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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка текстовых данных

Студент гр. 1304 _____ Макки К.Ю.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Макки К.Ю.

Группа 1304

Тема работы: обработка текстовых данных

Исходные данные:

Необходимо написать программу на языке Си, которая получает в качестве входных данных текст. Действия, которые необходимо совершить с введенным текстом выбирает пользователь.

Содержание пояснительной записки:

1. Аннотация
2. Введение
3. Теоретическая часть
4. Заключение
5. Список использованных источников
6. Приложение

Предполагаемый объем пояснительной записки:

Не менее 14 страниц.

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 20.12.2021

Дата защиты реферата: 22.12.2021

Студент _____ Макки К.Ю.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

С. 24. Ил. 4. Литература 3 назв. Прил. 1

Объектом исследования курсовой работы является алгоритм работы программы, реализующей различные операции для работы с текстом.

Целью данного курсового проекта является создание программы, позволяющей пользователю взаимодействовать с интерфейсом для выполнения различных действий над текстом.

В ходе подготовки к курсовой работе использовался: структурно-функциональный метод, позволяющий выделить из целостных систем структуры; анализ – метод, который позволяет разложить изучаемый материал и использовать то, что необходимо для данной работы.

В ходе работы был изложен краткий теоретический материал по данному вопросу, процесс решения задачи, приводится код на языке Си, а также результаты отладки.

СОДЕРЖАНИЕ

	Введение	5
1.	Ход работы	6
1.1.	Техническое задание	6
1.2.	Разбиение кода	7
2.	Описание и использование входных и промежуточных данных	8
2.1.	Запись и сохранение кода	8
2.2.	Обработка текста	9
3.	Тестирование	10
	Заключение	13
	Список использованных источников	14
	Приложение А. Алгоритм программы	15

ВВЕДЕНИЕ

Цель курсовой работы – разработка алгоритма, позволяющего пользователю совершать операции над текстом.

Задачи:

1. Изучение и анализ теоретических материалов: структур, функций стандартной библиотеки
2. Разработка программного кода
3. Сборка программы
4. Тестирование приложения и отладка

1. ХОД РАБОТЫ

1.1. Техническое задание

Вариант 5

- Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв, цифр и символ ‘/’. 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. Вывести все предложения в которых есть даты в формате “DD/MM/YYYY”, даты которые еще не наступили надо выделить красным цветом.
2. Удалить все предложения в которых каждое слово содержит нечетное количество букв.
3. Преобразовать предложения так, чтобы перед каждым словом стояло количество букв в нем. Например, для предложения “57 ab:e r4” результатом будет “57 3ab:e 1r4”.
4. Отсортировать предложения по возрастанию длины последнего слова.

- 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.

1.1. Разбиение кода

Разбиение кода на подзадачи можно сравнить с разбиением на функции. Одна функция, чаще всего, выполняет одну подзадачу.

1. Чтение текста из потока ввода в динамический массив структур
2. Печать текста в поток вывода
3. Написание интерфейса для работы с пользователем
4. Освобождение выделенной памяти
5. Удаление повторяющихся предложений в тексте
6. Вывод предложений, в которых есть даты в формате DD/MM/YYYY, даты, которые не наступили надо выделить красным цветом.
7. Удаление предложений, в которых каждое слово содержит нечетное количество букв.
8. Преобразовать предложения так, чтобы перед каждым словом стояло количество букв в нем. Например, для предложения “57 ab:e r4” результатом будет “57 3ab:e 1r4”.
9. Отсортировать предложения по возрастанию длины последнего слова.

ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ВХОДНЫХ И ПРОМЕЖУТОЧНЫХ ДАННЫХ

Запись и хранение кода

Для хранения текста используется динамическая память.

Для более удобной работы с текстом необходимо реализовать три структуры: Text, Sentence, Word. Это поможет обращаться к тем или иным частям текста для реализации различных функций.

```
struct Word{
        char * begin; - массив символов слова
        int size; - размер слова
        int alph; - количество(букв) в слове
        char end; - указатель на последний символ в слове
};
struct Sentence{
        char *str; - массив слов в предложении
        int size; - размер строк
        struct Word* words; - массив структур
        int wordsCount; - количество слов в предложении
};
struct Text{
        struct Sentence** sents; - массив структур
        int size; -
        int n; - количество предложений
};
```

Функция readSentence() – в ней создается временный буфер, в который помещается текст в качестве одной строки. Удаление пробелов в начале строк. В структуру sentence передаются разделенные строки.

