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: Указатели и массивы

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Целью работы является освоение работы с указателями и динамической памятью.

Задание.

Вариант 1

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 цифра 7 (в любом месте, в том числе внутри слова), должны быть удалены.
- 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**

Основные теоретические положения.

* – оператор разыменования;

& – оператор взятия адреса;

Функции заголовочного файла *stdlib.h* :

void * *malloc* (size_t *sizemem*) - выделяет блок памяти, размером *sizemem* байт, и возвращает указатель на начало блока. Содержание выделенного блока памяти не инициализируется, оно остается с неопределенными значениями.

void * *calloc* (size_t *number*, size_t *size*) - выделяет блок памяти для массива размером — *number* элементов, каждый из которых занимает *size* байт, и инициализирует все свои биты в нулями. В результате выделяется блок памяти размером *number* * *size* байт, причём весь блок заполнен нулями.

void * *realloc* (void * *ptrmem*, size_t *size*) - выполняет перераспределение блоков памяти. Размер блока памяти, на который ссылается параметр *ptrmem* изменяется на *size* байтов. Блок памяти может уменьшаться или увеличиваться в размере.

void free (void * ptrmem) - функция освобождает место в памяти. Блок памяти, ранее выделенный с помощью вызова *malloc, calloc* или *realloc* освобождается. То есть освобожденная память может дальше использоваться программами или ОС.

Функции заголовочного файла *string.h:*

int strcmp (const char * string1, const char * string2) - функция сравнивает символы двух строк, *string1* и *string2*. Начиная с первых символов функция сравнивает поочередно каждую пару символов, и продолжается это до тех пор, пока не будут найдены различные символы или не будет достигнут конец строки.

size_t strlen(const char * string) - функция считает длину строки не считая нулевого символа.

Выполнение работы.

Макроопределения:

BASIC_LEN_SENT — начальная, базовая длина предложения.

BASIC_LEN_TEXT — начальное , базовое количество предложений в тексте .

Собственная функция char* *get_sent()*:

Описание :

Функция считывает предложение (по буквенно , пока не встретит «.», «;», «?», «!»), добавляет в конце нулевой символ и возвращает указатель на эту строку.

Переменные :

char **sent, *re_sent;* - указатель на строку , и временный указатель на строку , соответственно.

char *ch = 0;* - текущий считанный символ.

int *len_sent = 0;* - текущая длина строки(предложения) .

int *max_len_sent = BASIC_LEN_SENT;* - текущее максимальное допустимое количество символов в строке.

int *in_sent = 0;* - флаг нахождения внутри предложения.

Ход работы:

- Выделение начальной динамической памяти для предложения.
- Проверка на удачное выполнение предыдущего действия .
- Цикл считывания предложения , завершается по считыванию «.», «;», «?», «!» (символов конца предложения).
  - Считывание текущего символа .
  - Проверка на вхождение в предложение.
  - Проверка на наличие свободного места в строке (динамическом массиве) для добавления считанного символа . Если места нет , добавление места и проверка на успешность добавления места.
  - Добавление считанного символа в динамический массив.
  - Увеличение длины строки на 1.
- Добавление в конец строки нулевого символа.
- Возврат указателя на считанное предложение.

Собственная функция int *find_seven* (char **sent*):

Описание :

Функция определяет находится ли число «7» в строке на которую указывает *sent.* В случае нахождения возвращает 1, иначе 0.

Переменные :

int i — счетчик .

Ход работы:

- Цикл от 0 до длинны строки *sent*.
  - Проверка каждого символа строки sent на равенство с «7», в случае успеха конец функции, возврат 1.
- Функция не прервалась , значит «7» нет в строке *sent* , конец функции , возврат 0.

Главная функция int *main* ():

Описание :

Выполнение поставленной задачи .

Переменные :

char **text, **re_text ; - указатель на массив указателей на строки , и временный указатель , соответственно.

char *sent = ""; - считанное текущее предложение.

int len_text = 0,i,kol_sent_bef = 0; - текущее количество предложений записанных в память , счетчик , количество введенных предложений .

int max_len_text = BASIC_LEN_TEXT; - текущее максимальное допустимое количество предложений в массиве.

Ход работы:

- Выделение начальной динамической памяти для предложений.
- Проверка на удачное выполнение предыдущего действия .
- Цикл считывания текста, заканчивается при считывании терминального предложения "Dragon flew away!"
  - Считывания предложения собственной функцией *get_sent()*
  - Плюс 1 к количеству введенных предложений.

