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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Программирование»
Тема: Обзор стандартной библиотеки

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

Изучить стандартную библиотеку языка программирования Си и научится использовать ее при написании программ.

Задание.

Вариант 1

Напишите программу, на вход которой подается текст на **английском** языке (длина текста не превышает **1000** символов) и слово **str** (длина слова не превышает **30** знаков). Слова в тексте разделены пробелами или точкой. Программа должна вывести строку "exists", если **str** в тексте есть и "doesn't exist" в противном случае.

Программа должна реализовать следующий алгоритм:

- разбить текст на слова, используя **функции стандартной библиотеки**
- отсортировать слова, используя алгоритм быстрой сортировки (см. **функции стандартной библиотеки**)
- определить, присутствует ли в тексте **str**, используя алгоритм двоичного поиска (для реализации алгоритма двоичного поиска используйте **функцию стандартной библиотеки)**
- вывести строку "exists", если **str** в тексте есть и "doesn't exist" в противном случае.

Основные теоретические положения.

При выполнении поставленной задачи были использованы следующие функции :

Из заголовочного файла **stdlib.h**:

void * **malloc**(size_t sizemem); - функция выделяет блок памяти, размером *sizemem* байт, и возвращает указатель на начало блока.

void * **realloc** (void * ptrmem, size_t size); - функция выполняет перераспределение блоков памяти. Размер блока памяти, на который ссылается параметр ptrmem изменяется на size байтов.

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.

void **qsort** (void * first, size_t number, size_t size, int (* comparator) (const void *, const void *)); - функция выполняет сортировку num элементов массива, на который ссылается указатель *first*. Для каждого элемента массива устанавливается размер в байтах, который передается через параметр *size*. Последний параметр функции *qsort* — указатель *comparator* на функцию сравнения, которая используется для определения порядка следования элементов в отсортированном массиве.

void * **bsearch** (const void * searchkey, const void * baseptr, size_t number, size_t size, int (* funccompar) (const void *, const void *)); - функция выполняет двоичный поиск в массиве. Алгоритм поиска запрашивает искомое значение, которое передается через параметр *searchkey,* в массиве, на который ссылается указатель *baseptr*. Количество элементов массива передается через параметр number, каждый элемент массива имеет размер *size* байт. В конце, функция возвращает указатель *void** на первое вхождение искомого значения. Для выполнения поиска, функция сравнивает элементы с искомым элементом, путем вызова функции через параметр **comparator,* указанного в качестве последнего аргумента.

Из заголовочного файла **string.h**:

size_t **strlen**(const char * string); - функция определяет длину Си-строки .

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

char* **strtok**(char * string, const char * delim); - Функция *strtok* выполняет поиск лексем в строке *string*. Последовательность вызовов этой функции разбивают строку *string* на лексемы, которые представляют собой последовательности символов, разделенных символами разделителями.

Выполнение работы.

Структуры:

Создана была *структура* ***Words*** ,которая содержит в себе *char*** ***word*** - указатель на массив слов, *int* ***cur_size*** - текущее количество слов и *int* ***max_size*** - максимальное возможное количество слов на данный момент.

Основная функция ***main()***:

Описание :

Основная функция программы,которая выполняет поставленную задачу.

Переменные :

*char** ***str —*** хранит вводимый текст .

c*har** ***find_word —*** хранит слово, которое нужно найти .

*char*** ***point*** — двойной указатель на найденное слово .

struct Words ***arr_words*** — структура слов.

*char** ***temp_p*** — временный указатель.

int ***i*** — счетчик.

Ход работы :

- Выделяем память для ***str , find_word , arr_words.word***.
- Считываем текст в ***str*** и слово в ***find_word***.
- С помощью функции ***strtok(str," .")*** разрезаем текст ***str*** на слова и запоминаем каждое слово в массив ***arr_words.word***, при необходимости перевыделяем память для ***arr_words.word.***
- С помощью функции ***qsort(arr_words.word, arr_words.cur_size, sizeof (char*), cmp)*** сортируем лексика-графически (используя *strcmp()*) список слов ***arr_words.word***. Функция компаратора ***cmp*** будет рассмотренно позже.
- С помощью функции бинарного поиска ***bsearch(&find_word, arr_words.word, arr_words.cur_size, sizeof (char*), cmp)*** , находим искомое слово ***find_word*** ,в отсортированном массиве ***arr_words.*** Функция компаратора ***cmp*** будет рассмотренно позже.
- Выводим «*exists»,* если слово было найдено или «*doesn't exist*», в противном случае . Не забываем очистить память с помощью функции ***free()***.

Функция компаратора ***cmp:***

Описание:

Функция компаратор для сравнения слов, основывается на использовании функции ***strcmp()*** и используется в функциях ***qsort()*** и ***bsearch()***.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Java is a general-purpose computer programming language that is . is	exists	Программа работает правильно
2.	Java a general-purpose computer programming language that . is	doesn't exist	Программа работает правильно

Выводы.

Была изучена стандартная библиотека языка программирования Си и использована при написании программы.

Разработана программа, выполняющая поставленную задачу, а именно считывание с клавиатуры исходного текста и поиск в нем вводимого слова . Для решения этой задачи были использованы следующие функции стандартной библиотеки : для разрезания текста на слова была использована

функция ***strtok()***, для сортировки слов была использована функция ***qsort()*** , а для поиска вводимого слова использовалась функция ***bsearch().***

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_1.c

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

struct Words{
    char **word;
    int cur_size;
    int max_size;
};

int cmp(const void* p1, const void* p2){
    char *word1 = *((char**)p1);
    char *word2 = *((char**)p2);

    return strcmp(word1,word2);
}

int main(){
    char *str = malloc(1001 * sizeof(char));
    char *find_word = malloc(31 * sizeof(char));
    char **point;
    struct Words arr_words =  {.cur_size = 0};
    char* temp_p;
    int i;
```

```
    arr_words.max_size = 50;
    arr_words.word = malloc(arr_words.max_size * sizeof(char*));

    fgets(str,1001,stdin);
    fgets(find_word,31,stdin);

    str[strlen(str)-1] = '\0';

    temp_p = strtok(str," .");
    for (i = 0; temp_p ; i++){
        if (arr_words.cur_size == arr_words.max_size-2){
            arr_words.max_size = arr_words.max_size + 30;
            arr_words.word = realloc(arr_words.word,arr_words.max_size *
sizeof(char*));
        }
        arr_words.word[i] = temp_p;
        arr_words.cur_size++;

        temp_p = strtok(NULL, " .");
    }

    qsort(arr_words.word,arr_words.cur_size,sizeof (char*),cmp);

    point = (char**)bsearch(&find_word,arr_words.word,arr_words.cur_size,sizeof
(char*),cmp);


    if (point == NULL){
        printf("doesn't exist");
```

```
    }else{
        printf("exists");
    }

    free(find_word);
    free(str);
    free(arr_words.word);

}
```