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

Студент гр. 8303 _____ Гришин К. И.

Преподаватель _____ Чайка К. В.

Санкт-Петербург
2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Гришин К. И.

Группа 8303

Тема работы: Обработка текстовых данных

Исходные данные:

Написать программу, содержащую базовые функции для считывания и обработки текста.

Содержание пояснительной записки:

- Содержание
- Введение
- Описание работы программы
- Подробное описание функций
- Примеры работы программы
- Заключение
- Список используемых источников
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 27 страниц.

Дата выдачи задания: 19.11.2018

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Гришин К. И.

Преподаватель _____ Чайка К. В.

АННОТАЦИЯ

В этой работе была написана программа на языке С для обработки текста. Написаны функции для чтения текста из файла и его дальнейшего вывода, разбиения его на предложения и слова, дальнейшей его обработки. Обработка текста совершается путем введения пользователем команд в терминал. Сборка программы происходит посредством утилиты «make».

Исходный код приведен в приложении А.

SUMMARY

In this work the text processing program in the C programming language was written. There are the functions for text reading and output, splitting into the sentences and words, further processing. Text processing is performed by entering the commands into the terminal. The program is built using “make” utility.

The full source code can be found in application A.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка и написание программы, обрабатывающей исходный текст путем введения команд в терминал. Исходная длина текста и количество в нем предложений заранее неизвестны.

Для достижения поставленной цели требуется: написать функции для считывания и вывода текста, обработки текста, циклического ввода пользователем команд в терминал, создать *Makefile* для сборки программы.

1. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

На вход программе подается текст из файла, текст считывается по предложениям, пока не будет достигнут конец файла. Во время считывания проверяется наличие такого предложения (независимо от регистра) в тексте, и если факт повтора обнаружен, предложение игнорируется. Весь текст хранится в памяти разделенным на предложения. После удачного чтения текста запускается цикл запроса команд, пока пользователь не введет команду выхода из программы или не прервет программу путем подачи сигналов «ctrl-c» или «ctrl-d».

1.1. Ввод и вывод

В связи с тем, что заранее размер текста неизвестен, а выделение заведомо большого количества памяти нерационально, используется принцип посимвольного чтения из документа и динамического выделения памяти с помощью функций стандартной библиотеки. Для чтения текста используется функция *readText*, которая считывает предложения и динамически выделяет по 5 сегментов памяти под хранение предложений. Сами же предложения считываются с помощью функции *readSentence* и для хранения предложения динамически выделяется по 10 сегментов памяти для хранения символов. Функции ввода имеют схожие сигнатуры и в целом построение:

int readText(struct Text text);* и *int readSentence(struct Sentence* text);*

Эти функции циклически забивают информацию в память пока не достигнут терминального значения, для *readSentence* — это один из знаков окончания предложения (*«. ? !»*), для *readText* — это конец файла. Память под хранения выделяется посредством функций *«malloc»* и *«realloc»* стандартной библиотеки *«stdlib.h»*.

Для вывода написаны функции *printText* и *printSentence*, первая циклически (в зависимости от количества предложений) вызывает вторую, которая собственно выводит строки.

1.2. Форматирование текста

Хранение исходного и отформатированного текста решено было хранить одновременно, что позволяет обрабатывать текст несколько раз. Отформатированный текст можно вывести после его успешной обработки, для того, чтобы отформатировать исходный текст снова, программа потребует от пользователя ввести команду очистки «формата» — памяти в которой хранится отформатированный текст.

Пользователю доступны следующие способы обработки текста:

1. Удаление предложений, в которых есть слова начинающиеся с маленькой буквы. Используется функция *sentenceCaps*.
2. Сортировка слов в каждом предложении по количеству гласных букв. Используется функция *vowSort*.
3. Вывод повторяющихся слов в каждом предложении, а также количества таких повторений. Используется функция *wordsCount*.
4. Составление общей маски из всех слов для каждого предложения. Используется функция *strPattern*.

2. ПОДРОБНОЕ ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ

Используемые функции классифицируются на функции для обработки текста и функции для обработки предложений. Первые имеют вид цикла, который последовательно применяет вторые для каждого предложения.

Функции для работы с текстом хранятся в файле *«text.c»*, а объявления этих функций хранятся в заголовочном файле *«text.h»*. Функции для работы с предложениями хранятся в файле *«sentence.c»*, а объявления этих функций хранятся в заголовочном файле «sentence.h».

Также, помимо объявлений функций, заголовочные файлы хранят объявления структур для хранения текста и предложений и нужные для обработки предложений макросы: END — обозначающий конец файла, VOWELS — в котором хранятся гласные и DEL — в котором хранятся разделители.

