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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текстовых данных

Студентка гр. 1304 _____ Чернякова В.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Чернякова В.А.

Группа 1304

Тема работы: обработка текстовых данных

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.

Технические требования:

Для хранения предложения и для хранения текста требуется реализовать структуры Sentence и Text. Программа должна сохранять текст в виде динамического массива предложений и оперировать далее только с ним.

Содержание пояснительной записки:

«Содержание», «Введение», «Основные теоретические положения», «Разработка программного кода», «Пользовательская инструкция», «Заключение», «Список использованных источников».

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 24.02.2022

Дата защиты реферата: 02.03.2022

Студентка _____ Чернякова В.А.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, которая обрабатывает введённый пользователем с клавиатуры текст. Программа сохраняет данный текст в виде динамического массива строк и далее обрабатывает его. Реализованы функции, обрабатывающие текст до выбора пользователем доступных действий: считывание текста, удаление повторно встречающих предложений без учета регистра, вывод текста на экран, освобождение динамически выделенной памяти для текста. Созданы функции, выполняющие действия, согласно запросу пользователя: сдвиг слов в предложении на целое положительное число, вывод уникальных кириллических и латинских символов, подсчет и вывод количества слов определенных длин, удаление всех слов, оканчивающихся на заглавный символ.

SUMMARY

In this course work, a program was implemented that processes the text entered by the user from the keyboard. The program saves this text as a dynamic array of strings and then processes it. Implemented functions that process text before the user selects available actions: reading text, deleting case-insensitive repeated sentences, displaying text on the screen, freeing up dynamically allocated memory for text. Functions have been created that perform actions according to the user's request: shifting words in a sentence by a positive integer, outputting unique Cyrillic and Latin characters, counting and outputting the number of words of certain lengths, deleting all words ending in a capital character.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка программы, которая сохраняет введённый с клавиатуры пользователем текст в виде динамического массива строк и далее обрабатывает его: удаляет повторно встречающиеся предложения без учета регистра. Для дальнейшей обработки программа запрашивает у пользователя одно из доступных действий и модернизирует текст на основе выбранной функции.

На основе вышеизложенной цели были сформулированы задачи, необходимые для ее достижения:

1. Изучить особенности работы со строками и широкими символами в языке C;
2. Изучить работу со структурами в языке C;
3. Изучить работу с динамической памятью в языке C;
4. Изучить работу с указателями и массивами в языке C;
5. Изучить и применить функции стандартных библиотек языка C;
6. Создать Makefile.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Широкие символы в языке программирования С

Широкими символами (wide character) называются многонациональные символы расширенного набора, имеющие двухбайтовое представление. Для описания широких символов используется тип *wchar_t*, определенный в заголовочном файле *ctype.h*. Широкие символы описываются кодовым набором *Unicode*.

1.2. Строки в языке программирования С

Строки в С представляют собой массив символов, заканчивающихся нулевым символом *\0*.

1.3. Указатели и массивы в языке программирования С

В Си существует связь между указателями и массивами. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен с помощью указателя. По определению значение переменной или выражения типа массив есть адрес нулевого элемента массива. Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель — это переменная, поэтому можно написать. Но имя массива не является переменной. Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес.

1.4. Основные сведения о структурах в языке программирования С

Объявление структуры начинается с ключевого слова *struct* и содержит список объявлений, заключенный в фигурные скобки. За словом *struct* может следовать имя, называемое тегом структуры. Тег дает название структуре данного вида и далее может служить кратким обозначением той части объявления, которая заключена в фигурные скобки. Перечисленные в структуре

переменные называются элементами (members). Имена элементов и тегов без каких-либо коллизий могут совпадать с именами обычных переменных (т. е. не элементов), так как они всегда различимы по контексту. Объявление структуры определяет тип. За правой фигурной скобкой, закрывающей список элементов, могут следовать переменные точно так же, как они могут быть указаны после названия любого базового типа. Объявление структуры, не содержащей списка переменных, не резервирует памяти; оно просто описывает шаблон, или образец структуры. Однако если структура имеет тег, то этим тегом далее можно пользоваться при определении структурных объектов. Доступ к отдельному элементу структуры осуществляется посредством конструкции вида: имя-структуры.элемент Оператор доступа к элементу структуры . (точка) соединяет имя структуры и имя элемента.

