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

# КУРСОВАЯ РАБОТА

**по дисциплине «Программирование»**

**Тема: Обработка текста**

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

Студент Колованов Р.А.

Группа 9381

Тема работы: Обработка текста

Исходные данные:

- Язык программирования C
- Компилятор GCC
- Система Linux

Содержание пояснительной записки:

- Введение
- Формулировка задания
- Написание программы
- Тестирование программы
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 15.10.2019

Дата сдачи реферата: 12.12.2019

Дата защиты реферата: 12.12.2019

Студент _____ Колованов Р.А.

Преподаватель _____ Жангиров Т.Р.

## АННОТАЦИЯ

Была разработана программа, выполняющая обработку текста. На вход программе подается текст заранее неизвестной длины из латинских и кириллических букв. Программа удаляет все повторяющиеся предложения и в зависимости от команды пользователя выполняет определенную обработку текста. Был использован метод восходящей разработки. Для хранения текста были реализованы структуры *Sentence* и *Text*. Операции со строками осуществляются при помощи функций стандартной библиотеки. Помимо этого, был написан Makefile для компиляции исходного кода программы. Было проведено тестирование программы.

## SUMMARY

A program has been developed that performs text processing. The text of an unknown length of Latin and Cyrillic letters is input to the program. The program deletes all duplicate sentences and, depending on the user's command, performs certain text processing. An upstream development method was used. Sentence and Text structures were implemented to store text. String operations are performed using the functions of the standard library. In addition, a Makefile was written to compile the source code of the program. The program was tested.

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

**Цель работы.**

Разработка программы, которая считывает и обрабатывает текст в зависимости от команды пользователя.

**Задачи.**

- Изучение языка программирования C
- Написание исходного кода программы
- Написание Makefile-а
- Компиляция программы
- Тестирование программы

# 1. ФОРМУЛИРОВКА ЗАДАНИЯ

**Вариант 3.**

Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой) Длина текста и каждого предложения заранее не известна.

Для хранения предложения и для хранения текста требуется реализовать структуры Sentence и Text.

Программа должна сохранить (считать) текст в виде динамического массива предложений и оперировать далее только с ним. Функции обработки также должны принимать на вход либо текст (Text), либо предложение (Sentence).

Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).

Далее, программа должна запрашивать у пользователя одно из следующих доступных действий (программа должна печатать для этого подсказку. Также следует предусмотреть возможность выхода из программы):

1. Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении. Строка условия содержит: символы, ? - 1 или больше любых символов, в начале и конце образца могут быть символы * - обозначающие 0 или больше символов. Например, для слов “Аристотель” и “Артишок”, строка образец будет иметь вид “Ар???о?*”.
2. Удалить все предложения, в которых нет заглавных букв в начале слове.
3. Отсортировать слова в предложении по количеству гласных букв в слове.
4. Для каждого предложения вывести количество одинаковых слов в строке.

Все сортировки и операции со строками должны осуществляться с использованием функций стандартной библиотеки. Использование

собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

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

# 2. НАПИСАНИЕ ИСХОДНОГО КОДА ПРОГРАММЫ

Для удобства хранения, работы с текстом и предложениями были разработаны структуры *Word*, *Sentence*, *SplitSentence*, *Text*. Каждая структура была вынесена в отдельный файл для последующего выборочного подключения (*WordStructure.h*, *SentenceStructure.h*, *SplitSentenceStructure.h*, *TextStructure.h*). Помимо этого, для каждой структуры определяется синоним для последующего удобного использования при помощи оператора *typedef*. Для хранения символов кириллицы и латиницы используется тип *wchar_t*, для хранения размеров – беззнаковый целочисленный тип *size_t*. Данные типы объявлены в заголовочном файле *stddef.h* стандартной библиотеки. В заголовочных файлах структур также создано несколько макроопределений, которые хранят начальный размер выделяемой памяти для разных структур:

- INITIAL_SENTENCES_NUMBER (5)
- INITIAL_WORDS_NUMBER (4)
- INITIAL_CHARACTERS_NUMBER (10)

## 2.1. Структура Word

Используется для хранения одного слова. Слово хранится в виде динамического массива широких символов (C-style string). Содержит следующие поля:

| Название и тип поля | Предназначение |
|---|---|
| *wchar_t* characters* | Хранит адрес первого элемента массива широких символов. |
| *size_t charactersNumber* | Хранит количество символов в слове. |

## 2.2. Структура Sentence

Используется для хранения одного предложения. Предложение хранится в виде динамического массива широких символов (C-style string). Содержит следующие поля:

| Название и тип поля | Предназначение |
|---|---|
| *wchar_t* characters* | Хранит адрес первого элемента массива широких символов. |
| *size_t charactersNumber* | Хранит количество символов в предложении. |
| *wchar_t endCharacter* | Хранит символ, находящийся сразу после предложения. |

### 2.3. Структура SplitSentence

Используется для хранения одного предложения. Предложение хранится в виде динамического массива структур *Word* и динамического массива широких символов, в котором записаны знаки препинания. Содержит следующие поля:

| Название и тип поля | Предназначение |
|---|---|
| *Word* words* | Хранит адрес первого элемента массива слов. |
| *size_t charactersNumber* | Хранит количество слов в предложении. |
| *wchar_t* punctuationMarks* | Хранит адрес первого элемента массива знаков препинания. Массив содержит знаки пунктуации после каждого слова в предложении. Если после слова отсутствует знак препинания, то тогда символ пунктуации будет равен пробелу (что означает, что знак препинания отсутствует). |
| *wchar_t endCharacter* | Хранит символ, находящийся сразу после предложения. |

### 2.4. Структура Text

Используется для хранения текста. Текст хранится в виде динамического массива структур *Sentence*. Содержит следующие поля:

| Название и тип поля | Предназначение |
|---|---|
| *Sentence* sentences* | Хранит адрес первого элемента массива. |

| | |
|---|---|
| *size_t sentencesNumber* | Хранит количество предложений в тексте. |

Далее были написаны различные функции для работы с текстом и предложениями. Они были разделены на два файла: *Functions.c* (функции обработки текста и предложений) и *InputOutputFunctions.c* (функции считывания и вывода текста и предложений). Для данных файлов также были написаны заголовочные файлы *Functions.h* и *InputOutputFunctions.h*, в которые подключены заголовочные файлы стандартной библиотеки для использования некоторых функций.

### 2.5. Функция readSentence()

*Sentence readSentence(FILE* stream).*

Позволяет считать одно предложение с потока *stream*, передаваемого в функцию в качестве аргумента. Возвращает структуру *Sentence*.

Для начала создается экземпляр *sentence* структуры Sentence, который потом будет возвращен из функции. Происходит выделение динамической памяти для массива при помощи функции стандартной библиотеки *calloc()*. Указатель на участок выделенной памяти присваивается полю *characters* экземпляра sentence.

**Листинг 1.**

```
Sentence sentence;
sentence.characters = calloc(INITIAL_CHARACTERS_NUMBER,
sizeof(wchar_t));
sentence.charactersNumber = 0;
```

Также объявляется переменная *characterMaxNumber* типа size_t для хранения размера выделенного динамического массива. Далее происходит посимвольное считывание предложения при помощи функции *fgetwc()* до тех пор, пока в потоке не встретится один из символов L‘.’, L‘!’, L‘?’ и L‘\n’. Проверка на равенство одному из этих символов производится при помощи функции *wcschr()*. При этом цикле происходит проверка, хватает ли в

динамическом массиве места для хранения очередного считанного символа. Если нет, то происходит расширение участка выделенной памяти при помощи функции *realloc()*. После того, как было считано предложение, в конец массива добавляется нулевой символ.

**Листинг 2.**

```
    do {
        if (sentence.charactersNumber + 1 >= characterMaxNumber) {
            characterMaxNumber *= 2;
            sentence.characters = realloc(sentence.characters,
characterMaxNumber * sizeof(wchar_t));
        }

        sentence.characters[sentence.charactersNumber++] =
fgetwc(stream);

    } while (wcschr(L".!?\n",
sentence.characters[sentence.charactersNumber - 1]) == NULL);

    sentence.characters[sentence.charactersNumber] = L'\0';
```

В конце производится освобождение лишней памяти и возвращение предложения из функции.