2.1. Функции readText, readSentence и sentenceTest

В *readText* сначала открывается файл и выделяется начальная память под 5 структур *sentence*, затем начинается цикл, который работает пока не будет достигнут конец файла. Цикл каждую пятую иттерацию добавляет к памяти по пять сегментов памяти для следующих структур *sentence* с помощью функции *realloc* стандартной библиотеки. Как было описано ранее, большинство функций файла *text* иттеративно вызывают аналогичные функции файла *sentence*, так и целью цикла функции *readText* является вызов функции *readSentence*, после прочтения предложения оно проходит проверку с помощью функции *sentenceTest* и если такого предложения ранее не встречалось в тексте, то оно добавляется в *text*.

Функция *sentenceTest* получает на вход предложение и текст, и проверяет текущее предложение с каждым в тексте. Сравнивание двух предложений происходит следующим образом, вначале сравниваются длины предложений, если они совпадают, то далее идет посимвольное сравнивание двух предложений, при несовпадении цикл прерывается.

Функция *readSentence* изначально запрашивает память под 10 символов, затем с помощью функции стандартной библиотеки *fgetc* пропускает все пробельные символы, а именно: пробел, табуляция, перенос строки. После того, как все пробельные символы были пропущены запускается цикл по посимвольному чтению строки и дальнейшему динамическому выделению памяти с помощью функции *realloc*, чтение происходит, пока не будет прочитан символ конца предложения (*«. ? !»*) .

2.2. Функции printText и printSentence

Функция *printText* принимает на вход структуру *Text* и далее используя поле *sizeText* отвечающее за количество предложений запускает цикл, в котором вызывает функцию *printSentence*.

Функция *printSentence*, выводит на экран строку *sentence* в структуре *Sentence* функцией *wprintf* стандартной библиотеки.

2.3. Функция cleanText

Функция иттеративно, используя поле *sizeText* структуры *Text* освобождает память, занимаемую строками структур *Sentence*.

2.4. Функции textCaps и sentenceCaps

В функцию *textCaps* сообщается структура *Text* с прочитанным текстом — input, и свободная структура *Text* — output. Сама же функция иттеративно, с помощью поля *sizeText*, проверяет каждую строку input с помощью функции *sentenceCaps*, и строка удовлетворяет условию, под нее выделяется память и она переносится в output.

Функция *sentenceCaps* используя функцию *wcstok* стандартной библиотеки разбивает строку на слова и проверяет первую букву каждого слова, является ли она строчной, как только было обнаружено несовпадение функция возвращает 0, если все слова удовлетворяют условию, функция возвращает 1.

2.5. Функции textVowSort, vowSort и cmp

Функция *textVowSort* получает на вход структуру *Text* — input и свободную структуру *Text* — output. Полям *output.sizeSentence* и *output.memSentence* присваиваются значения структуры input. По значению *output.memSentence* по полю *output.Text* выделяется память с помощью функции *malloc* стандартной библиотеки. Далее циклично в output добавляются строки, обработанные функцией *vowSort*.

Функция *vowSort* получает на вход структуру *Sentence*. Предложение разбивается на слова и помещается в массив *words*. Далее, с помощью функции *qsort* стандартной библиотеки, слова в массиве сортируются нужным образом с помощью написанного компаратора *cmp*.

Компаратор *cmp* считает количество гласных в каждом слове и возвращает разность этих количеств.

2.6. Функции textWordsCount и wordsCount

На вход функции *textWordsCount* подается две структуры *Text*, одна с исходным текстом — input, другая пустая — output. Полям *output.memSentence* и *output.sizeSentence* присваиваются значения структуры input, затем по значению *output.memSentence* выделяется память в структуре output. Затем иттеративно с, помощью значения *output.sizeText*, каждая строка обрабатывается функцией *wordsCount*.

Функция *wordsCount* создается массив слов из предложения с помощью функции стандартной библиотеки *wcstok*. а также массив слов в который будут забиваться повторяющиеся слова. Затем циклично по каждому из слов ищется количество его повторений в массиве слов, если повторений больше одного (то есть слово встречается больше одного раза) и этого слова нет в массиве повторяющихся слов, оно добавляется в этот массив и записывается в строку вывода — *result* с помощью функции *swprintf* стандартной библиотеки. После цикла, если строка вывода оказывается пустой, в нее записывается строка *„Повторений нет“*.

2.7. Функции textPattern, strPatter и getMask

На вход функции *textWordsCount* подается две структуры *Text*, одна с исходным текстом — input, другая пустая — output. Полям *output.memSentence* и *output.sizeSentence* присваиваются значения структуры input, затем по значению *output.memSentence* выделяется память в структуре output. Затем иттеративно с, помощью значения *output.sizeText*, каждая строка обрабатывается функцией *strPattern*.

