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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текста с помощью языка Си

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Азаров М.С.

Группа 0382

Тема работы : Обработка текста с помощью языка 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. Вывести все предложения, которые являются анаграммами друг для друга. Учитывать надо только буквы и цифры.
2. Отсортировать предложения (фактически, массив структур) по количеству заглавных букв в предложении.
3. Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту. Например, “ясЕнь” должно

быть преобразовано в “абсЁЖнь”.

4. Заменить все вхождения одного слова (заданного пользователем) на другое слово (заданного пользователем).

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.

Предполагаемый объем пояснительной записки:

Не менее 17 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

Курсовая работа заключается в выполнении поставленного задания и отработки полученный знаний. В процессе выполнения работы , были использованы следующие методы и знания . Для хранения текста были применены умения работать с динамической памятью . Для понятности программы и облегчения написания кода, структуры данных. Для эффективной сборки программы был создан Makefile. Также были отработаны умения роботы с указателями и многомерными массивами.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель курсовой работы заключается в создание примитивной программы , с минимальным пользовательским меню , которая обрабатывает текста в зависимости от введенной команды пользователем.

Для достижения поставленной цели потребовалось решить следующие задачи:

- Реализовать работу программы программы как с латинскими символами , так и кириллических. Для этого была освоена работа с типом широких символов wchar_t и библиотекой <wchar.h>.
- Считывание и сохранение текста неизвестной длинны , для этого была использована библиотека <stdlib.h> и ее функции для работы с динамической памяти.
- Сделать программу понятной и читабельной , для этого были применены структуры данных и разбиение программы на функции.
- Сделать эффективней сборку программы, была использована утилита make , и создан Makefile.

Программа была создана на операционной среде Linux , написана на языке Си с использованием текстового редактора Vim и интерактивной средой разработки Clion для отладки .

1.ХОД РАБОТЫ

1.1. Структуры данных

```
15
16 struct Sentence{
17     wchar_t *str ;
18     int len,avlb_len,  group_anagram; //group_anagram = 0 -> не анаграмма
19     int vol_upper;//количество сиволов верхнего регистра
20
21 };
22 typedef struct Sentence Sentence ;
23
24 struct Text{
25     Sentence *sents;
26     int size , avlb_size;
27     int volume_gr_anagram; // количество групп анаграмм
28 };
29 typedef struct Text Text;
30
31
```

Элементы структура Sentence(предложение):

- wchar_t *str - указатель на строку символов.
- int len — текущая длинна предложения ;
- int avlb_len — максимальная доступная на данный момент длина предложения .
- int group_anagram — переменная необходимая для 1-ой подзадачи , показывает принадлежность предложения к какой-либо группе анаграмм .
- int vol_upper — переменная необходимая для 2-ой подзадачи , указывает на количество символов в верхнем регистре в предложении.

Элементы структура Text(предложение):

- Sentence *sents — указатель на массив предложений.
- int size - текущее количество предложений .
- int avlb_size - максимальное доступное на данный момент количество предложений в тексте.
    - int volume_gr_anagram — количество групп анаграмм в тексте.

1.2. Функции считывания текста ,из файла read_text.c

Считывание и запоминание в память происходит в функциях Text read_text() , Sentence get_sent(int *ext) и int in_text(Sentence sent , Text _text).

Функция Sentence get_sent(int *ext) посимвольно считывает ввод с помощью функции getwchar() , до тех пор пока не встретит «.» , что означает конец предложения и возвращает структуру Sentence (считанное предложение). Сохраняет считанное предложение в динамическую память и при необходимости увеличивает базовый объем доступной динамической памяти. Также в функцию передается указатель на переменную ext , отвечающую за конец считываемого текста . Конец ввода текста считается **двойное нажатие Enter** , изначально ext равно 0 , но если считывается два «\n\n» то ext принимает значение 1 и функция возвращает последнее считанное предложение .(Уточнение : если два «\n\n» вводится в середине не законченного предложения часть введенного предложения не запоминается , т.к. не является предложением, отсутствует точка).

Функция Text read_text() считывает ввод по предложениям с помощью функции Sentence get_sent(int *ext) и сохраняет в структуру Text , которую потом возвращает . Выделяет динамическую память для массива указателей на Sentence и при необходимости увеличивает базовый объем доступной динамической памяти. Считывает до тех пор пока ext не станет равным 1. Также с помощью функции int in_text(Sentence sent , Text _text) сохраняет только не повторяющиеся предложения (без учета регистра).

Функция int in_text(Sentence sent , Text _text) проверяет есть ли уже предложение sent в тексте _text (без учета регистра) , если нет возвращает 0, иначе 1.

1.3. Главная функция main().

Это функция с которой происходит старт программы .Первым делом функция настраивает локальные переменные с помощью функции :

```
setlocale(LC_ALL,"");
```

Далее программа просит ввести текст и считывает его , с помощью описанной выше функции read_text().

```
//считывание текста
wprintf(L"Введите  текст (конец ввода два enter):\n");
_text = read_text();
```

Затем выводит меню , считывает команду пользователя и действует в соответствии с введенной командой пока пользователь не введет команду «0»(Выход).

Вывод меню:

```
wprintf(L"Введите команду :\n\t"
        "1 - Анаграммы ;\n\t"
        "2 - Отсортировать предложения  по количеству заглавных букв в предложении;\n\t"
        "3 - Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту;\n\t"
        "4 - Заменить одно слово  на другое во всем тексте ;\n\t"
        "5 - Ввести новый текст;\n\t"
        "6 - Вывести текущий текст;\n\t"
        "0 - Выход;\n");
```

Итак , рассмотрим выполнение каждой подзадачи .