1.5. Управление памятью в языке программирования С

Функции *malloc* и *calloc* динамически запрашивают блоки свободной памяти. Функция *malloc void *malloc(size_t n)* возвращает указатель на *n* байт неинициализированной памяти или *NULL*, если запрос удовлетворить нельзя. Функция *calloc void *calloc(size_t n, size_t size)* возвращает указатель на область, достаточную для хранения массива из *n* объектов указанного размера (*size*), или *NULL*, если запрос не удается удовлетворить. Выделенная память обнуляется. Указатель, возвращаемый функциями *malloc* и *calloc*, будет выдан с учетом выравнивания, выполненного согласно указанному типу объекта. Функция *free(p)* освобождает область памяти, на которую указывает *p,* — указатель, первоначально полученный с помощью *malloc* или *calloc*.

2. РАЗРАБОТКА ПРОГРАММНОГО КОДА

2.1. Функция main

В главной функции *int main()* функция *setlocale* задает локаль, которая будет использоваться текущей программой. Локаль содержит информацию о том, как интерпретировать и выполнять определенные операции ввода, вывода и преобразования с учетом географического расположения, и специфики языков в определённых условиях. Создается переменная типа *struct Text user_text*, которой присваивается значение введённого текста с помощью функции *readText()*. Далее с помощью другой функции *del_repeat_sent()* из текста удаляются повторно встречающиеся предложения без учета регистра символов. Текст, готовый к дальнейшей обработке, выводится на экран с помощью функции *print_text()*. Далее с помощью цикла *while* и конструкции *switch-case* с текстом выполняются действия, на основе доступных и выбираемых пользователем. После выхода пользователя из программы память, выделенная на текст, освобождается с помощью функции *free_text()*.

2.2. Функция считывания текста

Для работы функции, считывающей текст, были реализованы структуры.

Struct Sentence – *str*(массив символов), *len*(длина предложения без учета /0), *size*(объем выделенной памяти), *words*(количество слов в предложении).

Struct Text – *sent*(массив из указателей на предложения), size(объем выделенной памяти), *cnt_sent*(количество предложений в тексте).

Далее реализованы сами функции, позволяющие считать и сохранить текст соответственно.

readSentence() – возвращает указатель на структура *Sentence*. В теле функции происходит считывание символов, пока не встретиться точка или знак переноса строки.

readText() – возвращает структуру *Text*. В теле функции происходит запись предложений с помощью вышеописанной функции до тех пор, пока считанная строка не окажется состоящей только из символа переноса строки.

2.3. Функция удаления повторно встречающихся предложений.

Функция *Struct Text del_repeat_sent()* принимает на вход переменную *text* типа *struct Text*. В теле функции реализовано посимвольное сравнение символов одного предложения с последующими. При совпадении всех символов, без учета регистра, все повторно встречающиеся предложения удаляются, а первое остается. Выделенная раннее память на эти предложения освобождается с помощью функции *free()*, а с помощью функции *memmove()* копирует значения числа байтов из удаляемого предложения, в блок памяти, предназначенный для следующих за ним предложений.

2.4. Функция вывода текста на экран

В теле функции *print_text()* с помощью цикла *for* осуществляется вывод всех предложений текста на экран.

2.5. Функция освобождения выделенной памяти для текста

В теле функции *free_text()* с помощью цикла *for* осуществляется освобождение выделенной памяти для каждого предложения текста.

2.6. Функция сдвига слов в предложении на целое положительное число

Функция *shift_words()* принимает на вход введённый пользователем текст. Внутри функции считывается целое положительное число *N*, на которое необходимо осуществить сдвиг слов в каждом предложении текста, если это возможно. С помощью цикла *for* до тех пор, пока не произойдет передвижение слов равное числу *N* в каждом предложении последнее слово переносится вперед. Вначале высчитывается его длина *len_word*, затем это слово записывается в переменную *word*. Далее с помощью циклов *for* мы меняем предложение: последнее слово уходит в начало, остальные сдвигаются. Так происходит, пока счетчик первого цикла *for* меньше введённого

положительного целого числа *N*. При введении некорректного значения функция прекращает работу.

