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 _____ Злобин А. С.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Злобин А. С.

Группа 0382

Тема работы: обработка строк на языке 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.
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. Распечатать каждое слово и количество его повторений в тексте.

2. Заменить каждый символ, который не является буквой, на его код.

3. Отсортировать предложения по количеству латинских букв в предложении.

4. Удалить все предложения, которые содержат специальные символы и не содержат заглавные 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

Содержание пояснительной записки:

Отчет должен содержать подробное описание выполненной вами работы, программной реализации того или иного функционала.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Злобин А. С.

Преподаватель _____ Жангиров Т. Р.

АННОТАЦИЯ

В процессе выполнения курсовой работы была реализована программа для обработки текста на языке Си. Для хранения текста использовались структуры. Обработка и вывод текста производились при помощи использования функций следующих стандартных заголовочных файлов: stdio.h, stdlib.h, wchar.h, wctype.h, locale.h. Для удобства пользователя реализован вывод на консоль контекстного меню выбора необходимой опции обработки текста. При некорректном вводе номера опции выводится сообщение об ошибке. Также был написан makefile для удобной сборки программы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является создание стабильной работоспособной программы на языке СИ для обработки текста. Обработка введённого пользователем текста производится в соответствии с выбранной опцией обработки.

Программа хранит введённый текст в структурах Sentence и Text. Для работы с кириллическими символами, не входящими в таблицу ASCII, был использован тип данных wchar_t из заголовочного файла <wchar.h>, а для работы с этим типом данных использовались функции из <wctype.h>. Память для хранения текста выделяется динамически в процессе работы программы. Выделение памяти происходит с помощью стандартных функций работы с памятью из заголовочного файла stdlib.h. Сборка программы реализуется с помощью утилиты make и файла Makefile.

Программа была разработана для устройств работающих на базе операционных систем Linux. Разработка велась на ОС Linux Ubuntu 20.04.

Программа была написана с использованием текстового редактора vim и терминала.

1. ЦЕЛИ И ЗАДАЧИ

Цель: создать стабильную программу по обработке считанного текста в соответствии с поставленным заданием.

Для достижения поставленной цели необходимо решить следующие задачи:

- Считать текст.
- Реализовать функции по обработки текста в соответствии с поставленным заданием
- Создать Makefile
- Произвести отладку программы

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1 Считывание текста

Для хранения текста реализованы структуры Sentence и Text:

```
struct Sentence
{
	wchar_t * sym_sentence;
	int size_sentence;
	int is_end;
};

struct Text
{
	struct Sentence * list_sentences;
	int size_text;
};
```

Структура Sentence состоит из следующих элементов:

- wchar_t * sym_sentence - указатель на начало предложения
- int size_sentence – количество символов в тексте
- int is_end – переменная, которая хранит ненулевое значение, если ввод предложений окончен

Структура Text состоит из следующих элементов:

- struct Sentence * list_sentences – указатель на первое предложение
- int size_text – количество предложений в тексте

Считывание текста осуществляется с помощью функций get_text() и get_sentence().

Функция get_sentence() возвращает переменную типа struct Sentence и осуществляет посимвольное считывание предложений. Считывания предложения происходит до тех пор, пока не будет считан символ ‘.’. При считывании следующего символа происходит выделение памяти для его хранения и проводится проверка, закончился ли ввод. После считывания предложения указатель на первый символ считанной строки присваивается полю sym_sentence структуры Sentence.

Функция get_text() возвращает переменную типа struct text и осуществляет считывание текста по предложениям. Ввод текста прерывается, когда переменная is_end вновь полученного предложения отлична от нуля. Последнее полученное предложение, отвечающее за признак конца ввода, не сохраняется, так как является пустым. В цикле, где происходит считывание текста, также происходит удаление повторно встречающихся предложений без учёта регистра. Сравнение осуществляется с помощью функции wcscasecmp(), и если предложение уже встречалось, то его сохранения не происходит. При сохранении следующего предложения так же, как и в функции get_sentence(), происходит выделение памяти для следующего предложения.

2.2. Реализация функций обработки текста.

2.2.1. Реализация первой подзадачи

Для реализации первой подзадачи была написана фунция prin_dict(). Она распечатывает каждое слово и количество его повторений в тексте. Функция принимает на вход структуру Text и ничего не возвращает. В начале создаётся указатель на указатель word_list и указатель count_list, в которых хранится список слов, встречающихся в тексте и их количество соответственно. Далее в цикле for() для каждое предложение делится на слова с помощью вложенного цикла for(). Для каждого вновь найденного слова выполняется проверка, было ли это слово сохранено в word_list и если да, то сохраняется индекс, под которым это слово встречалось в word_list, а затем увеличивается на единицу соответствующее значение в count_list. Если же слово встречается впервые, то выделяется память под ещё один элемент в count_list и word_list и слово сохраняется в word_list, а соответствующее значение count_list принимает значение равное 1. После проверки всех предложений осуществляется вывод получившихся динамических массивов и освобождение выделенной под них памяти с помощью функции free().

