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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа со строками в языке Си

Студентка гр. 9304 _____ Каменская Е.К.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2019

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Каменская Е.К.

Группа 9304

Тема работы: Работа со строками в языке Си

Исходные данные:
Текст, состоящий из предложений, разделителями между которыми являются последовательности точка-пробел и точка-символ переноса строки (\n). Предложения состоят из слов, разделенных пробелом или последовательностью запятая-пробел, точка на конце предложения является его частью. Слова состоят из букв и цифр, точка и запятая также относятся к слову, за которым они следуют. Ввод текста заканчивается двумя символами переноса строки.

Содержание пояснительной записки: «Содержание», «Введение», «Исходное задание», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 15.10.2019

Дата сдачи реферата: 21.12.2019

Дата защиты реферата: 21.12.2019

Студентка _____ Каменская Е.К.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Курсовая работа представляет собой программу для считывания и обработки текста в соответствии с выбранным пользователем действием. Код программы написан на языке программирования Си с использованием функций стандартных библиотек и управляющих конструкций и предназначен для запуска на операционных системах семейства Linux.

Для выполнения подзадач, а также ввода и вывода текста были реализованы собственные функции. Для проверки работоспособности и использования памяти было проведено тестирование.

Исходный код программы и результаты тестирования представлены в соответствующих разделах.

SUMMARY

The coursework is a program for reading and processing text in accordance with the user-selected action. The program code is written in the C programming language using the functions of standard libraries and control structures and is designed to run on Linux operating systems.

To perform subtasks, as well as input and output of text, custom functions were created and implemented. To test the efficiency and memory usage, testing was conducted.

The source code of the program and the test results are presented in the according sections.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – разработка программы на языке Си согласно заданию, используя для этого знания, полученные в течение первого семестра по предмету Программирование. Для конкретной задачи: создать программу для считывания текста, его обработки в соответствии с указанием пользователя и последующего вывода измененного текста на экран.

1. ИСХОДНОЕ ЗАДАНИЕ

Вариант 6

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) Распечатать каждое слово и количество его повторений в тексте.

2) Преобразовать каждое предложение так, что символы в каждом слове шли в обратном порядке.

3) Удалить предложения в котором встречается запятая.

4) Отсортировать предложения по уменьшению значения кода 5 символа предложения. Если 5 символ является разделителем между словами, то брать следующий символ. Если символов в предложении меньше 5, то для этого предложения значение равняется -1.

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.

2. ОПИСАНИЕ КОДА ПРОГРАММЫ

2.1. Описание структур

- Предложение: список слов (двумерный массив символов), предложение в обычном виде (одномерный массив символов), количество слов.

```
typedef struct Sentence{
	wchar_t** tokenized;
	wchar_t* normal;
	int length;
} Sentence;
```

- Текст: массив предложений, количество предложений.

```
typedef struct Text{
	Sentence* arr;
	int size;
} Text;
```

- Количество вхождений слова: список слов (двумерный массив символов), список вхождений, количество слов.

```
typedef struct WordCount{
	wchar_t** words;
	int* counts;
	int size;
} WordCount;
```

2.2. Краткое описание функций

Общие подзадачи:

void initializeText(Text *text), void initializeSentence(Sentence *snt)– инициализация текста и предложения по умолчанию.

void textReduce(Text* text), void sentenceReduce(wchar_t** snt), void wordCountReduce(WordCount* OList) – уменьшение занимаемой памяти.

Считываение текста:

wchar_t* readNormalSentence() – считывание предложения посимвольно из stdin.
void readText(Text* text) – добавление новых предложений.

Первичная обработка текста:
void tokenizeSentence(Sentence* snt) – разбиение предложения на слова по пробелу.
void removeRepeated(Text* text) – удаление повторяющихся предложений без учета регистра.
void deleteSentenceFromArray(Text* text, int index) – удаление предложения из массива в структуре Text.

Вывод:
void printUserMenu() – подсказка для пользователя.
void printText(Text* text) – измененный текст, каждое предложение с новой строки.
void printWordCount(WordCount* OList) – строка «слово = количество повторений» для всех различных слов с новой строки.

Первая подзадача:
WordCount* countOccurrences(Text* text) – создание структуры WordCount и ее заполнение.
int isInList(wchar_t* token, wchar_t** list, int n) – проверка наличия слова в списке words структуры WordCount.

Вторая подзадача:
void wordReverse(wchar_t* word, unsigned long int end) – инверсия одного слова.
void backwardTokens(Text* text) – замена слов в предложении на инвертированные.