**Листинг 3.**

```
    if (sentence.charactersNumber + 1 < characterMaxNumber) {
        sentence.characters = realloc(sentence.characters,
(sentence.charactersNumber + 1) * sizeof(wchar_t));
    }

    return sentence;
```

### 2.6. Функция readText()

*Text* readText(FILE* stream).*

Позволяет считать текст с потока *stream*, передаваемого в функцию в качестве аргумента. Возвращает указатель на структуру *Text*.

Для начала выделяется память под экземпляр *text* структуры *Text*, указатель на который будет возвращен из функции. Выделение осуществляется при помощи функции *malloc()*. Также происходит выделение участка памяти

под массив структур Sentence, указатель на который присваивается полю sentences экземпляра text.

**Листинг 4.**

```
Text* text = malloc(sizeof(Text));
text->sentences = calloc(INITIAL_SENTENCES_NUMBER,
sizeof(Sentence));
text->sentencesNumber = 0;
```

Далее объявляются переменные следующие переменные:

- *wchar_t endCharacter* – хранит символ, находящийся сразу после предложения. Позволяет запоминать, после каких предложений в тексте идет перевод на новую строку (абзацы текста).
- *size_t sentencesMaxNumber* – хранит размер динамического массива предложений.
- *char loopEnabled* – хранит значение 1 или 0, управляет работой цикла.

После идет цикл, в котором происходит считывание предложений при помощи функции *readSentence()* до тех пор, пока переменная *loopEnabled* не станет равным 0. При этом цикле происходит проверка, хватает ли в динамическом массиве места для хранения очередного считанного предложения. Если нет, то происходит расширение участка выделенной памяти. После того, как было считано предложение, идет проверка на конце текста. Концом текста считается два подряд идущих символа перевода на новую строку сразу после предложения. Тогда если текущее предложение равно строке L“\n”, а *endCharacter* предыдущего предложения равен L’\n’, то достигнут конец текста. Проверка, что текущее предложение будет равно строке L“\n”, происходит при помощи функции *isEndSentence()*, описание которой будет приведено далее. В этом случае переменной loopEnabled присваивается значение 0, а последнее предложение L”\n” удаляется из текста, после происходит выход из цикла. Иначе мы считываем символ, расположенный после считанного предложения, и записываем его в

переменную *endCharacter*. Также если у только что считанного предложения символ *endCharacter* равен *WEOF* (символ конца файла), то в этом случае *loopEnabled* так же устанавливается в 0 и происходит выход из цикла.

**Листинг 5.**

```
do {
        if (text->sentencesNumber >= sentencesMaxNumber) {
            sentencesMaxNumber *= 2;
            text->sentences = realloc(text->sentences, sentencesMaxNumber
* sizeof(Sentence));
        }

        text->sentences[text->sentencesNumber++] = readSentence(stream);

        if (endCharacter == L'\n' && isEndSentence(text->sentences[text-
>sentencesNumber - 1])) {
            freeSentenceMemory(text->sentences + (text->sentencesNumber -
1));
            text->sentencesNumber--;
            loopEnabled = 0;
        } else {
            endCharacter = fgetwc(stream);
            text->sentences[text->sentencesNumber - 1].endCharacter =
endCharacter;
        }

        if (endCharacter == WEOF) {
            loopEnabled = 0;
        }

    } while (loopEnabled);
```

В конце производится освобождение лишней памяти и возвращение предложения из функции.

**Листинг 6.**

```
     if (text->sentencesNumber < sentencesMaxNumber) {
          text->sentences = realloc(text->sentences, text-
>sentencesNumber * sizeof(Sentence));
     }

     return text;
```

### 2.7. Функция isEndSentence()

*char isEndSentence(Sentence sentence).*

Проверяет, является ли предложение sentence предложением вида L”\n”. Если да – возвращает 1, иначе возвращает 0. Сравнивание строк происходит при помощи функции *wcscmp()*.

### 2.8. Функция printText()

*void printText(Text* text).*

Выводит текст **text* на экран. В цикле проходит по массиву предложений *sentences* и выводит каждое предложение с его конечным символом *endCharacter* при помощи функции *wprintf()*.

**Листинг 7.**

```
      for (int i = 0; i < text->sentencesNumber; i++) {
          wprintf(L"%ls%lc", text->sentences[i].characters, text-
>sentences[i].endCharacter);
      }
```

### 2.9. Функция freeWordMemory()

*void freeWordMemory(Word* word).*

Освобождает динамическую память, занимаемую полями экземпляра **word* структуры *Word*. Для освобождения используется функция *free()*. Полям *characters* и *charactersNumber* присваиваются значения *NULL* и 0 соответственно.

**Листинг 8.**

```
      free(word->characters);
      word->characters = NULL;
      word->charactersNumber = 0;
```

### 2.10. Функция freeSplitSentenceMemory()

*void freeSplitSentenceMemory(SplitSentence* splitSentence).*

Освобождает динамическую память, занимаемую полями экземпляра **splitSentence* структуры *SplitSentence*. Для освобождения используется функция *free()*. Для начала функция освобождает память, выделенную для хранения полей каждого элемента массива слов при помощи функции

*freeWordMemory()*. После освобождается память самого массива слов *words*, а также массива знаков пунктуации *punctuationMarks*. Полям *words* и *wordsNumber* присваиваются значения *NULL* и 0 соответственно.

**Листинг 9.**

```
for (size_t i = 0; i < splitSentence->wordsNumber; i++) {
    freeWordMemory(splitSentence->words + i);
}

free(splitSentence->punctuationMarks);
splitSentence->punctuationMarks = NULL;
splitSentence->words = NULL;
splitSentence->wordsNumber = 0;
```

### 2.11. Функция freeSentenceMemory()

*void freeSentenceMemory(Sentence* sentence).*

Освобождает динамическую память, занимаемую полями экземпляра **sentence* структуры *Sentence*. Для освобождения используется функция *free()*. Полям *characters* и *charactersNumber* присваиваются значения *NULL* и 0 соответственно.

**Листинг 10.**

```
free(sentence->characters);
sentence->characters = NULL;
sentence->charactersNumber = 0;
```

### 2.12. Функция freeTextMemory()

*void freeTextMemory(Text* text).*

Освобождает динамическую память, занимаемую полями экземпляра **text* структуры *Text*. Для освобождения используется функция *free()*. Для начала функция освобождает память, выделенную для хранения полей каждого элемента массива предложений при помощи функции *freeSentenceMemory()*. После освобождается память самого массива слов *sentences*. Полям *sentences* и *sentencesNumber* присваиваются значения *NULL* и 0 соответственно.

**Листинг 11.**

```
      free(sentence->characters);
      sentence->characters = NULL;
      sentence->charactersNumber = 0;
```

### 2.13. Функция deleteSentence()

*void deleteSentence(Text* text, size_t index).*

Удаляет предложение под индексом *index* в тексте *text*. Для начала освобождаем память, занимаемую полями удаляемого предложения **(sentence + index)* при помощи функции *freeSentenceMemory()*. Далее сдвигаются элементы массива, стоящие от удаленного предложения справа, влево. В конце декрементируется поле количества предложений и освобождается лишняя память, занимаемая массивом предложений *sentences*.

**Листинг 12.**

```
      freeSentenceMemory(text->sentences + index);

      for (size_t i = index; i < text->sentencesNumber - 1; i++) {
          text->sentences[i] = text->sentences[i + 1];
      }

      text->sentencesNumber--;
      text->sentences = realloc(text->sentences, text->sentencesNumber *
sizeof(Sentence));
```

### 2.14. Функция deleteDuplicateSentences()

*void deleteDuplicateSentences(Text* text).*

Удаляет одинаковые предложения в тексте **text*. Функция пробегается по массиву предложений и для каждого предложения выполняет сравнивание со всеми предложениями, стоящими справа. В случае, если предложения равны, предложение, находящееся справа, удаляется при помощи функции *deleteSentence()*. Сравнение предложений осуществляется без учета регистра символов при помощи функции *wcscasecmp()*. Также надо учитывать то, что при удалении предложения положение предложений, стоящих справа,

сдвигаются влево. Поэтому переменную *j* следует увеличивать лишь в том случае, когда предложение не удаляется.