2.2.2. Реализация второй подзадачи

Для реализации второй подзадачи была написана фунция change_symb(). Она заменяет каждый символ, который не является буквой, на его код. Условимся, что код символа будет записан в десятичном формате для экономии места. Один символ типа wchar_t занимает 2 байта, соответственно, если представить код символа в десятичном формате, он будет занимать 5 разрядов.

Функция принимает на вход структуру Text и возвращает так же структуру Text. Для каждого предложения, которые перебираются в цикле for() определяется количество букв, а затем вычисляется новый объём, который предложение будет занимать после заметы всех символов на их численное представление. Далее выделяется память под новое предложение с заменёнными символами. Затем производится посимвольное копирование предложения в выделенную память. Если встречается символ, не являющийся буквой, код символа переводится в строковый формат путём последовательного деления на основание системы счисления, что позволит при необходимости представить код символа в другой системе счисления. Далее строковая форма представления числа посимвольно копируется в новое предложение и цикл продолжается.

После копирования всех символов указатель на предложение принимает значение указателя на новую строку и цикл переходит к следующему предложению.

После копирования всех предложений, функция возвращает структуру Text.

2.2.3. Реализация третьей подзадачи

Для реализации третьей подзадачи была написана фунция sort_lat() и compare_sentences(). В функции sort_lat() происходит вызов функции qsort() для списка предложений структуры text и функции и compare_sentences(). Последняя функция принимает на вход указатели на структуры sentence и считает количество латинских букв в каждом предложении. Функция compare_sentences() возвращает разность между количеством латинских букв в первом и втором предложении.

Функция sort_lat() возвращает структуру Text.

2.2.4. Реализация четвёртой подзадачи

Для реализации четвёртой подзадачи была написана функция del_sent_spec(). Она удаляет все предложения, которые содержат специальные символы и не содержат заглавные буквы. Под специальными символами будем понимать все символы кроме букв, цифр, и символов: точка , запятая и пробел,- так как по условию они являются разделителями и по умолчанию присутствуют в каждом предложении.

В цикле for выполняется посимвольная проверка каждого предложения на наличие специальных символов и заглавных букв (проверка на наличие последних осуществляется с помощью функции iswupper()). Если предложение удовлетворяет условию задачи, то оно копируется в новый список предложений. В противном случае, оно удаляется с помощью функции free().

После проверки каждого предложения указатель на новый список предложений копируется в text.list_sentences и функция возвращает структуру text.

2.2.5. ФУНКЦИЯ MAIN

Для работы с кириллическими символами используется функция setlocale().

В начале программа считывает текст, с которым она будет в дальнейшем работать. Далее запускается цикл while(), который необходим пользователю для выбора всех необходимых опций обработки текста и выхода из программы при необходимости.

В цикле сначала выводится меню со всеми возможными опциями, затем считывается опция, выбранная пользователем, и с помощью конструкции switch вызывается нужная функция или выводится сообщение о том, что выбрано невозможное действие.

При выборе выхода из программы производится очистка памяти, которая была выделена под текст.

2.2.6. Разделение на файлы и создание Makefile.

Программа раздела на следующие файлы:

- main.c – основной файл программы. В нём содержится функция main(). С помощью include в него включены все необходимые заголовочные файлы
- sort_lat.c - содержит функцию для решения третьей подзадачи
- sort_lat.h – содержит объявление функции для решения третьей подзадачи
- print_dict.c - содержит функцию для решения первой подзадачи
- print_dict.h - содержит объявление функции для решения первой подзадачи
- change_symb.c - содержит функцию для решения второй подзадачи
- change_symb.h - содержит объявление функции для решения второй подзадачи
- del_sent_spec.c - содержит функцию для решения четвёртой подзадачи
- del_sent_spec.h - содержит объявление функции для решения четвёртой подзадачи
- input.c – содержит функцию для ввода текста
- input.h – содержит заголовочный файл для функции ввода текста
- text_struct.h – содержит объявления структур

3. ТРЕБОВАНИЯ ПО ВВОДУ И ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СЛУЧАЕВ

Для корректной работы программы необходимо, чтобы пользователь закончил ввод текста двумя символами переноса строки (двумя нажатиями на клавишу Enter). Также необходимо, чтобы каждое предложение заканчивалось точкой (как сказано в условии

задачи). Для удобства использования разделительные символы между предложениями (символ переноса строки, табуляция и пробел) пропускаются и не являются частью какого-либо предложения.