На вход функции *strPattern* подается структура *Sentence*. С помощью функции *wcstok* стандартной библиотеки предложение разбивается на слова и заносится в массив слов *words*. Далее, используя *„жадный алгоритм“,* с помощью написанной функции getMask, выделяется маска из всех слов в массиве. *„Жадный алгоритм“* означает, что вначале находится маска по первым двум

словам, потом по текущей маске и третьему слову, потом по текущей маске и четвертому слову и т.д.

Функция *getMask* получает на вход две строки, условно маску — *mask* и слово, которое должно дополнить маску, — *word*, а также длину наибольшего слова в массиве слов который был построен в функции *strPattern*. Из маски убираются крайние звездочки если они есть и при этом запоминается их наличие или отсутствие. Инициализируются переменные *befTmp*, *aftTmp*, *upStr* и *dwStr* для перемещения двух слов относительно друг друга и их посимвольного сравнивания. Инициализируются переменные: *emptyMask* — для хранения пустой маски (то есть состоящей только из знаков вопроса), *unstarMask* — для хранения текущей маски без учета крайних звездочек и *outMask* — для хранения маски которая пойдет на выход функции и которая содержит в себе крайние звездочки. Далее циклично две строки перемещаются относительно друг друга, сравниваются посимвольно и считается количество совпадений символов, по этому количеству определяется самая оптимальная маска эта маска записывается в переменную *unstarMask*, если совпадений не было, то в *unstarMask* записывается значение *emptyMask*. После того, как была получена маска без крайних звездочек, происходит проверка на надобность этих звездочек, а также побитовое или с исходными звездочками. Как только мы получаем готовые значения для *unstarMask* и крайних звездочек все это вместе конкатенируется с помощью функции *swprintf* стандартной библиотеки и идет на выход.

2.8. Функция main

Пользователю предлагается ввести текст в файл *input.txt*, при выполнении запроса требуется выполнить команду *«y»*, либо завершить работу программу нажатием клавиши *«enter»*, далее запускается бесконечный цикл, ожидающий нужного ввода. После успешного чтения файла, пользователю выводится список доступных функций:

0: Повтор меню
1: Вывод текста
2: Формат. Удаление предложений в которых есть слова начинающиеся с маленькой буквы
3: Формат. Сортировка слов в предложениях по количеству гласных букв
4: Формат. Вывод повторяющихся слов, а также количество повторений
5: Формат. Составление общей маски из слов в предложении
6: Очистка формата
7: Вывод формата

затем запускается бесконечный цикл, который ожидает ввода требуемой функции либо нажатия клавиши *«enter»* для выхода из программы. Определение введенной пользователем команды происходит с помощью оператора *switch*, при вводе существующей функции программа обработает текст так, как то оговорено в условии, после чего отформатированный текст можно будет вывести.

3. ПРИМЕР РАБОТЫ ПРОГРАММЫ

Исходный текст:

From Fairest Creatures We Desire Increase, That Thereby From Beautys Rose Might We Never Die We. But as the riper should by time riper decease, His tender heir might bear his memory: but thou, but contracted to thou thine thine own bright eyes but. flame Feed'st thy light's flame thy flame with self-substantial fuel flame. Making A Famine Where Abundance Lies. Chester Cheetah Chews Cheeps.

Обработанный текст (функции указаны в соответствие с меню данным пользователю):

2	From Fairest Creatures We Desire Increase, That Thereby From Beautys Rose Might We Never Die We. Making A Famine Where Abundance Lies. Chester Cheetah Chews Cheeps. end of file
3	Creatures Increase Beautys Fairest Desire Thereby Rose Never Die From We That From Might We We decease memory contracted eyes riper should time riper tender heir bear thou thou thine thine But as the by His might his but but to own bright but

	substantial flame Feed flame flame fuel flame thy light thy with self st s Abundance Famine Making Where Lies A Cheetah Chester Cheeps Chews end of file
4	From: 2; We: 3; riper: 2; but: 3; thou: 2; thine: 2; flame: 4; thy: 2; Повторений нет Повторений нет end of file
5	*??* *??* *?* *?* che??* end of file

