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. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

Цель работы.

1. Изучить понятие регулярного выражения.
2. Изучить функции, представленные в библиотеке *regex.h*.

Задание.

Вариант лабораторной работы №1.

Реализуйте программу, которая находит ссылки на файлы.

На вход программе подаётся 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.

Основные теоретические положения.

Регулярные выражения – формальный язык поиска и осуществления действий с подстроками в тексте. Для поиска используется строка-образец, состоящая из символов и метасимволов и задающая правило поиска.

Для работы с регулярными выражениями в языке С используется библиотека *regex.h.* Библиотека содержит 4 основные функции для работы с регулярными выражениями.

```
int regcomp(regex_t preg, const char *pattern, int cflags);
```

Функция принимает на вход специальную структуру типа *regex_t,* указатель на строку с регулярным выражением, и флаг, указывающий . Функция проверяет регулярное выражение на коррентность, выделяет память для структуры *preg* и заполняет её поле. В случае успеха функция возвращает 0 и записывает количество подгрупп регулярного выражения в поле структуры *preg*; в случае ошибки возвращает ненулевое значение, в данном случае значение *preg* не определено.

```
int regexec(const regex_t preg, const char *string, size_t nmatch,
regmatch_t pmatch[], int eflags);
```

Функция принимает на вход структуру *preg*, указатель на строку-источник *string*, количество рассматриваемых групп *nmatch* (для данной переменной допустимы тип *int* или *size_t*) и массив структур *regmatch_t pmatch[]* размера, равному числу рассматриваемых групп. Структура представляет собой поля, обозначающие индексы начала и конца подстроки для данной группы в регулярном выражении. Если строка подходит для данного регулярного выражения, функция возвращает 0 и заполняет поля структур массива для каждой группы (если группа отсутствует, то индексы равны *-1*. В случае отсутствия совпадений функция возвращает ненулевое значение.

```
size_t regerror(int errcode, const regex_t *preg, char *errbuf, size_t errbuf_size);
```

Функция принимает на вход возвращаемое значение функций *regcomp* и *regexec*, указатель на массив символов *errbuf* и его размер в байтах *errbuf_size* Функция формирует из кодов ошибок строку с информацией о том, что именно произошло. Функция возвращает количество байт, выделенное для строки. Функция возвращает 0, если обе функции выполнились без ошибок.

```
void regfree(regex_t preg);
```

Функция очищает память, выделенную для структуры *regex_t preg*.

Для записи регулярных выражений используют метасимволы, которые можно представить в виде следующей таблицы.

Метасимвол	Значение
\w	Любой символ в следующих диапазонах: 0-9, a-z, A-Z, _
\d	Любой символ цифры
.	Любой символ
\s	Любой непечатный символ (\t, \n …)
[...]	Один символ из набора
[^…]	Один любой символ не из набора

[B-Y0-3]	Один символ из диапазонов B-Y, 0-3
x+	Хотя бы одно повторение данного символа/группы символов
x*	Любое количество повторений данного символа/группы символов
^	Поиск с начала строки (помещается в начало строки)
$	Поиск с конца строки (помещается в конец строки)
x{a,b}	Повторение данного символа N раз в пределах: a≦N≦b (если значение b отсутствует, то b = ∞)
|	Операция ИЛИ для двух примыкающих к оператору групп
(...)	Символы группировки

Выполнение работы.

В программе использовались следующие переменные:

1. Целого типа (int):
    a. *sentcount* – число строк, поступивших на вход программе.
    b. *const siteName* – константа, обозначающая номер подгруппы регулярного выражения, содержащего информацию о названии сайта.
    c. *const fileName* – константа, обозначающая номер подгруппы регулярного выражения, содержащего информацию о названии файла.
    d. *const maxGroups* – константа, обозначающая количество подгрупп в регулярном выражении.
2. Символьного типа (char):
    a. *textFrag[SENT_UNIT]* – фрагмент считываемого текста, размер которого задаётся макросом *SENT_UNIT,* по умолчанию равный 64.
3. Указатели типа char:
    a. *char* tempp* – указатель на строку-буфер, используется в функции.

b. *char* rawText* – указатель на строку, содержащую все символы с потока ввода.
c. *const char* regexString* – указатель на строку, содержащую регулярное выражение, необходимое для решения поставленной задачи. Имеет следующий вид: *(\w+:\/\/)?(www\.)?((\w+\.)+\w+)\/(\w+\/)*(.+\.\w+)*

   Для корректной обработки регулярного выражения, каждый знак «\» дублируется.
d. *char** text* – указатель на динамический массив строк.

