) == 0) &&
(iswdigit(Sent.sentence[i]) == 0) && (Sent.sentence[i] != L'.')){
            for (j=Sent.len_s-1; j>i; j--){
                Sent.sentence[j+15] = Sent.sentence[j];
            }
            Sent.len_s += 15;
            Sent.sentence[Sent.len_s] = L'\0';
            for (j=0; j<wcslen(s); j++){
                Sent.sentence[i] = s[j];
                i++;
            }
        }
        else{
            if (Sent.sentence[i] == L'.'){
                Sent.len_s += 15;
                Sent.sentence[Sent.len_s] = L'\0';
                for (j=0; j<wcslen(s); j++){
                    Sent.sentence[i] = s[j];

```

```

        i++;
    }
}
else{
    i++;
}
}
}
return Sent;
}

int delete_sent(struct Sentence Sent){
    int i,j, flag1, flag2;
    for (i = 0; i<Sent.count_w; i++){
        flag1 = 1;
        flag2 = 1;
        for (j = 0; j<Sent.len_w[i]; j++){
            if (((Sent.words[i][j] >= L'a') && (Sent.words[i][j] <=
L'я')) || ((Sent.words[i][j] >= L'A') && (Sent.words[i][j] >= L'Я'))){
                flag1 = 0;
            }
            else {
                if (isalpha(Sent.words[i][j])) {
                    flag2 = 0;
                }
            }
        }
        if ((flag1 == 0) && (flag2 == 0)){
            return 1;
        }
    }
    return 0;
}

```

Файл other_funcs.h

```
#pragma once
```

```
int sort_sent(const void* arg1, const void* arg2);
```

Файл other_funcs.c

```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <wctype.h>
#include <wchar.h>
#include <locale.h>

#include "structures.h"
#include "other_funcs.h"

#define INIT_SIZE 50

```

```

int sort_sent(const void* arg1, const void* arg2){
    int i;
    int count_lat1 = 0;
    int count_lat2 = 0;
    struct Sentence* Sent1 = (struct Sentence*)arg1;
    struct Sentence* Sent2 = (struct Sentence*)arg2;
    for (i=0; i<Sent1->len_s; i++){
        if (isalpha(Sent1->sentence[i])) count_lat1++;
    }
    for (i=0; i<Sent2->len_s; i++){
        if (isalpha(Sent2->sentence[i])) count_lat2++;
    }
    return count_lat2 - count_lat1;
}

```

Файл main.c

```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <wctype.h>
#include <wchar.h>
#include <locale.h>

#include "funcs_for_tasks.h"
#include "other_funcs.h"
#include "input_text.h"
#include "structures.h"

#define INIT_SIZE 50

int main(){
    setlocale(LC_ALL, "");
    wprintf(L"Введите текст: \n");
    int i,j;
    int res1, res4;
    struct Text text = read_text();

    wprintf(L"Выберите команду, которую необходимо выполнить:\n"
           "0. Выход.\n"
           "1. Для каждого предложения вывести 'Палиндромы на\n"
           "месте!' если все слова являются палиндромами,\n"
           "'Кто-то потерял все палиндромы.' если ни одного слово\n"
           "не является палиндромом, в остальных случаях 'Чего-то не хватает.';\n"
           "2. Заменить все специальные символы (включая .,;) в\n"
           "тексте на подстроку ">special symbol<";\n"
           "3. Отсортировать предложения по уменьшению количества\n"
           "латинских букв;\n"
           "4. Удалить все предложения в которых в одном слове\n"
           "встречаются латинские и кириллические буквы;\n"
           "5. Вывести измененный текст.\n");
    int command;
    wscanf(L"%d", &command);
    while(command != 0) {

```

```

switch (command) {
    case 1: {
        for (i = 0; i < text.len_t; i++) {
            res1 = is_palindrom(text.sentences[i]);
            if (res1 == 0) {
                wprintf(L"Предложение №%d: Кто-то потерял
все палиндромы.\n", i + 1);
            }
            if (res1 == 1) {
                wprintf(L"Предложение №%d: Палиндромы на
месте!\n", i + 1);
            }
            if (res1 == 2) {
                wprintf(L"Предложение №%d: Чего-то не
хватает.\n", i + 1);
            }
        }
    }
    break;
    case 2:{
        for (i = 0; i < text.len_t; i++) {
            text.sentences[i] =
special_sym(text.sentences[i]);
        }
    }
    break;
    case 3:{
        qsort(text.sentences, text.len_t, sizeof(struct
Sentence), sort_sent);
    }
    break;
    case 4: {
        for (i = 0; i < text.len_t; i++) {
            res4 = delete_sent(text.sentences[i]);
            if (res4) {
                for (j = i; j < text.len_t; j++) {
                    text.sentences[j+1].sentence =
text.sentences[j].sentence;
                    text.sentences[j+1].len_s =
text.sentences[j].len_s;
                    text.sentences[j+1].count_w =
text.sentences[j].count_w;
                    text.sentences[j+1].len_w =
text.sentences[j].len_w;
                    text.sentences[j+1].words =
text.sentences[j].words;
                }
                free(text.sentences[text.len_t].sentence);
                for (j = 0; j <
text.sentences[text.len_t].count_w; j++) {
                    free(text.sentences[text.len_t].words[j]);
                }
                free(text.sentences[text.len_t].words);
                text.len_t -= 1;
            }
        }
    }
}

```



```

        break;
    case 5:{
        for (i=0; i<text.len_t; i++) {
            wprintf(L"%ls ", text.sentences[i].sentence);
        }
        wprintf(L"\n");
    }
    break;
    default:{
        wprintf(L"Данные некорректны.\n");
    }
    break;
}
wscanf(L"%d", &command);
}
for (i=0; i<text.len_t; i++){
    for (j=0; j<text.sentences[i].count_w; j++){
        free(text.sentences[i].words[j]);
    }
    free(text.sentences[i].sentence);
}
free(text.sentences);
return 0;
}

```