При выборе опции, все действия, произведённые с текстом, сохраняются и доступа к предыдущей версии текста не будет (Кроме первой функции, так как она выводит количество повторений и исходный текст не меняет). Чтобы получить промежуточные результаты достаточно вывести текст (введя цифру 5) перед вызовом следующей функции. Также необходимо учесть, что все разделительные символы, стоящие внутри предложения (символ переноса строки, табуляция и пробел) считываются и являются частью текста.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была написана программа по обработке текста. В ходе разработки кода программы были изучены основные принципы обработки строк на языке СИ, получены навыки работы с динамической памятью и широкими символами типа wchar_t.

В ходе разработки программы была достигнута цель работы: была создана стабильно работающая программа на языке СИ, способная обрабатывать текст по команде пользователя.

При достижении поставленной цели были решены следующие задачи:

- Считывание текста и динамическое выделение памяти под текст
- Реализация функции по обработки текста в соответствии с поставленным заданием
- Создание Makefile и разделение программа была разделена на файлы
- отладка программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

пРОГРАММИРОВАНИЕ Учебно-методическое пособие / сост: К. В. Кринкин, Т А. Берленко, М. М. Заславский, К. В. Чайка.: СПбГУ «ЛЭТИ»
2. Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.

ПРИЛОЖЕНИЕ А

ПРИМЕР РАБОТЫ ПРОГРАММЫ

Вывод с консоли после вызова утилиты make

```
drdrew@drdrew-TWC:~/Zlobin_Andrey_cw/src$ make
gcc -c input.c
gcc -c print_dict.c
gcc -c change_symb.c
gcc -c sort_lat.c
gcc -c del_sent_spec.c
gcc -c main.c
gcc input.o print_dict.o change_symb.o sort_lat.o del_sent_spec.o main.o -o main
drdrew@drdrew-TWC:~/Zlobin_Andrey_cw/src$ ./main
```

Ввод текста с консоли

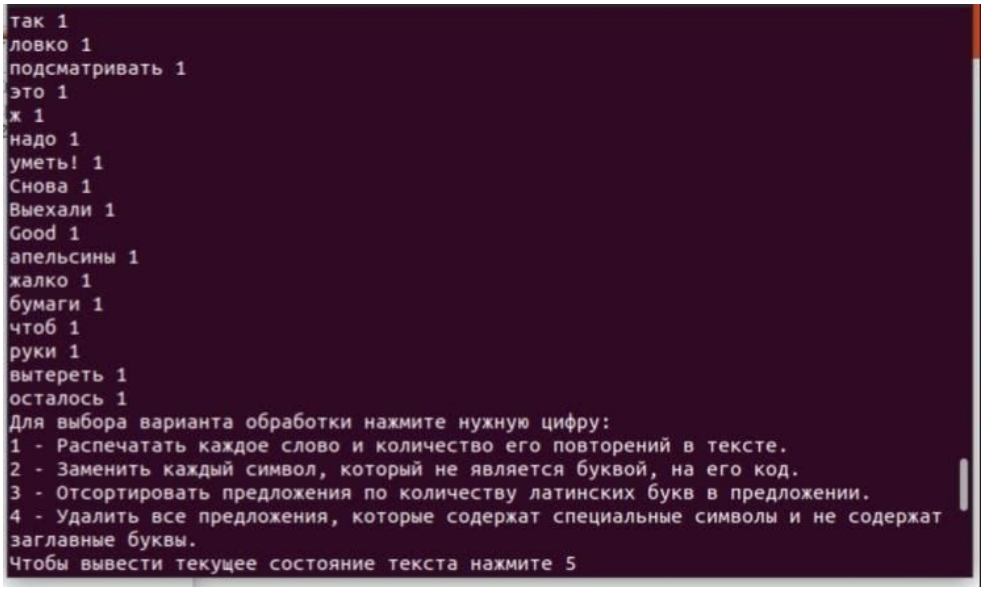
```
drdrew@drdrew-TWC:~/Zlobin_Andrey_cw/src$ ./main
Введите текст, который необходимо обработать:
Едут в поезде Мюллер, Шелленберг, и Штирлиц, направляются из Италии в Берлин. На
 столе стоит коробка с wery tasty and big апельсинами, рядом же с ними секретные
 документы отдела гестапо.
заежзают в туннель, ничего! не видать. Тут раздаётся шелест.
Поезд выезжает из туннеля, weater was fine. мюллер думает :
Вот Шелленберг подлец, знает же что документы МОЕГО отдела, а не его.
поезд снова заезжает в туннель: снова темнота, ничего не видно, снова шелест.
Выезжают из туннеля. Шелленберг думает:
Вот Штирлиц молодец, так ловко подсматривать документы это ж надо уметь!
Снова туннель, снова темнота, снова шелест.
Выехали. Штирлиц думает: Good апельсины, жалко бумаги, чтоб руки вытереть, не ос
талось.
```

