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. 9303 _____ Куршев Е.О.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Куршев Е. О.

Группа 9303

Тема работы : Обработка текста

Исходные данные:
Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой) Длина текста и каждого предложения заранее не известна.
Дополнительные условия:
1. Между словами может быть только пробел или запятая.
2. Между предложениями может быть только пробел или символ переноса строки.
3. Все предложения оканчиваются только на точку, восклицательный знак или вопросительный знак.
4. Ограничение по длине вводимого числа для выбора функции прописано в самой программе. Строка считается некорректной, если содержится хоть одна буква.
5. Ограничение по длине вводимого вводимого имени файла прописано в самой программе.
Содержание пояснительной записки:
«Аннотация», «Содержание», «Введение», «Описание кода программы», «Примеры работы программы», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 15.10.2019

Дата сдачи реферата: 23.12.2019

Дата защиты реферата: 23.12.2019

Студент _____ Куршев Е.О.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Программа написана на языке C. 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):

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?*”.

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.

Все сортировки и операции со строками должны осуществляться с использованием функций стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Создание программы для обработки текста.

Для достижения поставленной цели требуется решить следующие задачи:

1) Изучение функции для работы с широкими строками, а также с динамической памятью.

2) Поэтапное выполнение мини-подзадач, которые будут решать ту или определенную мини-задачу. Например, считывание текста, разбиение его на слова и тд.

3) Группировка мини-подзадач в одну для решение определенной задачи.

4) Написание Make-файла.

5) Тестирование программы.

Основные теоретические положения

Структура — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры. Язык С позволяет определять имена новых типов данных с помощью ключевого слова typedef. На самом деле здесь не создается новый тип данных, а определяется новое имя существующему типу. Он позволяет облегчить создание машинно-независимых программ.

Указатель (англ. pointer) — переменная, диапазон значений которой состоит из адресов ячеек памяти или специального значения — нулевого адреса. Последнее используется для указания того, что в данный момент указатель не ссылается ни на одну из допустимых ячеек.

Функции для динамического выделения памяти:

void* calloc (size_t num, size_t size) — функция для чистого выделения памяти.

void* malloc (size_t size) — функция для выделения памяти.

void* realloc (void* ptr, size_t size) — функция для перераспределения выделенной памяти.

void free (void* ptr) — функция для освобождения выделенной памяти.

Управляющие конструкции:

Оператор IF — ELSE используется при необходимости сделать выбор. Формально синтаксис имеет вид

IF (выражение)

оператор-1

ELSE

оператор-2,

Где часть ELSE является необязательной. Сначала вычисляется выражение; если оно "истинно", то выполняется оператор1. Если оно ложно, и если есть часть с ELSE, то вместо оператора-1 выполняется оператор-2.

Циклы:

В конструкции

WHILE (выражение)

оператор

вычисляется выражение. Если его значение отлично от нуля, то выполняется оператор и выражение вычисляется снова. Этот цикл продолжается до тех пор, пока значение выражения не станет нулем, после чего выполнение программы продолжается с места после оператора.

Оператор

FOR (выражение 1; выражение 2; выражение 3) оператор эквивалентен последовательности

выражение 1;

WHILE (выражение 2) оператор выражение 3;

Оператор выбора switch.

```
switch (выражение) {
 case константа_1 : операторы ; break;
 ...
 case константа_n : операторы ; break;
 default : операторы ; break; }
```

Оператор switch вычисляет выражение и переходит к первому совпадающему

значению после case. Далее выполняются операторы этого блока и по команде break происходит выход из структуры. Если ни одно из значений не совпадает с константами из case, то выполняются операторы блока default.

Оператор break - это выход из цикла или конструкции switch.

Оператор continue - переход на конец цикла (т.е. пропуск всех операторов от continue до конца структуры цикла).

1. Описание функций

1.1 main.

Точка входа в программу. С помощью функции setlocale задается текущая локаль, создается переменная text типа данных Text. После выполняются функции, которые нужны для выполнения поставленной задачи.

1.2 Пользовательский интерфейс

- void choice1() и void choice2()

Данные функции выводят подсказки для пользователя, а также меню, чтобы пользователь мог ориентироваться в действиях в программе.

- int userChoice()

Функция, запоминающая выбор пользователя.

1.3 Выбор пользователя

- int readAll(Text* text)

Функция, в которой выполняется считывание текста с клавиатуры или из файла (на усмотрение пользователя). Возвращает число, по значению которого определяется, выделилась ли запрашиваемая память и оказался ли текст пустым.