1.4. Выполнение первой подзадачи и функции из файла anagram.c

При выборе команды номер 1 (Нахождение анаграмм в тексте), функция main() вызывает функцию Text find_anagram(Text _text) передает ей считанный текст , а результат выполнения передает в функцию write() с флагом 1 (которая будет рассмотрена чуть позже).

```
case 1 :
    write(find_anagram(_text), 1);
    break;
```

Функция Text find_anagram(Text _text) как было уже сказано принимает на вход текст ищет в нем анаграммы-предложения . Для этого , она каждое

предложения сравнивает с каждым предложением текста на то , являются эти предложения анаграммами или нет с помощью функции int is_anagram(Sentence s1 , Sentence s2), и если являются ставит каждому предложению соответствующую группу в переменную _text.sents[i].group_anagram , иначе в переменная _text.sents[i].group_anagram ставит 0. И еще присваивает переменной _text.volume_gr_anagram количество найденных групп анаграмм.

Функция int is_anagram(Sentence s1 , Sentence s2) проверяет являются ли предложения s1 и s2 анаграммами и если да возвращает 1, иначе 0. Для этого функция создает копии s1.str и s2.str , т.к. при поиске строки нужно изменить . Далее удаляет из копий лишние символы (лишние — не цифры и не буквы). Сортирует буквы строк в порядке уменьшения числового значения букв . И если после всех эти процедур строки s1.str и s2.str стали раны значит ,они анаграммы возвращается функцией 1, иначе 0. И очищается память выделенная для копий .

1.5. Выполнение второй подзадачи и функции из файла sort_upper.c

При выборе команды номер 2 (Сортировка предложений по количеству заглавных букв в нем), функция main() вызывает функцию Text sort_text_upper (Text _text) , передает ей считанный текст и запоминает результат функции (новый отсортированный текст) в другую переменную , выводит результат с помощью функции warite() с флагом 2 (которая будет рассмотрена чуть позже). И очищает новый отсортированный текст .Копия текста в функции Text sort_text_upper (Text _text) создается для того ,чтобы сортировка текста не повлияла на исходный текст .(Также во время вывод будут удалятся все \n ,для красивого и компактного вывода , поэтому копирование происходит не только указателей на предложения ,но и строк предложений)

```
case 2 :
    new_text = sort_text_upper(_text);
    write(new_text, 2);
    for (i = 0; i < new_text.size; i++) {
        free(new_text.sents[i].str);
    }
    free(new_text.sents);
    break;
```

Итак , функция Text sort_text_upper (Text _text) создает копию текста _text после считает в каждом предложении количество заглавных букв и присваивает это значение в переменную new_text.sents[i].vol_upper соответствующему предложению.

```
//подсчет количества заглавных букв в new_text
for (i = 0 ; i < new_text.size ; i++){
    for (j = 0; j < new_text.sents[i].len; j++){
        if (iswupper(new_text.sents[i].str[j])){
            new_text.sents[i].vol_upper++;
        }
    }
}
```

Далее сортирует указатели предложений по убыванию количества заглавных букв в них, с помощью стандартной функции qsort().

```
qsort(new_text.sents, new_text.size, sizeof(Sentence), sort_upper);
```

В данном случая функции sort_upper() выступает в роли компаратора.

1.6. Выполнение третьей подзадачи и функции из файла one_on_two_char.c

При выборе команды номер 3 (Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту) главная функция main()

```
case 3 :
    _text = replace_substr(_text);
    write(_text, 3);
    break;
```

Вызывает функцию Text replace_substr(Text _text) которой передается считанный текст и эта функция возвращает этот текст изменяет его в

соответствии с поставленной задачей . После результат печатается с помощью функции write() с флагом 3 (которая будет рассмотрена чуть позже).

Итак , функция Text replace_substr(Text _text) заменяет каждую гласную буквы двумя другими буквами идущими следующими по алфавиту в переданном ей тексте , и возвращает измененный текст . В функции хранится массив который указывает какие буквы нужно заменить и на какие .

```
wchar_t  dict[][2][3] ={{L"а",L"бв"},{L"е",L"ёж"},{L"ё",L"жз"},{L"и",L"йк"},{L"о",L"пр"},//руский
                        {L"у",L"фх"},{L"ы",L"ьэ"},{L"э",L"юя"},{L"ю",L"яа"},{L"я",L"аб"},
                        {L"А",L"БВ"},{L"Е",L"ЁЖ"},{L"Ё",L"ЖЗ"},{L"И",L"ЙК"},{L"О",L"ПР"},
                        {L"У",L"ФХ"},{L"Ы",L"ЬЭ"},{L"Э",L"ЮЯ"},{L"Ю",L"ЯА"},{L"Я",L"АБ"},
                        {L"a",L"bc"},{L"e",L"fg"},{L"i",L"jk"},{L"o",L"pq"},{L"u",L"vw"},//английский
                        {L"y",L"za"},
                        {L"A",L"BC"},{L"E",L"FG"},{L"I",L"JK"},{L"O",L"PQ"},{L"U",L"VW"},//английский
                        {L"Y",L"ZA"}};
```

Далее при нахождении требуемой буквы в тексте она удаляется функцией void delt_char(int num , wchar_t* str) ,которая описана в файле primitiv_func.c и на это место вставляется соответствующие две буквы ,с помощью функции void str_append(Sentence* sent,int num,wchar_t* substr).

Функция void str_append(Sentence* sent,int num,wchar_t* substr) вставляет в предложение sent , на место num , подстроку substr. При необходимости перевыделяет память.