**Листинг 13.**

```
      for (size_t i = 0; i < text->sentencesNumber - 1; i++) {
          size_t j = i + 1;

          while (j < text->sentencesNumber) {
              if (wcscasecmp(text->sentences[i].characters, text-
>sentences[j].characters) == 0) {
                  deleteSentence(text, j);
              } else {
                  j++;
              }
          }
      }
```

### 2.15. Функция getSplitSentenceFromSentence()

*SplitSentence getSplitSentenceFromSentence(Sentence sentence).*

Преобразует предложение, хранящееся в структуре *Sentence* в эквивалентное предложение, хранящееся в структуре *SplitSentence*. Возвращает преобразованное предложение типа *SplitSentence*. Для начала создается экземпляр *splitSentence* структуры *SplitSentence*, который будет возвращен из функции. Поля инициализируются начальными значениями, для поля *words* выделяется участок динамической памяти, а полю *endCharacter* присваивается поле *endCharacter* структуры *sentence*.

**Листинг 14.**

```
      SplitSentence splitSentence;
      splitSentence.words = calloc(INITIAL_WORDS_NUMBER, sizeof(Word));
      splitSentence.punctuationMarks = NULL;
      splitSentence.wordsNumber = 0;
      splitSentence.endCharacter = sentence.endCharacter;
```

Далее создаются следующие переменные:

- *size_t splitSentenceMaxSize* – хранит размер массива слов *words* и массива символов пунктуации *punctuationMarks*.
- wchar_t* sentenceCopy – хранит копию предложения sentence (требуется, так как далее будет использоваться функция *wcstok()*).

- wchar_t* sentenceCopySave – хранит копию указателя *sentenceCopy* (нужна для отчистки выделенной под *sentenceCopy* динамической памяти).
- wchar_t* p – используется функцией wcstok для хранения внутреннего состояния парсера.

Далее производится копирование строки из *sentence.characters* в *sentenceCopy* при помощи функции *wcscpy()*.

**Листинг 15.**

```
      wchar_t* sentenceCopy = calloc(sentence.charactersNumber + 1,
sizeof(wchar_t));
      wchar_t* sentenceCopySave = sentenceCopy;
      wchar_t* p;

      wcscpy(sentenceCopy, sentence.characters);
```

Далее при помощи функции wcstok() делим предложения на слова. Сначала производится первый вызов wcstok() с указателем на строку в качестве параметра для получения первого слова. После в цикле происходят аналогичные вызовы функции wcstok(), только уже с параметром NULL, для получения остальных слов до тех пор, пока не будут получены все слова в предложении (пока sentenceCopy не станет равным NULL). При этом цикле происходит проверка, хватает ли в динамическом массиве места для хранения очередного считанного слова. Если нет, то происходит расширение участка выделенной памяти при помощи функции *realloc()*. Внутри цикла подсчитывается размер считанного слова при помощи функции *wcslen()*, и если оно не равно 0, то тогда создаем экземпляр *word* структуры *Word*, выделяем для хранения динамическую память, и добавляем его в массив слов *words*.

**Листинг 16.**

```
      sentenceCopy = wcstok(sentenceCopy, L"!?., ", &p);

      while (sentenceCopy != NULL) {
          size_t wordLength = wcslen(sentenceCopy);
```

```
          if (splitSentence.wordsNumber >= splitSentenceMaxSize) {
              splitSentenceMaxSize *= 2;
              splitSentence.words = realloc(splitSentence.words,
splitSentenceMaxSize * sizeof(Word));
          }

          if (wordLength != 0) {
              Word *word =
&(splitSentence.words[splitSentence.wordsNumber++]);
              word->characters = calloc(wordLength + 1, sizeof(wchar_t));
              word->charactersNumber = wordLength;

              wcscpy(word->characters, sentenceCopy);
              word->characters[wordLength] = L'\0';
          }

          sentenceCopy = wcstok(NULL, L"!?., ", &p);
      }
```

В конце происходит освобождение лишней памяти, получение массива знаков пунктуации при помощи функции *getPunctuationMarks()*(будет описана далее) и возвращение *splitSentence* из функции.

**Листинг 17.**

```
      sentenceCopy = wcstok(sentenceCopy, L"!?., ", &p);

      while (sentenceCopy != NULL) {
          size_t wordLength = wcslen(sentenceCopy);

          if (splitSentence.wordsNumber >= splitSentenceMaxSize) {
              splitSentenceMaxSize *= 2;
              splitSentence.words = realloc(splitSentence.words,
splitSentenceMaxSize * sizeof(Word));
          }

          if (wordLength != 0) {
              Word *word =
&(splitSentence.words[splitSentence.wordsNumber++]);
              word->characters = calloc(wordLength + 1, sizeof(wchar_t));
              word->charactersNumber = wordLength;

              wcscpy(word->characters, sentenceCopy);
              word->characters[wordLength] = L'\0';
          }

          sentenceCopy = wcstok(NULL, L"!?., ", &p);
      }
```

### 2.16. Функция getPunctuationMarks()

*wchar_t* getPunctuationMarks(wchar_t* wcs1, const wchar_t* wcs2, size_t marksNumber).*

Получает из строки символы пунктуации после каждого слова. На вход принимает *wcs1* – строка, в которой будет происходить поиск символов пунктуации, *wcs2* – строка, в которой содержатся символы пунктуации, *marksNumber* – число, равное количеству слов в предложении, а значит и количеству знаков пунктуации после каждого слова. Считается, что если после слова отсутствует знак пунктуации, то тогда знак пунктуации равен пробелу. Поиск знаков пунктуации в предложении происходит при помощи функции *wcspbrk(),* которая возвращает указатель на первый встретившийся знак пунктуации. Для массива символов пунктуации заранее выделяется память. Сначала происходит первый вызов функции *wcspbrk(),* после чего в цикле записывается полученный символ пунктуации в массив, далее пропускаются все последующие символы пунктуации до начала следующего слова, и происходит повторный вызов функции *wcspbrk()*. Цикл выполняется до тех пор, пока не встретится конец строки или количество полученных символов стало равно количеству слов *marksNumber*. В конце возвращаем полученный массив из функции.

**Листинг 18.**

```
    wchar_t* punctuationMarks = calloc(marksNumber, sizeof(wchar_t));

    wcs1 = wcspbrk(wcs1, wcs2);

    for (size_t i = 0; i < marksNumber && wcs1 != NULL; i++) {
        punctuationMarks[i] = *wcs1;

        while (wcschr(wcs2, *wcs1) != NULL) {
            wcs1 += 1;
        }

        wcs1 = wcspbrk(wcs1, wcs2);
    }

    return punctuationMarks;
```

### 2.17. Функция getSentenceFromSplitSentence()

*Sentence getSentenceFromSplitSentence(SplitSentence splitSentence).*

Преобразует предложение, хранящееся в структуре *SplitSentence* в эквивалентное предложение, хранящееся в структуре *Sentence*. Возвращает преобразованное предложение типа *Sentence*. Для начала создается экземпляр *sentence* структуры *Sentence*, который будет возвращен из функции. Поля инициализируются начальными значениями, полю *endCharacter* присваивается поле *endCharacter* структуры *splitSentence*. После чего подсчитывается количество символов, которые будут содержаться в строке для того, чтобы знать требуемый размер памяти для выделения. Количество символов в предложении складывается из размера слов предложения и количества символов между словами (зависят от знаков пунктуации). Далее происходит выделение памяти для предложения. Далее ставится нулевой символ в начало строки, чтобы потом при помощи функции *wcscat()* добавлять слова в строку.

**Листинг 19.**

```
      Sentence sentence;
      sentence.characters = NULL;
      sentence.charactersNumber = 0;
      sentence.endCharacter = splitSentence.endCharacter;

      for (size_t i = 0; i < splitSentence.wordsNumber; i++) {
          sentence.charactersNumber +=
(splitSentence.words[i].charactersNumber + 1);

          if (splitSentence.punctuationMarks[i] != L' ') {
              sentence.charactersNumber++;
          }
      }

      sentence.characters = calloc(sentence.charactersNumber + 1,
sizeof(wchar_t));
      sentence.characters[0] = L'\0';
```

После этого в цикле происходит последовательное добавление слов при помощи функции *wcscat()* и знаков пунктуации до тех пор, пока не будут добавлены все слова. Объявляется переменная *index*, равная позиции в строке, на которой остановилась запись. Нужна для добавления знаков препинания

вручную. После того, как произойдет выход из цикла, в конец строки добавится нулевой символ, и возвращается переменная *sentence* из функции.