ЗАКЛЮЧЕНИЕ

В ходе работы была написана программа обрабатывающая исходный текст по запросу пользователя. Были написаны функции, которые позволяют обработать текст или предложение. Все функции классифицированы на два основных типа и разбиты по соответствующим файлам. Для сборки программы написан *Makefile*. Были закреплены знания по использованию функций стандартной библиотеки для работы с широкими символами и строками.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация к языку C++ // Reference — C++ Reference. URL: http://www.cplusplus.com/reference/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл: main.c

```
#include "sentence.h"
#include "text.h"

void menu(){
   wprintf(L"\nВам предложено меню, выберите требуемую функцию нажатием вводом
соответствующей цифры\n");
   wprintf(L"0: Повтор меню\n");
   wprintf(L"1: Вывод текста\n");
   wprintf(L"2: Формат. Удаление предложений в которых есть слова начинающиеся с
маленькой буквы\n");
   wprintf(L"3: Формат. Сортировка слов в предложениях по количеству гласных букв\n");
   wprintf(L"4: Формат. Вывод повторяющихся слов, а также количество повторений\n");
   wprintf(L"5: Формат. Составление общей маски из слов в предложении\n");
   wprintf(L"6: Очистка формата\n");
   wprintf(L"7: Вывод формата\n");
   wprintf(L"Для выхода из программы нажмите enter или ctrl+c\n\n");
}

int main(){
   setlocale(LC_CTYPE, "");

   wprintf(L"Поместите требуемый для обработки текст в файл input.txt\n");
   wprintf(L"Если текст введен в требуемый файл, для продолжения нажмите
введите \"y\" \n");
   wprintf(L"Если хотите выйти из программы, нажмите enter или ctrl+c\n");
   for(;;){
      wchar_t exitC = fgetwc(stdin);
      if (exitC == L'\n') return 0;
      ungetwc(exitC, stdin);
      wchar_t f = fgetwc(stdin);
      wchar_t inpC = fgetwc(stdin);
      if (inpC == L'\n'){
         if (f == L'y') break;
         else continue;
      }
      else{
         do
            inpC = fgetwc(stdin);
         while (inpC != L'\n');
         wprintf(L"Wrong input\n");
         continue;
      }
   }
   putwc(L'\n', stdout);


   struct Text input;
   struct Text format;
   int formatInd = 0;
   readText(&input);

   menu();


   for(;;){
```

```
int f;
wchar_t exitC = fgetwc(stdin);
if (exitC == L'\n') break;
ungetwc(exitC, stdin);
wscanf(L"%d", &f);
wchar_t inpC = fgetwc(stdin);
if (inpC == L'\n'){

   switch (f){
      case 0:
         menu();
         break;

      case 1:
         printText(input);
         putwc(L'\n', stdout);
         break;

      case 2:
         if (formatInd) { wprintf(L"Требуется очистка формата\n"); putwc(L'\n', stdout); }
         else{
            textCaps(input, &format);
            formatInd = 1;
            putwc(L'\n', stdout);
         }
         break;

      case 3:
         if (formatInd) { wprintf(L"Требуется очистка формата\n"); putwc(L'\n', stdout); }
         else{
            textVowSort(input, &format);
            formatInd = 1;
            putwc(L'\n', stdout);
         }
         break;

      case 4:
         if (formatInd) { wprintf(L"Требуется очистка формата\n"); putwc(L'\n', stdout); }
         else{
            textWordsCount(input, &format);
            formatInd = 1;
            putwc(L'\n', stdout);
         }
         break;

      case 5:
         if (formatInd) { wprintf(L"Требуется очистка формата\n"); putwc(L'\n', stdout); }
         else{
            textPattern(input, &format);
            formatInd = 1;
            putwc(L'\n', stdout);
         }
         break;

      case 6:
         if (formatInd){
            cleanText(format);
            wprintf(L"Формат очищен\n");
            formatInd = 0;
            putwc(L'\n', stdout);
         }
```

```
                else { wprintf(L"Формат пуст\n"); putwc(L'\n', stdout); }
                break;

            case 7:
                if (!formatInd) { wprintf(L"Формат пуст\n"); putwc(L'\n', stdout); }
                else{
                    printText(format);
                    putwc(L'\n', stdout);
                }
                break;

            default: wprintf(L"Wrong function\n");  break;
        }

    }
    else{
        do
            inpC = fgetwc(stdin);
        while (inpC != L'\n');
        wprintf(L"Wrong input\n");
        continue;
    }
  }
  return 0;
}
```