4. Структуры (struct):
   a. *regex_t regexStringCompiled* – структура, содержащая количество заключенных в скобки группы регулярного выражения.
   b. *regmatch_t groupArray[maxGroups]* – структура, содержащая 2 массива – индексы начал и концов подстрок, подходящих регулярному выражению для каждой группы регулярного выражения.

В программе реализованы следующие функции:

1. Функция *char* readText()* считывает все элементы с потока ввода и записывает их в динамический массив символов *rawText*. Возвращает указатель на массив символов.

Разработанный программный код см. в приложении А.

Результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	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/	google.com - track.mp3 google.com.edu - hello.avi qwe.edu.etu.yahooo.org.net.ru - qwe.q skype.com - qwe.avi	Тест пройден

	qwe.q Some other protocols ftp://skype.com/qqwe/qweqw/qwe.avi Fin.		
2.	skype.com/qqwe/qweqw/qwe.avi skype.com/qwq.info s jsldkf js s sf www.skype.com/qwq.info dsfsd s sf sdf ftp://skype.com/qwq.info Fin.	skype.com - qwe.avi skype.com - qwq.info skype.com - qwq.info skype.com - qwq.info	Тест пройден

Выводы.

В ходе лабораторной работы были изучены регулярные выражения и функции библиотеки языка С для работы с ними.

Была разработана программа, которая считывает текст с потока ввода и выводит ссылки на различные файлы в сети интернет. Для обработки текста использовались функции библиотеки *regex.h*.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define SENT_UNIT 64

char* readText(){

    char textFrag[SENT_UNIT];
    char* tempp;
    char* rawText = (char*)malloc(sizeof(char)*SENT_UNIT);

    while (1){
        tempp = fgets(textFrag, SENT_UNIT, stdin);
        if (tempp == NULL)
            return rawText;;
        rawText = (char*)realloc(rawText,
sizeof(char)*SENT_UNIT+strlen(rawText));
        strcpy(rawText+strlen(rawText)*sizeof(char), tempp);
    }
}

int main(){

    const char* regexString = "(\\w+:\\/\\/)?(www\\.)?((\\w+\\.)+\\w+)\\/
(\\w+\\/)*(.+\\.\\w+)";
     const int siteName = 3;
     const int fileName = 6;
     const int maxGroups = 7;
    int sentcount = 0;
    char** text = (char**)malloc(sizeof(char*));

    regex_t regexStringCompiled;
    regmatch_t groupArray[maxGroups];

     char* rawText = readText();
    char* tempp = strtok (rawText, "\n");

    while (tempp != NULL){
        text[sentcount] = tempp;
        sentcount++;
        text = (char**)realloc(text, (sentcount+1)*sizeof(char*));
        tempp = strtok(NULL, "\n");
    }

    regcomp(&regexStringCompiled, regexString, REG_EXTENDED);
    for(int i=0; i<sentcount; i++){
        if (regexec(&regexStringCompiled, text[i], maxGroups, groupArray,
0) == 0){
            for(int j=groupArray[siteName].rm_so;
j<groupArray[siteName].rm_eo; j++)
                printf("%c", text[i][j]);
            printf(" - ");
```

```
            for(int j=groupArray[fileName].rm_so;
j<groupArray[fileName].rm_eo; j++)
                printf("%c", text[i][j]);
        }
         printf("\n");
    }

    regfree(&regexStringCompiled);
    free(rawText);
    free(text);

    return 0;
}
```