- int functions(Text text)

Функция, в которой реализовано выполнение подзадач, в зависимости от выбора пользователя. Возвращает число, в зависимости от которого происходит печать текста, так как в некоторых функциях подразумевается печать своего текста.

1.4 Ввод и считывание текста

- Sentence ReadSentence(FILE* source)

Функция для посимвольного считывания предложения. В зависимости от параметра source происходит считывание либо с клавиатуры, либо из файла. Возвращает предложение (структуру Sentence).

- Text ReadText(FILE* source)

Функция для считывание текста. Она используется функцию ReadSentence. Учитывается случай, когда не удается выделить запрашиваемую память. Возвращается текст (структура Text).

1.5 Обработка текста до функций

- void delete(Text* text)

Функция для удаления одинаковых (без учета регистра) предложений.

- int SplitSentence(Sentence* ptr_sentence)

Функция, которая разбивает предложение на слова, создает массив структур Word. Для разбиения всего текста используется в функции ReadText.

- int SplitText(Text* text_ptr)

Функция, которая разбивает текст на слова с помощью функции SplitSentence.

1.6 Реализованные функции

1.6.1 Маска для слов

- void pattern(Text text)

Функция для нахождения маски слов в предложении. Сохраняет предложение во временный массив и производит его обработку.

1.6.2 Удаление предложений, попадающих под условие

- void del(Text text)

Для каждого слова происходит проверка на условие для первой буквы с помощью функции iswupper.

1.6.3 Сортировка предложений

- void sort(Text text)

Функция сохраняет предложение во временный массив. Далее в слове находится количество гласных с помощью функции wcschr, и производится сортировка.

1.6.4 Подсчёт одинаковых слов

- void same(Text text)

Функция сохраняет предложение во временный массив, сортирует предложения с помощью функции wcscmp. Далее прогоном массива сравниваются слова.

1.7 Очистка памяти

- void FreeMemory(Text text)

Функция для очистки динамически выделенной памяти.

2. Описание дополнительных компонентов

2.1 Структуры

Структура Word

```
struct Word{
   wchar_t* ptr; //указатель на слово
   wchar_t punctuation; //символ пунктуации
}
```

Структура Sentence

```
struct Sentence{
   Word* words; //указатель на массив структур Word
   wchar_t* ptr; //указатель на предложение
   int size; //количество символов предложения
   int words_counter; //количество слов
   int newline; //наличие символа перевода строки
   int empty; //пустое ли предложение
   int only_newline; //состоит ли предложение только из символов перевода строки
   int EOR; //закончилось ли считывание текста
}
```

Структура Text

```
struct Text{
   Sentence* ptr; //указатель на предложение
   int size; //количество предложений
   int read_sent; //количество считанных предложений
   int split_sent; //количество предложений разбитых на слова
}
```

2.2 Библиотеки

- libraries.h

Файл, со всеми используемыми библиотеками языка С.

2.3 Макроопределения

#define SENT_LEN 100 - начальная длина предложения

#define TEXT_LEN 100 - начальная длина массива структур Sentence

#define WORDS_NUM 50 - начальная длина массива структур Word

#define PATH_LEN 52 - максимальная длина пути файла

#define USER 17 - максимальная длина строки выбора пользователя

3. Тестирование

№	Входные данные	Выходные данные	Примечание
1	Аристотель Артишок Арсенал	Предложению 1 соответсвует данная маска: Ар?*	Выполнение подзадачи 1
2	Привет,я новая программа. Всем Пока.	Всем Пока.	Выполнение подзадачи 2
3	ааа а аааaа аа.	а аа ааа аааaа.	Выполнение подзадачи 3
4	Привет привет ПРИвет. Пока Googbay.	В предложении 1: привет:3 В предложении 2: все слова в предложении уникальны!	Выполнение подзадачи 4

4. Заключение

В результате выполнения курсовой работы получена программа, которая считывает введенный пользователем текст, динамически выделяет память под его хранение и обрабатывает его по выбору пользователя.