Функция splitWords() – разделяет строку на слова

Функция readText() – собирает все предложения в структуру Text. В этой функции вызывается функция delete_sents, которая удаляет повторяющиеся строки.

Обработка текста

Обработка текста выполняется четырьмя основными функциями.

Функция date_check() – проверяет если введенный набор символов является датой.

Функция print_date() – проверяет если введенная дата относится к будущему, если дата ещё не наступила выводит красным цветом.

Функция odd_sents() – удаляет предложение, которое содержит нечетное количество букв.

Функция num_word() – преобразует предложения так, чтобы перед каждым словом стояло количество букв в нем.

Функция sortLast() – сортирует предложения по возрастанию длины слова.

ТЕСТИРОВАНИЕ

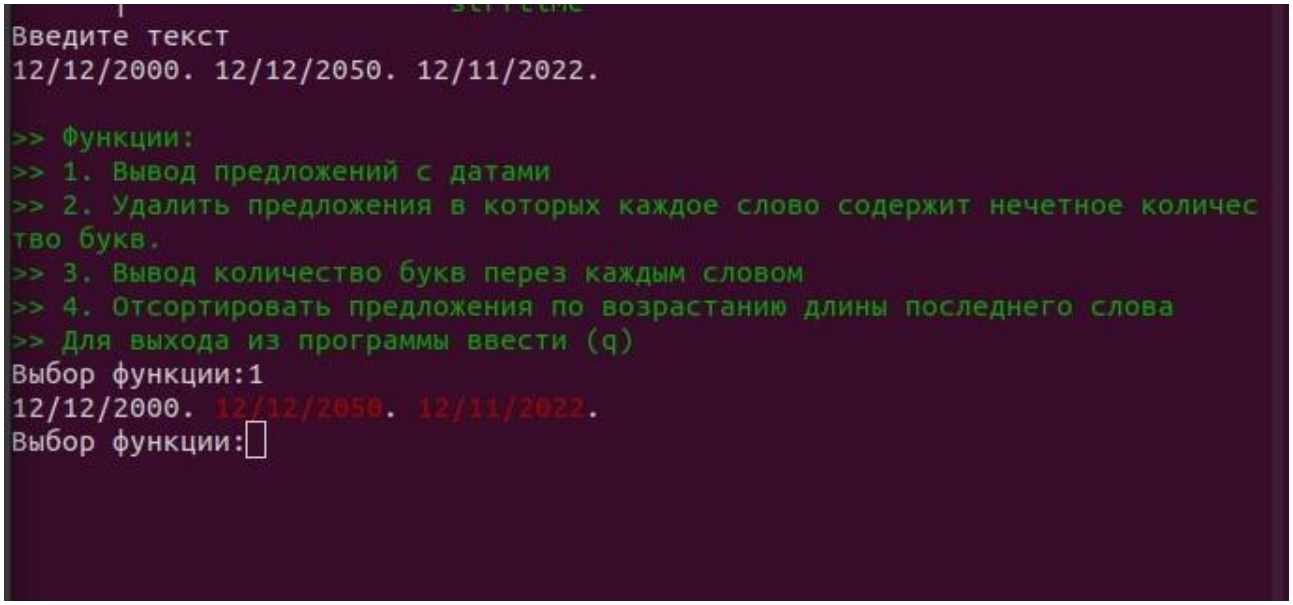

Рисунок 1 - Тестирование первой функции

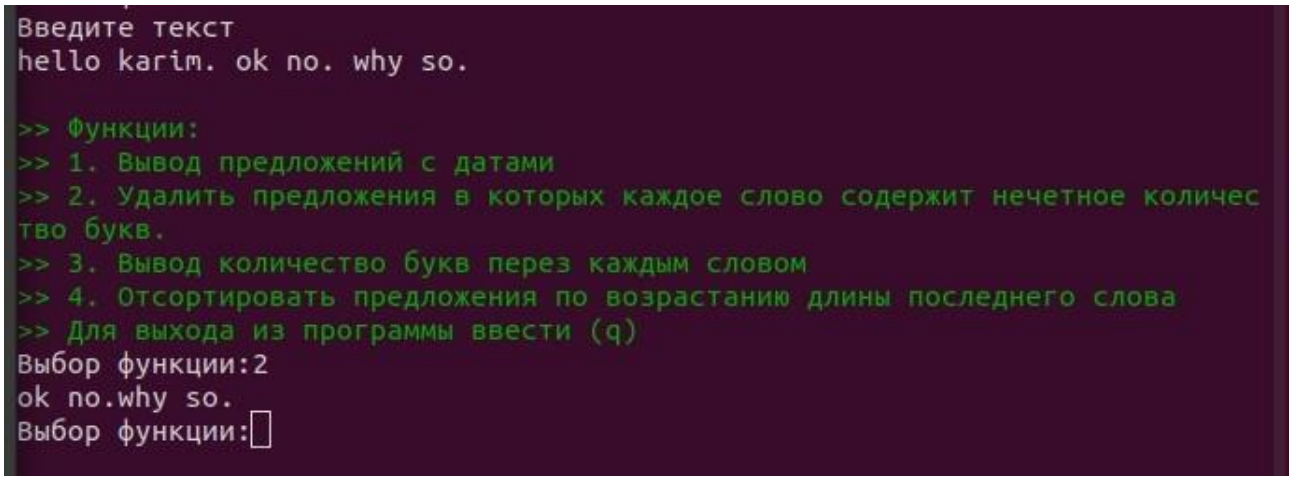

Рисунок 2 - Тестирование второй функции

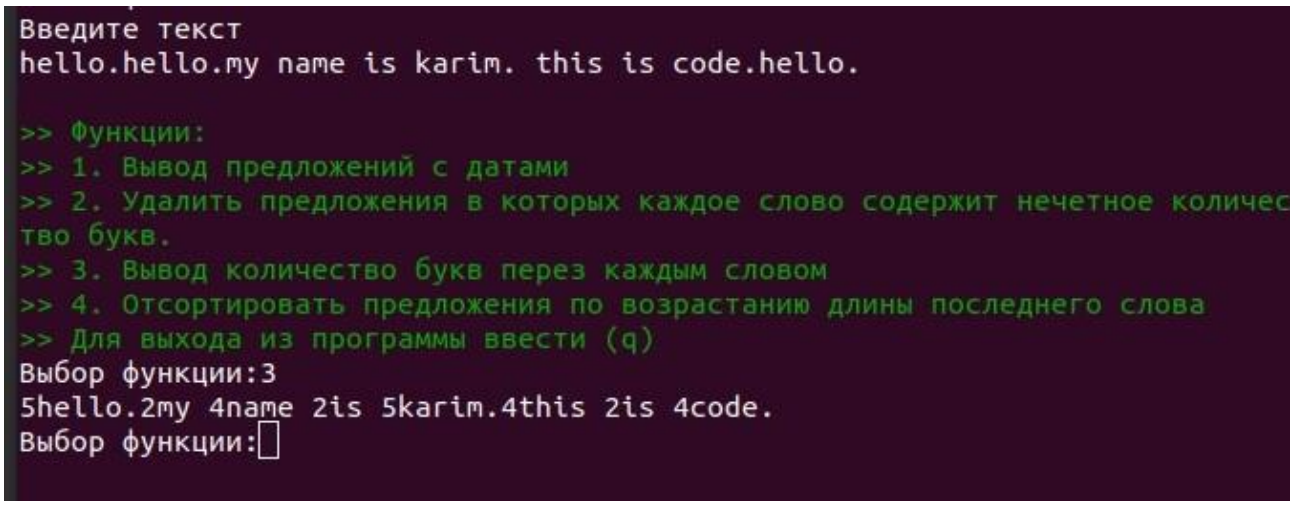

Рисунок 3 - Тестирование третьей функции

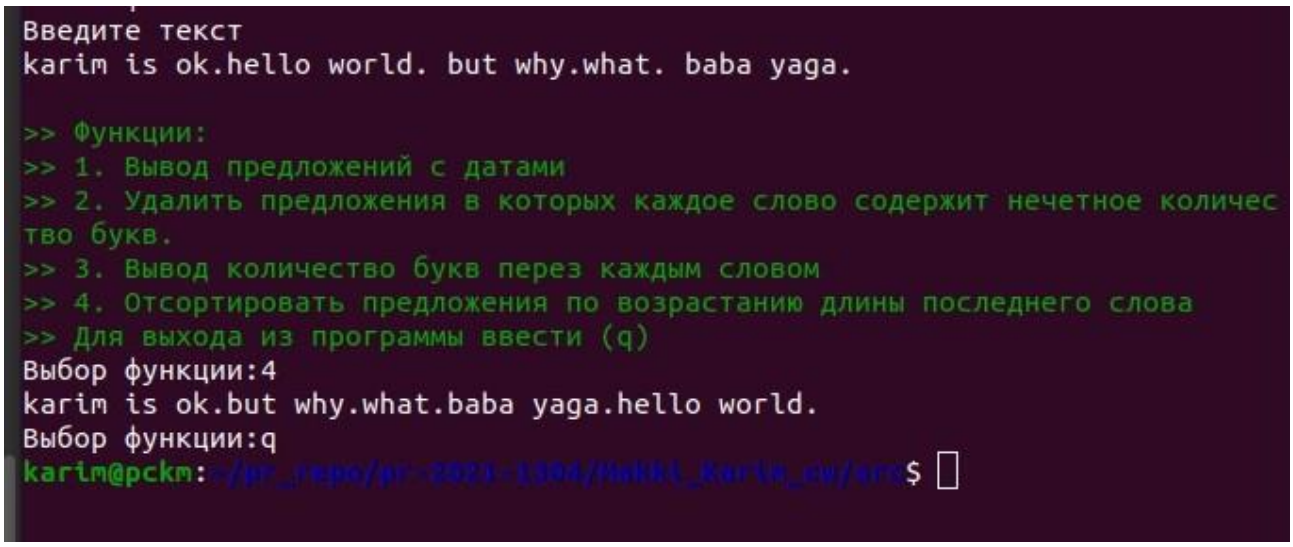