1.7. Выполнение четвертой подзадачи и функции из файла swap_word.c

При выборе команды номер 4 (Замена одно слово на другое во всем тексте), главная функция main() просит , чтобы пользователь ввел слово которое хочет заменить , и слово на которое хочет заменить, и считывает их.

```
wprintf(L"\nВведите  слово которое хотите заменить:\n");
fgetws(word_old,MAX_LEN_WORD,stdin);
wprintf(L"\nВведите  слово НА которе  хотите заменить:\n");
fgetws(word_new,MAX_LEN_WORD,stdin);
```

Удаляет «\n» в конце слов и вызывает для кождого предложения функцию Sentence find_and_swap_word(Sentence sent , wchar_t* word_old, wchar_t*

word_new) , которая заменяет одно слово на другое . И осуществляет вывод функцией write() с флагом 4.

Функции Sentence find_and_swap_word(Sentence sent , wchar_t* word_old, wchar_t* word_new) поступает на вход предложение в котором нужно произвести замену слов , слово которое заменяется и слово на которое заменяется .Функция возвращает изменённое предложение. Для этого используется функция wcsstr(sent.str,word_old) , которая находит первое вхождение подстроки word_old в строке ,sent.str. Далее проверяется найденная подстрока является в предложении словом или частью слова,если является словом ,то удаляется из предложения с помощью функции delt_char(numb, sent.str) и добавляется новое слово с помошью функции Sentence add_substr (Sentence sent , int numb , wchar_t* word) .

Функция add_substr (Sentence sent , int numb , wchar_t* word) работает аналогично функции str_append(Sentence* sent,int num,wchar_t* substr) из 3 подзадачи.

Также в файле swap_word.c присутствует функция Text cut_sent(Sentence sent), которая разрезает предложение sent ,на слова и возвращает их в структуре Text , но она не понадобилась.

1.8. Функции из файла primitiv_func.c

В файле описаны функции :

- void* mem_alloc(int len, int size) -выделение памяти размера len*size , обработка ошибки в случае возврата функции malloc() NULL.
- void delt_all_char(Sentence* sent,wchar_t ch) - функция удаляет все символы ch в предложении sent.

- void delt_char(int num , wchar_t* str) -функция удаляет символ с индексом num в строке str.

Эти функции часто используются в других файлах программы , поэтому были вынесены как отдельный файл который подключается при необходимости использования данных функция .

1.9. Дополнительные команды

В программу и в меню также добавлены следующие команды :

- 5 — позволяет ввести новый текст и следующие команды обработки будут уже работать с этим текстом. Старый текст не сохраняется.
- 6 — просмотр текста , выводит текущий сохраненный текст с изменениями которые произошли при выполнении других команд . Команды 1 и 2 не изменяют сохраненный текст.(Главной целью данной команды является проверка работоспособности других команд)
- 0 — команда выхода из программы .

1.10. Ввод, функция write().

Функция void write(Text _text, int flag_n) находится в файле write.c .Отвечает за вывод результатов работы всех команд , и было вынесенна в отдельную функцию для облегчения главной функции main(). flaf_n отвечает для какой команды выводится результат, что означает как именно выводить результат.

Для подзадачи :

- 1 — если анаграммы небыли найдены выводится «Ненайденно», иначе по группа выводятся предложения которые являются анаграммами.

```
case 1:
    if (_text.volume_gr_anagram == 0){
        wprintf(L"Ненайденно \n\n");
        break;
    }

    for (int i = 1; i <= _text.volume_gr_anagram; i++) {//вывод по групам анаграмм
        wprintf(L"%i.\n",i);

        for (int j = 0;j < _text.size;  j++){//проверка каждого sent на принадлежность к группе анаграмм
            if (_text.sents[j].group_anagram == i){
                wprintf(L"\t%ls\n", _text.sents[j].str);
            }
        }
        wprintf(L"\n");

    }
    break;
```

- 2 — выводится отсортированный текст с удалением всех «\n» , для более красивого вывода.
- 3 ,4,6 — просто выводится измененный или не измененный текст.

```
case 3:
    for (int i = 0; i < _text.size; i++) {
        wprintf(L"%ls", _text.sents[i].str);
    }
    wprintf(L"\n\n");
    break;
```

Для удобности просмотра результата работы команды , меню откроется снова только после нажатия enter.

2. УКАЗАНИЯ ПО РАБОТЕ С ПРОГРАММОЙ И ОСОБЕННОСТИ

При вводе текста считывание прекратится только после после **двойного нажатия enter** . Если будет дважды нажат Enter в момен ввода незаконченого предложения , то часть введенного предложения запоминатся не будет.

Изменяют сохраненный текст команды обработки 3 и 4 , и **не** изменяют текст команды 1,2 .При введении не существующей команды , программа попросит пользователя ввести другую команду.

Команда выводит выводит предложения во убыванию заглавных символов и нумерует их , если количество заглавных символов совпадает с предыдущим предложением , номер предложения тоже совпадает с предыдущим. В конце предложения в ( ) пишется количество заглавных символов.

ЗАКЛЮЧЕНИЕ

В ходе работы была создана программа удовлетворяющая поставленным требованиям. А именно программа считывает и запоминает вводимый текста в динамическую память , у неё присутствует пользовательское меню и реализована работа поставленных подзадач на обработку текста. Программа работает как с латинскими символа , так и с кириллическими. Был создан Makefile для более эффективной сборки программы.

Также были приобретены новые знания в написании программ на языке Си и отработаны уже имеющиеся .