Работа первой функции

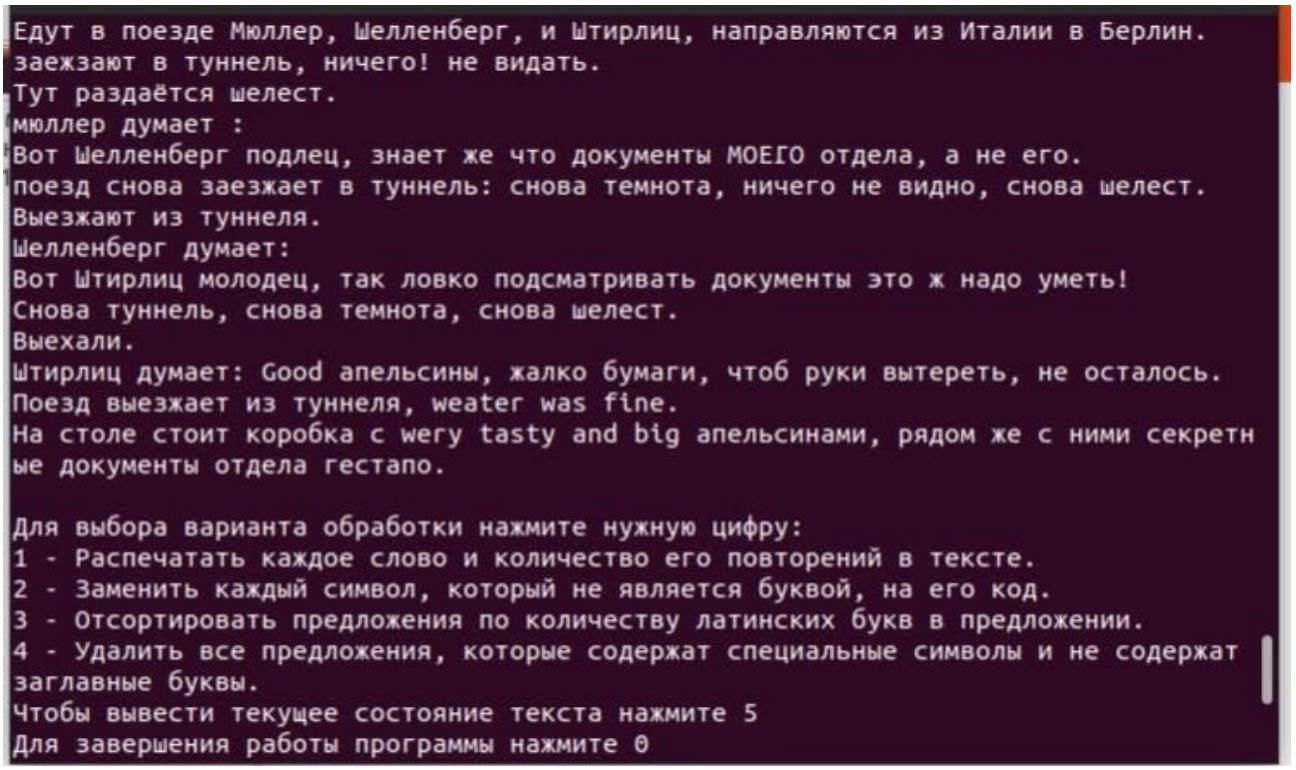
```
Для выбора варианта обработки нажмите нужную цифру:
1 - Распечатать каждое слово и количество его повторений в тексте.
2 - Заменить каждый символ, который не является буквой, на его код.
3 - Отсортировать предложения по количеству латинских букв в предложении.
4 - Удалить все предложения, которые содержат специальные символы и не содержат
заглавные буквы.
Чтобы вывести текущее состояние текста нажмите 5
Для завершения работы программы нажмите 0
1
Едут 1
в 4
поезде 1
Мюллер 1
Шелленберг 3
и 1
Штирлиц 3
направляются 1
из 3
Италии 1
Берлин 1
На 1
столе 1
стоит 1
коробка 1
```


с 2
wery 1
tasty 1
and 1
big 1
апельсинами 1
рядом 1
же 2
ними 1
секретные 1
документы 3
отдела 2
гестапо 1
заежзают 1
туннель 2
ничего! 1
не 4
видать 1
Тут 1
раздаётся 1
шелест 3
Поезд 1
выезжает 1
туннеля 2