Рисунок 4 - Тестирование четвертой функции

ЗАКЛЮЧЕНИЕ

В результате курсовой работы была создана программа на языке Си, позволяющая пользователю взаимодействовать с интерфейсом для выполнения различных действий над текстом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Герберт Шилдт C: полное руководство, классическое издание = C: The Complete Reference, 4th Edition. — М.: «Вильямс», 2010. — С. 704.

2. Язык программирования C: Лекции и упражнения = C Primer Plus. — 1-е изд. — М.: Вильямс, 2006. — С. 960.

3. Язык программирования Си для «чайников» = C For Dummies. — М.: Диалектика, 2006. — С. 352.

ПРИЛОЖЕНИЕ А

АЛГОРИТМ ПРОГРАММЫ

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <ctype.h>
#include <time.h>
#define MEM_STEP 5


struct Word{
	char * begin;
	int size;
	int alph;
	char end;
};

struct Sentence{
	char *str;
	int size;
	struct Word* words;
	int wordsCount;
};

struct Text{
	struct Sentence** sents;
	int size;
	int n;
};
//*********************************************************************
*******
int splitWords(struct Sentence* sent){

	int size = MEM_STEP;
	struct Word* buf = malloc(size*sizeof(struct Word));

	int j = 0;
	int lenCount = 0;
```

```
	int inWord = 0;
	char *temp_buf;
	int alphaCount;

	for (int i = 0; i < sent->size; i++){

		if (inWord == 0 && (isspace(sent->str[i]) || sent->str[i] ==
',') == 0){

			buf[j].begin = &sent->str[i];
			inWord = 1;
			lenCount = 1;
			alphaCount = 0;
			if (isalpha(sent->str[i])){
				alphaCount++;
			}

		} else if (inWord == 1 && (isspace(sent->str[i]) || sent-
>str[i] == ',' || sent->str[i] == '.')){
			buf[j].end = sent->str[i];
			buf[j].alph = alphaCount;
			buf[j++].size = lenCount;

			inWord = 0;
			temp_buf = malloc((lenCount+1)*sizeof(char));
			temp_buf = memcpy(temp_buf, buf[j-1].begin,
lenCount*sizeof(char));
			temp_buf[lenCount] = '\0';
			buf[j-1].begin = temp_buf;



			if (j == size) {
				size += MEM_STEP;
				buf = realloc(buf, size*sizeof(struct Word));
			}

		} else {
```

```
                lenCount++;
                if (isalpha(sent->str[i])){
                        alphaCount++;
                }
        }