Третья подзадача:

void deleteByComma(Text* text) – поиск и удаление всех предложений с запятой.

Четвертая подзадача:

void sortByFifth(Text* text) – сортировка по коду 5 символа.

int symcodecmp(const void* a, const void* b) – алгоритм для qsort.

Освобождение памяти:

void freeText(Text* text);

void freeSentence(Sentence *snt);

void freeWordCount(WordCount* OList);

3. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

```
Введите текст:
Big city life. Me try fi get by.Pressure nah ease up, no matter how hard me try.
Big city life. Here my heart have no base. And right now Babylon de pon me case.

Выберите действие:
        1 - Распечатать каждое слово и количество его повторений в тексте.
        2 - Преобразовать каждое предложение так, чтобы символы в каждом слове шли в обратном порядке.
        3 - Удалить предложения, в которых встречается запятая.
        4 - Отсортировать предложения по уменьшению значения кода 5 символа предложения.
        Остальные цифры - Выйти.

Введите соответствующий номер: 1
Результат:
Big = 1
city = 1
life. = 1
Me = 3
try = 1
fi = 1
get = 1
by. = 1
Pressure = 1
nah = 1
ease = 1
up, = 1
no = 2
matter = 1
how = 1
hard = 1
try. = 1
Here = 1
my = 1
heart = 1
have = 1
base. = 1
And = 1
right = 1
now = 1
Babylon = 1
de = 1
pon = 1
case. = 1
```

```
Выберите действие:
        1 - Распечатать каждое слово и количество его повторений в тексте.
        2 - Преобразовать каждое предложение так, чтобы символы в каждом слове шли в обратном порядке.
        3 - Удалить предложения, в которых встречается запятая.
        4 - Отсортировать предложения по уменьшению значения кода 5 символа предложения.
        Остальные цифры - Выйти.

Введите соответствующий номер: 4
Результат:
Pressure nah ease up, no matter how hard me try.
Me try fi get by.
And right now Babylon de pon me case.
Here my heart have no base.
Big city life.
```

```
Выберите действие:
        1 - Распечатать каждое слово и количество его повторений в тексте.
        2 - Преобразовать каждое предложение так, чтобы символы в каждом слове шли в обратном порядке.
        3 - Удалить предложения, в которых встречается запятая.
        4 - Отсортировать предложения по уменьшению значения кода 5 символа предложения.
        Остальные цифры - Выйти.

Введите соответствующий номер: 3
Результат:
Me try fi get by.
And right now Babylon de pon me case.
Here my heart have no base.
Big city life.
```

```
Выберите действие:
        1 - Распечатать каждое слово и количество его повторений в тексте.
        2 - Преобразовать каждое предложение так, чтобы символы в каждом слове шли в обратном порядке.
        3 - Удалить предложения, в которых встречается запятая.
        4 - Отсортировать предложения по уменьшению значения кода 5 символа предложения.
        Остальные цифры - Выйти.

Введите соответствующий номер: 2
Результат:
eM yrt if teg .yb
dnA thgir won nolybaB ed nop em .esac
ereH ym traeh evah on .esab
giB ytic .efil
```

4. КОД ПРОГРАММЫ

Файл cw_main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#include "print.h"
#include "sentence.h"
#include "text.h"
#include "wordcount.h"
#include "1_occurrences.h"
#include "2_reverse.h"
#include "3_del.h"
#include "4_sort.h"

int main()
{
	setlocale(LC_ALL, "");
	Text* text = malloc(sizeof(Text));
	initializeText(text);
	int choice = 0;
	wprintf(L"\033[1;96mВведите текст:\033[0m\n");
	readText(text);
	removeRepeated(text);
	while(1){
		printUserMenu();
	  wscanf(L"%d", &choice);
		switch(choice){
			case 1:;
				WordCount* OList = countOccurrences(text);
				printWordCount(OList);
				freeWordCount(OList);
				break;
			case 2:
				backwardTokens(text);
			printText(text);
				break;
			case 3:
				deleteByComma(text);
				printText(text);
```

```
                        break;
                    case 4:
                        sortByFifth(text);
                        printText(text);
                        break;
                    default:
                      freeText(text);
                        return 0;
                }
            }
        }
```

Файл print.h:

```
#pragma once

#include <stdio.h>
#include <wchar.h>
#include "text.h"
#include "wordcount.h"

void printUserMenu();
void printText(Text* text);
void printWordCount(WordCount* OList);
```

Файл print.c:

```
#include "print.h"

void printUserMenu(){
    wprintf(L"\033[1;96mВыберите действие:\033[0m\n"
            L"\t\033[1;33m1 - Распечатать каждое слово и количество его повторений в тексте.\n"
            L"\t2 - Преобразовать каждое предложение так, чтобы символы в каждом слове шли в обратном порядке.\n"
            L"\t3 - Удалить предложения, в которых встречается запятая.\n"
            L"\t4 - Отсортировать предложения по уменьшению значения кода 5 символа предложения.\n"
            L"\tОстальные цифры - Выйти.\033[0m\n"
            L"\n\033[1;96mВведите соответствующий номер: \033[0m");
}

void printText(Text* text){
    wprintf(L"\033[1;92mРезультат:\n\033[0m");
    for(int i=0; i<text->size;i++)
```

```
        wprintf(L"%ls\n",text->arr[i].normal);
}

void printWordCount(WordCount* OList){
    wprintf(L"\033[1;92mРезультат:\n\033[0m");
    for(int i = 0; i < OList->size; i++)
        wprintf(L"%ls = %d\n", OList->words[i], OList->counts[i]);
}
```

Файл sentence.h:

```
#pragma once

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <wchar.h>
#define DEF_SIZE 8

typedef struct Sentence{
      wchar_t** tokenized;
      wchar_t* normal;
      int length; // число слов
} Sentence;

void initializeSentence(Sentence *snt);
void sentenceReduce(wchar_t** snt);
wchar_t* readNormalSentence();
void tokenizeSentence(Sentence* snt);
void freeSentence(Sentence *snt);
```

Файл sentence.c:

```
#include "sentence.h"

void initializeSentence(Sentence *snt) {
    snt->tokenized = NULL;
    snt->normal = NULL;
    snt->length = 0;
}

void sentenceReduce(wchar_t** snt){
    *snt = realloc(*snt, (wcslen(*snt)+1) * sizeof(wchar_t));
}
```

```
wchar_t* readNormalSentence(){
    int size = DEF_SIZE;
    int n = 0;
    wchar_t* input = malloc(size * sizeof(wchar_t));
    if(!input){
        wprintf(L"\033[1;91mНедостаточно памяти для нового предложения.\
033[0m\n");
        exit(1);
    }
    wchar_t* buf = NULL;
    wchar_t wc;
    do{
        wc = fgetwc(stdin);
        if((n != 0)||(wc != L' ')){
            input[n] = wc;
            n++;
        }
        if(n == size - 2){
            size += DEF_SIZE;
            buf = realloc(input, size * sizeof(wchar_t));
            if(!buf){
                free(input);
                wprintf(L"\033[1;91mНедостаточно памяти для предложения.\
033[0m\n");
                exit(1);
            }
            input = buf;
        }
    }while(!wcschr(L".\n", wc));
    input[n] = L'\0';
    sentenceReduce(&input);
    return input;
}

void tokenizeSentence(Sentence* snt){
    wchar_t* wcs = malloc( (wcslen(snt->normal)+1) * sizeof(wchar_t));
    memcpy(wcs, snt->normal, (wcslen(snt->normal)+1) * sizeof(wchar_t));
    wchar_t* pwc;
    wchar_t* tmp;
    wchar_t** tok = malloc(1 * sizeof(wchar_t*)); // array of words
    int n = 0;
    pwc = wcstok(wcs, L" ", &tmp);
```

```
    while (pwc != NULL)
    {
        tok[n] = pwc;
        n++;
        tok = realloc(tok, (n+1)*sizeof(wchar_t*));
        pwc = wcstok(NULL, L" ", &tmp);
    }
    snt->tokenized = tok;
    snt->length = n;
}

void freeSentence(Sentence *snt) {
    free(*snt->tokenized);
    free(snt->tokenized);
    free(snt->normal);
    snt->length = 0;
}
```

Файл text.h:

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#define DEF_SIZE 8
#include "sentence.h"

typedef struct Text{
      Sentence* arr;
      int size;
} Text;

void initializeText(Text *text);
void textReduce(Text* text);
void readText(Text* text);
void removeRepeated(Text* text);
void deleteSentenceFromArray(Text* text, int index);
void freeText(Text* text);
```

Файл text.c:

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

```
void initializeText(Text *text) {
    text->arr = NULL;
    text->size = 0;
}

void textReduce(Text* text){
    text->arr = realloc(text->arr, text->size * sizeof(Sentence));
}

void readText(Text* text){
    int arr_size = DEF_SIZE;
    Sentence* buf = NULL;
    int newline = 0;
    text->arr = malloc(arr_size * sizeof(Sentence));
    while(newline < 2) {
        wchar_t *ws = readNormalSentence();
        if (!wcscmp(ws, L"\n")) {
            free(ws);
            newline += 1;
        } else {
            newline = 0;
            initializeSentence(&text->arr[text->size]);
            text->arr[text->size].normal = ws;
            tokenizeSentence(&text->arr[text->size]);
            text->size += 1;
            if (text->size == arr_size - 1) {
                arr_size += DEF_SIZE;
                buf = realloc(text->arr, arr_size * sizeof(Sentence));
                if (!buf) {
                    freeText(text);
                    wprintf(L"\033[1;91mНедостаточно памяти для текста\
033[0m\n");
                    exit(1);
                }
                text->arr = buf;
            }
        }
    }
    textReduce(text);
}

void removeRepeated(Text* text){
```

```
    int i = 0;
    while(i < text->size-1){
        int j = i+1;
        while(j < text->size){
            if(!wcscasecmp(text->arr[i].normal, text->arr[j].normal)){
                deleteSentenceFromArray(text, j);
            }
            else{
                j++;
            }
        }
        i++;
    }
    textReduce(text);
}

void deleteSentenceFromArray(Text* text, int index){
    freeSentence(&text->arr[index]);
    for(int k = index; k < text->size-1; k++)
        text->arr[k] = text->arr[k+1];
    text->size--;
}
```

Файл wordcount.h:

```
#pragma once

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <wchar.h>
#include <locale.h>
#define DEF_SIZE 8

typedef struct WordCount{
      wchar_t** words;
      int* counts;
      int size;
} WordCount;

void wordCountReduce(WordCount* OList);
void freeWordCount(WordCount* OList);
```

Файл wordcount.c:

```
#include "wordcount.h"

void wordCountReduce(WordCount* OList){
      OList->words = realloc(OList->words, OList->size *
sizeof(wchar_t*));
    OList->counts = realloc(OList->counts, OList->size * sizeof(int));
}

void freeWordCount(WordCount* OList){
    free(OList->words);
    free(OList->counts);
    free(OList);
}
```

Файл 1_occurrences.h:

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#define DEF_SIZE 8
#include "text.h"
#include "wordcount.h"

WordCount* countOccurrences(Text* text);
int isInList(wchar_t* token, wchar_t** list, int n);
```

Файл 1_occurrences.c:

```
#include "1_occurrences.h"

WordCount* countOccurrences(Text* text){
    WordCount* OList = malloc(sizeof(WordCount));
    OList->size = DEF_SIZE;
    OList->words = malloc(OList->size * sizeof(wchar_t*));
    OList->counts = calloc(OList->size, sizeof(int));
    wchar_t** wbuf;
    int* cbuf;
    int n = 0;
    int index;
    for(int i=0; i<text->size; i++){
        for(int j = 0; j < text->arr[i].length; j++){
            index = isInList(text->arr[i].tokenized[j], OList->words, n);
            if(index >= 0)
```

```
                    OList->counts[index] += 1;
                else {
                    OList->words[n] = text->arr[i].tokenized[j];
                    OList->counts[n] = 1;
                    n++;
                }
                if(n == OList->size){
                    OList->size+=DEF_SIZE;
                    wbuf = realloc(OList->words, OList->size *
sizeof(wchar_t*));
                    if(!wbuf){
                        wprintf(L"\033[1;91mНедостаточно памяти для массива
слов\n\033[1;91m");
                        freeWordCount(OList);
                        exit(1);
                    }
                    OList->words = wbuf;
                    cbuf = realloc(OList->counts, OList->size * sizeof(int));
                    if(!cbuf){
                        wprintf(L"\033[1;91mНедостаточно памяти для массива
вхождений\n\033[1;91m");
                        freeWordCount(OList);
                        exit(1);
                    }
                    OList->counts = cbuf;
                }
            }
        }
        OList->size = n;
        wordCountReduce(OList);
        return OList;
    }

    int isInList(wchar_t* token, wchar_t** list, int n){
        for(int i = 0; i < n; i++){
            if(!wcscasecmp(token, list[i]))
                return i;
        }
        return -1;
    }
```

Файл 2_reverse.h:

```
#pragma once
```

```
#include <string.h>
#include <wchar.h>
#include "text.h"

void wordReverse(wchar_t* word, unsigned long int end);
void backwardTokens(Text* text);
```

Файл 2_reverse.c:

```
#include "2_reverse.h"

void wordReverse(wchar_t* word, unsigned long int end){
    wchar_t temp;
    unsigned long int start = 0;
    while (start < end)
    {
        temp = word[start];
        word[start] = word[end];
        word[end] = temp;
        start++;
        end--;
    }
}

void backwardTokens(Text* text){
    unsigned long len;
    unsigned long index;
    for(int i = 0; i < text->size; i++){
        index = 0;
        for(int j = 0; j < text->arr[i].length; j++){
            len = wcslen(text->arr[i].tokenized[j]);
            wordReverse(text->arr[i].tokenized[j], len-1);
            memcpy(text->arr[i].normal + index, text->arr[i].tokenized[j],
len * sizeof(wchar_t));
            index += len + 1;
        }
    }
}
```

Файл 3_del.h:

```
#pragma once

#include <wchar.h>
```

```
#include "text.h"

void deleteByComma(Text* text);
```

Файл 3_del.c:

```
#include "3_del.h"

void deleteByComma(Text* text){
    int i = 0;
    while(i < text->size){
        if(wcschr(text->arr[i].normal, L','))
            deleteSentenceFromArray(text, i);
        else
            i++;
    }
    textReduce(text);
}
```

Файл 4_sort.h:

```
#pragma once

#include <stdlib.h>
#include <wchar.h>
#include "text.h"

void sortByFifth(Text* text);
int symcodecmp(const void* a, const void* b);
```

Файл 4_sort.c:

```
#include "4_sort.h"

void sortByFifth(Text* text){
    qsort(text->arr, text->size, sizeof(Sentence), symcodecmp);
}

int symcodecmp(const void* a, const void* b){
    Sentence* snt1 = (Sentence*)a;
    Sentence* snt2 = (Sentence*)b;

    if(wcslen(snt1->normal) < 5)
        return 1;
    if(wcslen(snt2->normal) < 5)
        return -1;
```

```
    int index = 4;
    wchar_t c1 = snt1->normal[index];
    if( c1 == L' ')
        c1 = snt1->normal[index+1];
    wchar_t c2 = snt2->normal[index];
    if( c2 == L' ')
        c2 = snt2->normal[index+1];
    return (int)c2 - (int)c1;
}
```

Makefile:

```
CC=gcc
FLAG=

all: vSIX clean

vSIX: cw_main.o print.o sentence.o text.o wordcount.o 1_occurrences.o
2_reverse.o 3_del.o 4_sort.o
	$(CC) $(FLAG) cw_main.o print.o sentence.o text.o wordcount.o
1_occurrences.o 2_reverse.o 3_del.o 4_sort.o -o vSIX

cw_main.o: cw_main.c print.h sentence.h text.h wordcount.h 1_occurrences.h
2_reverse.h 3_del.h 4_sort.h
	$(CC) $(FLAG) -c cw_main.c

print.o: print.c print.h
	$(CC) $(FLAG) -c print.c

sentence.o: sentence.c sentence.h
	$(CC) $(FLAG) -c sentence.c

text.o: text.c text.h
	$(CC) $(FLAG) -c text.c

wordcount.o: wordcount.c sentence.h
	$(CC) $(FLAG) -c wordcount.c

1_occurrences.o: 1_occurrences.c 1_occurrences.h
	$(CC) $(FLAG) -c 1_occurrences.c

2_reverse.o: 2_reverse.c 2_reverse.h
	$(CC) $(FLAG) -c 2_reverse.c
```

```
3_del.o: 3_del.c 3_del.h
	$(CC) $(FLAG) -c 3_del.c

4_sort.o: 4_sort.c 4_sort.h
	$(CC) $(FLAG) -c 4_sort.c

clean:
	rm -f *.o
```

ЗАКЛЮЧЕНИЕ

Для выполнения задания были использованы знания о работе с функциями стандартной библиотеки языка Си, динамической памятью, строками и структурами, полученные за семестр. Была написана и разделена на подзадачи собственная программа, а также проведено тестирование. Таким образом, в ходе написания курсовой работы были закреплены фундаментальные навыки программирования на языке Си.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы программирования на языках Си и C++ [Электронный ресурс URL: http://cplusplus.com

2. Керниган Б., Ритчи Д. Язык программирования СИ. СПб.: Издательство "Невский Диалект", 2001. 352 с.