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт https://www.cplusplus.com/
2. Книга Кернигана и Ритчи «Язык программирования Си»
3. Сайт https://pubs.opengroup.org/onlinepubs/7908799/xsh/wchar.h.html

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Запуск Makefile:

```
maxim@mi:~/Рабочий стол/Вуз/Прог/Курсач/src$ make
gcc -c main.c
gcc -c read_text.c
gcc -c primitiv_func.c
gcc -c write.c
gcc -c anagram.c
gcc -c sort_upper.c
gcc -c one_on_two_char.c
gcc -c swap_word.c
gcc main.o read_text.o primitiv_func.o write.o anagram.o sort_upper.o one
_on_two_char.o swap_word.o  -o  res
```

Запуск программы:

```
maxim@mi:~/Рабочий стол/Вуз/Прог/Курсач/src$ ./res
Введите  текст (конец ввода два enter):
```

Ввод текста и вывод меню:

```
Введите  текст (конец ввода два enter):
Анаграмма. Программе на вход подается текст (текст представляет собой предложения, разделенные точкой
. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллическ
их букв, цифр и других символов кроме точки, пробела или запятой) Длина текста и каждого предложения
заранее не известна.Аа-аагрн++м?м .

Введите команду :
        1 - Анаграммы ;
        2 - Отсортировать предложения  по количеству заглавных букв в нем;
        3 - Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту;
        4 - Заменить одно слово  на другое во всем тексте ;
        5 - Ввести новый текст;
        6 - Вывести текущий текст;
        0 - Выход;
```

Проверка команды 1:

```
Введите команду :
        1 - Анаграммы ;
        2 - Отсортировать предложения  по количеству заглавных букв в нем;
        3 - Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту;
        4 - Заменить одно слово  на другое во всем тексте ;
        5 - Ввести новый текст;
        6 - Вывести текущий текст;
        0 - Выход;
1

Результат:
1.
        Анаграмма.
        Аа-аагрн++м?м .

Чтобы продолжить нажмите Enter:
```

Проверка команды 2:

```
Результат:
1. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кирилличес
ких букв, цифр и других символов кроме точки, пробела или запятой) Длина текста и каждого предложения
 заранее не известна. (2)
2.Анаграмма. (1)
2. Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. (1)
2.Аа-аагрн++м?м . (1)
```

Проверка команды 3:

```
3

Результат:
ББнбвгрбвммбв. Прпргрбвммёж нбв вхпрд ппрдбвёжтсаб тёжкст (тёжкст прёждстбввлабёжт спрбпрй прёждлпржё
жнйкаб, рбвздёжлёжнньюабёж тпрчкпрй. Прёждлпржёжнйкаб - нбвбпрр слпрв, рбвздёжлёжнньюабёж прпрбёжлпрм
 йклйк збвпабтпрй, слпрвбв - нбвбпрр лбвтйкнскйкх йклйк кйкрйкллйкчёжскйкх бфхкв, цйкфр йк дрфхгйкх с
йкмвпрлпрв крпрмёж тпрчкйк, прпрбёжлбв йклйк збвпабтпрй) Длйкнбв тёжкстбв йк кбвждпргпр прёждлпржёжнй
каб збврбвнёжёж нёж йкзвёжстнбв.ББбв-бвбвгрн++м?м .

Чтобы продолжить нажмите Enter:
```

Проварка команды 5,4:

```
5

Введите  текст (конец ввода два enter):
Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функ
ции обработки текста в один, функции ввода/вывода в другой). Также, должен быть написан Makefile.

Введите команду :
        1 - Анаграммы ;
        2 - Отсортировать предложения  по количеству заглавных букв в нем;
        3 - Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту;
        4 - Заменить одно слово  на другое во всем тексте ;
        5 - Ввести новый текст;
        6 - Вывести текущий текст;
        0 - Выход;
4

Введите  слово которое хотите заменить:
в

Введите  слово НА которе  хотите заменить:
:)

Результат:
Каждую подзадачу следует вынести :) отдельную функцию, функции сгруппировать :) несколько файлов (например, фу
нкции обработки текста :) один, функции ввода/вывода :) другой). Также, должен быть написан Makefile.

Чтобы продолжить нажмите Enter:
```

Проверка команды 0:

```
Введите команду :
        1 - Анаграммы ;
        2 - Отсортировать предложения  по количеству заглавных букв в нем;
        3 - Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту;
        4 - Заменить одно слово  на другое во всем тексте ;
        5 - Ввести новый текст;
        6 - Вывести текущий текст;
        0 - Выход;
0
maxim@mi:~/Рабочий стол/Вуз/Прог/Курсач/src$
```