    }
    sent->words = buf;
    return j;

}


void delete_sents(struct Sentence **sents, int *n){

    struct Sentence *sent_1, *sent_2;

    int i = 0;
    int j;

    while (i < *n-1){

        sent_1 = sents[i];
        j = i+1;
        while (j < *n){

                sent_2 = sents[j];

                if (strcasecmp(sent_1->str, sent_2->str) == 0){
                        free(sent_2->str);
                        free(sent_2);
                        memmove(&sents[j], &sents[j+1], (*n-j-
1)*sizeof(struct Sentence*));
                        *n -= 1;
                } else {
                        j++;
                }
        }
```

```
            i++;
            j = i + 1;
      }
}
```

```
struct Sentence* readSentence();

struct Text readText(){
      printf("Введите текст\n");
      int size=MEM_STEP;
      struct Sentence** text = malloc(size*sizeof(struct Sentence*));
      int n=0;
      struct Sentence* temp;
      int nlcount=0;
      do{
            temp = readSentence();


            if(temp->str[0]=='\n'){
                  nlcount++;
                  free(temp->str);
                  free(temp);
            }else{
                  nlcount=0;
                  text[n]=temp;
                  n++;

                  if (n == size){
                        size += MEM_STEP;
                        text = realloc(text, size*sizeof(struct
Sentence*));
                  }
            }
      }while(nlcount<2);
```

```
	delete_sents(text, &n);

	struct Text txt;
	txt.size = size;
	txt.sents = text;
	txt.n = n;


	return txt;
}


struct Sentence* readSentence(){
	int size = 5;
	char *buf = malloc(size*sizeof(char));
	char temp;
	int n = 0;
	do{
		if(n>=size-2){
			char *t = realloc(buf,size+MEM_STEP);
			if (!t){
				printf("Error:");
				return NULL;
			}
			size += MEM_STEP;
			buf = t;
		}

		temp=getchar();
		buf[n]=temp;
		n++;

	}while(temp!='\n' && temp!='.' && temp!='!');
	buf[n]='\0';
	struct Sentence *sentence = malloc(sizeof(struct Sentence));

	int count = 0;
```

```
	for (int i = 0; buf[i] == ' '; i++){
		count++;
	}
	memmove(buf, buf+count, strlen(buf)-count+1);


	sentence->str = buf;
	sentence->size = size;


	sentence->wordsCount = splitWords(sentence);


	return sentence;
}

void free_text(struct Text* text){
	for(int i =0; i < text->n; i++){
		for(int j = 0 ;j < text->sents[i]->wordsCount;j++){
			free(text->sents[i]->words[j].begin);
		}
		free(text->sents[i]->words);
		free(text->sents[i]);
	}
	free(text->sents);
}
//*****************************************************************
*******
void sentProc(struct Sentence* sent, int k, int m){
	int len = strlen(sent->str);
	char *cur_str = sent->str;
	memmove(cur_str+k, cur_str+k+m, len-m-k+1);
}

//4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4

int cmp(const void* a, const void* b){
```

```
     struct Sentence **first_sent = (struct Sentence **) a;
     struct Sentence **second_sent = (struct Sentence **) b;

     if ((*first_sent)->words[(*first_sent)->wordsCount-1].size >
(*second_sent)->words[(*second_sent)->wordsCount-1].size){
          return 1;
     } else if ((*first_sent)->words[(*first_sent)->wordsCount-1].size <
(*second_sent)->words[(*second_sent)->wordsCount-1].size){
          return -1;
     }
     return 0;

}

void sortLast(struct Text* txt){

     qsort(txt->sents, txt->n, sizeof(struct Sentence*), cmp);

}

//4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
// 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
void num_word(struct Text* text){
     for (int i=0; i<text->n; i++){
          for (int j = 0; j < text->sents[i]->wordsCount; j++){
               if (text->sents[i]->words[j].alph != 0){
                    printf("%d%s%c", text->sents[i]->words[j].alph,
text->sents[i]->words[j].begin, text->sents[i]->words[j].end);
               } else{
                    printf("%s%c", text->sents[i]->words[j].begin,
text->sents[i]->words[j].end);
               }
          }
     }
     printf("\n");
}
```