5. Использованные источники

http://www.cplusplus.com/

http://www.c-cpp.ru/

Б.В. Керниган, Д.М. Ричи. - ЯЗЫК С

https://ru.wikipedia.org/wiki/

Приложение А. Код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <wchar.h>
#include <wctype.h>
#include <math.h>
#define SENT_LEN 100
#define TEXT_LEN 100
#define WORDS_NUM 50
#define PATH_LEN 52
#define USER 17

void del(Text text);
void FreeMemory(Text* text);
int function(Text text);
void choice1();
void choice2();
void pattern(Text text);
int read(Text* text);
Sentence ReadSentence(FILE* source);
Text ReadText(FILE* source);
void delete(Text* text_ptr);
void same(Text text);
void sort(Text text);
int SplitSentence(Sentence* ptr_sentence);
int SplitText(Text* text_ptr);
```

```
int userChoice();
void WriteText(Text text);

int main() {
    setlocale(LC_ALL, "");
    Text text;
    int a = read(&text);
    if (a == 0){
        return 0;
    }
    delete(&text);
    a = SplitText(&text);
    if (a == 0){
        wprintf(L"Невозможно выделить память. Программа завершена.\n");
        return 0;
    }
    if(function(text) == 1){
        FreeMemory(&text);
        return 0;
    }
    WriteText(text);
    FreeMemory(&text);
    return 0;
}

typedef struct Word{
    wchar_t* ptr;
    wchar_t punctuation;
}Word;
```

```
typedef struct Sentence{
    Word* words;
    wchar_t* ptr;
    int size;
    int words_counter;
    int newline;
    int empty;
    int only_newline;
    int EOR;
}Sentence;

typedef struct Text{
    Sentence* ptr;
    int size;
    int read_sent;
    int split_sent;
}Text;

void del(Text text){
    int n;
    for(int i = 0; i < text.size; i++){
        n = 0;
        for (int j = 0; j < text.ptr[i].words_counter; j++) {
            if (iswupper(*(text.ptr[i].words[j].ptr)) != 0) {
                n += 1;
            }
        }
        if (n != text.ptr[i].words_counter){
            Word *tmp1 = text.ptr[i].words;
            text.ptr[i].words = NULL;
```

```
            free(tmp1);

        }
    }
}

void FreeMemory(Text* text) {
    for (int i=0; i < (text -> size); i++) {
        if (text->ptr[i].words != NULL) {
            free((text->ptr)[i].words);
        }
        free((text->ptr)[i].ptr);
    }
    free(text -> ptr);
}

int function(Text text){
    int k;
    choice2();
    k = userChoice();
    while((k != 1) && (k != 2) && (k != 3) && (k != 4) && (k != 0)){
        wprintf(L"Некоректные данные. Повторите ввод: ");
        k = userChoice();
    }
    switch (k){
        case 1:
            pattern(text);
            return 1;
        case 2:
            del(text);
```

```
            break;
        case 3:
            sort(text);
            break;
        case 4:
            same(text);
            return 1;
        case 0:
            wprintf(L"Программа завершена.");
            break;
    }
    return 0;
}

void choice1(){
    wprintf(L"Выберите вариант ввода текста:\n1. Из стандартного потока ввода.\
n2. Из файла.\n0. Выход из программы.\n");
    wprintf(L"ОБРАТИТЕ ВНИМАНИЕ! Введенный текст не должен превышать
%d символов.\n", USER - 2);
    wprintf(L"Вы выбрали: ");
}

void choice2(){
    wprintf(L"Выберите одну из функций:\n1. Вывести маску для каждого слова
предложения.\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.\n0. Выйти из программы.\n");
    wprintf(L"ОБРАТИТЕ ВНИМАНИЕ! Введенный текст не должен превышать %d символов.\n", USER - 2);
    wprintf(L"Ваш выбор: ");
}

void pattern(Text text) {
    wprintf(L"----------------------------\n");
    wchar_t** patterns = malloc(text.size * sizeof(wchar_t*));
    for (int i = 0; i < text.size; i++) {
        wchar_t **tmp = malloc(text.ptr[i].words_counter * sizeof(wchar_t *));
        for (int j = 0; j < text.ptr[i].words_counter; j++) {
            tmp[j] = malloc(wcslen(text.ptr[i].words[j].ptr) * sizeof(wchar_t));
            tmp[j] = text.ptr[i].words[j].ptr;
        }
        for (int q = 0; q < text.ptr[i].words_counter; q++) {
            for (int w = text.ptr[i].words_counter - 1; w > q; w--) {
                if (wcslen(tmp[w - 1]) > wcslen(tmp[w])) {
                    wchar_t *s = tmp[q];
                    tmp[q] = tmp[w];
                    tmp[w] = s;
                }
            }
        }
        if (text.ptr[i].words_counter > 1){
            if (wcslen(tmp[0]) > wcslen(tmp[1])) {
                wchar_t *s = tmp[0];
                tmp[0] = tmp[1];
                tmp[1] = s;
```