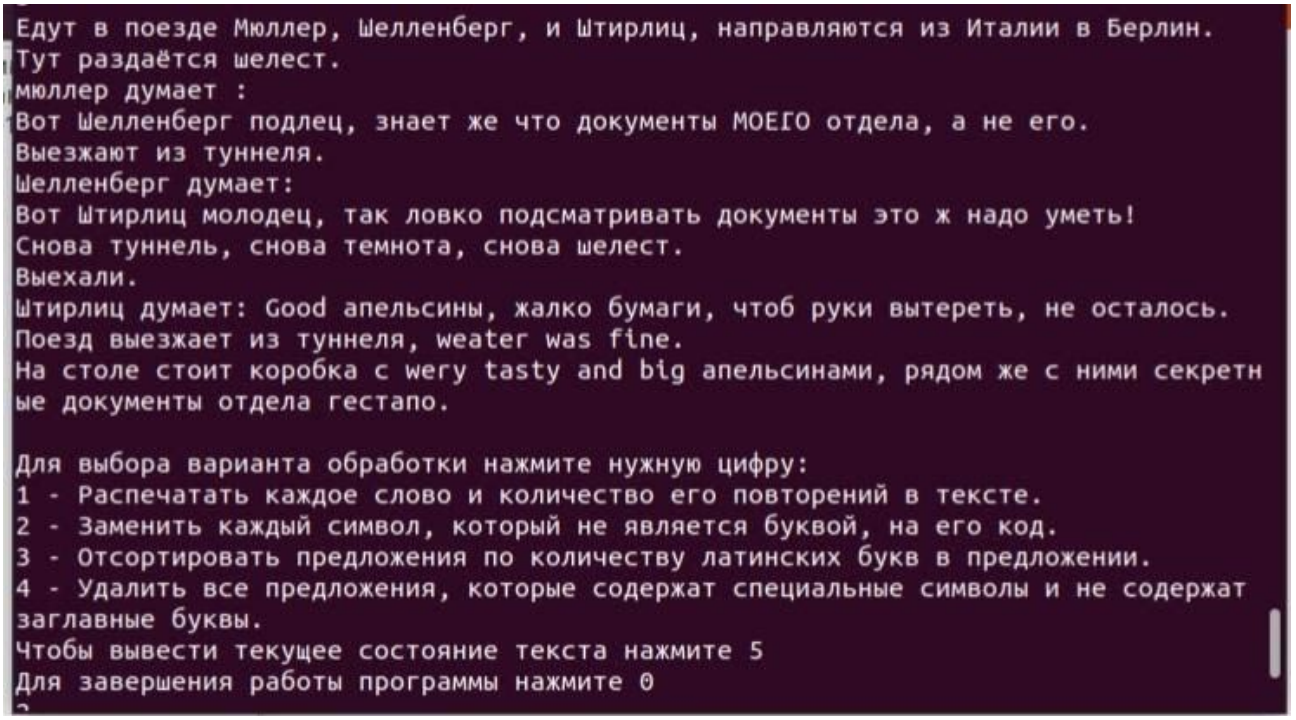

weater 1
was 1
fine 1
мюллер 1
думает 1
: 1
Вот 2
подлец 1
знает 1
что 1
МОЕГО 1
а 1
его 1
поезд 1
снова 5
заезжает 1
туннель: 1
темнота 2
ничего 1
видно 1
Выезжают 1
думает: 2
молодец 1
так 1


так 1
ловко 1
подсматривать 1
это 1
ж 1
надо 1
уметь! 1
Снова 1
Выехали 1
Good 1
апельсины 1
жалко 1
бумаги 1
чтоб 1
руки 1
вытереть 1
осталось 1
Для выбора варианта обработки нажмите нужную цифру:
1 - Распечатать каждое слово и количество его повторений в тексте.
2 - Заменить каждый символ, который не является буквой, на его код.
3 - Отсортировать предложения по количеству латинских букв в предложении.
4 - Удалить все предложения, которые содержат специальные символы и не содержат заглавные буквы.
Чтобы вывести текущее состояние текста нажмите 5