**Листинг 20.**

```
      size_t index = 0;

      for (size_t i = 0; i < splitSentence.wordsNumber; i++) {
          wcscat(sentence.characters, splitSentence.words[i].characters);
          index += splitSentence.words[i].charactersNumber;

          sentence.characters[index++] =
splitSentence.punctuationMarks[i];

          if (splitSentence.punctuationMarks[i] == L',') {
              sentence.characters[index++] = L' ';
          }
      }
      sentence.characters[index] = L'\0';

      return sentence;
```

### 2.18. Функция deleteSentencesWithoutCapitalLetters()

*void deleteSentencesWithoutCapitalLetters(Text* text).*

Удаляет в тексте **text* предложения, в которых отсутствуют слова, начинающиеся с заглавной буквы. Функция для каждого предложения осуществляет поиск слов, начинающихся с заглавной буквы, и если их нет, то выполняет удаление данного предложения. Объявляется переменная I, которая содержит индекс текущего предложения. Также надо учитывать то, что при удалении предложения положение предложений, стоящих справа, сдвигаются влево. Поэтому переменную *i* следует увеличивать лишь в том случае, когда предложение не удаляется. Для получения списка слов предложение преобразуется к структуре *SplitSentence* при помощи функции *getSplitSentenceFromSentence(),* после чего записывается в переменную *splitSentence*. Помимо этого, объявляется переменная *delete*, которая хранит значение 1 или 0 в зависимости от того, нужно ли удалить предложение. После чего в цикле функция пробегается по словам и проверяет, является ли первый символ в слове заглавной буквой. Происходит это при помощи функции *iswupper().* После того, как слова были проверены, проверяется переменная

*delete*: если она равна 1, то удаляем предложение при помощи функции deleteSentence, иначе инкрементируем *i*. В конце итерации производится освобождение выделенной под *splitSentence* динамической памяти при помощи функции *freeSplitSentenceMemory().*

**Листинг 21.**

```
      size_t i = 0;

      while (i < text->sentencesNumber) {
          SplitSentence splitSentence =
getSplitSentenceFromSentence(text->sentences[i]);

          char delete = 1;

          for (size_t j = 0; j < splitSentence.wordsNumber && delete;
j++) {
              if (iswupper(splitSentence.words[j].characters[0])) {
                  delete = 0;
              }
          }

          if (delete) {
              deleteSentence(text, i);
          } else {
              i++;
          }

          freeSplitSentenceMemory(&splitSentence);
      }
```

### 2.19. Функция getSentenceDuplicateWordsNumber()

*size_t getSentenceDuplicateWordsNumber(Sentence sentence).*

Позволяет найти максимальное количество одинаковых слов в предложении *sentence*. Возвращает это количество. Для получения списка слов предложение преобразуется к структуре *SplitSentence* при помощи функции *getSplitSentenceFromSentence(),* после чего записывается в переменную *splitSentence*. Объявляется переменная *maxNumber*, которая будет хранить максимальное количество одинаковых слов в предложении. В цикле для каждого слова в предложении находится количество его повторений в предложении, и записывается максимальное из них в переменную *maxNumber*. В конце производится освобождение выделенной под *splitSentence*

динамической памяти при помощи функции *freeSplitSentenceMemory()* и возвращение *maxNumber* из функции.

**Листинг 22.**

```
      SplitSentence splitSentence =
getSplitSentenceFromSentence(sentence);

      size_t maxNumber = 1;

      for (size_t i = 0; i < splitSentence.wordsNumber; i++) {
          size_t number = 1;

          for (size_t j = i + 1; j < splitSentence.wordsNumber; j++) {
              if (wcscmp(splitSentence.words[i].characters,
splitSentence.words[j].characters) == 0) {
                  number++;
              }
          }

          if (number > maxNumber) {
              maxNumber = number;
          }
      }

      freeSplitSentenceMemory(&splitSentence);

      return maxNumber;
```

### 2.20. Функция printSentencesDuplicateWordsNumber()

*void printSentencesDuplicateWordsNumber(Text* text).*

Выводит количество одинаковых слов в каждом предложении текста **text*. Для каждого предложения находится количество одинаковых слов при помощи функции *getSentenceDuplicateWordsNumber().* Если количество одинаковых слов равно 1 (в предложении нет одинаковых слов), то выводится *“Одинаковых слов не найдено в предложении №%d: %ls”*, иначе выводится *“Найдено %d одинаковых слов(а) в предложении №%d: %ls”,* где на места подставляются полученные значения.

**Листинг 23.**

```
      for (size_t i = 0; i < text->sentencesNumber; i++) {
          size_t number = getSentenceDuplicateWordsNumber(text-
>sentences[i]);

          if (number == 1) {
              wprintf(L"Одинаковых слов не найдено в предложении №%d:
```

```
%ls\n", i + 1, text->sentences[i].characters);
          } else {
              wprintf(L"Найдено %d одинаковых слов(а) в предложении №%d:
%ls\n", number, i + 1, text->sentences[i].characters);
          }
      }
```

### 2.21. Функция getNumberOfVowels()

*size_t getNumberOfVowels(Word word).*

Подсчитывает и возвращает количество гласных букв в слове *word*. Объявляется переменная-счетчик *number*, которая будет содержать количество гласных букв в слове. Далее в цикле функция пробегается по буквам в слове и при помощи функции wcschr() проверяет, является ли буква гласной. Если буква является гласной, то счетчик увеличивается на 1. После выхода из цикла переменная *number* возвращается из функции.

**Листинг 24.**

```
      size_t number = 0;

      for (size_t i = 0; i < word.charactersNumber; i++) {
          if (wcschr(L"aeiouyaeёиоуыэюя", towlower(word.characters[i]))
!= NULL) {
              number++;
          }
      }

      return number;
```

### 2.22. Функция compareByVowelsNumber()

*int compareByVowelsNumber(const void* word1, const void* word2).*

Функция-компаратор для сортировки массива структур *Word* по количеству гласных букв в них. Сравнивает два слова **word1* и **word2* по количеству гласных букв. Количество гласных букв в слове определяется при помощи функции *getNumberOfVowels()*. Если количество гласных в первом слове больше, чем во втором, то функция возвращает 1, если количество гласных во втором слове больше, чем в первом, то функция возвращает -1, в другом случае функция возвращает 0 (когда количества гласных равны).

**Листинг 25.**

```
    size_t number1 = getNumberOfVowels(*((const Word*)word1));
    size_t number2 = getNumberOfVowels(*((const Word*)word2));

    if (number1 > number2) return 1;
    if (number1 < number2) return -1;

    return 0;
```

### 2.23. Функция sortTextByVowels()

*void sortTextByVowels(Text* text).*

Сортирует слова в предложениях текста **text* по возрастанию количества гласных букв. Для каждого предложения в тексте выполняются следующие действия. Для получения списка слов предложение преобразуется к структуре *SplitSentence* при помощи функции *getSplitSentenceFromSentence()*, после чего записывается в переменную *splitSentence*. Далее при помощи функции qsort() массив слов сортируется, используя функцию-компаратор *compareByVowelsNumber()*. Далее полученное предложение типа *SplitSentence* преобразуется в предложение типа *Sentence*, и заменяет старое предложение на новое, при этом осуществляя освобождение неиспользуемой памяти при помощи функций *freeSentenceMemory()* и *freeSplitSentenceMemory()*.

