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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текста

Студентка гр. 1304 _____ Виноградова М.О.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Виноградова М.О.

Группа 1304

Тема работы : обработка текста

Исходные данные:
На вход программе подается текст. Текст состоит из предложений разделенных точкой. Предложения состоят из слов разделенных пробелом или запятой. Требуется использовать структуры для получения и обработки текста.

Содержание пояснительной записки: «Содержание», «Введение», «Разработка программного кода», «Сборка программы», «Тестирование», «Пользовательские инструкции», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 24.12.2021

Дата защиты реферата: 25.12.2021

Студентка _____ Виноградова М.О.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Реализована программа по обработке текста на языке программирования СИ. Для записи и работы с текстом реализованы структуры Text и Sentence. Пользователь должен ввести одну из предложенных команд (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: выход из программы), согласно команде будет выполнена обработка текста (или выход из программы).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью является написание программы, которая должна считать текст в виде динамического массива предложений, и выполнить ряд действий по обработке полученного текста.

Для реализации данной программы требуется:

- изучить следующие темы: широкие символы, работа с динамической памятью, структуры и их применение.

- создать Makefile

- использование стандартных функция языка СИ.

1. РАЗРАБОТКА ПРОГРАММНОГО КОДА

1.1. Техническое задание

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 слова.
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.

1.2. Запись и хранение текста

Необходимые структуры:

Sentence – массив символов(words), длина предложения без учета знака разделителя предложений(len), объем выделенной памяти (size).

Text – массив из указателей на предложения (sentences), количество предложений (n), объем выделенной памяти (size).

Функции:

readSentence() – возвращает указатель на структуру Sentence.В функции считывается предложение до точки (знака разделителя предложений).

readText() – возвращает структуру Text. В функции сохраняются предложения полученные в функции readSentence() до получения знака переноса строки(символа окончания ввода).

1.3. Обработка текста

Функция *del_rep*():

Функция получает на вход текст(struct Text text) из которого необходимо удалить все повторно встретившиеся предложения (если предложение встретилось несколько раз, то сохраняется только первое его вхождение). Для нахождения повтора реализовано посимвольное сравнение предложений без учета регистра. Функция возвращает структуру Text.

1.4. Функции обработки текста согласно выбранной команде

Необходимы структуры для обработки текста:

Mask – общая маска для предложения.
Arr – количество повторов слова(kol_repeat), слово (word), количество слов без повторов(len_rep).

Функции:

task1() – на вход функция получает текст. Каждая строчка разбивается на слова с помощью функции стандартной библиотеки strtok(). Далее происходит формирование маски по первым двум словам. В цикле слова сравниваются попарно и с полученной маской, в результате формируя маску в соответствие с условием. Полученная по предложению маска записывается в массив mask. Функция возвращает массив из структур Mask.

task2() - на вход функция получает текст. Каждая строчка разбивается на слова с помощью функции стандартной библиотеки strtok(). Слова проверяются на заглавные буквы в начале слова. Если в предложение есть слова начинающиеся с строчной буквы, то такие предложения удаляются. Функция возвращает структуру Text.

task3() - на вход функция получает текст. Каждая строчка разбивается на слова и разделители с помощью функции стандартной библиотеки strtok(). Внутри функции сортируются слова по количеству гласных букв. Функция возвращает структуру Text.

task4() – на вход функция получает текст. Каждая строчка разбивается на слова с помощью функции стандартной библиотеки strtok(). Далее находится количество повторов слова в предложение. В двумерный массив arr_get – записываются слово и количество его повторов. Функция возвращает двумерный массив из структур Arr.

1.5. Вывод результата

В функции *main()* выводится результат работы программы в зависимости от выбранной пользователем команды. При вводе команды 2 и 3 выводится обработанный текст. При вводе 1 выводятся маски по каждому предложению. При вводе 4 выводятся количество повторов слов в предложении и само слово.

2. СБОРКА ПРОГРАММЫ

Функции ввода-вывода(readSentence, readText) объединены в один файл main.c. Функции обработки текста(del_rep, task1, task2, task3, task4) объединены в файле text_processing.c , с заголовочным файлом text_processing.h.

Сборка программы осуществляется утилитой make в соответствие с Makefile.