ПРИЛОЖЕНИЕ В
КОД ПРОГРАММЫ

Файл main.c:

```
#include "top_header.h"
#include "read_text.h"
#include "primitiv_func.h"
#include "anagram.h"
#include "sort_upper.h"
#include "write.h"
#include "one_on_two_char.h"
#include "swap_word.h"


int main (){

    setlocale(LC_ALL,"");

    //объявление переменных
    Text _text,new_text ;
    int i,ans;
    wchar_t word_old[MAX_LEN_WORD],word_new[MAX_LEN_WORD];

    //считывание текста
    wprintf(L"Введите  текст (конец ввода два enter):\n");
    _text = read_text();

    while (1) {
        wprintf(L"Введите команду :\n\t"
                "1 - Анаграммы ;\n\t"
```

```
            "2 - Отсортировать предложения  по количеству заглавных букв в нем;\n\t"
                "3 - Заменить каждую гласную буквы двумя другими буквами идущими следующими по алфавиту;\n\t"
            "4 - Заменить одно слово  на другое во всем тексте ;\n\t"
            "5 - Ввести новый текст;\n\t"
            "6 - Вывести текущий текст;\n\t"
            "0 - Выход;\n");

    wscanf(L"%i", &ans);
    while ((getwchar ()) != '\n');//очистка буфера ввода

    switch (ans) {
        case 0 ://очистка
            for (i = 0; i < _text.size; i++) {
                free(_text.sents[i].str);
            }
            free(_text.sents);

            return 0;

        case 1 :
            write(find_anagram(_text), 1);
            break;

        case 2 :
            new_text = sort_text_upper(_text);
            write(new_text, 2);
            for (i = 0; i < new_text.size; i++) {
                free(new_text.sents[i].str);
```

```
            }
            free(new_text.sents);
            break;

        case 3 :
            _text = replace_substr(_text);
            write(_text, 3);
            break;

        case 4 :
            wprintf(L"\nВведите  слово которое хотите заменить:\n");
            fgetws(word_old,MAX_LEN_WORD,stdin);
            wprintf(L"\nВведите  слово НА которе  хотите заменить:\n");
            fgetws(word_new,MAX_LEN_WORD,stdin);

            delt_char((int)wcslen(word_old)-1,word_old);
            delt_char((int)wcslen(word_new)-1,word_new);

            for(i = 0; i < _text.size ;i++){
                                                    _text.sents[i]     =
find_and_swap_word(_text.sents[i],word_old,word_new);
            }
            write(_text, 4);

            break;

        case 5 :
            for (i = 0; i < _text.size; i++) {
                free(_text.sents[i].str);
            }
            free(_text.sents);
```

```
            wprintf(L"\nВведите  текст (конец ввода два enter):\n");
            _text = read_text();
            break;

        case 6 :
            write(_text, 6);
            break;

        default:
            wprintf(L"\nНет такой команды , введите другую!\n");
            wprintf(L"Чтобы продолжить нажмите Enter:\n");
            while ((getwchar ()) != '\n');
            break;
    }
  }
}
```

Файл top_header.h:

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


#define BASIC_LEN_SENT 50
#define BASIC_LEN_TEXT 20
#define MAX_LEN_WORD 100
```

```
//создание типов Sentence, Text

struct Sentence{
   wchar_t *str ;
   int len,avlb_len,  group_anagram; //group_anagram = 0 -> не анаграмма
   int vol_upper;//количество сиволов верхнего регистра

};
typedef struct Sentence Sentence ;

struct Text{
   Sentence *sents;
   int size , avlb_size;
   int volume_gr_anagram; // количество групп анаграмм
};
typedef struct Text Text;
```

Файл read_text.c:

```
#include "top_header.h"
#include "read_text.h"
#include "primitiv_func.h"


Sentence get_sent(int *ext){

   Sentence sent = { .len = 0};

   wchar_t *temp_sent;
   wchar_t ch = 0, temp_ch;
   int in_sent = 0;
```

```
sent.avlb_len = BASIC_LEN_SENT;
sent.str = mem_alloc(sent.avlb_len , sizeof(wchar_t));


while (ch != '.'){
    temp_ch = ch;
    ch = getwchar();

    if ((ch == '\n')&&(temp_ch == '\n')){
        *ext = 1;
        return sent;
    }

    if (sent.len  >= sent.avlb_len - 1) {

        sent.avlb_len = sent.avlb_len + BASIC_LEN_SENT;
        temp_sent = realloc(sent.str, sent.avlb_len*sizeof(wchar_t));

        if (temp_sent == NULL) {
            free(sent.str);
            fprintf(stderr, "Ошибка перевыделения памяти для sent");
            exit(1);
        } else {
            sent.str = temp_sent;
        }
    }

    sent.str[sent.len++] = ch;

}
```

```
    sent.str[sent.len] = '\0';


    return sent;
}



Text read_text(){

    Text _text = { .size = 0};
    Sentence *temp_text;
    Sentence temp_sent;
    int ext = 0,i;



    _text.avlb_size = BASIC_LEN_TEXT;
    _text.sents = mem_alloc(_text.avlb_size,sizeof(Sentence));

    while (ext == 0){

        //проверка на свободное место
        if (_text.size >= _text.avlb_size) {

            _text.avlb_size = _text.avlb_size + BASIC_LEN_TEXT;
            temp_text = realloc(_text.sents, _text.avlb_size*sizeof(Sentence));

            if (temp_text == NULL) {
                for (i = 0; i < _text.size; i++) {
                    free(_text.sents[i].str);
```

```
            }
            free(_text.sents);
            fprintf(stderr, "Ошибка перевыделения памяти для Text");
            exit(1);
        } else {
            _text.sents = temp_text;
        }
    }


    //_text.sents[_text.size] = get_sent(&ext);
    //проверка на повтор
    temp_sent = get_sent(&ext);

    if (ext == 0) {
        if (!in_text(temp_sent, _text)) {
            _text.sents[_text.size++] = temp_sent;
        }
    }
  }

  return _text;
}



int in_text(Sentence sent , Text _text){

  for (int i = 0 ; i<_text.size ; i++){
    if (!wcscasecmp(sent.str,_text.sents[i].str)){
        return 1;
```

```
        }
    }

    return 0;
}
```

Файл read_text.h:

```
Sentence get_sent(int *ext);
Text read_text();
int in_text(Sentence sent , Text _text);
```

Файл anagram.c:

```
#include "top_header.h"

#include "primitiv_func.h"
#include "anagram.h"

Text find_anagram(Text _text){

    int i,j;

    //обнуляем anagrm
    for(i = 0 ; i < _text.size ; i++){
        _text.sents[i].group_anagram = 0;
    }

    int numb = 0 ; //сквозная нумерация групп анаграмм
    int last_i = -1;//запоминает предыдушее значение i

    for(i = 0; i < _text.size ; i++){
        for(j = i+1; j < _text.size ; j++){
```

```
            if (is_anagram(_text.sents[i],_text.sents[j])&&(_text.sents[j].group_anagram == 0)){
                if (i != last_i){numb+=1;}
                _text.sents[i].group_anagram = numb;
                _text.sents[j].group_anagram = numb;
                last_i = i;
            }
        }
    }