**Листинг 26.**

```
    for (size_t i = 0; i < text->sentencesNumber; i++) {
        SplitSentence splitSentence =
getSplitSentenceFromSentence(text->sentences[i]);

        qsort(splitSentence.words, splitSentence.wordsNumber,
sizeof(Word), compareByVowelsNumber);

        freeSentenceMemory(text->sentences + i);
        text->sentences[i] =
getSentenceFromSplitSentence(splitSentence);

        freeSplitSentenceMemory(&splitSentence);
    }
```

### 2.24. Функция printSampleStrings()

*void printSampleStrings(Text* text).*

Для каждого предложения выводит строку-образец, удовлетворяющий каждому слову в предложении. Для каждого предложения в тексте **text* выполняет следующие действия. Для начала текущее предложение типа *Sentence* преобразуется к структуре *SplitSentence* при помощи функции *getSplitSentenceFromSentence(),* после чего записывается в переменную *splitSentence*. Если в предложении всего одно слово, то выводим его, так как для самого себя оно и является строкой-образцом. Иначе происходит поиск самого длинного и самого короткого слова в предложении (longWord и shortWord соответственно).

**Листинг 27.**

```
     SplitSentence splitSentence = getSplitSentenceFromSentence(text-
>sentences[i]);

          if (splitSentence.wordsNumber == 1) {
              wprintf(L"Строка-образец для предложения №%d: %ls\n", i +
1, splitSentence.words[0].characters);
          } else {
              Word longWord = splitSentence.words[0];
              Word shortWord = splitSentence.words[0];

              for (int j = 0; j < splitSentence.wordsNumber; j++) {
                  if (splitSentence.words[j].charactersNumber <
shortWord.charactersNumber) {
                      shortWord = splitSentence.words[j];
                  }

                  if (splitSentence.words[j].charactersNumber >
longWord.charactersNumber) {
                      longWord = splitSentence.words[j];
                  }
              }
```

Далее выделяется память под строку-образец *string* длиной, равной длине *shortWord* + 1, так как строка-образец не может по длине быть больше чем длина самого короткого слова + 1 (последний символ в строке-образце равен L’*’). В строку-образец копируется слово shortWord при помощи функции *wcscpy()*. В конец строки записывается символ L’*’. Далее происходит

посимвольное сравнивание текущей строки-образца с остальными словами в предложении. Если k-ый символ какого-либо слова в предложении не совпадает с k-ым символов строки-образца, значит на это место в строке-образце ставится символ L’?’, так как если этого не сделать, то строка-образец не будет удовлетворять этому слову. После того, как строка-образец была сравнена с остальными словами, происходит замена нескольких символов L’?’ (если они есть) в начале на один L’*’.

**Листинг 28.**

```
     wchar_t *string = calloc(shortWord.charactersNumber + 2,
sizeof(wchar_t));
             wcscpy(string, shortWord.characters);

             string[shortWord.charactersNumber] = L'*';
             string[shortWord.charactersNumber + 1] = L'\0';

             for (size_t j = 0; j < splitSentence.wordsNumber; j++) {
                 if (splitSentence.words[j].characters !=
shortWord.characters) {
                     for (size_t k = 0; k < shortWord.charactersNumber;
k++) {
                         if (splitSentence.words[j].characters[k] !=
string[k]) {
                             string[k] = L'?';
                         }
                     }
                 }
             }

             size_t endIndex = shortWord.charactersNumber;

             if (string[0] == L'?') {
                 string[0] = L'*';

                 while (wcschr(L”?*”, string[1]) != NULL && string[1] !=
L’\0’) {
                     for (size_t j = 1; j <= endIndex; j++) {
                         string[j] = string[j + 1];
                     }
                     endIndex--;
                 }
             }
```

В конце строка-образец выводится на экран, и происходит освобождение памяти.

### 2.25. Функция main()

Для начала при помощи функции setlocale() с параметрами *LC_ALL* и *“ru_RU.utf8”* устанавливаем возможность работы с кириллицей. Далее объявляются следующие переменные:

- *int userAction* – хранит выбор действия пользователя.
- *char loopEnabled* – хранит значение 1 или 0, от которого зависит, когда пользователь выйдет из цикла выбора действий.
- *Text* text* – хранит указатель на считанный текст.

Далее происходит вывод сообщения *“\n[ Выберите метод считывания текста ]\n 1: Считать текст с консоли.\n 2: Считать текст с файла (text.txt).\n 0: Выйти из программы.\n\n>>> ”* на экран. При помощи функции *wscanf()* считываем выбор пользователя. В случае, если пользователь введен некорректное значение, то программа сообщит ему об этом и даст возможность сделать выбор заново. Если пользователь введет 0, то произойдет выход из программы. Если пользователь введет 1, то текст будет считан с консоли, а если 2 – то с файла *text.txt*. Считывание текста происходит с помощью функции *readText(),* в которую передается поток – стандартный с консоли или с файла. После происходит удаление повторяющихся предложений при помощи функции *deleteDuplicateSentences().* Далее пользователь заходит в цикл выбора действий, в котором он может выбирать, какие действия нужно произвести с текстом. Доступны следующие действия:

- 1 - Для каждого предложения вывести строку образец, удовлетворяющую каждому слову в предложении.
- 2 - Удалить все предложения, в которых нет заглавных букв в начале слова.
- 3 - Отсортировать слова в предложении по количеству гласных букв в слове.

- 4 - Для каждого предложения вывести количество одинаковых слов в строке.
- 5 – Вывести текст.
- 0 – Выход из программы.

Действия выполняются при помощи функций *printSampleStrings(), deleteSentencesWithoutCapitalLetters(), sortTextByVowels(), printSentencesDuplicateWordsNumber()* и *printText(),* которые были описаны выше. В случае, если пользователь введен некорректное значение, то программа сообщит ему об этом и даст возможность сделать выбор заново. В конце программы, когда пользователь завершает работу с текстом, происходит освобождение занятой памяти при помощи функции *freeTextMemory()*.

**Листинг 29.**

```
     setlocale(LC_ALL, "ru_RU.utf8");

          int userAction;
          char loopEnabled = 1;

          Text* text = NULL;

          do {
              wprintf(L"\n[ Выберите метод считывания текста ]\n  1: 
Считать текст с консоли.\n  2: Считать текст с "
                      "файла (text.txt).\n  0: Выйти из прораммы.\n\n>>> 
");

              wscanf(L"%d", &userAction);
              fgetwc(stdin);

              FILE* stream = NULL;

              switch (userAction) {
                  case 0:
                      return 0;

                  case 1:
                      wprintf(L"\nВведите текст:\n");

                      text = readText(stdin);
                      deleteDuplicateSentences(text);

                      break;

                  case 2:
                      stream = fopen("text.txt", "r");

                      if (stream == NULL) {
```

```
                        wprintf(L"[ Ошибка ] Файл text.txt не
существует или не доступен для программы.");
                        return 0;
                    }

                    text = readText(stream);
                    deleteDuplicateSentences(text);

                    wprintf(L"\n[ Считанный текст ]\n\n");
                    printText(text);
                    wprintf(L"\n");

                    break;

                default:
                    wprintf(L"[ Вы ввели некорректное значение ]\n");
            }

        } while (userAction < 0 || userAction > 2);

        while (loopEnabled) {
            wprintf(L"\n[ Выберите действие ]\n  1: Для каждого
предложения вывести строку образец, удовлетво"
                    "ряющую каждому слову в предложении.\n  2: Удалить
все предложения, в которых нет заглавных букв в "
                    "начале слова.\n  3: Отсортировать слова в
предложении по количеству гласных букв в слове.\n  4: "
                    "Для каждого предложения вывести количество
одинаковых слов в строке.\n  5: Вывести текст на экран."
                    "\n  0: Выйти из программы.\n\n>>> ");
            wscanf(L"%d", &userAction);
            wprintf(L"\n");

            switch (userAction) {
                case 0:
                    loopEnabled = 0;
                    break;

                case 1:
                    printSampleStrings(text);
                    break;

                case 2:
                    deleteSentencesWithoutCapitalLetters(text);
                    break;

                case 3:
                    sortTextByVowels(text);
                    break;

                case 4:
                    printSentencesDuplicateWordsNumber(text);
                    break;

                case 5:
                    printText(text);
                    wprintf(L"\n");
```

```
                    break;

                default:
                    wprintf(L"\n[ Вы ввели некорректное значение
]\n\n");
            }
        }

        freeTextMemory(text);

        return 0;
```

## 2.26. Makefile

Для компиляции программы был написан Makefile:

**Листинг 30.**

```
all: build clean

build: main.o Functions.o InputOutputFunctions.o
	gcc main.o Functions.o InputOutputFunctions.o -o cw

main.o: main.c
	gcc -c main.c

Functions.o: functions/Functions.c
	gcc -c functions/Functions.c

InputOutputFunctions.o: functions/InputOutputFunctions.c
	gcc -c functions/InputOutputFunctions.c

clean:
	rm -rf *.o
```

# 3. ТЕСТИРОВАНИЕ РАБОТЫ ПРОГРАММЫ

## Тест №1: Проверка чтения текста с файла:

```
[ Выберите метод считывания текста ]
  1: Считать текст с консоли.
  2: Считать текст с файла (text.txt).
  0: Выйти из прораммы.

>>> 2

[ Считанный текст ]

Opens the file whose name is specified in the parameter filename and
associates it with a stream that can be identified in future operations
by the FILE pointer returned.
The operations that are allowed on the stream and how these are performed
are defined by the mode parameter.
The returned stream is fully buffered by default if it is known to not
refer to an interactive device (see setbuf).
The returned pointer can be disassociated from the file by calling fclose
or freopen. All opened files are automatically closed on normal program
termination. The running environment supports at least FOPEN_MAX files
open simultaneously.
Text files are files containing sequences of lines of text. Depending on
the environment where the application runs, some special character
conversion may occur in input/output operations in text mode to adapt
them to a system-specific text file format. Although on some environments
no conversions occur and both text files and binary files are treated the
same way, using the appropriate mode improves portability.
```

## Тест №2: Проверка чтения текста с консоли, проверка 1-ого действия, проверка выхода из программы:

```
[ Выберите метод считывания текста ]
  1: Считать текст с консоли.
  2: Считать текст с файла (text.txt).
  0: Выйти из прораммы.

>>> 1

Введите текст:
Hello, yelli, feold.


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
```

```
  0: Выйти из программы.

>>> 1

Строка-образец для предложения №1: *e?l?*

[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 0


Process finished with exit code 0
```

## Тест №3: Проверка 2-ого действия, проверка вывода текста в консоль:

```
[ Выберите метод считывания текста ]
  1: Считать текст с консоли.
  2: Считать текст с файла (text.txt).
  0: Выйти из прораммы.

>>> 1

Введите текст:
Hi. how are you. gooD morNINg. are You okay?


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 2


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
```

```
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 5

Hi. are You okay?


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 0


Process finished with exit code 0
```

## Тест №4: Проверка 3-ого действия:

```
[ Выберите метод считывания текста ]
  1: Считать текст с консоли.
  2: Считать текст с файла (text.txt).
  0: Выйти из прораммы.

>>> 1

Введите текст:
baaaaaa aaabbbbb abbbabbbb bababababa ba ab abab abababab.


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 3


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
```

```
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 5

ba ab abbbabbbb abab aaabbbbb abababab bababababa baaaaaa.


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 0


Process finished with exit code 0
```

## Тест №5: Проверка 4-ого действия:

```
[ Выберите метод считывания текста ]
  1: Считать текст с консоли.
  2: Считать текст с файла (text.txt).
  0: Выйти из прораммы.

>>> 1

Введите текст:
hi hi hi how type int.
type type int int int unicode.
allocator smart pointer.
std std std std std namespace.


[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.
```

```
>>> 4

Найдено 3 одинаковых слов(а) в предложении №1: hi hi hi how type int.
Найдено 3 одинаковых слов(а) в предложении №2: type type int int int
unicode.
Одинаковых слов не найдено в предложении №3: allocator smart pointer.
Найдено 5 одинаковых слов(а) в предложении №4: std std std std std
namespace.

[ Выберите действие ]
  1: Для каждого предложения вывести строку образец, удовлетворяющую
каждому слову в предложении.
  2: Удалить все предложения, в которых нет заглавных букв в начале
слова.
  3: Отсортировать слова в предложении по количеству гласных букв в
слове.
  4: Для каждого предложения вывести количество одинаковых слов в строке.
  5: Вывести текст на экран.
  0: Выйти из программы.

>>> 0


Process finished with exit code 0
```

# ЗАКЛЮЧЕНИЕ

В ходе работы над поставленным заданием удалось разработать программу, способную считывать и обрабатывать текст в зависимости от действий пользователя. Программа может считывать текст как с консоли, так и с файла. Пользователь может выбирать способ ввода текста, а также следующие действия обработки текста:

- Для каждого предложения вывести строку образец удовлетворяющую каждому слову в предложении.
- Удалить все предложения, в которых нет заглавных букв в начале слове.
- Отсортировать слова в предложении по количеству гласных букв в слове.
- Для каждого предложения вывести количество одинаковых слов в строке.

Программа соответствует требованиям, поставленным в задании. Программа была успешно протестирована на работоспособность.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

2. Кенриган Б. И Ритчи Д. Язык программирования Си М.: Вильямс, 1978 288 с.

2. Standart C++ Library reference // cplusplus.com. URL: http://www.cplusplus.com/reference/ (дата обращения: 8.12.2019).

3. CS106B Style Guide // Stanford University. URL: http://stanford.edu/class/archive/cs/cs106b/cs106b.1158/styleguide.shtml (дата обращения: 8.12.2019).

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл *WordStructure.h*:

```
#ifndef WORD_STRUCTURE_H
#define WORD_STRUCTURE_H

#include <stddef.h>

struct Word {
    wchar_t* characters;
    size_t charactersNumber;
};

typedef struct Word Word;

#endif
```

Файл *SentenceStructure.h*:

```
#ifndef SENTENCE_STRUCTURE_H
#define SENTENCE_STRUCTURE_H

#include <stddef.h>

#define INITIAL_CHARACTERS_NUMBER 10

struct Sentence {
    wchar_t* characters;
    size_t charactersNumber;
    wchar_t endCharacter;
};

typedef struct Sentence Sentence;

#endif
```

Файл *SplitSentenceStructure.h*:

```
#ifndef WORD_LIST_STRUCTURE_H
#define WORD_LIST_STRUCTURE_H

#include <stddef.h>
#include "WordStructure.h"

#define INITIAL_WORDS_NUMBER 4

struct SplitSentence {
    Word* words;
    size_t wordsNumber;
    wchar_t* punctuationMarks;
    wchar_t endCharacter;
};
```

```
typedef struct SplitSentence SplitSentence;

#endif
```

Файл *TextStructure.h*:

```
#ifndef TEXT_STRUCTURE_H
#define TEXT_STRUCTURE_H

#include "SentenceStructure.h"

#define INITIAL_SENTENCES_NUMBER 5

struct Text {
    Sentence* sentences;
    size_t sentencesNumber;
};

typedef struct Text Text;

#endif
```

Файл *InputOutputFunctions.h*:

```
#ifndef INPUT_OUTPUT_FUNCTIONS_H
#define INPUT_OUTPUT_FUNCTIONS_H

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include "../structures/SentenceStructure.h"
#include "../structures/TextStructure.h"
#include "Functions.h"

Sentence readSentence(FILE*);
Text* readText(FILE*);

void printText(Text*);

#endif
```