- Проверка предложения на наличие символа «7» , с помощью собственной функции *find_seven(sent)*. В случае нахождение предложение не запоминается .

  В противном случае выполняется проверка на наличие свободного места в массиве строк(динамическом массиве) для добавления новой строки . Если места нет , добавление места и проверка на успешность добавления места.
- Добавление нового считанного предложения в массив строк.
- Плюс 1 к количеству предложений записанных в память.

- Вывод сохраненных предложений .
- Вывод строки с количеством выведенных предложений и количество введенных предложений .
- Очистка памяти .
- Конец функции.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Dragon flew away!	Dragon flew away! Количество предложений до 0 и количество предложений после 0	Программа работает правильно
2.	Nu555llam auctor vehicula dui, quis lobortis nibh. Dragon flew away!	Nu555llam auctor vehicula dui, quis lobortis nibh. Dragon flew away! Количество предложений до 1 и количество предложений после 1	Программа работает правильно
3.	Nu555llam auctor vehicula dui, qu7is lobortis nibh. Dragon flew away!	Dragon flew away! Количество предложений до 1 и количество предложений после 0	Программа работает правильно

Выводы.

Были освоены работа с указателями и динамической памятью.

Разработана программа, выполняющая считывание с клавиатуры исходных данных и обработка и вывод их по заданны требованиям.

Для считывания использовалась функция *getchar()*. Для работы с динамической памятью функции *malloc*() , *calloc*(), *realloc*(), *free*(). Для работы со строками *strlen*() , *strcmp*(). Для вывода ошибок *fprintf*(stderr, «»). Для работы с указателями операторы * и &.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3_prog.c

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

#define BASIC_LEN_SENT 50
#define BASIC_LEN_TEXT 20

char* get_sent(){

   char *sent, *re_sent;
   char ch = 0;
   int len_sent = 0;
   int max_len_sent =  BASIC_LEN_SENT;
   int in_sent = 0;

   sent = calloc(BASIC_LEN_SENT,sizeof(char));// незабудь free

   if (sent == NULL){
      fprintf(stderr,"Ошибка выделения памяти для sent");
      exit(1);
   }

   while ((ch != '.')&&(ch != ';')&&(ch != '?')&&(ch != '!')){
      ch = getchar();
```

```
        if ((ch != ' ')&&(ch != '\n')&&(in_sent == 0)){in_sent = 1;}

        if ((ch != '\n')&&(in_sent == 1)) {
            if (len_sent  == max_len_sent - 1) {

                max_len_sent = max_len_sent + BASIC_LEN_SENT;
                re_sent = realloc(sent, max_len_sent*sizeof(char));

                if (re_sent == NULL) {
                    free(sent);
                    fprintf(stderr, "Ошибка перевыделения памяти для sent");
                    exit(1);
                } else {
                    sent = re_sent;
                }
            }

            sent[len_sent] = ch;
            len_sent ++;
        }

    }
    sent[len_sent] = '\0';

    return sent;
}

int find_seven(char *sent){
```

```
    for (int i = 0; i<strlen(sent);i++){
        if (sent[i] == '7') {
            return 1;
        }
    }
    return 0;
}


int main (){
    char **text, **re_text ;
    char *sent = "";
    int len_text = 0,i,kol_sent_bef = 0;
    int max_len_text = BASIC_LEN_TEXT;

    text = malloc(BASIC_LEN_TEXT*sizeof(char*));

    if (text == NULL){
        fprintf(stderr,"Ошибка выделения памяти для Text");
        exit(1);
    }

    while (strcmp(sent , "Dragon flew away!")){
        sent = get_sent();
        kol_sent_bef ++;

        if (!find_seven(sent)) {
            if (len_text == max_len_text) {
```

```
            max_len_text = max_len_text + BASIC_LEN_TEXT;
            re_text = realloc(text, max_len_text*sizeof(char*));

            if (re_text == NULL) {
               for (i = 0; i < len_text; i++) {
                  free(text[i]);
               }
               free(text);
               free(sent);
               fprintf(stderr, "Ошибка перевыделения памяти для Text");
               exit(1);
            } else {
               text = re_text;
            }
         }

         text[len_text] = sent;
         len_text++;
      }
   }

   for (int i = 0; i<len_text ; i++){
      printf("%s\n",text[i]);
   }
     printf("Количество предложений до %i и количество предложений после %i",kol_sent_bef-1,len_text-1);

   for (i = 0 ; i<len_text ;i++){
      free(text[i]);
```

```
    }
    free(text);


    return 0;


}
```