Файл: text.c

```
#include "text.h"
#include "sentence.h"

   // INPUT OUTPUT
int sentenceTest(struct Sentence sent, struct Text text)
   {
   int count = 0;
   for (int i = 0; i < text.sizeText; i++){
      if (sent.sizeSentence != text.text[i].sizeSentence) continue;
      for (int j = 0; j < sent.sizeSentence; j++){
         if (towupper(sent.sentence[j]) != towupper(text.text[i].sentence[j])) break;
      }
   count++;
   }
   if (count) return 1;
   else return 0;
}


int readText(struct Text* text){
   setlocale(LC_CTYPE, "");

   wprintf(L"Открытие файла: ");
   FILE* file = fopen("input.txt", "r");
   if (file == NULL) { wprintf(L"Ошибка.\n"); return -1;}
   else wprintf(L"Выполнено\n");

   text->memText = 6;
   text->text = (struct Sentence*) malloc(text->memText * sizeof(struct Sentence));

   readSentence(&(text->text)[0], file);

   for(text->sizeText = 1; wcscmp((text->text)[text->sizeText-1].sentence, END); text-
>sizeText++){
```

```
                // Выделение памяти по надобности.
        if (text->sizeText % 5 == 0){
            text->memText += 5;
            struct Sentence* textBuff = (struct Sentence*) realloc(text->text, text->memText *
sizeof(struct Sentence));
            if (textBuff == NULL) return -1; // -1 - TEXT REALLOC ERROR
            else text->text = textBuff;
        }
        struct Sentence tmp;
        readSentence(&tmp, file);
        int i = sentenceTest(tmp, *text);
        if (i) text->sizeText--;
        else text->text[text->sizeText] = tmp;
        //readSentence(&((text->text)[text->sizeText]), file);
        (text->text)[text->sizeText + 1].sentence = NULL;
    }
    wprintf(L"Чтение файла закончено\n");

    wprintf(L"Закрытие файла: ");
    if (fclose (file) == WEOF) wprintf(L"Ошибка\n");
    else wprintf (L"Выполнено\n");

    if(text->sizeText == 1){
        (text->text)[0].sentence = L"file is empty";
    }

    return text->sizeText;
}


int printText(struct Text text){
    int i;
    for(i = 0; i < text.sizeText/*text.text[i].sentence*/; i++) printSentence(text.text[i]);
    return i;
}


    // FORMAT
int cleanText(struct Text format){
    for(int i = 0; i < format.sizeText-1; i++) free(format.text[i].sentence);
    free(format.text);
    return 0;
}


int textCaps(const struct Text input, struct Text* output){
    output->memText = 1;
    output->sizeText = 0;
    wprintf(L"Выделение памяти: ");
    struct Sentence* buffer = malloc(sizeof(struct Sentence));
    if (buffer == NULL) { wprintf(L"Ошибка\n"); return -1;}
    else { output->text = buffer; wprintf(L"Выполнено\n"); }
    wprintf(L"Форматирование: ");
    for (int i = 0; i < input.sizeText-1; i++){
        if (sentenceCaps(input.text[i])){
            output->memText++;
            buffer = realloc(output->text, output->memText*sizeof(struct Sentence));
            if (buffer == NULL) { wprintf(L"Ошибка\n"); return -1; }
            else output->text = buffer;
            //output->text[output->sizeText] = input.text[i];
            output->text[output->sizeText].memSentence = input.text[i].memSentence;
```

```
            output->text[output->sizeText].sizeSentence = input.text[i].sizeSentence;
            wchar_t* strBuffer = malloc(input.text[i].memSentence * sizeof(wchar_t));
            if (strBuffer == NULL) { wprintf(L"Ошибка\n"); return -1; }
            output->text[output->sizeText].sentence = strBuffer;
            wcscpy(output->text[output->sizeText].sentence, input.text[i].sentence);
            output->sizeText++;
        }
    }
    wprintf(L"Выполнено\n");
    output->text[output->sizeText++] = input.text[input.sizeText-1];
    return 0;
}


int textVowSort(const struct Text input, struct Text* output){
    output->memText = input.memText;
    output->sizeText = input.sizeText;
    wprintf(L"Выделение памяти: ");
    struct Sentence* buffer = malloc(output->memText*sizeof(struct Sentence));
    if (buffer == NULL) { wprintf(L"Ошибка\n"); return -1; }
    else output->text = buffer; wprintf(L"Выполнено\n");
    wprintf(L"Форматирование: ");
    for(int i =0; i < input.sizeText-1; i++)
        output->text[i].sentence = vowSort(input.text[i]);
    output->text[output->sizeText-1].sentence = END;
    wprintf(L"Выполнено\n");
    return 0;
}


int textWordsCount(const struct Text input, struct Text* output){
    output->memText = input.memText;
    output->sizeText = input.sizeText;
    wprintf(L"Выделение памяти: ");
    struct Sentence* buffer = malloc(output->memText*sizeof(struct Sentence));
    if (buffer == NULL) { wprintf(L"Ошибка\n"); return -1; }
    else output->text = buffer; wprintf(L"Выполнено\n");
    wprintf(L"Форматирование: ");
    for(int i =0; i < input.sizeText-1; i++){
        output->text[i] = wordsCount(input.text[i]);
    }
    output->text[output->sizeText-1].sentence = END;
    wprintf(L"Выполнено\n");
    return 0;
}


int textPattern(const struct Text input, struct Text* output){
    //setlocale(LC_CTYPE, "");
    output->memText = input.memText;
    output->sizeText = input.sizeText;
    wprintf(L"Выделение памяти: ");
    struct Sentence* buffer = malloc(output->memText*sizeof(struct Sentence));
    if (buffer == NULL) { wprintf(L"Ошибка\n"); return -1; }
    else output->text = buffer; wprintf(L"Выполнено\n");
    wprintf(L"Форматирование: ");
    for(int i =0; i < input.sizeText-1; i++){
        output->text[i] = strPattern(input.text[i]);
    }
    output->text[output->sizeText-1].sentence = END;
    wprintf(L"Выполнено\n");
```

```
    return 0;
}
```

