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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Программа для обработки текста на языке Си

Студент гр. 9304 _____ Мохаммед А.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Мохаммед А. А.

Группа 9304

Тема работы: программа для обработки текста на языке Си

Исходные данные:

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

Содержание пояснительной записки:
«Аннотация», «Содержание», «Введение», «Описание кода программы», «Примеры работы программы», «Заключение», «Список использованных источников».
Предполагаемый объем пояснительной записки:
Не менее 25 страниц.

Дата выдачи задания: 15.10.2019

Дата сдачи реферата: 23.12.2019

Дата защиты реферата:23.12.2019

Студент _____ Мохаммед А.А.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

В данной работе была создана программа на языке программирования C, которая производит обработку текста посредством набора функций. Программе на вход подается текст (текст представляет собой предложения, разделенные точкой). Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв и цифр. Длина текста и каждого предложения заранее не известна.

Программа должна сохранить этот текст в динамический массив строк и оперировать далее только с ним.

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 посчитать количество слов “garbage” в нем (без учета регистра). В зависимости от количества вывести следующие строки: 0 - “Clean”, [1 5] - “Must be washed”, >5 - “It is a dump”.
2) Заменить все цифры в предложении на введенную строку.
3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.
4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

Все сортировки должны осуществляться с использованием функции стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

СОДЕРЖАНИЕ

Введение

Целью данной работы является изучение функций стандартной библиотеки языка С и разработка программы, которая редактирует текст по заранее заданным правилам.
Для достижения поставленной цели требуется решить следующие задачи:
1) изучение основ языка программирования С для реализации данной программы;
2) разработка программного кода;
3) сборка программы из полученного сегментированного программного кода;
4) тестирование программного кода.

Summary

The program is fed to the input text. The length of the text and each sentence is not known in advance. You want to execute commands entered by the user. To store the sentence and to store the text, you need to implement the Sentence and Text structures. During the program, the text will be changed based on the user's requirements. The result of the program is the output of the modified text after each user command.

Описание функций

Функция считывания введенного текста

```
char** readText(int *sentence_counter){
    char **text = (char**)calloc(1, sizeof(char*));
    if (text == NULL){return NULL;}
    int i=0;
    int sentence_max=0;
    char c = getchar();
    do{
        i=0;
        if((text=(char**)realloc(text,
((*sentence_counter)+1)*sizeof(char*)))==NULL)
            return NULL;
        do{
            if
((text[*sentence_counter]=(char*)realloc(text[*sentence_counter],
sizeof(char)*(i+3)))==NULL)
                return NULL;
            text[*sentence_counter][i]=c;
            i++;
        }while((c=getchar())!='.');
        text[*sentence_counter][i]='.';
        text[*sentence_counter][i+1]='\0';
        (*sentence_counter)++;
        if (i+2>sentence_max)
            sentence_max=i+2;
    }while((c = getchar())!='\n');


    char checker[*sentence_counter][sentence_max];
    int j, k;

    for(i=0; i<*sentence_counter; i++){
        strcpy(checker[i], text[i]);
        for(j=0; j<strlen(checker[i]); j++){
            checker[i][j]=tolower(checker[i][j]);
        }
    }

    for(i=0; i<*sentence_counter; i++)
        for(j=i+1; j<*sentence_counter; j++){

            int l=0;
            int found = 0;
            if (strlen(checker[i]) == strlen(checker[j]))
              while (l < strlen(checker[i])){
                if (checker[i][l] == checker[j][l]){
```

```
                found = 1;
                l++;
              }else{
                found = 0;
                break;
              }
            }
            if (found == 1){
              (*sentence_counter)--;
              for(k=j; k<*sentence_counter; k++){
                  text[k]=text[k+1];
                  strcpy(checker[k], checker[k+1]);
                  }
                  j--;
              }
          }
  return text;
     }
```

Ничего не принимает на вход. Программа получает данные из стандартного потока ввода. Функция записывает полученные предложения в динамический массив символов Sentence.
Возвращает указатель на массив предложений.