3. ТЕСТИРОВАНИЕ

№	Ввод	Введенная команда	Вывод
1	Аристотель Артишок.АРИСТОТЕЛЬ АРТИШОК. Столяр Сидел За Стулом. Hello, World.	**1**	0: mask= Ap???o?* 1: mask= ??* 2: mask= ???l?
2	Аристотель Артишок. Столяр Сидел За Стулом. Hello, World. hello.	**2**	Аристотель Артишок. Столяр Сидел За Стулом. Hello, World.
3	аеёи гласныее, эоя тоже гласные. abc, dio, europe, pancakes.	**3**	аеёи гласныее, эоя гласные тоже. europe, pancakes, dio, abc.
4	Аристотель Аристотель Артишок. hello, hello, hello, world. word not word, but word.	**4**	'Аристотель':1; 'Артишок':0; 'hello':2; 'world':0; 'word':2; 'not':0; 'but':0;
5	аеёи гласныее, эоя тоже гласные. abc, dio, europe, pancakes.	**5**	

4. ПОЛЬЗОВАТЕЛЬСКИЕ ИНСТРУКЦИИ

I – запустите программу (make && ./a.out)
II – введите текст. Предложение должно заканчиваться точкой. Перенос строки – символ конца ввода текста.
III - введите необходимую команду для обработки текста.
IV – для выхода из программы введите 5. Для применения еще одной команды повторите шаг III.

ЗАКЛЮЧЕНИЕ

Реализована программа для считывания текста и его дальнейшей обработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт cplusplus.com - https://www.cplusplus.com/reference/cstring/strcat/
2. Сайт all-ht.ru - http://all-ht.ru/inf/prog/c/func/strtok.html
3. Сайт Wikipedia.org - https://ru.wikipedia.org/wiki/String.h

ПРИЛОЖЕНИЕ А

main.c

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
#include <locale.h>
#include <string.h>
#include "text_processing.h"
#define STEP 5


struct Sentence{
    wchar_t *words;
    int size;
    int len;


};
struct Sentence* readSentence(){
    int size=STEP;

    wchar_t *buf = malloc(size* sizeof(wchar_t));
    wchar_t temp;
    int n=0;
    do{
        if(n<=size-2){
            wchar_t *t= realloc(buf,size+STEP);
            if(!t){puts("ERROR: закончилась память");}
            size+=STEP;
            buf=t;
            //printf("new size=%d\n",size);

        }
        temp=getwchar();
        buf[n]=temp;
        n++;
    }while(temp!='.' && temp!='\n');
    buf[n]='\0';
    struct Sentence *sentence= malloc(sizeof (struct Sentence));
    sentence->words=buf;
    sentence->size=size;
    sentence->len=n-1;

    return sentence;
}

struct Text{
    struct Sentence **sentences;
    int size;
    int n;
```

```
};

struct Text readText(){
   int size=STEP;
   struct Sentence** txt= malloc(size*sizeof(struct Sentence));
   int n=0;
   struct Sentence *temp;
   int null_kol=0;
   do{
      temp=readSentence();
      if(temp->words[0]=='\n'){
         null_kol++;
         free(temp->words);
         free(temp);
      }else{
         null_kol=0;
         txt[n]=temp;
         n++;
      }
   } while (null_kol<1);

   struct Text full_text;
   full_text.size=size;
   full_text.sentences=txt;
   full_text.n=n;

   return full_text;

}

struct Arr{
   int kol_repeat;
   wchar_t* word;
   int len_rep;
};

   struct Mask{
   wchar_t *masks;
};



int main() {

   setlocale(LC_ALL,"");

   struct Text new_text = readText();
   new_text = del_rep(new_text);

   for(int i=0;i<new_text.n;i++){
//       wprintf(L"\nstring---->  %d : %ls %d %d\n",i,new_text.sentences[i]-
>words,new_text.sentences[i]->len,new_text.n);
```