Файл: text.h

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


struct Text{
    struct Sentence* text;
    int memText;
    int sizeText;
};

int sentenceTest(struct Sentence sent, struct Text text);
int readText(struct Text* text);
int printText(struct Text text);

          // Format text
int cleanText(struct Text format);
int textCaps(const struct Text input, struct Text* output);
int textVowSort(const struct Text input, struct Text* output);
int textWordsCount(const struct Text input, struct Text* output);
int textPattern(const struct Text input, struct Text* output);
```

Файл: sentence.c

```
#include "sentence.h"


    // INPUT OUTPUT

int readSentence(struct Sentence* sent, FILE* file){
    setlocale(LC_CTYPE, "");

    if (file == NULL) { wprintf(L"Ошибка.\n"); return -1; }

    sent->memSentence = 12;
    sent->sentence = (wchar_t*) malloc(sent->memSentence * sizeof(wchar_t));
    if (sent->sentence == NULL) { wprintf(L"Ошибка.\n"); return -2;} // MALLOC ERROR

    wchar_t inpC;
    do
        inpC = fgetwc(file);
    while(inpC == L' ' || inpC == L'\t' || inpC == L'\n');

    *(sent->sentence) = inpC;
    for(sent->sizeSentence = 1; inpC != L'.' && inpC != L'?' && inpC != L'!' && inpC != WEOF;
sent->sizeSentence++){
                    // Выделение памяти по надобности
        if (sent->sizeSentence % 10 == 0){
            sent->memSentence += 10;
            wchar_t* sentenceBuff = (wchar_t*) realloc(sent->sentence, sent->memSentence *
sizeof(wchar_t));
            if (sentenceBuff == NULL) { wprintf(L"Ошибка расширения памяти.\n"); return -3; } //
SENTENCE REALLOC ERROR
            else sent->sentence = sentenceBuff;
        }
        inpC = fgetwc(file);
        if (inpC == '\n' || inpC == '\t') inpC = ' ';
        (sent->sentence)[sent->sizeSentence] = inpC;
        (sent->sentence)[sent->sizeSentence + 1] = 0;
```

```
    }

    if (inpC == WEOF){
        free(sent->sentence);
        sent->sentence = END;
    }

    return 0;
}

int printSentence(struct Sentence sent){
    wprintf(L"%ls\n", sent.sentence);
    return 0;
}


    // FORMAT

int sentenceCaps(const struct Sentence sent){
    wchar_t* buffer = malloc(sent.memSentence*sizeof(wchar_t));
    wcscpy(buffer, sent.sentence);
    wchar_t* bufferPoint = buffer;
    wchar_t* word = wcstok(buffer, DEL, &buffer);
    for(;word;){
        if (iswlower(*word)) return 0;
        word = wcstok(buffer, DEL, &buffer);
    }
    free(bufferPoint);
    return 1;
}


        // Компаратор для vowSort
int cmp(const void* a, const void* b){
    wchar_t* wordA = *(wchar_t**) a;
    wchar_t* wordB = *(wchar_t**) b;
    int countA = 0;
    int countB = 0;
    for (int i = 0; i < wcslen(wordA); i++)
        if (wcschr(VOWELS, wordA[i])) countA++;
    for (int i = 0; i < wcslen(wordB); i++)
        if (wcschr(VOWELS, wordB[i])) countB++;
    return countB - countA;
}


wchar_t* vowSort(struct Sentence sent){
    wchar_t** words = (wchar_t**) malloc(sizeof(wchar_t*));

    wchar_t* buffer = malloc(sent.memSentence*sizeof(wchar_t)); // выделенная память buffer
    wcscpy(buffer, sent.sentence);
    void* bufferPoint = buffer;

    wchar_t* word = wcstok(buffer, DEL, &buffer);
    words[0] = word;
    int count;
    for (count = 1; word; count ++){
        wchar_t** wordsBuffer = (wchar_t**) realloc(words, (count+1)*sizeof(wchar_t*));
        if (wordsBuffer == NULL) return NULL;
        else words = wordsBuffer;
        word = wcstok(buffer, DEL, &buffer);
```

