**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 |  | Иванов С.С. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Изучение регулярных выражений и реализация программы, которая осуществляет поиск строк текста, удовлетворяющих заданному виду и выводит их фрагменты на экран.

## 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.

## Основные теоретические положения.

Регулярные выражения (их еще называют regexp, или regex) — это механизм для поиска и замены текста.

С помощью regex можно искать как конкретные выражения, так и что-то более общее (например, любую букву или цифру).

Для обозначения второй категории существуют специальные символы. Вот некоторые из них:

«.» - любой символ;

«[...]» - любой символ из тех, что представлены в скобках;

«[^...]» - любой символ, кроме тех, что представлены в скобках;

«^» - начало строки;

«$» - конец строки;

«\» - экранирование специальных символов;

«|» - логическое «ИЛИ».

Также существуют специальные символы, наиболее часто из которых встречаются +, \* и ?. + используют, чтобы обозначить, что группа или символ используются 1 и более раз, \* - 0 и более, а ? сигнализирует о том, что символ или группа или отсутствуют, или встречаются не более 1 раза в этом месте.

\w\d\s используют, чтобы обозначить буквы, цифры или пробельные символы.

Когда в квадратных скобках указывается диапазон, подразумевается наличие одного из этих символов (или 0, или большего числа в зависимости от знака после, если он есть), причём в регулярных выражениях используется расположение символов в таблице ASCII. Таким образом, если написать, например, [A-z], будет считаться, что символ ^ также может входить в строку и быть на данном месте.

В Си для работы с регулярными выражениями подключается библиотека regex.h.

## Выполнение работы

char \* read\_text() – функция для считывания текста.

char \*\* split\_text(char \*text, int \*k) – функция получает на фход текст и количество предложений. Возвращает уже текст, разбитый по предложениям.

В функции main вызываются происходит считывание текста а также его разбиение на предложения (с помощью вышеописанных функций). В переменную regex\_pttrn записывается регулярное выражение, соответствующее условию задачи. В переменной maxGroups записано максимальное число групп в выражении. Создаются переменные regexCompiled (где будет храниться скомпилированное регулярное выражение) типа regex\_t и groupArray типа regmatch\_t.

Если компиляция не прошла успешно, выводится ошибка.

В цикле for проверяется каждое предложение считанного текста. Если для этого предложения результат вызова функции regexec равен нулю, то выводятся все символы групп, в которых содержится имя домена и название файла. Между этими выводами на экран выводится тире, после — символ переноса строки.

Далее с помощью функций free, regfree осуществляется освобождение памяти.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | Hello! That is my program  Try this simple URL:  http://www.google.com/kjwfwk/kitten.jpg  And what about this one?  ftp://jshkjshk.net.ru.edu/kjwhdjkh/kjhkhl/jh/ha-ha.zip  Here is the one you should NOT choose:  hgjhghjp:/i-am-the-wrong-choice.boo/jjjjjjj/whats'up.doc  Fin. | google.com - kitten.jpg  jshkjshk.net.ru.edu - ha-ha.zip |  |
|  | 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

Были изучены регулярные выражения, их использование.

Разработана программа, выполняющая считывание с клавиатуры текста. Используя регулярные выражения, программа находит URL файла и выводит в консоль название домена и имя файла. Для этого использовалась библиотека regex.h и циклы for, while, динамическое выделение памяти и её освобождение.

# Приложение А Исходный код программы

Название файла: Antipina\_Veronika\_lb1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <regex.h>

#include <stdbool.h>

#define BLOCK\_SIZE 10

void

error(const char \*\_msg)

{

fprintf(stderr, "%s\n", \_msg);

exit(0);

}

void

error\_if(bool \_cond, const char \*\_msg)

{

if (\_cond)

error(\_msg);

}

void \*

allocator(void \*\_old, size\_t \_n, size\_t \_size)

{

void \*\_ptr = realloc(\_old, \_n \* \_size);

error\_if(\_ptr == NULL, "ERROR: badalloc");

return \_ptr;

}

char \*

read\_text()

{

int cap = BLOCK\_SIZE;

char \*text = (char\*)allocator(NULL, cap, sizeof(char));

int k = 0;

char ch = getchar();

char ch\_1 = 'a';

char ch\_2 = 'a';

char ch\_3 = 'a';

char ch\_4 = 'a';

while( ch\_1 != 'F'

|| ch\_2 != 'i'

|| ch\_3 != 'n'

|| ch\_4 != '.')

{

text[k++] = ch;

if (k == cap-1)

{

cap += BLOCK\_SIZE;

text = (char\*)allocator(text, cap, sizeof(char));

}

ch\_1 = ch\_2;

ch\_2 = ch\_3;

ch\_3 = ch\_4;

ch\_4 = ch;

ch = getchar();

}

text[k] = '\0';

return text;

}

char \*\*

split\_text(char \*text, int \*k)

{

int cap = BLOCK\_SIZE;

char \*\*res\_text = (char\*\*)allocator(NULL, cap, sizeof(char\*));

char \*sentence = strtok(text, "\n");

while(sentence != NULL)

{

int sent\_len = strlen(sentence);

res\_text[\*k] = (char\*)allocator(NULL, sent\_len+1, sizeof(char));

strcpy(res\_text[(\*k)++], sentence);

if(\*k == cap-1)

{

cap += BLOCK\_SIZE;

res\_text = (char\*\*)allocator(res\_text, cap, sizeof(char\*));

}

sentence = strtok(NULL, "\n");

}

return res\_text;

}

int main(int argc, char \*\*argv)

{

char \*text = read\_text();

int k = 0;

char \*\*res\_text = split\_text(text, &k);

const char \*regex\_pttrn = "(([A-z]\*):\\/\\/)?(www.)?([A-z0-9]+([\_\\-\\.]+[A-z]+)+)\\/((([A-z]\*)\\/)\*)([A-z0-9\_\\-]+\\.[A-z0-9\_\\-]+)";

regex\_t regexCompiled;

regmatch\_t groupArray[BLOCK\_SIZE];

error\_if(regcomp(&regexCompiled, regex\_pttrn, REG\_EXTENDED),

"ERROR: Could not compile regular expression\n");

for(int index = 0; index < k; index++)

{

if(regexec(&regexCompiled, res\_text[index], BLOCK\_SIZE, groupArray, 0) == 0)

{

for(int j = groupArray[4].rm\_so; j < groupArray[4].rm\_eo; j++)

printf("%c",res\_text[index][j]);

printf(" - ");

for(int j = groupArray[9].rm\_so; j < groupArray[9].rm\_eo; j++)

printf("%c",res\_text[index][j]);

printf("\n");

}

}

free(text);

for(int i = 0; i < k; i++)

free(res\_text[i]);

free(res\_text);

regfree(&regexCompiled);

return 0;

}