Функция подсчета кол-ва слов «garbage» в предложении.

```
i nt count_garbage(char **text, int sentence_counter){

    char** text2 = convert_to_lowercase(text,sentence_counter);
    int count, letter = 0, total_count = 0;
    int search_word_len = strlen(SEARCH_WORD);
    char *word = (char*)calloc( 1,search_word_len * sizeof(char));
    if (word==NULL) return total_count;
    for(int i=0; i<sentence_counter; i++){
       count = 0;
       for (int j=0; j< strlen(text2[i]); j++){
         int k=0;
         for (k=0; k<search_word_len; k++){
           word[k] = text2[i][k+j];
         }
         word[strlen(word)] = '\0';
         if (strcmp(word, SEARCH_WORD) == 0){
           count++;
         }
       }
       printf("\n=========================\n");
       printf("For sentence: '%s'\n============\n",text2[i]);
       if(count == 0 )
         printf("Clean\n");
       else if(count > 5)
         printf("It is a dump\n");
       else
```

```
    printf("Must be washed\n");
  printf("=============\n");
  total_count+=count;
  }
  return total_count;
}
```

Принимает на вход указатель на массив предложений (Text) и кол-во предложений. Функция считает кол-во слов «garbage» без учета регистра в каждом предложении и выводит:

1) Если слов 0, то «Clear»;
2) Если слов меньше 5, то «Must be washed»;
3) Если слов больше 5, то «It‟s a dump»;

для каждого предложения, выделяя текстом результат, для удобства наблюдения результата

Функция ничегоне возвращает.

Функция проверки наличия трех подряд заглавных букв в предложении.

```
int three_consecutive_letters_in_upper_case(char *sentence){


    for(int i=0; i < strlen(sentence)-2; i++){
        if(isupper(sentence[i]) && isupper(sentence[i+1]) && isupper(sentence[i+2]))
          return 1;
    }
    return 0;
}


int delete_if_three_consecutive_letters_in_upper_case(char **text, int *sentence_counter){
  int num = 0;
    for(int i=0; i<*sentence_counter; i++){
        if(three_consecutive_letters_in_upper_case(text[i])){
            free(text[i]);
            (*sentence_counter)--;
            for (int j = i; j < *sentence_counter; j++)
              text[j] = text[j + 1];
          i--;
          num++;
        }
    }
    return num;
  }
```

Функция, если она находит три последовательные заглавные буквы в предложении, она выведет их из вывода.

Функция main

```
int main(){
    int sentence_counter=0;
    int command;
    int num;
    // char input_string[50];
    char* input_string = (char*) malloc(sizeof(char)*1);
    printf("Здравствуйте! Введите Ваш текст.\nПодсказка: текст состоит из предложений, которые состоят из набора слов, разделённых запятой ИЛИ пробелом.\nСлова состоят из цифр и латинских букв.\n\n");
    char** text = readText(&sentence_counter);
    if(text==NULL){
        printf("Ошибка выделения памяти.\n");
        return 1;}


    printText(text, sentence_counter);


    printf("---------------------------------------\n");
    printf("Введите одно из чисел ниже, чтобы выбрать действие:\n1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра). \n2) Заменить все цифры в предложении на введенную строку.\n3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.\n4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.\n0) Выйти из программы.\n");
    printf("---------------------------------------\n");
    printf("------Type command № >> ");
    scanf("%d", &command);
    printf("\n");
    int sentences_count = 0;
    while(command!=0){
      switch(command){
          case 1:
```

```
            printf("Number of '%s' occuring in text is %d\n", SEARCH_WORD,
count_garbage(text, sentence_counter));
            break;

        case 2:
            printf("Please input the string(50 chars) to replace digits with: ");
            scanf("%s", input_string);
            num = replace_all_digits(text, &sentence_counter,input_string);
            printText(text, sentence_counter);
            printf("-------------------------------------\n");
            printf("Number of Replaces is %d\n", num);
            break;

        case 3:
            sentences_count = sentence_counter;
            num = delete_if_three_consecutive_letters_in_upper_case(text,
&sentence_counter);
            printText(text, sentence_counter);
            printf("-------------------------------------\n");
            printf("Number of Deletions is %d\n", num);
            printf("Remaining sentences is %d\n", sentences_count - num);
            break;
        case 4:
          sort_by_descending_number_of_words_beginning_with_vowels(text,
sentence_counter);
          printText(text, sentence_counter);
          break;

        case 0:
          break;

        default:
```

```
            printf("Неправильный ввод!\n");
            break;
        }
        printf("-------------------------------------\n");
        printf("Введите одно из чисел ниже, чтобы выбрать действие:\n1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра). \n2) Заменить все цифры в предложении на введенную строку.\n3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.\n4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.\n0) Выйти из программы.\n");
        printf("-------------------------------------\n");
        printf("------Type command № >> ");
        scanf("%d", &command);
    }
    printf("Program is Exited Successfully");
    return 0;

}
```