```
        words[count] = word;
    }
    qsort(words, count-1, sizeof(wchar_t*), cmp);

    wchar_t* res = calloc(sent.memSentence, sizeof(wchar_t));
    for (int i = 0; words[i]; i++) wcscat(wcscat(res, words[i]), L" ");

    free(words);
    free(bufferPoint);
    return res;
}


struct Sentence wordsCount(struct Sentence sent){
    struct Sentence error = {NULL, -1, -1};
    int count;

    wchar_t** words = malloc(sizeof(wchar_t*));

    wchar_t* buffer = malloc(sent.memSentence*sizeof(wchar_t));
    wcscpy(buffer, sent.sentence);

    void* bufferPoint = buffer;

        // Разбиение строки на массив слов
    wchar_t* word = wcstok(buffer, DEL, &buffer);
    for(count = 0; word; count++){
        wchar_t** wordsBuffer = (wchar_t**) realloc(words, (count+1)*sizeof(wchar_t*));
        if (wordsBuffer == NULL) return error;
        else words = wordsBuffer;
        words[count] = word;
        word = wcstok(buffer, DEL, &buffer);
    }


    int h = 0;
    wchar_t* result = calloc(1, sizeof(wchar_t));
    int memResult = 0;
    wchar_t** repeats = malloc(sizeof(wchar_t*));
    for(int j = 0; j < count; j++){
        int sk = 0;
        int f = 0;
        for(int k = 0; k < count; k++)
            if (wcscmp(words[j], words[k]) == 0) sk++;
        for(int d = 0; d < h; d++)
            if (wcscmp(words[j], repeats[d]) == 0) f++;
        if(sk > 1 && f == 0){
            wchar_t** repeatsBuffer = (wchar_t**) realloc(repeats, (h+1)*sizeof(wchar_t*));
            if (repeatsBuffer == NULL) return error; // Ошибка выделения памяти;
            else repeats = repeatsBuffer;

            int tmpSk = sk;
            int digits;
            for(digits =0; tmpSk; digits++) tmpSk /= 10;
            int currentSize = wcslen(words[j]) + 5 + digits;
            wchar_t* currentRes = malloc(currentSize*sizeof(wchar_t));
            swprintf(currentRes, currentSize, L"%ls: %d; ", words[j], sk);

            memResult += currentSize;
            wchar_t* resultBuffer = (wchar_t*) realloc(result, memResult*sizeof(wchar_t*));
            if (resultBuffer == NULL) return error;
```

```
            else result = resultBuffer;

            wcscat(result, currentRes);
            free(currentRes);

            repeats[h] = words[j];
            h++;
        }
    }
    if (memResult == 0){
        memResult = 16;
        wchar_t* resultBuffer = (wchar_t*) realloc(result, memResult*sizeof(wchar_t*));
        if (resultBuffer == NULL) return error;
        else result = resultBuffer;

        swprintf(result, memResult, L"Повторений нет ");
    }


    result[wcslen(result)-1] = 0;
    struct Sentence out = {result, memResult, wcslen(result)};

    free(bufferPoint);
    free(repeats);
    free(words);
    return out;
}


struct Sentence strPattern (struct Sentence sent){
    struct Sentence error = {NULL, -1, -1};
    int count;

    wchar_t** words = malloc(sizeof(wchar_t*));

    wchar_t* buffer = malloc(sent.memSentence*sizeof(wchar_t));
    wcscpy(buffer, sent.sentence);

    void* bufferPoint = buffer;

        // Разбиение строки на массив слов
    wchar_t* word = wcstok(buffer, DEL, &buffer);
    int mlen = 0;
    for(count = 0; word; count++){
        for(int i = 0; i < wcslen(word); i++) word[i] = tolower(word[i]);

        int len = wcslen(word);
        if (len > mlen) mlen = len;

        wchar_t** wordsBuffer = (wchar_t**) realloc(words, (count+1)*sizeof(wchar_t*));
        if (wordsBuffer == NULL) return error;
        else words = wordsBuffer;
        words[count] = word;
        word = wcstok(buffer, DEL, &buffer);
    }

        // Выделение маски
    wchar_t* mask = calloc(mlen+3, sizeof(wchar_t));
    wcscpy(mask, words[0]);
    for (int i = 0; i < count; i++){
        mask = getMask(mask, words[i], mlen+1);
```