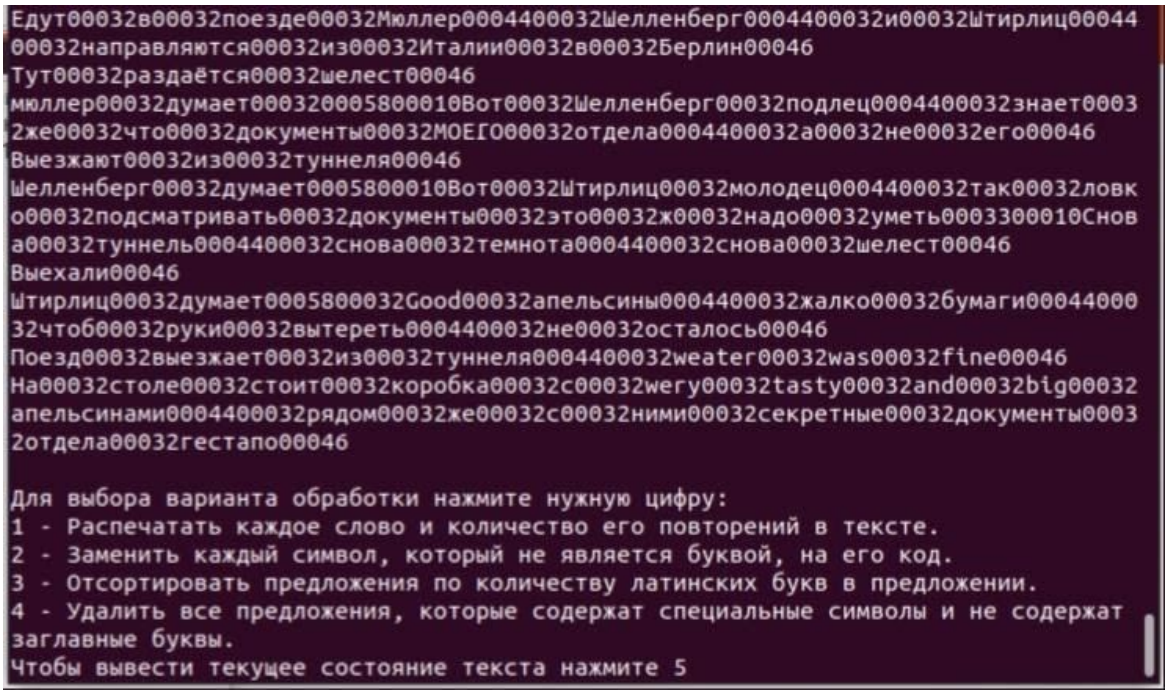
Работа второй функции (отсортированы предложения)



Работа третьей функции (Было удалено второе предложение, в котором были специальные символы и не было заглавных букв)



Работа четвёртой функции



ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл text_struct.h

```
#pragma once

struct Sentence
{
	wchar_t * sym_sentence;
	int size_sentence;
	int is_end;
};

struct Text
{
	struct Sentence * list_sentences;
	int size_text;
};
```

Файл main.c

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

#include "text_struct.h"
#include "input.h"
#include "print_dict.h"
#include "change_symb.h"
#include "sort_lat.h"
#include "del_sent_spec.h"



int main()
{
	setlocale(LC_ALL, "");
	struct Text new_text;
	int size;
	int edited_size;
	int i, n = -1;
	wchar_t sym;

	wprintf(L"%s\n", "Введите текст, который необходимо обработать:");
	new_text = get_text();



	while (n)
	{
		wprintf(L"%s\n", "Для выбора варианта обработки нажмите
нужную цифру:");
		wprintf(L"%s\n", "1 - Распечатать каждое слово и количество
его повторений в тексте.");
```

```
		wprintf(L"%s\n", "2 - Заменить каждый символ, который не
является буквой, на его код.");
		wprintf(L"%s\n", "3 - Отсортировать предложения по количеству
латинских букв в предложении.");
		wprintf(L"%s\n", "4 - Удалить все предложения, которые
содержат специальные символы и не содержат заглавные буквы.");
		wprintf(L"%s\n", "Чтобы вывести текущее состояние текста
нажмите 5");
		wprintf(L"%s\n", "Для завершения работы программы нажмите
0");
		wscanf(L"%d", &n);
		switch (n)
		{
			case 0:
				for (i=0; i<new_text.size_text; i++)

free(new_text.list_sentences[i].sym_sentence);
				free(new_text.list_sentences);
				return 0;
				break;
			case 1:
				print_dict(new_text);
				break;
			case 2:
				new_text = change_symb(new_text);
				break;
			case 3:
				new_text = sort_lat(new_text);
				break;
			case 4:
				new_text = del_sent_spec(new_text);
				break;
			case 5:
				for (i = 0; i <= new_text.size_text-1; i++)
				{

			wprintf(L"%ls\n",new_text.list_sentences[i].sym_sentence);
				}
				wprintf(L"\n");
				break;
			default:
					wprintf(L"%ls\n", "Введена неправильная
команда, попробуйте ещё раз:");
		}

	}









	for (int j=0;j<new_text.size_text;j++)
		free(new_text.list_sentences[i].sym_sentence);
	free(new_text.list_sentences);
```