ПРИМЕР РАБОТЫ ПРОГРАММЫ

Пример работы функции подсчета слов «garbage» в каждом предложении garbage_count

Before the early 1960s,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive. Computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for A-Z, a-z, and 0-9 would leave only 2 codes: nowhere near enough. Most computers opted not to support lower-case letters. Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.

==========

0| 'Before the early 1960s,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive.'

==========

1| ' Computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for A-Z, a-z, and 0-9 would leave only 2 codes: nowhere near enough.'

==========

2| ' Most computers opted not to support lower-case letters.'

==========

3| ' Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.'

Введите одно из чисел ниже, чтобы выбрать действие:

1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра).

2) Заменить все цифры в предложении на введенную строку.

3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.

4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

0) Выйти из программы.

------Type command № >> 1

==========

For sentence: 'before the early 1960s,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive.'

==========

Must be washed

==========

==========

For sentence: ' computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for a-z, a-z, and 0-9 would leave only 2 codes: nowhere near enough.'

==========

Clean

==========

==========

For sentence: ' most computers opted not to support lower-case letters.'

==========

Clean

==========

==========

For sentence: ' thus, early text projects such as roberto busa's index thomisticus, the brown corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.'

==========

Must be washed

==========

Number of 'garbage' occuring in text is 7

Пример работы функции замены всех цифр на введенную строку*change_digit_text*

Введите одно из чисел ниже, чтобы выбрать действие:

1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра).

2) Заменить все цифры в предложении на введенную строку.

3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.

4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

0) Выйти из программы.

------Type command № >> 2

Please input the string(50 chars) to replace digits with: QQ

==========

0| 'Before the early QQQQQQQQs,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive'

==========

1| ' Computers often allocated only QQ bits for each character, permitting only QQQQ characters—assigning codes for A-Z, a-z, and QQ-QQ would leave only QQ codes: nowhere near enough.'

==========

2| ' Most computers opted not to support lower-case letters.'

==========

3| ' Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.'

Number of Replaces is 10

Введите одно из чисел ниже, чтобы выбрать действие:

1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра).

2) Заменить все цифры в предложении на введенную строку.

3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.

4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

0) Выйти из программы.

Пример работы функции удаления предложений, содержащих 3 и более подряд заглавные буквы *delete_alpha*

IAM Abdulrahman who is from yemen. I Live in RASia with my frind WHOS nAME is ESKAnder.

==========

0| 'IAM Abdulrahman who is from yemen.'

==========

1| ' I Live in RASia with my frind WHOS nAME is ESKAnder.'

Введите одно из чисел ниже, чтобы выбрать действие:

1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра).

2) Заменить все цифры в предложении на введенную строку.

3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.

4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

0) Выйти из программы.

------Type command № >> 3

Number of Deletions is 2

Remaining sentences is 0

Пример работы функции сортировки предложений по количеству слов, начинающихся с гласной *sort_count*

Before the early 1960s,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive. Computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for A-Z, a-z, and 0-9 would leave only 2 codes: nowhere near enough. Most computers opted not to support lower-case letters. Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.

==========

0| '

Before the early 1960s,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive.'

==========

1| ' Computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for A-Z, a-z, and 0-9 would leave only 2 codes: nowhere near enough.'

==========

2| ' Most computers opted not to support lower-case letters.'

==========

3| ' Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.'

Введите одно из чисел ниже, чтобы выбрать действие:

1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра).