```
    }

    free(bufferPoint);
    free(words);
    struct Sentence output = {mask, mlen+3, wcslen(mask)};
    return output;
}



wchar_t* getMask(wchar_t* mask, wchar_t* word, int maxlen){
    setlocale(LC_CTYPE, "");

    int lstar_inp = 0;
    int rstar_inp = 0;
    if (mask[0] == L'*') lstar_inp = 1;
    if (mask[wcslen(mask) - 1] == L'*') rstar_inp = 1;
    mask[wcslen(mask) - rstar_inp] = 0;
    mask += lstar_inp;

    int mask_len = wcslen(mask);
    int word_len = wcslen(word);

    wchar_t* befTmp = calloc((maxlen+1), sizeof(wchar_t));
    wchar_t* aftTmp = calloc((maxlen+1), sizeof(wchar_t));
    int maxCongs = 0;
    wchar_t* unstarMask = calloc((maxlen+3), sizeof(wchar_t));
    wchar_t* outMask = calloc((maxlen+3), sizeof(wchar_t));
    wchar_t* emptyMask = malloc((maxlen+1)*sizeof(wchar_t));
    int lstar;
    int lstar_tmp;
    int rstar;
    int rstar_tmp;
    for(int i = -word_len + 1; /*i < 1*/ i < mask_len; i++){
        lstar_tmp = 0;
        rstar_tmp = 0;
        wchar_t* upStr = calloc((maxlen+1), sizeof(wchar_t));
        wchar_t* dwStr = calloc((maxlen+1), sizeof(wchar_t));
        if (i < 1){
            swprintf(befTmp, maxlen, L"%ls", word - i);
            wcsncpy(upStr, mask, wcslen(befTmp));
            wcsncpy(dwStr, befTmp, wcslen(befTmp));
        }
        else{
            swprintf(aftTmp, maxlen, L"%ls", mask + i);
            wcsncpy(upStr, aftTmp, wcslen(aftTmp));
            wcsncpy(dwStr, word, wcslen(aftTmp));
        }

        if (i != 0) lstar_tmp = 1;
        if (i != -word_len + mask_len) rstar_tmp = 1;

        wchar_t* tmpMask = malloc((maxlen+3)*sizeof(wchar_t));
        int congs = 0;
        int symbMarks = (wcslen(dwStr) < wcslen(upStr)) ? wcslen(dwStr) : wcslen(upStr);
        for (int j = 0; j < symbMarks; j++){
            if (upStr[j] == dwStr[j]){
                tmpMask[j] = upStr[j];
                congs++;
            }
            else tmpMask[j] = L'?';
```

```
            tmpMask[j+1] = 0;
        }

        if (i == 0) wcscpy(emptyMask, tmpMask);

        if (congs >= maxCongs){
            lstar = lstar_tmp;
            rstar = rstar_tmp;
            maxCongs = congs;
            wcscpy(unstarMask, tmpMask);
        }
        free(upStr);
        free(dwStr);
        free(tmpMask);
    }

    if (maxCongs == 0) wcscpy(unstarMask, emptyMask);

    lstar |= lstar_inp;
    rstar |= rstar_inp;
    switch (lstar){
        case 1:
            swprintf(outMask, maxlen+2, L"%lc%ls%lc", L'*', unstarMask, (rstar == 1) ? L'*' : L'\0');
            break;
        case 0:
            swprintf(outMask, maxlen+2, L"%ls%lc", unstarMask, (rstar == 1) ? L'*' : L'\0');
            break;
    }

    free(emptyMask);
    free(befTmp);
    free(aftTmp);
    free(unstarMask);
    return outMask;
}
```

Файл: sentence.h

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

#define END L"end of file"
#define VOWELS L"EYUIOAeyuioaУАОЭЯИЮЁуаоэяиюё"// гласные для функции vowSort
#define DEL L" ,;:'\"\\/.?!`_-\n"// разделители для wcstok

struct Sentence{
    wchar_t* sentence;
    int memSentence;
    int sizeSentence;
};


int readSentence(struct Sentence* sentence, FILE* file);
int printSentence(struct Sentence sentence);

        // Format Sentence
int sentenceCaps(const struct Sentence sent);
```

```
wchar_t* vowSort(struct Sentence sent);
struct Sentence wordsCount(struct Sentence sent);
struct Sentence strPattern(struct Sentence string);


    // Extra funcs
int cmp(const void* a, const void* b);
wchar_t* getMask(wchar_t* mask, wchar_t* word, int maxlen);
```

Файл: Makefile

```
all: start
	./start

start: main.o text.o sentence.o
	gcc -Wall main.o text.o sentence.o -o start

main.o: main.c text.h sentence.h
	gcc -c main.c

text.o: text.c text.h sentence.h
	gcc -c text.c

sentence.o: sentence.c sentence.h
	gcc -c sentence.c

clean:
	rm -rf *.o start
```