**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе№1**

**по дисциплине «Программирование»**

Тема: **Обзор стандартной библиотеки языка С.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Васильев А.А. |
| Преподаватель |  | Кринкин К.В. |

Санкт-Петербург

2017

**Оглавление**

[Оглавление 2](#_Toc477205270)

[Цель работы 3](#_Toc477205271)

[Формулировка задачи 3](#_Toc477205272)

[Ход работы 4](#_Toc477205273)

[1. Разбиение текста на слова 4](#_Toc477205274)

[2. Сортировка полученного массива строк по алфавиту 4](#_Toc477205275)

[3. Бинарный поиск строки 5](#_Toc477205276)

[Вывод 5](#_Toc477205277)

[Приложение 1 6](#_Toc477205278)

**Цель работы**

Написание программы, в которой все действия будут реализованы средствами стандартной библиотеки языка С.

**Формулировка задачи**

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
* определить, присутствует ли в тексте **str**, используя алгоритм двоичного поиска
* вывести строку "exists", если str в тексте есть и "doesn't exist" в противном случае.

**Ход работы**

1. *Разбиение текста на слова*

Для реализации разбиения используем функцию **strtok**:

int count = 0;

char\*\* words = (char\*\*)realloc(words, sizeof(char\*)); //Выделяем память под первое слово

words[count] = strtok(input\_text, " ."); // Ищем первую лексему и помещаем её в массив

while(words[count]) //Цикл выполняется пока элемент массива не равен NULL

{

words = (char\*\*)realloc(words, sizeof(char\*)\*(count+2)); //Выделяем память под новое слово

words[++count] = strtok(NULL, " ."); //и помещаем в неё следующую лексему }

words = (char\*\*)realloc(words, sizeof(char\*)\*(count));

/\*Перераспределяем память, так как последний элемент не указывает на строку, а указывает на NULL\*/

1. *Сортировка полученного массива строк по алфавиту*

После разбиения текста мы получили массив из строк. Для его сортировки по алфавиту с помощью библиотечной функции **qsort** требуется функция-компаратор:

int compare(const void \* a, const void \* b)

{

return strcmp(\*(char\*\*)a, \*(char\*\*)b); //Возвращает результат сравнения двух строк:

} //0 - "a" = "b"; >0 - "a" > "b"; <0 - "a" < "b"

Функция **strcmp** посимвольно сравнивает две строки. Функция возвращает ноль, когда строки одинаковы. Иначе функция возвращает разность первых двух неодинаковых символов (разность между кодами символов).

Теперь можно использовать **qsort**:

qsort (words, count, sizeof(char\*), compare);

1. *Бинарный поиск строки*

Для поиска используем функцию **bsearch**, для которой тоже требуется отсортированный массив и функция-компаратор. В нашем случае подходит компаратор, созданный раннее.

char\*\* text\_has\_str = (char\*\*)bsearch(&str, words, count, sizeof(char\*), compare);

if(text\_has\_str)

printf("exists\n");

else

printf("doesn't exist\n");

**Вывод**

Для реализации алгоритма мы использовали функции стандартной библиотеки языка Си. Применение готовых функций заметно сокращает количество ошибок в работе программы. Также уменьшается время, затраченное для выполнения операций. В нашем случае разница во времени не так существенна, но в более объемных программах она станет заметнее.

**Приложение 1**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int compare(const void \* a, const void \* b) //Функция сравнения

{

return strcmp(\*(char\*\*)a, \*(char\*\*)b); //Возвращает результат сравнения двух строк:

} //0 - "a" = "b"; >0 - "a" > "b"; <0 - "a" < "b"

int main()

{

char\* input\_text = (char\*)malloc(sizeof(char)\*1000);

char\* str = (char\*)malloc(sizeof(char)\*30);

// Выделяем память под текст и под слово для поиска

char\*\* words = (char\*\*)realloc(words, sizeof(char\*)); //Выделяем память под первое слово

fgets(input\_text, 1000, stdin); //Считываем исходный текст и слово для поиска

fgets(str,30, stdin);

//\*strchr (input\_text, '\n') = '\0'; //В Ubuntu требуется данные строки

//\*strchr (str, '\n') = '\0'; //которые удаляют знаки переноса строки

int count = 0;

words[count] = strtok(input\_text, " ."); // Ищем первую лексему и помещаем её в массив

while(words[count]) //Цикл выполняется пока элемент массива не равен NULL

{

words = (char\*\*)realloc(words, sizeof(char\*)\*(count+2)); //Выделяем память words[++count] = strtok(NULL, " ."); //Помещаем в память следующую лексему

}

words = (char\*\*)realloc(words, sizeof(char\*)\*(count));

/\*Перераспределяем память, так как последний элемент не указывает на строку, а указывает на NULL\*/

qsort (words, count, sizeof(char\*), compare);

/\*Сортируем массив строк библиотечной функцией\*/

char\*\* text\_has\_str = (char\*\*)bsearch(&str, words, count, sizeof(char\*), compare);

/\*Ищем строку "str" в массиве\*/

if(text\_has\_str)

printf("exists\n");

else

printf("doesn't exist\n");

return 0;

}