```
        }
    }

    int count = 0;
        patterns[i] = malloc((wcslen(tmp[text.ptr[i].words_counter - 1]) + 1) *
sizeof(wchar_t));
        wchar_t* s = malloc((wcslen(tmp[text.ptr[i].words_counter - 1]) + 1) *
sizeof(wchar_t));
    if(text.ptr[i].words_counter == 1){
        patterns[i] = tmp[0];
    }
    else{
      for(int t = 0; t <wcslen(tmp[0]); t++){
        wchar_t temp;
        for(int n = 1; n < text.ptr[i].words_counter; n++){
          if(tmp[n][t] == tmp[n - 1][t]){
            temp = tmp[n][t];
          }
          else{
            temp = '?';
            break;
          }
        }
        patterns[i][count] = temp;
        count += 1;
      }
      int len = count;
      for(int k = count - 1; k > 0; k--){
        if(patterns[i][k] == patterns[i][k-1] && patterns[i][k] == '?'){
          patterns[i][k] = '\0';
```

```
                len -= 1;
            }
            else{
                break;
            }
        }
        if(len != wcslen(tmp[text.ptr[i].words_counter - 1])){
            patterns[i][len] = L'*';
        }
    }
  }
  for (int z = 0; z < text.size; z++){
        wprintf(L"Предложению %d соответсвует данная маска: %ls\n", z + 1,
patterns[z]);
  }
  wprintf(L"-----------------------------\n");
}

int read(Text* text){
  int k;
  choice1();
  k = userChoice();
  while((k != 1) && (k != 2) && (k != 0)){
    wprintf(L"Некоректные данные. Повторите ввод: ");
    k = userChoice();
  }
  switch (k){
    case 1:
      wprintf(L"Введите текст, оканчивающийся ТРЕМЯ символами переноса
строки:\n");
```

```
        *text = ReadText(stdin);
        if (text->ptr == NULL){
            wprintf(L"Невозможно выделить память. Программа завершена.\n");
            return 0;
        }
        if (text->read_sent == 0){
            wprintf(L"Вы ввели пустой текст. Программа завершена.\n");
            return 0;
        }
        break;
    case 2:
        wprintf(L"ПРИМЕЧАНИЕ: Длина имени файла должна быть не более %d\n", PATH_LEN-2);
        wchar_t *file_name = (wchar_t*) calloc(PATH_LEN, sizeof(wchar_t));
        wprintf(L"Введите название файла, из которого нужно считать текст: ");
        fgetws(file_name, PATH_LEN, stdin);
        char *buffer = calloc(PATH_LEN * sizeof(wchar_t), sizeof(char));
        *wcschr(file_name, L'\n') = '\0';
        wcstombs(buffer, file_name, sizeof(wchar_t) * PATH_LEN);
        FILE *file_ptr;
        while ((file_ptr = fopen(buffer, "r")) == NULL){
            wprintf(L"Некорректное название файла! Повторите попытку!\n");
            wprintf(L"ОБРАТИТЕ ВНИМАНИЕ! Длина имени файла должна быть не более %d\n", PATH_LEN-2);
            wprintf(L"Введите название файла, из которого будет произведено считывание текста: ");
            fgetws(file_name, PATH_LEN, stdin);
            *wcschr(file_name, L'\n') = '\0';
            wcstombs(buffer, file_name, sizeof(wchar_t) * PATH_LEN);
        }
```

```
            *text = ReadText(file_ptr);
            fclose(file_ptr);
            free(file_name);
            free(buffer);
            if (text->ptr == NULL){
               wprintf(L"Невозможно выделить память!.\n");
               return 0;
            }
            if (text->read_sent == 0){
               wprintf(L"Введён пустой текст. Программа завершена!\n");
               return 0;
            }
            break;
         case 0:
            wprintf(L"Программа завершена.\n");
            return 0;
      }
   }

Sentence ReadSentence(FILE* source) {
   Sentence sentence = {NULL,
                        calloc(SENT_LEN, sizeof(wchar_t)),
                        0,
                        0,
                        0,
                        0,
                        0,
                        0};
   if (sentence.ptr == NULL) {
      return sentence;
```

