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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Использование указателей

Студент гр. 8303 _____ Гришин К. И.

Преподаватель _____ Чайка К. В.

Санкт-Петербург
2018

Цель работы

Изучить расположение объектов в памяти. Научиться использовать динамическую память для осуществления ввода с заранее неивестным размером вводимых данных.

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

Ход работы

Подключаются стандартные библиотеки:

- *«stdio.h»* — для ввода и вывода.
- *«stdlib.h»* — для работы с динамической памятью.
- *«string.h»* — для использования функций обработки строк.

Подключаем идентификатор *«STOP»*, который будет использоваться для обозначения терминальной строки ввода.

Объявляются функции *«processSentence»*, которая очищает предложение, если там есть слово, содержащее внутри цифры, и *«processWord»*, которая определяет содержатся ли внутри слова цифры.

Объявляются переменные *«text»* и *«memText»*, в первой хранятся адреса введенных предложений, вторая отражает текущий размер массива адресов.

Объявляются перменные, одна выступает в роли буффера, в котором хранится предыдущее предложение, другая содержит в себе очередной введенный символ.

Объявляется переменная для хранения размера текста (количество предложений) и запускается цикл считывания предложений, который будет считывать предложения ,пока не встретится *«STOP»* — стоп фраза.

Цикл очищает буффер и увеличивает память длч хранения адресов строк, если это требуется.

Объявляется перменная для хранения размера вводимого предложения и запускается цикл чтения предложения, пока не встретится знак конца предложения, либо пока введенная строка не станет равна терминальному предложению. В этом цикле пропускаются первые символы, если они пустые (равны пробелу или табуляции), остальные введенные символы записываются в динамически расширяющуюся строку. В буффер копируется значение введенной строки (буфер нужен для того, чтобы иметь возможность остановить цикл при его переходе на новую иттерацию).

После того, как текст был введен, запускается цикл вывода. Цикл сначала обрабатывает строку функцией «processSentence», а затем выводит ее, если она

не пустая, а также инкрементирует значение нового количества предложений *«newSizeText»*. После вывода предложения, память занимаемая строкой освобождается.

После того, как был выведен весь отформатированный текст, освобождается память выделенная под адреса с предложениями.

Выводится соотношение количества предложений до форматирования и после.

Выводы.

В ходе выполнения работы была написана программа на языке С, которая обрабатывает введный пользователем текст нефиксированной длины. Были изучены методы динамического выделения памяти и обработки строк, путем использования адресов.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

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

#define STOP "Dragon flew away!"  // Стоп-фраза

void processSentence(char** sentence);  // Обработка предложения
int processWord(char* word);            // Обработка слова

int main(){
    int memText = 0;
    char** text = (char**)malloc(sizeof(char*));
    char* buffer = (char*)malloc(2*sizeof(char));
    char inpC;

            // Ввод текста
    int sizeText;
    for(sizeText = 0; strcmp(buffer, STOP); sizeText++){  // Ввод текста, до тех пор, пока введенное
предложение не станет равно стоп-фразе
        free(buffer);   // Очистка буфера

               // Увеличение памяти, в которой хранятся адреса предложений,
               // по мере ее заполнения
        if (sizeText % 5 == 0){
            memText += 5;
            text = (char**)realloc(text, memText*sizeof(char*));
        }

        do
            inpC = fgetc(stdin);
        while(inpC == ' ' || inpC == '\t' || inpC == '\n');


               // Ввод предложения
        int memString = 10;
        text[sizeText] = (char*)malloc((memString+1)*sizeof(char));
        *(text[sizeText]) = inpC;
        int sizeString;
        for(sizeString = 1; inpC != '.' && inpC != ';' && inpC != '?'; sizeString++){
               // Увеличение памяти, в которой хранится строка,
               // по мере ее считывания
            if (sizeString % 10 == 0){
                memString += 10;
                text[sizeText] = (char*)realloc(text[sizeText], (memString+1)*sizeof(char));
            }

               // Пропуск пустых символов
            inpC = fgetc(stdin);
            if (inpC == '\n') inpC = ' ';
               // Считывание очередного символа строки
            text[sizeText][sizeString] = inpC;
            text[sizeText][sizeString+1] = 0;
               // Выход из цикла, если предложение приняло вид стоп-фразы
            if (strcmp(text[sizeText], STOP) == 0) break;
        }
               // Выделение памяти для буфера, размером с текущую строку
        buffer = (char*)malloc((strlen(text[sizeText])+1)*sizeof(char));
               // Копирование текущей строки в буфер
        strcpy(buffer, text[sizeText]);
    }

            // Форматирование и вывод
    int newSizeText = 0;
    for(int i = 0; i < sizeText; i++){
        processSentence(&(text[i]));
        if (*text[i] == 0) { free(text[i]); continue;}
        printf("%s\n", text[i]);
        free(text[i]);
        newSizeText++;
    }

    free(text);       // Очистка памяти выделенной под адреса с предложениями
    printf("Количество предложений до %d и количество предложений после %d\n", sizeText-1,
newSizeText-1);
```

```
    return 0;
}

int processWord(char* word){
    int identifier = 0;
    int len = strlen(word);
    int leftPointer;  // Индекс первой буквы справа
    for(leftPointer = 0; word[leftPointer] >= '0' && word[leftPointer] <= '9' && leftPointer != len
- 1; leftPointer++);
    int rightPointer; // Индекс первой буквы слева
    for(rightPointer = len - 1; word[rightPointer] >= '0' && word[rightPointer] <= '9' &&
rightPointer != 0; rightPointer--);

    if (rightPointer - leftPointer > 1)
        for(int i = leftPointer; i <= rightPointer; i++)
            if (word[i] >= '0' && word[i] <= '9')
                identifier = 1;
    if (identifier) return 1;
    else return 0;
}


void processSentence(char** sentence){
    char buffer[strlen(*sentence)+1];
    strcpy(buffer, *sentence);
    int identifier = 0;
    char* word = strtok(buffer, " ,.;?");
    for(int i = 0; word; i++){
        identifier += processWord(word);
        word = strtok(NULL, " ,.;?");
    }
    if (identifier) **sentence = 0;
}
```