2) Заменить все цифры в предложении на введенную строку.

3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.

4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.

0) Выйти из программы.

------Type command № >> 4

==========

0| ' Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be garbage garbage upper-case.'

==========

1| ' Computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for A-Z, a-z, and 0-9 would leave only 2 codes: nowhere near enough.'

==========

2| '

Before the early 1960s,grabage computers were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive.'

==========

3| ' Most computers opted not to support lower-case letters.'

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены основные библиотеки языка С и были получены навыки работы с функциями. В результате работы была разработана программа, которая редактирует текст по заранее заданным правилам и печатает на экран. Для каждой подзадачи были описаны функции для корректной работы программы. Сопоставленные входные и выходные данные программы удовлетворяют условию задания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. www.cplusplus.com
2. www.geeksforgeeks.com
3. другие ресурсы, такие как Youtube.

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include<math.h>


#define SEARCH_WORD "garbage"

#define TEXT_EXAMPLE "Before the early 1960s,grabage computers  were mainly garbage garbagegarbage garbage garbage used grabage for number-crunching rather than for text, and memory was extremely expensive. Computers often allocated only 6 bits for each character, permitting only 64 characters—assigning codes for A-Z, a-z, and 0-9 would leave only 2 codes: nowhere near enough. Most computers opted not to support lower-case letters. Thus, early text projects such as Roberto Busa's Index Thomisticus, the Brown Corpus, and others had to resort to conventions such as keying an asterisk preceding letters actually intended to be  garbage garbage upper-case."


char** readText(int *sentence_counter){
   char **text = (char**)calloc(1, sizeof(char*));
   if (text == NULL){return NULL;}
   int i=0;
   int sentence_max=0;
   char c = getchar();
   do{
      i=0;
      if((text=(char**)realloc(text, ((*sentence_counter)+1)*sizeof(char*)))==NULL)
         return NULL;
      do{
         if ((text[*sentence_counter]=(char*)realloc(text[*sentence_counter], sizeof(char)*(i+3)))==NULL)
            return NULL;
         text[*sentence_counter][i]=c;
         i++;
      }while((c=getchar())!='.');
      text[*sentence_counter][i]='.';
      text[*sentence_counter][i+1]='\0';
      (*sentence_counter)++;
```

```
    if (i+2>sentence_max)
        sentence_max=i+2;
}while((c = getchar())!='\n');


char checker[*sentence_counter][sentence_max];
int j, k;

for(i=0; i<*sentence_counter; i++){
    strcpy(checker[i], text[i]);
    for(j=0; j<strlen(checker[i]); j++){
        checker[i][j]=tolower(checker[i][j]);
    }
}

for(i=0; i<*sentence_counter; i++)
    for(j=i+1; j<*sentence_counter; j++){

        int l=0;
        int found = 0;
        if (strlen(checker[i]) == strlen(checker[j]))
          while (l < strlen(checker[i])){
            if (checker[i][l] == checker[j][l]){
              found = 1;
              l++;
            }else{
              found = 0;
              break;
            }
          }
        if (found == 1){
          (*sentence_counter)--;
          for(k=j; k<*sentence_counter; k++){
              text[k]=text[k+1];
```

```
                strcpy(checker[k], checker[k+1]);
                }
                j--;
            }
        }
    return text;
    }


void printText(char **text, int sentence_counter){
    for(int i=0; i<sentence_counter; i++){
        printf("========================================\n");
        printf("%d|   '%s'\n", i, text[i]);
        }
  }


char** convert_to_lowercase(char **text, int sentence_counter){
  char** text2 = (char**)malloc((sentence_counter)*sizeof(char*));
  int j = 0;
  if (text2 == NULL) return NULL;
    for(int i=0; i<sentence_counter; i++){
      text2[i] = (char*)realloc(text2[i], sizeof(char)*(strlen(text[i])+1));
      if (text2[i]==NULL) return NULL;
      j = 0;
      while(text[i][j] != '\0'){
        text2[i][j] = tolower(text[i][j]);
        j++;
      }
      text2[i][strlen(text2[i])] = '\0';
    }
    return text2;
  }
```

