**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: Регулярные выражения**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Роднов И.С. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Обучение работе с регулярными выражениями и создание программы на языке C, при помощи которой, производится поиск и вывод частей текста, которые подходят по изначальным параметрам.

## Задание

На вход программе подается текст, представляющий собой набор предложений с новой строки. Текст заканчивается предложением "Fin." В тексте могут встречаться ссылки на различные файлы в сети интернет. Требуется, используя регулярные выражения, найти все эти ссылки в тексте и вывести на экран пары <название\_сайта> - <имя\_файла>. Гарантируется, что если предложение содержит какой-то пример ссылки, то после ссылки будет символ переноса строки.

Ссылки могут иметь следующий вид:

Могут начинаться с названия протокола, состоящего из букв и :// после

Перед доменным именем сайта может быть www

Далее доменное имя сайта и один или несколько доменов более верхнего уровня

Далее возможно путь к файлу на сервере

И, наконец, имя файла с расширением.

## Выполнение работы

Функция make\_text принимает на входу указатель на длинну считываемого текста text\_len – которая в дальнешем увеличивается по мере считывания текста. В этой функции выделяется память под текст, а так же по каждое предложение, предложения счиытваются построчно при помощи цикла do-while и fgets() до момента пока не встретится предложение “Fin.”. Реализованны проверки выделения памяти.

В функции main создается и компилируется регулярное выражение. Получается текст из функции make\_text. Затем запускается цикл, в котором каждое предложение сверяется с регулярным выражение и в случае совпадения выводится на экран в нужном формате. В конце очищается память при помощи regfree() и вызывается функция free\_text для очистки памяти из под текста.

Функция free\_text принимает на вход текст и длинну текста. Внутри функции при помощи цикла и free() очищается память из под каждого предложения и из под текста в целом.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
|  | This is simple url:  http://www.google.com/track.mp3  May be more than one upper level  domain http://www.google.com.edu/hello.avi  Many of them.  Rly. Look at this!  http://www.qwe.edu.etu.yahooo.org.net.ru/qwe.q  Some other protocols  ftp://skype.com/qqwe/qweqw/qwe.avi  Fin. | google.com - track.mp3  google.com.edu - hello.avi  qwe.edu.etu.yahooo.org.net.ru - qwe.q  skype.com - qwe.avi |

## Выводы

Успешное обучение работе с регулярными выражениями. Освоение особенностей работы с регулярными выражениями на языке С, в следствии чего написание верно работающей программы.

# Приложение А Исходный код программы

Название файла: main.c

#include <regex.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#define MAX\_TEXT\_SIZE 1000

char\*\* make\_text(int\* text\_len)

{

int capacity = 10; // Initial capacity

char\*\* text = malloc(capacity \* sizeof(char\*));

if (text == NULL) {

exit(EXIT\_FAILURE);

}

int index = 0;

do {

if (index == capacity) {

capacity \*= 2; // Double the capacity

text = realloc(text, capacity \* sizeof(char\*));

if (text == NULL) {

exit(EXIT\_FAILURE);

}

}

text[index] = calloc(MAX\_TEXT\_SIZE, sizeof(char));

if (text[index] == NULL) {

exit(EXIT\_FAILURE);

}

if (fgets(text[index], MAX\_TEXT\_SIZE, stdin) == NULL) {

free(text[index]);

break; // Break the loop on fgets failure

}

index++;

} while (!strstr(text[index - 1], "Fin."));

\*text\_len = index;

return text;

}

void free\_text(char\*\* text, int text\_len)

{

for (int i = 0; i < text\_len; i++) {

free(text[i]);

}

free(text);

}

int main()

{

regex\_t regex;

int value;

int text\_len = 0;

char\*\* text = make\_text(&text\_len);

char\* pattern = "([a-zA-Z]+://)?([w]{3}\\.)?([a-zA-Z0-9-]+(\\.[a-zA-Z0-9]+)+)/?(([a-zA-Z0-9]+/)+)?([a-zA-Z0-9-]+(\\.[a-zA-Z0-9-]+))\n$";

regmatch\_t match[8];

value = regcomp(&regex, pattern, REG\_EXTENDED);

int count = 0;

int count\_2 = 0;

for (int i = 0; i < text\_len; i++) {

value = regexec(&regex, text[i], 8, match, 0);

if (value == 0) {

count++;

}

}

for (int i = 0; i < text\_len; i++) {

value = regexec(&regex, text[i], 8, match, 0);

if (value == 0) {

count\_2++;

if (count\_2 != count) {

printf("%.\*s - %.\*s\n",

(int)(match[3].rm\_eo - match[3].rm\_so), &text[i][match[3].rm\_so],

(int)(match[7].rm\_eo - match[7].rm\_so), &text[i][match[7].rm\_so]);

} else {

printf("%.\*s - %.\*s",

(int)(match[3].rm\_eo - match[3].rm\_so), &text[i][match[3].rm\_so],

(int)(match[7].rm\_eo - match[7].rm\_so), &text[i][match[7].rm\_so]);

}

}

}

regfree(&regex);

free\_text(text, text\_len);

return 0;

}