    _text.volume_gr_anagram = numb;

    return _text;

}



int is_anagram(Sentence s1 , Sentence s2) {
    int j;
    wchar_t* p1;// промежуточные укозатели
    wchar_t* p2;

    //делаем копии s1 s2 , чтобы можно было их изменять
    p1 = mem_alloc(s1.avlb_len+1,sizeof(wchar_t)); //+1 на всякий случай
    p2 = mem_alloc(s2.avlb_len+1,sizeof(wchar_t));

    wcscpy(p1,s1.str);
    wcscpy(p2,s2.str);

    s1.str = p1;
    s2.str = p2;

    //удаляем лишнее
    for (j = 0; j < s1.len; j++) {
        if (!iswalpha(s1.str[j]) && !iswdigit(s1.str[j])) {
```

```
        delt_char(j, s1.str);
        j = j - 1;
        s1.len--;
    }
}

for (j = 0; j < s2.len; j++) {
    if (!iswalpha(s2.str[j]) && !iswdigit(s2.str[j])) {
        delt_char(j, s2.str);
        j = j - 1;
        s2.len--;
    }
}

if(s1.len!=s2.len){
    free(p1);
    free(p2);
    return 0;
}
else{
    qsort(s1.str,s1.len,sizeof(wchar_t),cmp1);
    qsort(s2.str,s2.len,sizeof(wchar_t),cmp1);

    for (int i = 0 ;i < s1.len;i++){
        if (s1.str[i]!=s2.str[i]){
            free(p1);
            free(p2);
            return 0;
        }
    }
    free(p1);
    free(p2);
    return 1;
}
```

```
}


int cmp1(const void* a ,const void* b){
    wchar_t* ch1 = (wchar_t*)a;
    wchar_t* ch2 = (wchar_t*)b;

    if (*ch1>*ch2) {
        return 1;
    }else{
        return 0;}
}
```

Файл anagram.h:

```
Text find_anagram(Text _text);
int is_anagram(Sentence s1 , Sentence s2);
int cmp1(const void* a ,const void* b);
```

Файл sort_upper.c:

```
#include "top_header.h"
#include "primitiv_func.h"
#include "sort_upper.h"

Text sort_text_upper(Text _text){
    Text new_text;
    int i,j;

    //полное копирование текста
    new_text.sents = mem_alloc(_text.avlb_size,sizeof(Sentence));//незабудь очисить

    for (i = 0 ; i < _text.size; i++){
        new_text.sents[i].avlb_len = _text.sents[i].avlb_len;
        new_text.sents[i].str = mem_alloc(_text.sents[i].avlb_len,sizeof(wchar_t));//незабудь очисить
        new_text.sents[i].len = _text.sents[i].len;
```

```
        wcscpy(new_text.sents[i].str,_text.sents[i].str);
    }
    new_text.size = _text.size;
    new_text.avlb_size =_text.avlb_size;

    //обнуление vol_upper
    for(i = 0; i <new_text.size ; i++){
        new_text.sents[i].vol_upper = 0;
    }

    //подсчет количества заглавных букв в new_text
    for (i = 0 ; i < new_text.size ; i++){
        for (j = 0; j < new_text.sents[i].len; j++){
            if (iswupper(new_text.sents[i].str[j])){
                new_text.sents[i].vol_upper++;
            }
        }
    }


    qsort(new_text.sents, new_text.size, sizeof(Sentence), sort_upper);


    return new_text;
}


int sort_upper(const void* a ,const void* b){
    Sentence sent1 = *(Sentence*) a;
    Sentence sent2 = *(Sentence*) b;

    return    sent2.vol_upper - sent1.vol_upper    ;
}
```

Файл sort_upper.h:

```
int sort_upper(const void* a ,const void* b);
Text sort_text_upper(Text _text);
```

Файл one_on_two_char.c:

```
#include "top_header.h"
#include "primitiv_func.h"
#include "one_on_two_char.h"

void str_append(Sentence* sent,int num,wchar_t* substr ){
    wchar_t *temp_sent;
    int len_substr= (int)wcslen(substr) ,i,j;



    //перевыделение памяти
    while(sent->len >= sent->avlb_len - len_substr-1){//-1 на всякий случай
        sent->avlb_len = sent->avlb_len + BASIC_LEN_SENT;
        temp_sent = realloc(sent->str, sent->avlb_len*sizeof(wchar_t));

        if (temp_sent == NULL) {
            free(sent->str);
            fprintf(stderr, "Ошибка перевыделения памяти для sent");
            exit(1);
        } else {
            sent->str = temp_sent;
        }
    }

    //сдвиг
    for(j = 0 ; j < len_substr;j++) {
        for (i = sent->len; i >=
        num + j; i--) {
            sent->str[i + 1] = sent->str[i];
        }
```

```
        sent->len++;
    }

    //вставка
    for(i = num ; i < num+len_substr ; i++){
        sent->str[i] = substr[i-num];
    }
}