```
int count_garbage(char **text, int sentence_counter){

   char** text2 = convert_to_lowercase(text,sentence_counter);
   int count, letter = 0, total_count = 0;
   int search_word_len = strlen(SEARCH_WORD);
   char *word = (char*)calloc( 1,search_word_len * sizeof(char));
   if (word==NULL) return total_count;
   for(int i=0; i<sentence_counter; i++){
      count = 0;
      for (int j=0; j< strlen(text2[i]); j++){
        int k=0;
        for (k=0; k<search_word_len; k++){
          word[k] = text2[i][k+j];
        }
        word[strlen(word)] = '\0';
        if (strcmp(word, SEARCH_WORD) == 0){
          count++;
        }
      }
      printf("\n==========================\n");
      printf("For sentence: '%s'\n=============\n",text2[i]);
      if(count == 0 )
        printf("Clean\n");
      else if(count > 5)
        printf("It is a dump\n");
      else
        printf("Must be washed\n");
      printf("=============\n");
      total_count+=count;
      }
      return total_count;
}
```

```
int replace_all_digits(char **text, int *sentence_counter, char* input_string){
  int len = strlen(input_string);
  int count = 0;
  for(int i=0; i<*sentence_counter; i++){
    for (int j=0; j<strlen(text[i]); j++){
      if (isdigit(text[i][j])){
        count++;
        // Replace
        text[i] = (char*)realloc(text[i], (strlen(text[i])+len+1) * sizeof(char));// extend the string to add the new string
        if (text[i]==NULL) return count;
        char* temp = (char*)malloc(strlen(text[i])+sizeof(char));
        if(temp==NULL) return count;
        // copy all chars after the digit
        for (int k=j; k<strlen(text[i]); k++){
          temp[k-j] = text[i][k+1];
        }
        temp[strlen(temp)] = '\0';
        //adding the substring
        for (int k=0; k<len; k++){
          text[i][k+j] = input_string[k];
        }

        //returning all chars from the memory
        for(int k=0; k<strlen(temp); k++){
          text[i][j+k+len] = temp[k];
        }
        text[i][strlen(text[i])] = '\0';
        j--;
      }
    }
  }
  return count;
}
```

```
int three_consecutive_letters_in_upper_case(char *sentence){

   for(int i=0; i < strlen(sentence)-2; i++){
      if(isupper(sentence[i]) && isupper(sentence[i+1]) && isupper(sentence[i+2]))
        return 1;
   }
   return 0;
}

int delete_if_three_consecutive_letters_in_upper_case(char **text, int *sentence_counter){
  int num = 0;
   for(int i=0; i<*sentence_counter; i++){
      if(three_consecutive_letters_in_upper_case(text[i])){
         free(text[i]);
         (*sentence_counter)--;
         for (int j = i; j < *sentence_counter; j++)
           text[j] = text[j + 1];
        i--;
        num++;
      }
   }
   return num;
  }

int is_vowel(char ch){
   /* Condition for vowel */
  if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u' ||
     ch=='A' || ch=='E' || ch=='I' || ch=='O' || ch=='U')
     return 1;
  else
     return 0;
}
```

```
int num_words_beginning_with_vowels(char* sentence){
  int start=0;
  int num_words=0;
  if (is_vowel(sentence[start])){
      num_words++;
    }
  int i=0;
  while(sentence[i] != '.'){
    if (sentence[i]==' ' || sentence[i]==',') {
        start = i+1;
        if (is_vowel(sentence[start])){
          num_words++;
          }
      }
      i++;
  }
  return num_words;
}

int compare_func( const void* a, const void* b)
{
     int int_a = * ( (int*) a );
     int int_b = * ( (int*) b );

     if ( int_a == int_b ) return 0;
     else if ( int_a < int_b ) return -1;
     else return 1;
}
int cmp_ptr(const void *a, const void *b)
{
    const int **left  = (const int **)a;
    const int **right = (const int **)b;
```