```
	return 0;
}
```

Файл print_dict.c

```
#include "print_dict.h"

int print_dict(struct Text text)
{
	int i=0, j=0, word_size=0, list_size = 1, flag = -1;
	wchar_t letter = L'1';
	wchar_t ** word_list;
	int * count_list;
	wchar_t * word;
	word_list = calloc(1, sizeof(wchar_t * ));
	count_list = calloc(1, sizeof(int));
	word = calloc(1, sizeof(wchar_t));
	for (int i=0; i<text.size_text; i++)
	{
		for (int j=0; j<text.list_sentences[i].size_sentence; j++)
		{
		letter =(wchar_t) text.list_sentences[i].sym_sentence[j];

			if (letter != ' ' && letter != '\n' && letter != '\t'
&& letter != ',' && letter != '.')
			{

				*(word + word_size) = letter;
				word_size++;
			word = realloc(word, (word_size+1) * sizeof(wchar_t));
			}
			else
			{
				if (word_size != 0)
				{
					*(word + word_size) = '\0';
					for (int k = 0; k < list_size-1; k++)
					{
					if (wcscmp(*(word_list+k), word) == 0)
							flag = k;

					}
					if (flag == -1)
					{
*(word_list + list_size - 1) = calloc((word_size+1), sizeof(wchar_t));
					wcscpy(*(word_list + list_size - 1), word);
					*(count_list + list_size - 1) = 1;
						list_size++;
	word_list = realloc (word_list, list_size * sizeof(wchar_t * ));
	count_list = realloc (count_list, list_size * sizeof(int));
					}
					else
					{
			*(count_list + flag) = *(count_list + flag) + 1;
					}
					flag = -1;
					word_size = 0;
```

```
                                }
                        }
                }
        }
        for (i =0; i<list_size-1; i++)
                {
                        wprintf(L"%ls ", *(word_list + i));
                        wprintf(L"%d\n", *(count_list + i));
                        free(*(word_list + i));
                }
        free(word_list);
        free(count_list);
        return 0;
}
```

Файл print_dict.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include "text_struct.h"

int print_dict(struct Text text);
```

Файл change_symb.c

```
#include "change_symb.h"

struct Text change_symb(struct Text text)
{
        wchar_t * new_sentence;
        wchar_t buffer[5];
        int new_size=0;
        int counter=0, code = 0;
        int i=0, j=0, k=0, x=0;
        for (i=0; i<text.size_text; i++)
        {
                for (j=0; j<text.list_sentences[i].size_sentence; j++)
                {
                        if (iswalpha(text.list_sentences[i].sym_sentence[j]))
                                counter++;
                }
                new_size = text.list_sentences[i].size_sentence +
(text.list_sentences[i].size_sentence - counter) * SYM_SIZE;
                new_sentence = malloc(new_size * sizeof(wchar_t));
                j= 0;

                while (text.list_sentences[i].sym_sentence[j] != '\0')
                {
                        if (iswalpha(text.list_sentences[i].sym_sentence[j]))
                        {
```

```
                                new_sentence[k] =
text.list_sentences[i].sym_sentence[j];
                                k++;
                        }
                        else
                        {
                                for (x=0; x<SYM_SIZE; x++)
                                        buffer[x] = L'0';
                                code = (int)
text.list_sentences[i].sym_sentence[j];
                                x=SYM_SIZE - 1;
                                while (code!=0)
                                {
                                        buffer[x] = L'0' + code % 10;
                                        code = code / 10;
                                        x--;
                                }
                                x = 0;
                                for (x=0; x<SYM_SIZE; x++)
                                {
                                        new_sentence[k] = buffer[x];
                                        k++;
                                }
                                x = 0;
                        }

                                j++;

                }
                new_sentence[k] = '\0';
                k=0;
                j=0;
                free(text.list_sentences[i].sym_sentence);
                text.list_sentences[i].sym_sentence = new_sentence;
                new_sentence = NULL;
                counter = 0;
        }
        return text;
}
```

Файл change_symb.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include "text_struct.h"

#define SYM_SIZE 5

struct Text change_symb(struct Text text);
```

Файл del_sent_spec.c

```
#include "del_sent_spec.h"

struct Text del_sent_spec(struct Text text)
{
	int i = 0, j = 0, k = 0;
	int flag_spec = 0, flag_up = 0;
	wchar_t letter;
	struct Sentence * new_list;

	new_list = calloc(text.size_text, sizeof(struct Sentence));


	for (i=0; i<text.size_text; i++)
	{
		for (j = 0; j < text.list_sentences[i].size_sentence; j++)
		{
			letter = text.list_sentences[i].sym_sentence[j];
			if (!(iswalpha(letter) || iswdigit(letter)) && letter
!= ' ' && letter != ',' && letter != '.')
				flag_spec = 1;
			if (iswupper(letter))
				flag_up = 1;
		}
		if (!(flag_spec && !flag_up))
		{

			*(new_list + k) = text.list_sentences[i];
			k++;
		}
		else
		{
			free(text.list_sentences[i].sym_sentence);
		}
		flag_spec = 0;
		flag_up = 0;