```
    }


    ////////////////////////////////////////////////////////////////////////////////

    int k=1;
while(k){
    int operation;
    printf("\n\nДля получения маски по каждому предложению введите 1.\nЧтобы удалить все предложения, в которых нет заглавных букв в начале слова введите 2."
           "\nЧтобы отсортировать слова в предложении по количеству гласных букв в слове введите 3."
           "\nЧтобы для каждого предложения получить количество одинаковых слов в строке введите 4."
           "\nЧтобы завершить программу введите 5.\n--->");
    scanf("%d",&operation);
    switch (operation) {
        case 1:{
            struct Mask *mask=task1(new_text);
            for(int i=0;i<new_text.n;i++){
                wprintf(L"%d: mask= %ls\n",i,mask[i].masks);
                }

            free(mask);
            }
            break;
        case 2:{
            new_text= task2(new_text);
            for(int i=0;i<new_text.n;i++){
                wprintf(L"%ls",new_text.sentences[i]->words);
            }
            puts("");
        }
            break;
        case 3:{

            new_text= task3(new_text);
            for(int i=0;i<new_text.n;i++){
                wprintf(L"%ls",new_text.sentences[i]->words);
            }
            puts("");
        }
            break;
        case 4:{
            struct Arr **arr= task4(new_text);

            for(int i=0;i<new_text.n;i++){
                int len=arr[i][0].len_rep;
                //printf("%d",len);
                for(int z=0;z<len;z++){
                    //printf("s ");
```

```
                wprintf(L"'%ls':%d; ",arr[i][z].word,arr[i][z].kol_repeat);
            }
            puts("");



        }
        for(int f=0;f<new_text.n;f++){
            free(arr[f]);
        }
        free(arr);

    }
        break;
    case 5:{
        k=0;
    }break;
    default:
        puts("");


}
}



    for(int i=0;i<new_text.n;i++){
        free(new_text.sentences[i]->words);
    }
    for(int i=0;i<new_text.n;i++){
        free(new_text.sentences[i]);
    }
    }









    ////////////////////////////////////////////////////////////////////////////////////////////////////