Text replace_substr(Text _text){
    int i,j,k;
                wchar_t         dict[][2][3]    ={{L"а",L"бв"},{L"е",L"ёж"},{L"ё",L"жз"},{L"и",L"йк"},
{L"о",L"пр"},//руский
                    {L"у",L"фх"},{L"ы",L"ьэ"},{L"э",L"юя"},{L"ю",L"яа"},{L"я",L"аб"},
                    {L"А",L"БВ"},{L"Е",L"ЁЖ"},{L"Ё",L"ЖЗ"},{L"И",L"ЙК"},{L"О",L"ПР"},
                    {L"У",L"ФХ"},{L"Ы",L"ЬЭ"},{L"Э",L"ЮЯ"},{L"Ю",L"ЯА"},{L"Я",L"АБ"},
                        {L"a",L"bc"},{L"e",L"fg"},{L"i",L"jk"},{L"o",L"pq"},{L"u",L"vw"},//
английский
                    {L"y",L"za"},
                        {L"A",L"BC"},{L"E",L"FG"},{L"I",L"JK"},{L"O",L"PQ"},{L"U",L"VW"},//
английский
                    {L"Y",L"ZA"}};


    int len_dict = sizeof (dict)/(sizeof (wchar_t)*3*2);

    for (i = 0 ; i < _text.size; i++){   //в каждом предложение
        for (j = 0; j < _text.sents[i].len; j++){    //каждая буква
            for(k = 0; k < len_dict ;k++){     //сравнивается с каждым эл dict
                if (_text.sents[i].str[j] == dict[k][0][0]){

                    delt_char(j,_text.sents[i].str);
                    _text.sents[i].len--;
```

```
                str_append(&_text.sents[i],j,dict[k][1]);
                j++;
            }
        }
    }
}

return _text;
}
```

Файл one_on_two_char.h:

```
Text replace_substr(Text _text);
void str_append(Sentence* sent,int num,wchar_t* substr );
```

Файл swap_word.c:

```
#include "top_header.h"
#include "primitiv_func.h"
#include "swap_word.h"




Sentence  find_and_swap_word(Sentence sent , wchar_t* word_old, wchar_t* word_new){
    int word_old_len = (int)wcslen(word_old) ;
    int word_new_len = (int)wcslen(word_new) ;
    int i,numb;//номер вхождения в строку
    wchar_t* p;//указатель на вхождение подстроки в строку
    Text arr_words;

    //удаляем \n в крнце слов

    p = wcsstr(sent.str,word_old);
```

```
   if (p == NULL){
      return sent;
   }
   else{
      while(p != NULL){
         numb = p-sent.str;
         if(numb == 0 ) {
                                          if((sent.str[word_old_len]==' ')||(sent.str[word_old_len]==',')||
(sent.str[word_old_len]=='.')||(sent.str[word_old_len]=='\n')) {

               //удаляем слово
               for (i = 0; i < word_old_len; i++) {
                  delt_char(numb, sent.str);
               }
               sent.len -= word_old_len;
               //вставляем слово
               sent = add_substr(sent, numb, word_new);



               p = wcsstr(sent.str+numb, word_old);
            }
            else{
               p = wcsstr(sent.str+numb+word_old_len, word_old);
            }
         }
         else {
                              if(((sent.str[numb-1]==' ')||(sent.str[numb-1]==',')||(sent.str[numb-1]=='\
n'))&&((sent.str[numb+word_old_len]==' ')||(sent.str[numb+word_old_len]==',')||
(sent.str[numb+word_old_len]=='.')||(sent.str[numb+word_old_len]=='\n'))){

               //удаляем слово
               for (i = 0; i < word_old_len; i++) {
                  delt_char(numb, sent.str);
               }
               sent.len -= word_old_len;
```

```
                //вставляем слово
                sent = add_substr(sent, numb, word_new);

                p = wcsstr(sent.str+numb, word_old);
            }
            else{
                p = wcsstr(sent.str+numb+word_old_len, word_old);
            }
        }
    }
    return sent;
  }
}




Text cut_sent( Sentence sent){
  Text arr_words;
  wchar_t * temp_sent;
  Sentence* temp_text;
  int i,in_word = 0,numb_word = -1;

  //выделение памяти
  arr_words.avlb_size = BASIC_LEN_TEXT;
  arr_words.sents = malloc(sizeof ( Sentence)*arr_words.avlb_size);
  arr_words.size = 0;


  for(i = 0; i < sent.len ; i++){
    //проверка на нахождении в слове
    if ((sent.str[i] != ' ')&&(sent.str[i] != '\n')&&(sent.str[i] != '.')&&(sent.str[i] != ',')&&(sent.str[i]
!= '\0')){
      if (in_word == 0){
```

```
                                                    if (numb_word != -1)
{arr_words.sents[numb_word].str[arr_words.sents[numb_word].len] = '\0';}

            arr_words.size += 1;
            numb_word +=1;
            if (arr_words.size >= arr_words.avlb_size) {

               arr_words.avlb_size = arr_words.avlb_size + BASIC_LEN_TEXT;
               temp_text = realloc(arr_words.sents, arr_words.avlb_size*sizeof(Sentence));

               if (temp_text == NULL) {
                  for (i = 0; i < arr_words.size; i++) {
                     free(arr_words.sents[i].str);
                  }
                  free(arr_words.sents);
                  fprintf(stderr, "Ошибка перевыделения памяти для Text");
                  exit(1);
               } else {
                  arr_words.sents = temp_text;
               }
            }

            //выделение памяти для нового слова
            arr_words.sents[numb_word].avlb_len = BASIC_LEN_SENT;
                                                arr_words.sents[numb_word].str = malloc(sizeof
(wchar_t)*arr_words.sents[numb_word].avlb_len) ;
            arr_words.sents[numb_word].len = 0;
         }
         in_word = 1;
      }
      else{
         in_word = 0;
      }

      //добавление символа
```

```
        if (in_word == 1){
            if (arr_words.sents[numb_word].len  >= arr_words.sents[numb_word].avlb_len - 1) {

