МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Факультет КТИ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Указатели и массивы.

Студент гр. 1303 _____ Чубан Д.В.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2021

Цель работы.

Изучить поразрядные операции, указатели, связь указателей и массивов, передачу указателей в функцию, динамическую память, двумерные массивы, символьные массивы. После успешного изучения вышеперечисленного материала создать программу на языке Си, которая будет обрабатывать вводимую информацию по алгоритму и выводить результат согласно условию задания.

Задание.

Напишите программу, которая форматирует некоторый текст и выводит результат на консоль.

На вход программе подается текст, который заканчивается предложением "Dragon flew away!".

Предложение (кроме последнего) может заканчиваться на:

. (точка)

; (точка с запятой)

? (вопросительный знак)

Программа должна изменить и вывести текст следующим образом:

- Каждое предложение должно начинаться с новой строки.
- Табуляция в начале предложения должна быть удалена.
- Все предложения, в которых есть цифры внутри слов, должны быть удалены (это не касается слов, которые начинаются/заканчиваются цифрами). Если слово начинается с цифры, но имеет и цифру в середине, удалять его все равно требуется (4a4a).
- Текст должен заканчиваться фразой "Количество предложений до n и количество предложений после m", где n - количество предложений в изначальном тексте (**без учета** терминального предложения "Dragon flew away!") и m - количество предложений в отформатированном тексте (без учета предложения про количество из данного пункта).

* Порядок предложений не должен меняться

* Статически выделять память под текст нельзя

* Пробел между предложениями является разделителем, а не частью какого-то предложения

Выполнение работы.

Все функции находятся в файле lb3_finale.*c.*

В основной функции *main()* создается объект типа txt, которому присваивается результат функции *read_text(),* после чего с помощью цикла *for* и функции *memmove()* убираются пробелы и/или символы табуляции *(‘\t’*). После этого полученный результат построчно выводится в консоль и программа выводит предложение "Количество предложений до n и количество предложений после m", где n - количество предложений в изначальном тексте и m - количество предложений в отформатированном тексте.

Структура *txt* имеет 3 поля:

- *char** *sentences* - хранит массив указателей на массивы символов (текст).
- *Int total_sent_size* - количество введенных предложений.
- *Int after_sent_size* - количество предложений после обработки текста.

Функция *read_text()* возвращает объект типа *txt.*

В функции создаются три переменные типа *int:*

- *Overall_sent* - общее количество предложений в тексте, считая удаленные.
- *Size* - количество байт, которое выделяется под вводимый текст.
- *N* - счетчик добавляемых предложений в текст (впоследствии это будет количество предложений в тексте после обработки).

Создается переменная *char** *res_text,* под которую сразу выделяется память с помощью функции *malloc().*

Далее запускается цикл *while* с проверкой текущего считываемого предложения на равность терминальному предложению.

В данном цикле сначала производится проверка на необходимость довыделения памяти под текст, после чего с помощью функции read_sentence() считывается

новое предложение и оно передается функции *sentence_check(),* которая возвращает 0 или 1.
Если функция вернула 0, то предложение добавляется в массив с текстом и переменные *n* и *overall_sent* увеличиваются на 1. В случае, если функция вернула 1, то предложение не добавляется в массив, а *overall_sent* увеличивается на 1.
После завершения цикла *while* создается объект *result* типа *txt,* а его полям присваиваются следующие значения:

- *Result.sentences = res_text*
- *Result.total_sent_size = overall_sent*
- *Result.after_sent_size = n*

После чего функция завершается, возвращая как результат *result.*

Функция *read_sentence()* посимвольно считывает предложения, пока не встретит один из символов признака конца строки (‘.’, ‘;’,’?’,’!’), после чего добавляет в конец строки символ конца строки ‘\0’. Динамическое выделение памяти под считываемую строку происходит так же, как и в функции *read_text().*

Функция *sentence_check()* получает на вход ссылку на первый элемент строки, после чего с помощью функции *strtok()* делит предложение на отдельные слова, которые проверяются на наличие цифр внутри слов. Программа возвращает 1 или 0, в зависимости от того, нужно ли удалять предложение.

Тестирование.