    return 0;
}
```

text_processing.c

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
#include <locale.h>
#include <string.h>
#include "text_processing.h"
struct Sentence{
   wchar_t *words;
   int size;
   int len;

};
struct Text{
   struct Sentence **sentences;
   int size;
   int n;
};

struct Mask{
   wchar_t *masks;
};



struct Text del_rep(struct Text text){


   wchar_t *check_words;//строчки для сравнения
   wchar_t *word_now;
   struct Text new_text=text;
   int len_of_txt=new_text.n;


   for(int i=0;i<len_of_txt;i++){
      check_words=new_text.sentences[i]->words;//check_words пробегается по всем
предложениям
      int len_w_check=new_text.sentences[i]->len;
      for(int j=i+1;j<len_of_txt;j++){
         word_now=new_text.sentences[j]->words;//текущее предложение
         int len_w_now=new_text.sentences[j]->len;
         int len=new_text.sentences[j]->len;
         int check_symb=0;
         ////посимвольное сравнение
         if(len_w_check==len_w_now) {
            for (int k = 0; k < len; k++) {
               if (towupper(word_now[k]) == towupper(check_words[k])) {

                  check_symb = 1;
               } else {
```

```
                    check_symb = 0;
                    break;
                }
            }
        }
        ////
        if(check_symb){//если предложения совпали сдвигаем массив так, чтобы удалить повторяющееся предложение

            //wprintf(L"%ls %ls\n",word_now,check_words);
            free(new_text.sentences[len_of_txt]);
            memmove(&new_text.sentences[j],&new_text.sentences[j+1],(len_of_txt-j)*sizeof(struct Sentence*));
            len_of_txt--;
            j--;

        }


    }

}

new_text.n=len_of_txt;

return new_text;
}


struct Mask* task1(struct Text text){
    int len_of_txt=text.n;
    struct Mask *mask= malloc(sizeof (struct Mask)*len_of_txt);

    for(int i=0;i<len_of_txt;i++) {
        int max=0,min;
        int full_len = text.sentences[i]->len;
        wchar_t str[full_len + 2];
        wcscpy(str, text.sentences[i]->words);

        wchar_t *tr;
        wchar_t *token = wcstok(str, L" ,.", &tr);

        wchar_t *list_of_words[full_len];
        int kol = 0;
        //массив из слов
        while (token != NULL) {
            list_of_words[kol++] = token;

            token = wcstok(NULL, L" ,.", &tr);
        }
```

```
min= wcslen(list_of_words[0]);

for(int h=0;h<kol;h++ ){

   if(max < wcslen(list_of_words[h])){
      max= wcslen(list_of_words[h]);

   }
   if(min>wcslen(list_of_words[h])){
      min=wcslen(list_of_words[h]);
   }

}


wchar_t *mask_first = malloc(sizeof (wchar_t)*(max+2));//максимальной длины

if(kol>1){
   int z;
   int min_len;
   if(wcslen(list_of_words[0])<wcslen(list_of_words[1])){
      min_len=wcslen(list_of_words[0]);
   }else{
      min_len=wcslen(list_of_words[1]);
   }

   for(z=0;z< min_len;z++){
      if(list_of_words[0][z]==list_of_words[1][z]){
         mask_first[z]=list_of_words[0][z];
      }else{
         mask_first[z]='?';
      }
   }
   if(wcslen(list_of_words[0])!= wcslen(list_of_words[1])){
      mask_first[z]='*';
   }
}else{
   wcscpy(mask_first,list_of_words[0]);
   mask[i].masks=mask_first;
   continue;
}



for(int q=1;q<kol-1;q++){

   wchar_t* word_now=list_of_words[q];
   wchar_t* word_next=list_of_words[q+1];
   int index;

   for(index=0;index< min;index++){
```

```
            if((word_now[index]==word_next[index]) && (word_next[index]==mask_first[index])
&& (word_now[index]==mask_first[index])){
                //если совпали не изменять


            }else{

                mask_first[index]='?';
            }
        }

        if(wcslen(word_now)!= wcslen(word_next)){
            if(mask_first[index-1]!='*'){
                mask_first[index]=L'*';
                mask_first[index+1]=L'\0';
            }else{
                mask_first[index]= L'\0';
            }

        }


    }
    mask[i].masks=mask_first;

  }



  return mask;
}



struct Text task2(struct Text text){


  int len_of_txt=text.n;

  for(int i=0;i<len_of_txt;i++){
    int full_len=text.sentences[i]->len;
    wchar_t str[full_len+2];
    wcscpy(str,text.sentences[i]->words);


    wchar_t *tr;
    wchar_t *token=wcstok(str,L" ,.",&tr);


    wchar_t *list_of_words[full_len];
    int kol=0;
    do {
```

```
            list_of_words[kol++]=token;
            token = wcstok(NULL, L" ,.", &tr);
        } while (token!=NULL);



        int kol_up=0;

        for(int q=0;q<kol;q++){
            if(list_of_words[q][0]== towupper(list_of_words[q][0])){
                kol_up++;
            }

        }
        if(kol_up!=kol){//если не все слова с заглавной буквы

            free(text.sentences[len_of_txt]);
            memmove(&text.sentences[i],&text.sentences[i+1],(len_of_txt-i)*sizeof(struct Sentence*));
            len_of_txt--;
            i--;
        }

    }
    text.n=len_of_txt;