```
// 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3

// 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
void odd_sents(struct Text* text){
    struct Sentence** sp;
    struct Word* wp;

    for (sp = text->sents+text->n-1; sp >= text->sents; sp--){
        for (wp = (*sp)->words; wp < (*sp)->words+(*sp)->wordsCount;
wp++)
            if (wp->alph % 2 == 0) break;
        if (wp < (*sp)->words+(*sp)->wordsCount) continue;
        for (wp = (*sp)->words; wp < (*sp)->words+(*sp)->wordsCount;
wp++)
            free(wp->begin);
        free((*sp)->words);
        free((*sp)->str);
        free((*sp));
        text->n--;
        memmove(sp, sp+1, (text->n+text->sents-sp)*sizeof(struct
Sentence*));
    }
}
// 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
// 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
int date_check(struct Word* word_p){
    if (word_p->size != 10) return 0;

    char* ch= word_p->begin;
    if ( ! (isdigit(ch[0]) && isdigit(ch[1])&&(ch[2]=='/') &&
            isdigit(ch[3]) && isdigit(ch[4])&&(ch[5]=='/') &&
            isdigit(ch[6]) && isdigit(ch[7])&&isdigit(ch[8]) &&
            isdigit(ch[9]) && (ch[10]=='\0')))
    {
```

```
            return 0;
      }
      struct tm* datetime = malloc(sizeof(struct tm));
      long int status= strptime(ch, "%d/%m/%Y", datetime);
      free(datetime);

      if ((char*)status == NULL) {
            return 0;
      }

      return 1;
}



void print_date(struct Text* text){
      char* ch;
      int flag;

      struct tm tm = *localtime(&(time_t){time(NULL)});
    time_t current_time_in_second= mktime(&tm);
      for (struct Sentence** sent_p = text->sents;
            sent_p < text->sents+text->n;
            sent_p++){
            int is_sent_have_date = 0;
            for (struct Word* word_p = (*sent_p)->words;
                  word_p < (*sent_p)->words+(*sent_p)->wordsCount;
                  word_p++)
            {
                  is_sent_have_date = date_check(word_p);
                  if(is_sent_have_date != 0) break;
            }
            if(is_sent_have_date == 0) continue;
            for(struct Word* word_p = (*sent_p)->words;
                  word_p < (*sent_p)->words+(*sent_p)->wordsCount;
                  word_p++)
                  {     int is_date = date_check(word_p);
                        if(is_date == 0){
```

```
                        printf("%s%c", word_p->begin,word_p->end);
                    }else{
                        struct tm* datetime = malloc(sizeof(struct
tm));
                        strptime(ch, "%d/%m/%Y", datetime);
                        time_t date_in_seconds= mktime(datetime);
                        free(datetime);
                        if (date_in_seconds > current_time_in_second)
{
                            printf("\033[0;31m");
                            printf("%s", word_p->begin);
                            printf("\033[0m");
                        } else {
                            printf("%s", word_p->begin);
                        }
                        printf("%c ",word_p->end);
                    }
                }
                //printf("\n");
    }
    printf("\n");
}
// 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
int main(){

    struct Text text = readText();


    char mode = -1;
    printf("\033[0;32m");
    printf(">> Функции:\n");
    printf(">> 1. Вывод предложений с датами\n");
    printf(">> 2. Удалить предложения в которых каждое слово содержит
нечетное количество букв. \n");
    printf(">> 3. Вывод количество букв перез каждым словом\n");
    printf(">> 4. Отсортировать предложения по возрастанию длины
последнего слова\n");
```

```
	printf(">> Для выхода из программы ввести (q)\n");
	printf("\033[0m");
	char trash;

	while(mode != 0) {
		printf("Выбор функции:");
		scanf("%c", &mode);
		scanf("%c", &trash);
		switch(mode){
			case '1':
				print_date(&text);
				break;
			case '2':
				odd_sents(&text);
				for (int i = 0; i < text.n; i++){
					printf("%s", text.sents[i]->str);
				}
				printf("\n");
				break;
			case '3':
				num_word(&text);
				break;
			case '4':
				sortLast(&text);
				for (int i = 0; i < text.n; i++){
					printf("%s", text.sents[i]->str);
				}
				printf("\n");
				break;
			case 'q':
				free_text(&text);
				return 0;
			default:
				printf("Повторите попытку :)\n");

		}
	}
}
```