2.7. Функция вывода уникальных кириллических и латинских символов

Функция *unic_symbols()* принимает на вход введённый пользователем текст. Далее происходит проверка каждого символа на его уникальность. С помощью циклов *for* осуществляется сравнение выбранного символа с другими символами каждого предложения. Сравнение происходит только в случае, если выполнено условие *iswalpha() != 0*, то есть выбранный символ является буквенным. Все уникальные символы выводятся через пробел, в случае их отсутствия пользователь информируется о том, что уникальные символы не найдены.

2.8. Функция подсчёта и вывода слов различной длины

Функция *len_words()* принимает на вход введённый пользователем текст. В теле функции создается динамический массив *words_len* автоматически заполняемый нулями с помощью функции *calloc*, в котором под соответствующими длине слова индексам будет храниться количество таких слов. С помощью циклов *for* рассматриваются все слова каждого предложения: другой цикл считает количество символов пока не встретит пробел, точку или запятую, по полученному значению длины ячейка созданного массива с индексом соответствующему найденной длине увеличивается на один. После обработки все значения ячеек не равные 0 выводятся на экран с помощью цикла *for*.

2.9. Функция удаления всех слов, оканчивающихся на заглавный символ

Функция *del_up()* принимает на вход введённый пользователем текст. С помощью цикла *for* происходит обработка каждого предложение. С помощью функции стандартной библиотеки *wcstok()* строчка разбивается на слова. Далее каждое слово обрабатывается, если для последнего символа слова выполняется условие *iswupper()*, то есть последний символ является заглавным, то

происходит удаление данного слова из строки, символ окончания строки *\0* и точка сдвигаются. С помощью функции *wmemmove()* происходит перераспределение памяти.

3. ПОЛЬЗОВАТЕЛЬСКАЯ ИНСТРУКЦИЯ

1) Откройте терминал и запустите программу из папки с помощью команды make && ./a.out;
2) Программа встретит вас следующими словами «Пожалуйста, введите текст, который хотите обработать:»
3) Необходимо ввести текст, который вы хотите обработать.
   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.
4) Далее программа запросит одно из доступных действий.
   На экране отображается результат обработки текста, соответствующей функции действия.
   При некорректном вводе на экране отобразится «Пожалуйста, введите корректное значение! Целое число от 1 до 5.»
5) Для завершения работы программы введите число 5. Программа отобразит на экране «Работа программы окончена.»

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была освоена работа с широкими символами и структурами, усовершенствованы навыки работы с динамической памятью: выделение, расширение и освобождение, обработкой строк, указателями.

Разработана программа, считывающая вводимый пользователем с клавиатуры текст и обрабатывающая его с помощью доступных действий, выбираемых также пользователем. Для выполнения действия использовались функции стандартных библиотек языка, так и собственно написанные. Реализован Makefile для удобной сборки программы, состоящей из нескольких файлов, в которых содержатся функции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брайан Керниган Деннис Ритчи Язык программирования СИ
2. Основы программирования на языках Си и C++ для начинающих: http://cppstudio.com/
3. Сайт https://www.cplusplus.com/
4. Сайт https://ru.wikipedia.org/wiki/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.c

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

#include "processing_entered_text.h"

#include "shift_words.h"
#include "unic_symbols.h"
#include "len_words.h"
#include "del_up.h"

int main(){
    setlocale(LC_ALL, "");
    struct Text user_text;
    wprintf(L"Пожалуйста,введите текст,который хотите
обработать:\n\n");
    user_text = readText();
    wprintf
(L"_____
_____\n");
    wprintf (L"Введеный вами текст после обработки(удаления
повторяющихся предложений): \n\n");
    user_text = del_repeat_sent(user_text);
    print_text(&user_text);
    wprintf
(L"\n_____
_____\n");
    wprintf(L"Введите номер действия,которое необходимо
выполнить:\n\n");
    int command;
    while (command != 5) {
        wprintf(L"1 - Сделать сдвиг слов в предложении на
положительное целое число N;\n");
         wprintf(L"2 - Вывести все уникальные кириллические и латинские
символы в тексте;\n");
         wprintf(L"3 - Подсчитать и вывести количество слов (плюс слова
в скобках) длина которых равна 1, 2, 3, и.т.д.;\n");
```

