МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Программирование»

ТЕМА: ОБЗОР СТАНДАРТНОЙ БИБЛИОТЕКИ

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить стандартную библиотеку языка Си, освоить ее возможности, используя основные функции этой библиотеки в программном коде.

Задание.

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 exis*t" в противном случае.

Основные теоретические положения.

В заголовочном файле <*stdlib.h*> собраны объявления различных функций, частью из которых мы уже пользовались ранее.

- Функции для работы с динамической памятью
- Функции для преобразования строки в число
- Генерации псевдослучайных чисел
- Функции для управления процессом выполнения программы
- Функции для вычисления абсолютного значения и деления целых чисел
- Функции для сортировки и поиска

Для написания функции нахождения минимума в массиве элементов неизвестного типа указатель на функцию нужен, чтобы сравнивать элементы. Похожая логика используется во многих языках программирования, а функцию сравнения двух элементов обычно называют компаратор (англ *compare* - сравнивать). Компаратор работает по следующему принципу: если элементы равны, результатом сравнения будет 0, если первый больше - результат 1 или любое положительное число, иначе -1 или любое отрицательное.

Как мы уже говорили, в *stdlib.h* есть функции для сортировки и поиска в массиве любого типа. Давайте рассмотрим как такое возможно на примере функции *qsort*:

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

Функция принимает указатель на начальный элемент массива, количество элементов и размер одного элемента, а также указатель на функцию для сравнения двух элементов.

Так как тип элементов может быть любым, то и указатель на первый элемент массива имеет тип *void*. Это позволяет, зная адрес первого элемента и размер каждого элемента вычислить адрес любого элемента массива в памяти и обратиться к нему. Остается только сравнить 2 элемента имея 2 указателя на них. Это выполняет функция *compar*, указатель на которую передается функции *qsort* в качестве одного из параметров.

Функция *compar* принимает 2 указателя типа *void*, но в своей реализации может привести их к конкретному типу (так как её реализация остается за программистом, он точно знает элементы какого типа он сортирует) и сравнивает их. Результат сравнения определяется знаков возвращаемого функций *qsort* числа.

Выполнение работы.

1. Для того, чтобы реализовать ввод текста и слова, выделим память под строку *char* str*, которая будет хранить исходный текст, и под строку *char* word*, которая будет хранить введенное пользователем слово, с помощью функции *calloc*() из стандартной библиотеки языка<*stdlib.h*>. Размер каждого символа определим с помощью функции *sizeof(),* а так как строка – это массив символов, на каждый символ будем выделять *sizeof(char)*.

2. Считаем строку-текст и строку-слово с помощью функций *fgets(),* которые считывают символы из потока и сохраняют их в виде строки до тех пор, пока не наступит конец строки. После ввода текста пользователь перейдет на новую строку. Следовательно, символ *'\n'* запишется последним в первую строку *str* перед нуль-терминатором. Аналогично считается строка-слово.

3. Так как перед нуль-терминатором запишется символ перевода строки, "переприсвоим" последний элемент каждой строки(*'\n'*) на нуль-терминатор(*'\0'*).

4. Разбивать текст на слова будем с помощью алгоритма с использованием функции *strtok()* из заголовочного файла <*string.h*>. Инициализируем переменную *char* pch*, которая будет хранить возвращаемый функцией *strtok()* указатель на последнюю найденную лексему в строке или пустой указатель, если найденных лексем нет. Аргументами функции для первого вызова передаем указатель на строку, содержащую исходный текст, а также строковый литерал, содержащий символы-разделители слов.

5. Далее выделим память на динамический массив указателей на строки *char** text*, который будет хранить указатели на найденные лексемы, с помощью функции стандартной библиотеки языка *calloc()*: выделим память на небольшое количество элементов(*int sent_num*) типа *char***, на каждой итерации цикла *while(pch!=NULL)* будем присваивать каждому элементу массива *text[i++]* возвращенный функцией *strtok(NULL," .")* указатель на найденную лексему. Если выделенная память "заканчивается", расширяем ее с помощью известной нам функции *realloc()*. Таким образом, вызывая функцию

strtok() в цикле, мы постепенно заполним массив указателями на слова из текста.