№ теста	Входные данные	Выходные данные
1	qwerty; test111test; qwerty. Dragon flew away!	qwerty; qwerty. Dragon flew away! Количество предложений до 3 и количество предложений после 2
2	qwerty; 111testtest111; qwerty. Dragon flew away!	qwerty; 111testtest111; qwerty. Dragon flew away! Количество предложений до 3 и количество предложений после 3
3	qwerty; 111test111test; qwerty. Dragon flew away!	qwerty; qwerty. Dragon flew away! Количество предложений до 3 и количество предложений после 2
4	qwerty; test111test111; qwerty. Dragon flew away!	qwerty; qwerty. Dragon flew away! Количество предложений до 3 и количество предложений после 2

Выводы.

Работа поразрядных операций, указателей, связь указателей и массивов, передача указателей в функцию, динамическая память, двумерные массивы, символьные массивы изучены. После успешного изучения вышеперечисленного материала создана программа на языке Си, которая обрабатывала вводимую информацию по алгоритму и выводила результат согласно условию задания.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lb3_finale.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MEM_STEP 100

struct txt{
    char** sentences;
    int total_sent_size;
    int after_sent_size;
};

char* read_sentence(){

    int size = MEM_STEP;
    char *sentence = malloc(size*sizeof(char));
    char temp;
    int n = 0;
    do{
        if (n >= size-2){
```

```
            char* t = realloc(sentence, (size + MEM_STEP)*sizeof(char*));
            if(!t){
            }else{
                size += MEM_STEP;
                sentence = t;
            }
        }
        temp = getchar();
        sentence[n] = temp;
        n++;
    } while (temp != '.' && temp != ';' && temp != '?' && temp != '!');
    sentence[n] = '\0';
    return sentence;
}


int sentence_check(char* temp){
    int digit_flag = 0;
    char* checking_ = malloc(strlen(temp)*sizeof(char));
    strcpy(checking_, temp);
    char* word_token = strtok(checking_, " ,\t");
    while (word_token != NULL){
        int checkflag = 0;
        int for_exit = 0;
        for (int i = 0; i < strlen(word_token); i++){
            if (isalpha(word_token[i]) && !checkflag){
                checkflag = 1;
            }else{
                if(isdigit(word_token[i]) && !checkflag){
                    checkflag = 0;
                }
                if (isdigit(word_token[i]) && checkflag){
                    for_exit = 1;
                }
```

```
                if (isalpha(word_token[i]) && checkflag){
                    checkflag = 1;
                }
                if(isalpha(word_token[i]) && for_exit){
                    digit_flag = 1;
                    return digit_flag;
                }

            }
        }
        word_token = strtok(NULL, " ,\t");
    }
    return digit_flag;

};
struct txt read_text(){
    int overall_sent =0;
    int size = MEM_STEP;
    char** res_text = malloc(size*sizeof(char*));
    int n = 0;
    char* temp;
    do{
        if (n >= size-2){
            char** t = realloc(res_text, (size +
MEM_STEP)*sizeof(char**));
            if(!t){}
            else{
                size += MEM_STEP;
                res_text = t;
            }
        };
        temp = read_sentence();
        int digit_flag = sentence_check(temp);
        if (digit_flag != 0){
```

```
            overall_sent++;
        }else{
            res_text[n] = temp;
            n++;
            overall_sent++;
        }
    }while (strcmp(temp, "Dragon flew away!") != 0 && strcmp(temp, "
Dragon flew away!") != 0 && strcmp(temp, "\tDragon flew away!") != 0 &&
strcmp(temp, "\nDragon flew away!") != 0);
    struct txt result;
    result.sentences = res_text;
    result.total_sent_size = overall_sent;
    result.after_sent_size = n;
    return result;
}

int main(){
    struct txt test = read_text();
    for (int i = 0; i < test.after_sent_size; i++){
        int curr = 0;
        char* curr_char;
        for(curr_char = test.sentences[i]; *curr_char == ' ' ||
*curr_char == '\t'; curr_char++ ){
            curr++;
        }
        memmove(test.sentences[i], curr_char, strlen(test.sentences[i]));
    };
    for (int i = 0; i < test.after_sent_size; i++){
        puts(test.sentences[i]);
    };
    printf("Количество предложений до %d и количество предложений
после %d", test.total_sent_size-1, test.after_sent_size-1);
    return 0;
}
```