```
            wprintf(L"4 - Удалить все слова которые заканчиваются на
заглавный символ;\n");
            wprintf(L"5 - Завершение работы программы.\n");
            wprintf
(L"_____________________________________________________________________
_____________________________\n");
            wscanf(L"%d", &command);
               switch (command){
                   case 1:
                       shift_words(&user_text);
                       wprintf (L"Введеный вами текст после сдвига
предложений на целое число N:\n");
                       print_text(&user_text);
                       wprintf
(L"\n_____________________________________________________________________
_____________________________\n");
                       break;
                   case 2:
                       wprintf (L"ВЫВОД УНИКАЛЬНЫХ ЛАТИНСКИХ И
КИРИЛЛИЧЕСКИХ СИМВОЛОВ В ТЕКСТЕ:\n");
                       unic_symbols(&user_text);
                       wprintf
(L"_____________________________________________________________________
_____________________________\n");
                       break;
                   case 3:
                       wprintf (L"ПОДСЧЕТ И ВЫВОД СЛОВ ОПРЕДЕЛЕННЫХ
ДЛИН:\n");
                       len_words(&user_text);
                       wprintf
(L"_____________________________________________________________________
_____________________________\n");
                       break;
                   case 4:
                       del_up(&user_text);
                       wprintf (L"Введеный вами текст после удаления из
него всех слов,оканчивающихся на заглавный символ:\n");
                       print_text(&user_text);
                       wprintf
(L"\n_____________________________________________________________________
_____________________________\n");
                       break;
                   case 5:
                       wprintf(L"Работа программы окончена.\n");
```

```
                    break;
                default:
                    wprintf(L"Пожалуйста, введите корректное
значение!Целое число от 1 до 5.\n");
                    break;
            }
        }
        free_text(&user_text);
        return 0;
    }
```

Файл processing_entered_text.h

```
struct Sentence{
    wchar_t* str;
    int len;
    int size;
    int words;
};

struct Text{
    struct Sentence** sent;
    int size;
    int cnt_sent;
};

struct Sentence* readSentence();

struct Text readText();

struct Text del_repeat_sent(struct Text text);

void print_text(struct Text* text);

void free_text(struct Text* text);
```

Файл processing_entered_text.c

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

```
#define MEM_STEP 15

struct Sentence{
    wchar_t* str;
    int len;
    int size;
    int words;
};

struct Text{
    struct Sentence** sent;
    int size;
    int cnt_sent;
};

struct Sentence* readSentence(){
    int size = MEM_STEP;
    wchar_t* buf = malloc(size * sizeof(wchar_t));
    wchar_t temp;
    int ind_symb = 0;
    int words = 0;
    do{
        if (ind_symb <= size-2){
            wchar_t* t = realloc(buf, size + MEM_STEP);
            if(!t){
                wprintf(L"Ошибка!Закончилась память.");
            }
            size += MEM_STEP;
            buf = t;
        }
        temp = getwchar();
        buf[ind_symb] = temp;
        if (iswpunct(buf[ind_symb]) || iswspace(buf[ind_symb]))
            words++;
        ind_symb++;
    }while(temp != '.' && temp != '\n');
    buf[ind_symb] = '\0';
    struct Sentence* sentence= malloc(sizeof(struct Sentence));
    sentence -> str = buf;
    sentence -> size = size;
    sentence -> len = ind_symb - 1;
    sentence -> words = words;
    return sentence;
}
```

```
struct Text readText(){
    int size = MEM_STEP;
    struct Sentence** text = malloc(size*sizeof(struct Sentence));
    int ind_sent = 0;
    struct Sentence* temp;
    int null = 0;
    do{
        temp = readSentence();
        if(temp -> str[0] == '\n'){
            null++;
            free(temp -> str);
            free(temp);
        }
        else{
            null=0;
            text[ind_sent] = temp;
            ind_sent++;
        }
    }while (null < 1);
    struct Text enter_text;
    enter_text.size = size;
    enter_text.sent = text;
    enter_text.cnt_sent = ind_sent;
    return enter_text;
}

struct Text del_repeat_sent(struct Text text)
{
    wchar_t* words;
    wchar_t* words_compare;
    struct Text right_text = text;
    int text_len = right_text.cnt_sent;
    for(int i = 0; i < text_len; i++){
        words = right_text.sent[i] -> str;
        int words_len = right_text.sent[i] -> len;
        for(int j = i + 1; j < text_len; j++){
            words_compare = right_text.sent[j] -> str;
            int compare_len = right_text.sent[j] -> len;
            int same_symb = 0;
            if(words_len == compare_len) {
                for (int k = 0; k < right_text.sent[j] -> len; k++)
{
```