Файл *InputOutputFunctions.c*:

```
#include "InputOutputFunctions.h"

Sentence readSentence(FILE* stream) {
    Sentence sentence;
    sentence.characters = calloc(INITIAL_CHARACTERS_NUMBER,
sizeof(wchar_t));
    sentence.charactersNumber = 0;

    size_t characterMaxNumber = INITIAL_CHARACTERS_NUMBER;
```

```
    do {
        if (sentence.charactersNumber + 1 >= characterMaxNumber) {
            characterMaxNumber *= 2;
            sentence.characters = realloc(sentence.characters,
characterMaxNumber * sizeof(wchar_t));
        }

        sentence.characters[sentence.charactersNumber++] =
fgetwc(stream);

    } while (wcschr(L".!?\n",
sentence.characters[sentence.charactersNumber - 1]) == NULL);

    sentence.characters[sentence.charactersNumber] = L'\0';

    if (sentence.charactersNumber + 1 < characterMaxNumber) {
        sentence.characters = realloc(sentence.characters,
(sentence.charactersNumber + 1) * sizeof(wchar_t));
    }

    return sentence;
}

Text* readText(FILE* stream) {
    Text* text = malloc(sizeof(Text));
    text->sentences = calloc(INITIAL_SENTENCES_NUMBER, sizeof(Sentence));
    text->sentencesNumber = 0;

    wchar_t endCharacter = L'\n';
    size_t sentencesMaxNumber = INITIAL_SENTENCES_NUMBER;
    char loopEnabled = 1;

    do {
        if (text->sentencesNumber >= sentencesMaxNumber) {
            sentencesMaxNumber *= 2;
            text->sentences = realloc(text->sentences, sentencesMaxNumber
* sizeof(Sentence));
        }

        text->sentences[text->sentencesNumber++] = readSentence(stream);

        if (endCharacter == L'\n' && isEndSentence(text->sentences[text-
>sentencesNumber - 1])) {
            freeSentenceMemory(text->sentences + (text->sentencesNumber -
1));
            text->sentencesNumber--;
            loopEnabled = 0;
        } else {
            endCharacter = fgetwc(stream);
            text->sentences[text->sentencesNumber - 1].endCharacter =
endCharacter;
        }

        if (endCharacter == WEOF) {
            loopEnabled = 0;
        }
```

```
    } while (loopEnabled);

    if (text->sentencesNumber < sentencesMaxNumber) {
        text->sentences = realloc(text->sentences, text->sentencesNumber
* sizeof(Sentence));
    }

    return text;
}

void printText(Text* text) {
    for (int i = 0; i < text->sentencesNumber; i++) {
        wprintf(L"%ls%lc", text->sentences[i].characters, text-
>sentences[i].endCharacter);
    }
}
```

Файл *Functions.h*:

```
#ifndef TEXT_FUNCTIONS_H
#define TEXT_FUNCTIONS_H

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <wctype.h>
#include "../structures/SentenceStructure.h"
#include "../structures/TextStructure.h"
#include "../structures/WordStructure.h"
#include "../structures/SplitSentenceStructure.h"

wchar_t* getPunctuationMarks(wchar_t*, const wchar_t*, size_t);
SplitSentence getSplitSentenceFromSentence(Sentence);
Sentence getSentenceFromSplitSentence(SplitSentence);
size_t getSentenceDuplicateWordsNumber(Sentence);
size_t getNumberOfVowels(Word);
int compareByVowelsNumber(const void*, const void*);
char isEndSentence(Sentence);

void sortTextByVowels(Text*);
void printSampleStrings(Text*);
void printSentencesDuplicateWordsNumber(Text*);
void deleteSentencesWithoutCapitalLetters(Text*);
void deleteDuplicateSentences(Text*);
void deleteSentence(Text*, size_t);

void freeWordMemory(Word*);
void freeSplitSentenceMemory(SplitSentence*);
void freeSentenceMemory(Sentence*);
void freeTextMemory(Text*);

#endif
```

Файл *Functions.c*:

```
#include "Functions.h"
```

```
wchar_t* getPunctuationMarks(wchar_t* wcs1, const wchar_t* wcs2, size_t
marksNumber) {
    wchar_t* punctuationMarks = calloc(marksNumber, sizeof(wchar_t));

    wcs1 = wcspbrk(wcs1, wcs2);

    for (size_t i = 0; i < marksNumber && wcs1 != NULL; i++) {
        punctuationMarks[i] = *wcs1;

        while (wcschr(wcs2, *wcs1) != NULL) {
            wcs1 += 1;
        }

        wcs1 = wcspbrk(wcs1, wcs2);
    }

    return punctuationMarks;
}

SplitSentence getSplitSentenceFromSentence(Sentence sentence) {
    SplitSentence splitSentence;
    splitSentence.words = calloc(INITIAL_WORDS_NUMBER, sizeof(Word));
    splitSentence.punctuationMarks = NULL;
    splitSentence.wordsNumber = 0;
    splitSentence.endCharacter = sentence.endCharacter;

    size_t splitSentenceMaxSize = INITIAL_WORDS_NUMBER;

    wchar_t* sentenceCopy = calloc(sentence.charactersNumber + 1,
sizeof(wchar_t));
    wchar_t* sentenceCopySave = sentenceCopy;
    wchar_t* p;

    wcscpy(sentenceCopy, sentence.characters);

    sentenceCopy = wcstok(sentenceCopy, L"!?., ", &p);

    while (sentenceCopy != NULL) {
        size_t wordLength = wcslen(sentenceCopy);

        if (splitSentence.wordsNumber >= splitSentenceMaxSize) {
            splitSentenceMaxSize *= 2;
            splitSentence.words = realloc(splitSentence.words,
splitSentenceMaxSize * sizeof(Word));
        }

        if (wordLength != 0) {
            Word *word =
&(splitSentence.words[splitSentence.wordsNumber++]);
            word->characters = calloc(wordLength + 1, sizeof(wchar_t));
            word->charactersNumber = wordLength;

            wcscpy(word->characters, sentenceCopy);
            word->characters[wordLength] = L'\0';
        }

        sentenceCopy = wcstok(NULL, L"!?., ", &p);
```

```
    }

    if (splitSentence.wordsNumber < splitSentenceMaxSize) {
        splitSentence.words = realloc(splitSentence.words,
splitSentence.wordsNumber * sizeof(Word));
    }

    splitSentence.punctuationMarks =
getPunctuationMarks(sentence.characters, L"!?., ",
splitSentence.wordsNumber);

    free(sentenceCopySave);

    return splitSentence;
}

Sentence getSentenceFromSplitSentence(SplitSentence splitSentence) {
    Sentence sentence;
    sentence.characters = NULL;
    sentence.charactersNumber = 0;
    sentence.endCharacter = splitSentence.endCharacter;

    for (size_t i = 0; i < splitSentence.wordsNumber; i++) {
        sentence.charactersNumber +=
(splitSentence.words[i].charactersNumber + 1);

        if (splitSentence.punctuationMarks[i] != L' ') {
            sentence.charactersNumber++;
        }
    }

    sentence.characters = calloc(sentence.charactersNumber + 1,
sizeof(wchar_t));
    sentence.characters[0] = L'\0';

    size_t index = 0;

    for (size_t i = 0; i < splitSentence.wordsNumber; i++) {
        wcscat(sentence.characters, splitSentence.words[i].characters);
        index += splitSentence.words[i].charactersNumber;

        sentence.characters[index++] = splitSentence.punctuationMarks[i];

        if (splitSentence.punctuationMarks[i] == L',') {
            sentence.characters[index++] = L' ';
        }
    }

    sentence.characters[index] = L'\0';

    return sentence;
}

void printSampleStrings(Text* text) {
    for (size_t i = 0; i < text->sentencesNumber; i++) {
        SplitSentence splitSentence = getSplitSentenceFromSentence(text-
>sentences[i]);
```

```
        if (splitSentence.wordsNumber == 1) {
            wprintf(L"Строка-образец для предложения №1: %ls\n",
splitSentence.words[0].characters);

        } else {
            Word longWord = splitSentence.words[0];
            Word shortWord = splitSentence.words[0];

            for (int j = 0; j < splitSentence.wordsNumber; j++) {
                if (splitSentence.words[j].charactersNumber <
shortWord.charactersNumber) {
                    shortWord = splitSentence.words[j];
                }

                if (splitSentence.words[j].charactersNumber >
longWord.charactersNumber) {
                    longWord = splitSentence.words[j];
                }
            }

            wchar_t *string = calloc(shortWord.charactersNumber + 2,
sizeof(wchar_t));
            wcscpy(string, shortWord.characters);

            string[shortWord.charactersNumber] = L'*';
            string[shortWord.charactersNumber + 1] = L'\0';

            for (size_t j = 0; j < splitSentence.wordsNumber; j++) {
                if (splitSentence.words[j].characters !=
shortWord.characters) {
                    for (size_t k = 0; k < shortWord.charactersNumber;
k++) {
                        if (splitSentence.words[j].characters[k] !=
string[k]) {
                            string[k] = L'?';
                        }
                    }
                }
            }

            size_t endIndex = shortWord.charactersNumber;

            if (string[0] == L'?') {
                string[0] = L'*';

         while (wcschr(L"?*", string[1]) != NULL && string[1] != L'\0') {
                    for (size_t j = 1; j <= endIndex; j++) {
                        string[j] = string[j + 1];
                    }
                    endIndex--;
                }
            }

            wprintf(L"Строка-образец для предложения №%d: %ls\n", i + 1,
string);
            free(string);
```

```
        }

        freeSplitSentenceMemory(&splitSentence);
    }
}

char isEndSentence(Sentence sentence) {
    if (wcscmp(sentence.characters, L"\n") == 0) {
        return 1;
    } else {
        return 0;
    }
}

void deleteSentencesWithoutCapitalLetters(Text* text) {
    size_t i = 0;

    while (i < text->sentencesNumber) {
        SplitSentence splitSentence = getSplitSentenceFromSentence(text-
>sentences[i]);

        char delete = 1;

        for (size_t j = 0; j < splitSentence.wordsNumber && delete; j++)
{
            if (iswupper(splitSentence.words[j].characters[0])) {
                delete = 0;
            }
        }

        if (delete) {
            deleteSentence(text, i);
        } else {
            i++;
        }

        freeSplitSentenceMemory(&splitSentence);
    }

}

size_t getSentenceDuplicateWordsNumber(Sentence sentence) {
    SplitSentence splitSentence = getSplitSentenceFromSentence(sentence);

    size_t maxNumber = 1;

    for (size_t i = 0; i < splitSentence.wordsNumber; i++) {
        size_t number = 1;

        for (size_t j = i + 1; j < splitSentence.wordsNumber; j++) {
            if (wcscmp(splitSentence.words[i].characters,
splitSentence.words[j].characters) == 0) {
                number++;
            }
        }

        if (number > maxNumber) {
```

```
            maxNumber = number;
        }
    }

    freeSplitSentenceMemory(&splitSentence);

    return maxNumber;
}

size_t getNumberOfVowels(Word word) {
    size_t number = 0;

    for (size_t i = 0; i < word.charactersNumber; i++) {
        if (wcschr(L"aeiouyaeёиоуыэюя", towlower(word.characters[i])) !=
NULL) {
            number++;
        }
    }

    return number;
}

int compareByVowelsNumber(const void* word1, const void* word2) {
    size_t number1 = getNumberOfVowels(*((const Word*)word1));
    size_t number2 = getNumberOfVowels(*((const Word*)word2));

    if (number1 > number2) return 1;
    if (number1 < number2) return -1;

    return 0;
}

void sortTextByVowels(Text* text) {
    for (size_t i = 0; i < text->sentencesNumber; i++) {
        SplitSentence splitSentence = getSplitSentenceFromSentence(text-
>sentences[i]);

        qsort(splitSentence.words, splitSentence.wordsNumber,
sizeof(Word), compareByVowelsNumber);

        freeSentenceMemory(text->sentences + i);
        text->sentences[i] = getSentenceFromSplitSentence(splitSentence);

        freeSplitSentenceMemory(&splitSentence);
    }
}

void printSentencesDuplicateWordsNumber(Text* text) {
    for (size_t i = 0; i < text->sentencesNumber; i++) {
        size_t number = getSentenceDuplicateWordsNumber(text-
>sentences[i]);

        if (number == 1) {
            wprintf(L"Одинаковых слов не найдено в предложении №%d:
%ls\n", i + 1, text->sentences[i].characters);
        } else {
```

```
            wprintf(L"Найдено %d одинаковых слов(а) в предложении №%d:
%ls\n", number, i + 1, text->sentences[i].characters);
        }
    }
}

void deleteSentence(Text* text, size_t index) {
    freeSentenceMemory(text->sentences + index);

    for (size_t i = index; i < text->sentencesNumber - 1; i++) {
        text->sentences[i] = text->sentences[i + 1];
    }

    text->sentencesNumber--;
    text->sentences = realloc(text->sentences, text->sentencesNumber *
sizeof(Sentence));
}

void deleteDuplicateSentences(Text* text) {
    for (size_t i = 0; i < text->sentencesNumber - 1; i++) {
        size_t j = i + 1;

        while (j < text->sentencesNumber) {
            if (wcscasecmp(text->sentences[i].characters, text-
>sentences[j].characters) == 0) {
                deleteSentence(text, j);
            } else {
                j++;
            }
        }
    }
}

void freeWordMemory(Word* word) {
    free(word->characters);
    word->characters = NULL;
    word->charactersNumber = 0;
}

void freeSplitSentenceMemory(SplitSentence* splitSentence) {
    for (size_t i = 0; i < splitSentence->wordsNumber; i++) {
        freeWordMemory(splitSentence->words + i);
    }

    free(splitSentence->punctuationMarks);
    splitSentence->words = NULL;
    splitSentence->punctuationMarks = NULL;
    splitSentence->wordsNumber = 0;
}

void freeSentenceMemory(Sentence* sentence) {
    free(sentence->characters);
    sentence->characters = NULL;
    sentence->charactersNumber = 0;
}

void freeTextMemory(Text* text) {
```

```
    for (size_t i = 0; i < text->sentencesNumber; i++) {
        freeSentenceMemory(text->sentences + i);
    }

    free(text->sentences);
    text->sentences = NULL;
    text->sentencesNumber = 0;
}
```