```
    return (**left < **right) - (**right < **left);
}

int * order_indices(const int *a, int n)
{
    const int **pointers = malloc(n * sizeof(const int *));
    if (pointers==NULL) return NULL;
    for (int i = 0; i < n; i++) pointers[i] = a + i;

    qsort(pointers, n, sizeof(const int *), cmp_ptr);

    int *indices = malloc(n * sizeof(int));
    if (indices==NULL) return NULL;
    for (int i = 0; i < n; i++) indices[i] = pointers[i] - a;

    free(pointers);

    return indices;
}



void sort_by_descending_number_of_words_beginning_with_vowels(char** text, int sentence_counter){
  int* indices = (int*)malloc(sentence_counter*sizeof(int));
  if (indices == NULL) return;

  for(int i=0; i<sentence_counter; i++){
    int num = num_words_beginning_with_vowels(text[i]);
    indices[i] = num;
  }
  indices = order_indices(indices, sentence_counter);

  char** text_copy = (char**)malloc(sentence_counter*sizeof(char*));
  if (text_copy == NULL) return;
  for(int i=0; i<sentence_counter; i++){
```

```
    text_copy[i] = (char*)malloc((strlen(text[i])+1)*sizeof(char));
    if (text_copy[i]==NULL) return;
    strcpy(text_copy[i], text[i]);
  }
  for(int i=0; i<sentence_counter; i++){
    text[i] = (char*)realloc(text[i],sizeof(char)*(strlen(text_copy[indices[i]])+1));
    if (text[i]==NULL) return;
    text[i] = text_copy[indices[i]];
  }
}




int main(){
    int sentence_counter=0;
    int command;
    int num;
    // char input_string[50];
    char* input_string = (char*) malloc(sizeof(char)*1);
    printf("Здравствуйте! Введите Ваш текст.\nПодсказка: текст состоит из предложений, которые состоят из набора слов, разделённых запятой ИЛИ пробелом.\nСлова состоят из цифр и латинских букв.\n\n");
    char** text = readText(&sentence_counter);
    if(text==NULL){
        printf("Ошибка выделения памяти.\n");
        return 1;}


    printText(text, sentence_counter);


    printf("----------------------------------------\n");
    printf("Введите одно из чисел ниже, чтобы выбрать действие:\n1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра). \n2) Заменить все цифры в предложении на введенную строку.\n3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.\n4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.\n0) Выйти из программы.\n");
    printf("----------------------------------------\n");
```

```
printf("------Type command № >> ");
scanf("%d", &command);
printf("\n");
int sentences_count = 0;
while(command!=0){
 switch(command){
    case 1:
       printf("Number of '%s' occuring in text is %d\n", SEARCH_WORD, count_garbage(text, sentence_counter));
       break;

    case 2:
       printf("Please input the string(50 chars) to replace digits with: ");
       scanf("%s", input_string);
       num = replace_all_digits(text, &sentence_counter,input_string);
       printText(text, sentence_counter);
       printf("---------------------------------------\n");
       printf("Number of Replaces is %d\n", num);
       break;

    case 3:
       sentences_count = sentence_counter;
       num = delete_if_three_consecutive_letters_in_upper_case(text, &sentence_counter);
       printText(text, sentence_counter);
       printf("---------------------------------------\n");
       printf("Number of Deletions is %d\n", num);
       printf("Remaining sentences is %d\n", sentences_count - num);
       break;
    case 4:
     sort_by_descending_number_of_words_beginning_with_vowels(text, sentence_counter);
     printText(text, sentence_counter);
     break;

    case 0:
     break;
```

```
        default:
          printf("Неправильный ввод!\n");
          break;
      }
      printf("---------------------------------------\n");
      printf("Введите одно из чисел ниже, чтобы выбрать действие:\n1) Для каждого предложения посчитать количество слов “garbage” в нем (без учета регистра). \n2) Заменить все цифры в предложении на введенную строку.\n3) Удалить все предложения в которых есть три подряд идущие буквы в верхнем регистре.\n4) Отсортировать по уменьшению количества слов начинающихся с гласной буквы.\n0) Выйти из программы.\n");
      printf("---------------------------------------\n");
      printf("------Type command № >> ");
      scanf("%d", &command);
   }
   printf("Program is Exited Successfully");
   return 0;

}
```