```
                        if              (towupper(words[k])             ==
towupper(words_compare[k]))
                            same_symb = 1;
                        else{
                            same_symb = 0;
                            break;
                        }
                    }
                }
                if(same_symb){
                    free(right_text.sent[text_len]);
                    memmove(&right_text.sent[j],  &right_text.sent[j+1],
(text_len-j)*sizeof(struct Sentence*));
                    text_len--;
                    j--;
                }
            }
        }
        right_text.cnt_sent = text_len;
        return right_text;
    }

    void print_text(struct Text* text){
         for (int i = 0; i < text -> cnt_sent; i++)
              wprintf(L"%ls", text -> sent[i] -> str);
    }

    void free_text(struct Text* text){
         for(int i = 0; i < text -> cnt_sent; i++)
              free(text -> sent[i]);
    }
```

Файл shift_words.h

```
#pragma once

void shift_words(struct Text *text);
```

Файл shift_words.c

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

```
#include "processing_entered_text.h"
#include "shift_words.h"

void shift_words(struct Text* text){
    int N;
    wprintf (L"Введите целое положительное число N,на которое
хотите сделать сдвиг предложений:\n");
    wscanf (L"%d", &N);
    if(N > 0){
        int len_word;
        int index_last;
        int ind_symb = 0;
        int word_size = 10;
        wchar_t symb;
        wchar_t wchar;
        wchar_t* word;
        word = (wchar_t*)malloc(sizeof(wchar_t) * word_size);
        for (int l = 0; l < N; l++){
            for (int i = 0; i < text -> cnt_sent; i++){
                ind_symb = 0;
                len_word = 0;
                index_last = text -> sent[i] -> len - 1;
                int j = index_last;
                while (text -> sent[i] -> str[j] != L' ' && text ->
sent[i] -> str[j] != L',' && j != 0){
                    len_word++;
                    j--;
                    if (j == 0)
                        len_word++;
                }
                if (text -> sent[i] -> str[j] == L' ')
                    symb = L' ';
                else if (text -> sent[i] -> str[j] == L',')
                    symb = L',';
                for (int m = j + 1; m < wcslen(text -> sent[i] ->
str) - 1; m++){
                    if (ind_symb == word_size - 2)
                    {
                        word_size *= 2;
                        word = (wchar_t*)realloc(word,
sizeof(wchar_t) * word_size);
                    }
                    word[ind_symb] = text -> sent[i] -> str[m];
```

```
                        ind_symb++;
                    }
                    for (int m = text -> sent[i] -> len - 1; m !=
(len_word); m--){
                        wchar = text -> sent[i] -> str[m - (len_word +
1)];
                        if (wchar == L' ' || wchar == L','){
                            text -> sent[i] -> str[m] = symb;
                            symb = wchar;
                        }
                        else
                            text -> sent[i] -> str[m] = text -> sent[i]
-> str[m - (len_word + 1)];
                    }
                    for (int m = 0; m < len_word; m++){
                        text -> sent[i] -> str[m] = word[m];
                        if (m == len_word - 1)
                            text -> sent[i] -> str[m + 1] = symb;
                    }
                    for (int m = 0; m < word_size; m++)
                        word[m] = '\0';
                    text -> sent[i] -> str[index_last + 2] = '\0';
                }
            }
            free(word);
        }
        else
            wprintf(L"Введено неккоректное значение для свдига
предложений!Выберете другую функцию или обратитесь к этой же,но с
корректным значением!\n");
    }
```

Файл unic_symbols.h

```
#pragma once

void unic_symbols(struct Text *text);
```

Файл unic_symbols.c

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