                        arr_words.sents[numb_word].avlb_len = arr_words.sents[numb_word].avlb_len + BASIC_LEN_SENT;
                                                    temp_sent = realloc(arr_words.sents[numb_word].str, arr_words.sents[numb_word].avlb_len*sizeof(wchar_t));

                if (temp_sent == NULL) {
                    free(arr_words.sents[numb_word].str);
                    fprintf(stderr, "Ошибка перевыделения памяти для sent");
                    exit(1);
                } else {
                    arr_words.sents[numb_word].str = temp_sent;
                }
            }
            arr_words.sents[numb_word].str[arr_words.sents[numb_word].len] = sent.str[i];
            arr_words.sents[numb_word].len++;

        }


    }
    return arr_words;
}




Sentence add_substr(Sentence sent , int numb , wchar_t* word) {
    wchar_t *temp_sent;
    int word_len = (int) wcslen(word), i, j;
```

```
    //перевыделение памяти
    if (sent.len >= sent.avlb_len - 1 - word_len) {

        sent.avlb_len = sent.avlb_len + word_len + 2;
        temp_sent = realloc(sent.str, sent.avlb_len * sizeof(wchar_t));

        if (temp_sent == NULL) {
            free(sent.str);
            fprintf(stderr, "Ошибка перевыделения памяти для sent");
            exit(1);
        } else {
            sent.str = temp_sent;
        }
    }

    for (i = 0; i < word_len; i++) {
        for (j = sent.len; j >= numb+i ; j--) {
            sent.str[j+1] = sent.str[j];

        }
        sent.str[numb+i] = word[i];
        sent.len++;
    }

    return sent;

}
```

Файл swap_word.h:

```
Sentence  find_and_swap_word(Sentence sent , wchar_t* word_old, wchar_t* word_new);
Sentence add_substr(Sentence sent , int numb , wchar_t* word);
Text cut_sent( Sentence sent);
```

Файл primitiv_func.c:

```
#include "top_header.h"
#include "primitiv_func.h"



void* mem_alloc(int len, int size){ // длинаа , размер типа

   void* p;

   p = malloc(len*size);

   if (p == NULL){
      fprintf(stderr,"Ошибка выделения памяти ");
      exit(1);
   }

   return p;
}


void delt_all_char( Sentence* sent,wchar_t  ch ){
   int i;
   for(i = 0; i < sent->len; i++){
      if (sent->str[i] == ch){
         delt_char(i,sent->str);
         i --;
         sent->len;
      }
   }

}
```

```
void delt_char(int num , wchar_t* str ){

   int i;

   for (i = num; str[i]!='\0';i++){
      str[i] = str[i+1];
   }
}
```

Файл primitive_func.h:

```
void* mem_alloc(int len, int size);
void delt_char(int num , wchar_t* str);
void delt_all_char( Sentence* sent,wchar_t  ch );
```

Файл write.c:

```
#include "top_header.h"
#include "primitiv_func.h"
#include "write.h"


void write(Text _text, int flag_n){
   wprintf(L"\nРезультат:\n");
   switch (flag_n) {

      case 1:
         if (_text.volume_gr_anagram == 0){
            wprintf(L"Ненайденно \n\n");
            break;
         }

         for (int i = 1; i <= _text.volume_gr_anagram; i++) {//вывод по групам анаграмм
            wprintf(L"%i.\n",i);
```

```
            for (int j = 0;j < _text.size;  j++){//проверка каждого sent на принадлежность к группе анаграмм
                if (_text.sents[j].group_anagram == i){
                    wprintf(L"\t%ls\n", _text.sents[j].str);
                }
            }
            wprintf(L"\n");

        }
        break;

    case 2:
        for (int i = 0, j = 1 ; i < _text.size; i++,j++) {
            delt_all_char(&_text.sents[i] , '\n');
            wprintf(L"%i.%ls (%i)\n",j, _text.sents[i].str,_text.sents[i].vol_upper);
            if (_text.sents[i].vol_upper == _text.sents[i+1].vol_upper){j-=1;}
        }
        wprintf(L"\n");
        break;

    case 3:
        for (int i = 0; i < _text.size; i++) {
            wprintf(L"%ls", _text.sents[i].str);
        }
        wprintf(L"\n\n");
        break;

    case 4:
        for (int i = 0; i < _text.size; i++) {
            wprintf(L"%ls", _text.sents[i].str);
        }
        wprintf(L"\n\n");
        break;

    case 6:
```

```
            for (int i = 0; i < _text.size; i++) {
                wprintf(L"%ls", _text.sents[i].str);
            }
            wprintf(L"\n\n");
            break;

    }
    wprintf(L"Чтобы продолжить нажмите Enter:\n");
    while ((getwchar ()) != '\n');
}
```

Файл write.h:

```
void write(Text _text, int flag_n);
```

Файл Makefike:

```
all: main.o read_text.o primitiv_func.o write.o anagram.o sort_upper.o one_on_two_char.o swap_word.o
        gcc main.o read_text.o primitiv_func.o write.o anagram.o sort_upper.o one_on_two_char.o swap_word.o -o res

main.o: main.c top_header.h read_text.h primitiv_func.h write.h anagram.h sort_upper.h one_on_two_char.h swap_word.h
        gcc -c main.c

read_text.o: read_text.c top_header.h read_text.h
        gcc -c read_text.c

primitiv_func.o: primitiv_func.c top_header.h primitiv_func.h
        gcc -c primitiv_func.c

write.o: write.c top_header.h write.h
        gcc -c write.c

anagram.o: anagram.c top_header.h anagram.h
        gcc -c anagram.c
```

```
sort_upper.o: sort_upper.c top_header.h primitiv_func.h sort_upper.h
	gcc -c sort_upper.c

one_on_two_char.o: one_on_two_char.c top_header.h one_on_two_char.h
	gcc -c one_on_two_char.c

swap_word.o: swap_word.c top_header.h swap_word.h
	gcc -c swap_word.c

clear:
	rm *.o res
```