6. Так как функция двоичного поиска работает только с отсортированными массивами, нам нужно выполнить сортировку слов. Чтобы отсортировать слова в массиве, воспользуемся функцией быстрой сортировки *qsort()* из стандартной библиотеки языка. В качестве аргументов функции *qsort()* передаем сам массив *char** text,* количество элементов, которые нужно сортировать, т.е. количество элементов массива: *int i* - счетчик, который увеличивался по мере заполнения массива *text[],* размер элементов массива *sizeof(char**)* и функцию-компаратор *cmpr*, которая будет сравнивать наши элементы(определим ее далее).

7. Определим функцию-компаратор *cmpr*. Функция будет возвращать целое значение *int для* того, чтобы *qsort()* выполнил сортировку строк. Причем возвращаемое значение будет регулироваться функцией *strcmp()* из заголовочного файла <*string.h*>. Аргументами функции будут два элемента *const void* a, const void* b,* для того, чтобы компаратору могли передаваться указатели любого типа. А далее уже указатели будут приводиться к нужному нам типу и разыменовываться, а их значения сравниваться. Строка, которая будет возвращать результат будет выглядеть так: *return strcmp(*(char**)a,*(char**)b);*

8. После вызова функции *qsort()* массив *char** text* отсортируется. И можно приступать к двоичному поиску слова в тексте.

9. Инициализируем переменную *char** pItem,* которая будет хранить в себе возвращенный функцией *bsearch()* указатель на слово, совпадающее с ключом поиска, если таковое нашлось. Если искомое слово не найдется, то вернется *NULL*.

10. В аргументы функции *bsearch()* передадим указатель на слово-ключ поиска *&word*, указатель на первый элемент массива, в котором будет выполняться поиск *text,* количество элементов массива *i,* размер элементов массива *sizeof(char**),* а также функцию-компаратор, которая будет сравнивать

элементы массива с самим ключом *cmpr*(определенная нами на предыдущем шаге функция).

11. Таким образом, если в *char** pItem* запишется ненулевой указатель, то значит, слово, совпадающее с ключом поиска присутствует в тексте, т.е. выводим *“exists”;* если же *bsearch()* вернул *NULL*, то выводим *“doesn’t exist”*. Эти действия можно записать с помощью конструкции *if-else:*

```
if(pItem!=NULL)
        printf(“exists”);
else
        printf(“doesn’t exist”);
```

12. Основная задача программы выполнена, программа обеспечивает все заданные условия. Завершающим шагом не забываем очистить выделенную для выполнения задачи память. «Очистку» памяти произведем с помощью функции *free().*

Разработанный программный код см. в приложении А.

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 concurrent class-based object-oriented and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA) meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016 Java is one of the most popular programming languages in use particularly for client-	exists	Вывод верный

	server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems Java platform. is		
2.	Hello world.Direct message. world	exists	Вывод верный
3.	Hello world.Direct message. unique	doesn’t exist	Вывод верный

Выводы.

Была изучена стандартная библиотека языка Си, а также освоены ее возможности посредством использования основных функций этой библиотеки в программном коде.

Разработана программа, которая обрабатывает полученный на вход текст: разбивает его на слова, используя основные функции стандартной библиотеки, отсортировывает слова, используя функцию быстрой сортировки, и определяет, есть ли среди них введенное пользователем слово, с помощью алгоритма двоичного поиска.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int cmpr(const void* a,const void* b){
    return strcmp(*(char**)a,*(char**)b);
}
int main(){
    char* str=calloc(1001,sizeof(char));
    char* word=calloc(31,sizeof(char));
    fgets(str,1001,stdin);
    fgets(word,31,stdin);
    str[strlen(str)-1]='\0';
    word[strlen(word)-1]='\0';

    char* pch=strtok(str," .");
    int sent_num=15;
    int i=0;
    char** text=calloc(sent_num,sizeof(char*));
    while(pch!=NULL){
        text[i++]=pch;
        if(i==sent_num){
            sent_num*=2;
            text=realloc(text,sent_num*sizeof(char*));
        }
        pch=strtok(NULL," .");
    }

    qsort(text,i,sizeof(char**),cmpr);
    char** pItem;
    pItem=bsearch(&word,text,i,sizeof(char**),cmpr);
    if(pItem!=NULL)
        printf("exists");
    else
        printf("doesn't exist");
    free(text);
    free(str);
    free(word);
     return 0;
}
```