```
#include "processing_entered_text.h"
#include "unic_symbols.h"

void unic_symbols(struct Text* text){
    wchar_t *sent_now;
    wchar_t *sent_comp;
    int unic_symb;
    int cnt_unic = 0;
    wprintf (L"Уникальные символы:\n");
    for(int i = 0; i < text->cnt_sent; i++){
        sent_now = text -> sent[i] -> str;
        for (int j = 0; j < wcslen(sent_now); j++){
            if (iswalpha(sent_now[j]) != 0){
                unic_symb = 0;
                for (int k = 0; k < text -> cnt_sent; k++){
                    sent_comp = text -> sent[k] -> str;
                    for (int l = 0; l < wcslen(sent_comp); l++){
                        if (iswalpha(sent_comp[l]) != 0){
                            if ((sent_now[j] == sent_comp[l]) && (j
!= l)){
                                unic_symb = 1;
                                break;
                            }
                        }
                    }
                }
                if (unic_symb == 0){
                    wprintf (L"%lc ", sent_now[j]);
                    cnt_unic += 1;
                }
            }
        }
    }
    if (cnt_unic == 0)
        wprintf (L"Уникальные символы отсутсвуют.\n");
    wprintf (L"\n");
}
```

Файл len_words.h

```
#pragma once

void len_words(struct Text *text);
```

Файл len_words.c

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

#include "processing_entered_text.h"
#include "len_words.h"

void len_words(struct Text* text)
{
    int* words_len;
    int len_w_now = 0;
    int size = 100;
    int size_last;
    wchar_t symb;
    words_len = (int*)calloc(size, sizeof(int));
    for (int i = 0; i < text -> cnt_sent; i++){
        for (int j = 0; j < (wcslen(text->sent[i]->str)); j++){
            symb = text -> sent[i] -> str[j];
            if (symb != L' ' && symb != L',' && symb != L'.' &&
symb != L'(' && symb != L')')
                len_w_now++;
            else{
                if (len_w_now >= size){
                    size_last = size;
                    size = len_w_now + 10;
                    words_len = (int*)realloc(words_len,
sizeof(int) * size);
                    for (int k = size_last; k < size; k++)
                        words_len[k] = 0;
                }
                words_len[len_w_now] += 1;
                len_w_now = 0;
            }
        }
    }
    for (int i = 1; i < size; i++){
        if (words_len[i] != 0)
            wprintf(L"Количество слов длины %d = %d\n", i,
words_len[i]);
    }
```

```
        free(words_len);
    }
```

Файл del_up.h

```
#pragma once

void del_up(struct Text *text);
```

Файл del_up.c

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

#include "processing_entered_text.h"
#include "del_up.h"

void del_up(struct Text* text){
    int len_word;
    int size;
    int len_no_up;
    int i = 0;
    wchar_t* copy_str;
    wchar_t* buffer;
    wchar_t* ptr;
    wchar_t* del_word;
    size = wcslen(text->sent[i]->str) + 2;
    copy_str = (wchar_t*)malloc(sizeof(wchar_t) * size);
    while (i < text -> cnt_sent){
        if (wcslen(text -> sent[i] -> str) > size){
            size = wcslen(text -> sent[i] -> str) + 2;
            copy_str = (wchar_t*)realloc(copy_str, sizeof(wchar_t)
* size);
        }
        wcsncpy(copy_str, text -> sent[i] -> str, size);
        ptr = wcstok(copy_str, L", .", &buffer);
        while (ptr != NULL){
            len_word = wcslen(ptr);
            if (iswupper(ptr[len_word - 1])){
                del_word = wcsstr(text -> sent[i] -> str, ptr);
                len_no_up = wcslen(del_word) - len_word;
                if (del_word[len_word] == L'.'){
```

```
                        *(del_word) = '\0';
                        if (text-> sent[i] -> words != 1){
                            *(del_word - 1) = '.';
                        }
                    }
                    else{
                        wmemmove(del_word, del_word + len_word + 1,
len_no_up);
                        *(del_word + len_no_up) = '\0';
                    }
                    text -> sent[i] -> len = wcslen(text -> sent[i] ->
str);
                    text -> sent[i] -> words -= 1;
                }
                ptr = wcstok(NULL, L", .", &buffer);
            }
            i++;
        }
        free(copy_str);
    }
```

Файл Makefile