	}
	free(text.list_sentences);
	text.list_sentences = new_list;
	text.size_text = k;
	return text;
}
```

Файл del_sent_spec.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include "text_struct.h"

struct Text del_sent_spec(struct Text text);
```

Файл sort_lat.c

```
#include "sort_lat.h"

int compare_sentences(const void * s1, const void * s2)
{
	int i=0, counter1 = 0, counter2 = 0;
	struct Sentence sent1;
	struct Sentence sent2;
	sent1 = *(struct Sentence * ) s1;
	sent2 = *(struct Sentence * ) s2;
	for(i=0; i<sent1.size_sentence; i++)
	{
		if ((sent1.sym_sentence[i] >= L'A' && sent1.sym_sentence[i]
<= L'Z') ||(sent1.sym_sentence[i] >= L'a' && sent1.sym_sentence[i] <= L'z'))
			counter1++;
	}
	for(i=0; i<sent2.size_sentence; i++)
	{
		if ((sent2.sym_sentence[i] >= L'A' && sent2.sym_sentence[i]
<= L'Z') ||(sent2.sym_sentence[i] >= L'a' && sent2.sym_sentence[i] <= L'z'))
			counter2++;

	}
	return (counter1 - counter2);
}

struct Text sort_lat(struct Text text)
{
	qsort(text.list_sentences, text.size_text, sizeof(struct Sentence),
compare_sentences);
	return text;
}
```

Файл sort_lat.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include "text_struct.h"

int compare_sentences(const void * s1, const void * s2);
struct Text sort_lat(struct Text text);
```

Файл input.c

```
#include "input.h"

struct Sentence get_sentence() {
	struct Sentence new_sentence;
	int sent_mem_size = 1;
	int is_end = 0;
	int begin = 1;
```

```
	new_sentence.sym_sentence = calloc(sent_mem_size, sizeof(wchar_t));

	wchar_t letter;
	wchar_t prev_letter = L'1';
	do {
		letter = (wchar_t) fgetwc(stdin);
		if (letter == L'\n' && prev_letter == L'\n')
		{
			is_end = 1;
			break;
		}
		if ((letter == L'\t' || letter == L'\n' || letter == L' ') &&
begin == 1)
			begin = 1;
		else
		{
			begin = 0;
			new_sentence.sym_sentence[sent_mem_size-1] = letter;
			sent_mem_size++;
			new_sentence.sym_sentence
=realloc(new_sentence.sym_sentence, sent_mem_size * (sizeof(wchar_t)));
		}
		prev_letter = letter;
	} while (letter != L'.');
	new_sentence.sym_sentence[sent_mem_size-1] = L'\0';

	new_sentence.size_sentence = sent_mem_size - 1;
	new_sentence.is_end = is_end;
	return new_sentence;
}

struct Text get_text() {
	int text_mem_size = 1;
	struct Text text;
	int text_len = 0;
	int i;
	int flag = 1;
	text.list_sentences = calloc(text_mem_size, sizeof(struct Sentence));

	while (1)
	{
		struct Sentence sentence = get_sentence();
		for (i = 0; i < text_len; i++)
		{
			if (wcscasecmp(text.list_sentences[i].sym_sentence,
sentence.sym_sentence)==0)
				flag = 0;
		}

		if (sentence.is_end)
			break;
		if (flag)
		{
			text.list_sentences[text_len++] = sentence;
			text_mem_size++;
			text.list_sentences = realloc(text.list_sentences,
text_mem_size * (sizeof(struct Sentence)));
```

```
                }
                flag = 1;
        }

        text.size_text = text_len;
        return text;
}
```

Файл input.h

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include <wctype.h>
#include "text_struct.h"


struct Sentence get_sentence();
struct Text get_text();
```

Файл Makefile

```
all: input.o print_dict.o change_symb.o sort_lat.o del_sent_spec.o main.o
	gcc input.o print_dict.o change_symb.o sort_lat.o del_sent_spec.o
 main.o -o main
 main.o: main.c input.h print_dict.h change_symb.h sort_lat.h
 del_sent_spec.h text_struct.h
	gcc -c main.c
 input.o: input.c input.h text_struct.h
	gcc -c input.c
 print_dict.o: print_dict.c print_dict.h text_struct.h
	gcc -c print_dict.c
 change_symb.o: change_symb.c change_symb.h text_struct.h
	gcc -c change_symb.c
 sort_lat.o: sort_lat.c sort_lat.h text_struct.h
	gcc -c sort_lat.c
 del_sent_spec.o: del_sent_spec.c del_sent_spec.h text_struct.h
	gcc -c del_sent_spec.c
 clean:
	rm *.o
```