    return text;


}

struct Text task3(struct Text text){
    int len_of_txt=text.n;

    wchar_t sym[]=L"aeyuioEUIOAуеёэоаыяиюЮИЯЫАОЭЁЕУ";
    wchar_t
alphabet[]=L"qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNMйцукенгшщ
зхъфывапролджэёячсмитьбюЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЁЯЧСМИТЬБЮ";

    for(int i=0;i<len_of_txt;i++) {
        int full_len = text.sentences[i]->len;
        wchar_t str[full_len +2];
        wchar_t str_a[full_len+2];
        wcscpy(str, text.sentences[i]->words);
        wcscpy(str_a, text.sentences[i]->words);

        wchar_t *tr;
        wchar_t *token = wcstok(str, L" ,.", &tr);

        wchar_t *list_of_words[full_len];
        int kol = 0;
        while (token != NULL){
            list_of_words[kol++] = token;
            token = wcstok(NULL, L" ,.", &tr);
```

```
}

wchar_t *token_a = wcstok(str_a, alphabet, &tr);
wchar_t *list_of_sp[full_len];
int kol_sp = 0;
while (token_a != NULL){
   list_of_sp[kol_sp++] = token_a;
   token_a = wcstok(NULL, alphabet, &tr);
}

wchar_t change[full_len +2];

for(int q=0;q<kol-1;q++){
   for(int w=q;w<kol;w++){
      wchar_t *vowels_a,*vowels_b;



      wchar_t *word_a=list_of_words[q];
      wchar_t *word_b=list_of_words[w];



      vowels_a = wcspbrk (word_a, sym);

      int kol_a=0,kol_b=0;

      while (vowels_a != NULL)
      {
         kol_a++;
         vowels_a = wcspbrk (vowels_a+1,sym);
      }


      vowels_b = wcspbrk (word_b, sym);
      while (vowels_b != NULL)
      {
         kol_b++;
         vowels_b = wcspbrk (vowels_b+1,sym);
      }

      if(kol_b>kol_a){
         list_of_words[q]=word_b;
         list_of_words[w]=word_a;


      }


   }

}
```

```
        wchar_t change_now[full_len +2];
        if(kol==kol_sp && kol>1){
            wcscpy(change,list_of_words[0]);
            wcscat(change,list_of_sp[0]);

            for(int k=1;k<kol;k++){
                int len_sym= wcslen(list_of_words[k]), len_sep= wcslen(list_of_sp[k]);
                wchar_t pair[len_sym+len_sep+1];
                wcscpy(pair,list_of_words[k]);
                wcscat(pair,list_of_sp[k]);

                wcscat(change,pair);

            }

            wcscpy(text.sentences[i]->words,change);

        } else{
            wcscpy(change,list_of_words[0]);
            wcscat(change,list_of_sp[1]);

            for(int k=1;k<kol;k++){
                int len_sym= wcslen(list_of_words[k]), len_sep= wcslen(list_of_sp[k+1]);

                wchar_t pair[len_sym+len_sep+1];

                wcscpy(pair,list_of_words[k]);
                wcscat(pair,list_of_sp[k+1]);
                //wprintf(L"'%ls'\n",pair);
                wcscat(change,pair);
                //wprintf(L"%d %d\n",len_sym,len_sep);
                //wcscat(change,wcscat(list_of_words[k],list_of_sp[k]));
                //wprintf(L"STR=='%ls'\n",change);
            }

            wcscpy(change_now,list_of_sp[0]);
            wcscat(change_now,change);
            wcscpy(text.sentences[i]->words,change_now);

        }



    }


    return text;
}
```

```
struct Arr{
   int kol_repeat;
   wchar_t* word;
   int len_rep;
};

struct Arr** task4 (struct Text text){
   int len_of_txt=text.n;
   struct Arr** arr_get= malloc(len_of_txt*sizeof (struct Arr*));



   for(int i=0;i<len_of_txt;i++) {
      int full_len = text.sentences[i]->len;


      wchar_t* str= malloc(sizeof (wchar_t)*(full_len+2));
      wcscpy(str,text.sentences[i]->words);


      wchar_t *tr;
      wchar_t *token = wcstok(str, L" ,.", &tr);

      wchar_t *list_of_words[full_len];
      int kol = 0;
      while (token != NULL) {
         list_of_words[kol++] = token;
         token = wcstok(NULL, L" ,.", &tr);
      }


      arr_get[i]= malloc(kol*sizeof (struct Arr));


      // int max=0;
      int kol_arr=0;
      for(int q=0;q<kol;q++){
         int kol_rep=0;
         wchar_t *word_check = list_of_words[q];

         for(int w=q+1;w<kol;w++){


            if(wcscmp(word_check,list_of_words[w])==0){
               kol_rep++;
               memmove(&list_of_words[w],&list_of_words[w+1],(kol-w)*sizeof (wchar_t*));
               w--;
               kol--;
            }
```

```
            }


            arr_get[i][kol_arr].word=word_check;
            arr_get[i][kol_arr].kol_repeat=kol_rep;

            kol_arr++;







        }
        arr_get[i][0].len_rep=kol;
    }

    return arr_get;
}
```

text_processing.h

```
#pragma once
#include<stdio.h>
#include<string.h>
struct Text del_rep(struct Text text);
struct Text task2(struct Text text);
struct Arr** task4 (struct Text text);
struct Mask* task1 (struct Text text);
struct Text task3 (struct Text text);
```

Makefile

```
all: main.o text_processing.o
	gcc main.o text_processing.o

main.o: main.c text_processing.h
	gcc -c main.c

text_processing.o: text_processing.c text_processing.h
	gcc -c text_processing.c

clean:
	 rm *.o a.out
```