```
all: main.o shift_words.o unic_symbols.o len_words.o del_up.o
processing_entered_text.o
	gcc main.o shift_words.o unic_symbols.o len_words.o del_up.o
processing_entered_text.o

main.o: main.c shift_words.h unic_symbols.h len_words.h del_up.h
processing_entered_text.h
	gcc -c main.c

shift_words.o: shift_words.c shift_words.h
processing_entered_text.h
	gcc -c shift_words.c

unic_symbols.o: unic_symbols.c unic_symbols.h
processing_entered_text.h
	gcc -c unic_symbols.c

len_words.o: len_words.c len_words.h processing_entered_text.h
	gcc -c len_words.c

del_up.o: del_up.c del_up.h processing_entered_text.h
	gcc -c del_up.c
```

```
clean:
	rm *.o a.out
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Удаление повторно встречающихся предложений без учета регистра.

```
Пожалуйста,введите текст,который хотите обработать:

I live in SPB.i LIvE In SPb.Я очень love этот город.Petersберг,пожалуй,самый красивый city #on Earth123.

Введеный вами текст после обработки(удаления повторяющихся предложений):

I live in SPB.Я очень love этот город.Petersберг,пожалуй,самый красивый city #on Earth123.
```

Сдвиг слов в предложении на некоторое число.

```
Введите номер действия,которое необходимо выполнить:

1 - Сделать сдвиг слов в предложении на положительное целое число N;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д.;
4 - Удалить все слова которые заканчиваются на заглавный символ;
5 - Завершение работы программы.

1
Введите целое положительное число N,на которое хотите сделать сдвиг предложений:
-1
Введено неккоректное значение для свдига предложений!Выберете другую функцию или обратитесь к этой же,но с корректным значением!
Введеный вами текст после сдвига предложений на целое число N:
I live in SPB.Я очень love этот город.Petersберг,пожалуй,самый красивый city #on Earth123.

1 - Сделать сдвиг слов в предложении на положительное целое число N;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д.;
4 - Удалить все слова которые заканчиваются на заглавный символ;
5 - Завершение работы программы.

1
Введите целое положительное число N,на которое хотите сделать сдвиг предложений:
3
Введеный вами текст после сдвига предложений на целое число N:
live in SPB I.love этот город Я очень.city,#on,Earth123 Petersберг пожалуй самый красивый.
```

Вывод уникальных латинских и кириллических символов.

```
1 - Сделать сдвиг слов в предложении на положительное целое число N;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д.;
4 - Удалить все слова которые заканчиваются на заглавный символ;
5 - Завершение работы программы.

2
ВЫВОД УНИКАЛЬНЫХ ЛАТИНСКИХ И КИРИЛЛИЧЕСКИХ СИМВОЛОВ В ТЕКСТЕ:
Уникальные символы:
l v S B I l v э д Я ч н ь с у E a h s б п ж л у м к и в
```

Подсчёт и вывод количества слов различной длины.

```
3
ПОДСЧЕТ И ВЫВОД СЛОВ ОПРЕДЕЛЕННЫХ ДЛИН:
Количество слов длины 1 = 2
Количество слов длины 2 = 1
Количество слов длины 3 = 2
Количество слов длины 4 = 4
Количество слов длины 5 = 3
Количество слов длины 7 = 1
Количество слов длины 8 = 2
Количество слов длины 10 = 1

1 - Сделать сдвиг слов в предложении на положительное целое число N;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д.;
4 - Удалить все слова которые заканчиваются на заглавный символ;
5 - Завершение работы программы.
```

Удаление всех слов, заканчивающихся на заглавный символ.

```
1 - Сделать сдвиг слов в предложении на положительное целое число N;
2 - Вывести все уникальные кириллические и латинские символы в тексте;
3 - Подсчитать и вывести количество слов (плюс слова в скобках) длина которых равна 1, 2, 3, и.т.д.;
4 - Удалить все слова которые заканчиваются на заглавный символ;
5 - Завершение работы программы.

4
Введеный вами текст после удаления из него всех слов,оканчивающихся на заглавный символ:
live in.love этот город очень.city,#on,Earth123 Petersберг пожалуй самый красивый.
```

Завершение работы программы.