```
}
int i = 0;
wchar_t *tmp_ptr;
wchar_t wc;
int size = SENT_LEN;
do {
    wc = fgetwc(source);
    if (feof(source)) {
        free(sentence.ptr);
        sentence.EOR = 1;
        return sentence;
    }
    sentence.ptr[i++] = wc;
    if (i == size - 1) {
        size += SENT_LEN;
        tmp_ptr = realloc(sentence.ptr, sizeof(wchar_t) * size);
        if (tmp_ptr == NULL) {
            free(sentence.ptr);
            sentence.ptr = NULL;
            return sentence;
        }
        sentence.ptr = tmp_ptr;
    }
} while(!wcschr(L".!?\n", wc));
sentence.ptr[i] = L'\0';
if (wcschr(L".!?", sentence.ptr[0])) {
    free(sentence.ptr);
    fgetwc(source);
    sentence.empty = 1;
}
```

```
    else {
        if (sentence.ptr[0] == L'\n') {
            free(sentence.ptr);
            sentence.only_newline = 1;
        } else {
            sentence.size = i;
            if (fgetwc(source) == L'\n')
                sentence.newline = 1;
        }
    }
    return sentence;
}

Text ReadText(FILE* source);
Sentence ReadSentence(FILE* source);

Text ReadText(FILE* source){
    Text text = {calloc(TEXT_LEN, sizeof(Sentence)),
                 0,
                 0,
                 0};
    if (text.ptr == NULL){
        return text;
    }
    int i = 0;
    int n = 0;
    Sentence* tmp_ptr;
    Sentence ws;
    int size = TEXT_LEN;
    do {
```

```
ws = ReadSentence(source);
if (ws.ptr == NULL) {
    for (int j = 0; j < text.read_sent; j++)
        free(text.ptr[j].ptr);
    free(text.ptr);
    text.ptr = NULL;
    return text;
}
if (ws.only_newline) {
    n += 1;
    if (n == 2)
        break;
    continue;
}
else {
    n = 0;
}
if (ws.EOR){
    break;
}
if (ws.empty){
    continue;
}
text.ptr[i++] = ws;
if (i == size - 1){
    size += TEXT_LEN;
    tmp_ptr = realloc(text.ptr, sizeof(Sentence) * size);
    if (tmp_ptr == NULL){
        for (int j = 0; j < text.read_sent; j++)
            free(text.ptr[j].ptr);
```

```
            free(text.ptr);
            text.ptr = NULL;
            return text;
        }
        text.ptr = tmp_ptr;
    }
    text.read_sent++;
  }while (1);
  if (text.read_sent == 0) {
    free(text.ptr);
    return text;
  }
  text.size = i;
  return text;
}

void delete(Text* text) {
  int i = 0, j = 1;
  while (i < text->size) {
    while (j < text->size) {
      if (!wcscasecmp((text->ptr)[i].ptr, (text->ptr)[j].ptr))  {
        free((text->ptr)[j].ptr);
        for (int k=j; k < text->size - 1; k++)
          (text->ptr)[k] = (text->ptr)[k+1];
        (text->size) -= 1;
        continue;
      }
      j++;
    }
    i++;
```

```
        j = i + 1;
    }
}

void same(Text text){
    wprintf(L"----------------------------\n");
    for (int i = 0; i < text.size; i++){
        int k = 1;
        int num = 0;
        wchar_t** tmp = malloc(text.ptr[i].words_counter * sizeof(wchar_t*));
        for (int j = 0; j < text.ptr[i].words_counter; j++){
            tmp[j] = malloc(wcslen(text.ptr[i].words[j].ptr) * sizeof(wchar_t));
            for (int v = 0; v < wcslen(text.ptr[i].words[j].ptr); v++){
                text.ptr[i].words[j].ptr[v] = towlower(text.ptr[i].words[j].ptr[v]);
            }
            tmp[j] = text.ptr[i].words[j].ptr;
        }
        for (int r = 0; r < text.ptr[i].words_counter; r++){
            for (int r1 = 0; r1 < text.ptr[i].words_counter - 1; r1++){
                if (wcscmp(tmp[r], tmp[r1]) > 0){
                    wchar_t *temp = tmp[r];
                    tmp[r] = tmp[r1];
                    tmp[r1] = temp;
                }
            }
        }
        wprintf(L"В предложении %d:", i + 1);
        for (int z = 0; z < text.ptr[i].words_counter - 1; z++){
            if (wcscmp(tmp[z], tmp[z + 1]) == 0){
                k += 1;
```