Файл *main.c*:

```
#include <locale.h>
#include "functions/Functions.h"
#include "functions/InputOutputFunctions.h"

int main() {
    setlocale(LC_ALL, "ru_RU.utf8");

    int userAction;
    char loopEnabled = 1;

    Text* text = NULL;

    do {
        wprintf(L"\n[ Выберите метод считывания текста ]\n  1: Считать
текст с консоли.\n  2: Считать текст с "
                "файла (text.txt).\n  0: Выйти из прораммы.\n\n>>> ");

        wscanf(L"%d", &userAction);
        fgetwc(stdin);

        FILE* stream = NULL;

        switch (userAction) {
            case 0:
                return 0;

            case 1:
                wprintf(L"\nВведите текст:\n");

                text = readText(stdin);
                deleteDuplicateSentences(text);

                break;

            case 2:
                stream = fopen("text.txt", "r");

                if (stream == NULL) {
                    wprintf(L"[ Ошибка ] Файл text.txt не существует или
не доступен для программы.");
                    return 0;
                }

                text = readText(stream);
                deleteDuplicateSentences(text);
```

```
                wprintf(L"\n[ Считанный текст ]\n\n");
                printText(text);
                wprintf(L"\n");

                break;

            default:
                wprintf(L"[ Вы ввели некорректное значение ]\n");
        }

    } while (userAction < 0 || userAction > 2);

    while (loopEnabled) {
        wprintf(L"\n[ Выберите действие ]\n  1: Для каждого предложения
вывести строку образец, удовлетво"
                "ряющую каждому слову в предложении.\n  2: Удалить все
предложения, в которых нет заглавных букв в "
                "начале слова.\n  3: Отсортировать слова в предложении по
количеству гласных букв в слове.\n  4: "
                "Для каждого предложения вывести количество одинаковых
слов в строке.\n  5: Вывести текст на экран."
                "\n  0: Выйти из программы.\n\n>>> ");
        wscanf(L"%d", &userAction);
        wprintf(L"\n");

        switch (userAction) {
            case 0:
                loopEnabled = 0;
                break;

            case 1:
                printSampleStrings(text);
                break;

            case 2:
                deleteSentencesWithoutCapitalLetters(text);
                break;

            case 3:
                sortTextByVowels(text);
                break;

            case 4:
                printSentencesDuplicateWordsNumber(text);
                break;

            case 5:
                printText(text);
                wprintf(L"\n");
                break;

            default:
                wprintf(L"\n[ Вы ввели некорректное значение ]\n\n");
        }
    }

    freeTextMemory(text);
```

```
    return 0;
}
```

Файл *Makefile*:

```
all: build clean

build: main.o Functions.o InputOutputFunctions.o
	gcc main.o Functions.o InputOutputFunctions.o -o cw

main.o: main.c
	gcc -c main.c

Functions.o: functions/Functions.c
	gcc -c functions/Functions.c

InputOutputFunctions.o: functions/InputOutputFunctions.c
	gcc -c functions/InputOutputFunctions.c

clean:
	rm -rf *.o
```