```
            }
            else if (k > 1){
                wprintf(L"\n%ls:%d", tmp[z], k);
                num += 1;
                k = 1;
            }
        }
        if (k > 1){
            wprintf(L"\n%ls:%d", tmp[text.ptr[i].words_counter - 1], k);
            num += 1;
        }
        if (num == 0){
            wprintf(L" все слова в предложении уникальны!");
        }
        wprintf(L"\n");
    }
    wprintf(L"-----------------------------\n");
}

void sort(Text text){
    wchar_t s[] = L"уеыаоэяиюёУЕЫАОЭЯИЮЁeyuioaEYUIOA";
    for(int i = 0; i < text.size; i++){
        for (int j = 0; j < text.ptr[i].words_counter; j++){
            for (int m = j + 1; m < text.ptr[i].words_counter; m++){
                int n1 = 0;
                int n2 = 0;
                for(int l = 0; l < wcslen(text.ptr[i].words[j].ptr); l++){
                    if (wcschr(s, *(text.ptr[i].words[j].ptr + l)) != NULL){
                        n1 += 1;
                    }
```

```
            }
            for(int l = 0; l < wcslen(text.ptr[i].words[m].ptr); l++){
                if (wcschr(s, *(text.ptr[i].words[m].ptr + l)) != NULL){
                    n2 += 1;
                }
            }
            if (n1 > n2){
                wchar_t *tmp = text.ptr[i].words[m].ptr;
                text.ptr[i].words[m].ptr = text.ptr[i].words[j].ptr;
                text.ptr[i].words[j].ptr = tmp;
            }
        }
    }
}

int SplitSentence(Sentence* ptr_sentence) {
    int size = WORDS_NUM;
    Word *words = calloc(size, sizeof(Word));
    if (words == NULL) {
        free(words);
        return 0;
    }
    int i = 0;
    wchar_t *state;
    Word *temp_ptr;
    wchar_t *word_ptr = wcstok(ptr_sentence->ptr, L" ", &state);
    do{
        ptr_sentence->words_counter += 1;
        words[i].punctuation = 0;
```

```
        if (wcschr(L".!?,", word_ptr[wcslen(word_ptr) - 1])) {
            words[i].punctuation = word_ptr[wcslen(word_ptr) - 1];
            word_ptr[wcslen(word_ptr) - 1] = L'\0';
        }
        words[i++].ptr = word_ptr;
        if (i == size - 1){
            size += WORDS_NUM;
            temp_ptr = realloc(words, size * sizeof(Word));
            if (temp_ptr == NULL) {
                free(words);
                return 0;
            }
            words = temp_ptr;
        }
        word_ptr = wcstok(NULL, L" ", &state);
        if (word_ptr == NULL)
            break;
    }while(1);
    ptr_sentence->words = words;
    return 1;
}

int SplitText(Text* text) {
    for (int i = 0; i < text->size; i++){
        if (SplitSentence((text->ptr) + i)){
            text->split_sent += 1;
            continue;
        }
        else{
            for (int j = 0; j < text->split_sent; j++) {
```

```
            free((text->ptr)[j].words);
        }
        for (int j = 0; j < text->size; j++) {
            free((text->ptr)[j].ptr);
        }
        free(text->ptr);
        return 0;
    }
  }
  return 1;
}

int userChoice(){
  wchar_t buffer[USER];
  fgetws(buffer, USER, stdin);
  int num;
  for (int i = 0; i < wcslen(buffer); i++){
    if(iswalpha(buffer[i])){
      return -1;
    }
  }
  swscanf(buffer, L"%d", &num);
  return num;
}

void WriteText(Text text) {
  wprintf(L"----------------------------\n");
  for (int i=0; i < text.size; i++){
    if (text.ptr[i].words != NULL){
      for (int j = 0; j < text.ptr[i].words_counter - 1; j++){
```

```
            wprintf(L"%ls", text.ptr[i].words[j].ptr);
            if (text.ptr[i].words[j].punctuation){
               wprintf(L"%lc", text.ptr[i].words[j].punctuation);
            }
            wprintf(L" ");
         }
         wprintf(L"%ls", text.ptr[i].words[text.ptr[i].words_counter - 1].ptr);
         if (text.ptr[i].words[text.ptr[i].words_counter - 1].punctuation){
                        wprintf(L"%lc",  text.ptr[i].words[text.ptr[i].words_counter  -
1].punctuation);
         }
         if (text.ptr[i].newline){
            wprintf(L"\n");
         }
         else{
            wprintf(L" ");
         }
      }
   }
   wprintf(L"\n-----------------------------\n");
}
```