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

Студентка гр. 9304 _____ Каменская Е.К.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2020

Цель работы.

Изучение регулярных выражений, работа с ними на базе языка Си.

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

Sample Input 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/qwe.q
Some other protocols ftp://skype.com/qqwe/qweqw/qwe.avi
Fin.
```

Sample Output 1:

```
google.com - track.mp3
google.com.edu - hello.avi
qwe.edu.etu.yahooo.org.net.ru - qwe.q
skype.com - qwe.avi
```

Выполнение работы.

Макросы *TERM* и *ADDSIZE* равны *«Fin.»* и 16 соответственно. Переменной *regexString* присваивается строка регулярного выражения, *maxGroups* – количество групп в выражении. После инициализации переменных происходит компиляция регулярного выражения, и, если это сделать не удалось, программа выводит соответствующее сообщение и завершается.

Считывание текста осуществляется с помощью функций *get_sentence* и *read_til_terminal*. Если в какой-то из них не получилось выделить память под вводимый текст, программа выводит соответствующее сообщение, очищает уже выделенную память и завершается.

Далее программа итерируется по массиву предложений текста и с помощью функции *regexec* ищет совпадения с регулярным выражением. Если совпадение найдено, программа выводит искомые фрагменты на экран. После окончания этого цикла память, выделенная под регулярное выражение, очищается функцией *regfree*, аналогично очищается память под текст.

Разработанный программный код см. в приложении А.

Выводы.

В ходе выполнения лабораторной работы были изучены регулярные выражения и принципы работы с ними на языке Си. Была реализована программа для считывания текста, поиска подстрок заданного формата и вывода нужных их фрагментов на экран.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h>
#define TERM "Fin."
#define ADDSIZE 16

int read_til_terminal(char*** totext, char* terminal);

char* get_sentence();

int main(){
    char* regexString = "(\\w+\\:\\/\\/)?(www\\.)?((\\w+\\.)+(\\
w+))\\/(\\w+\\/)*(\\w+\\.\\w+)";
    size_t maxGroups = 8;
    regmatch_t groupArray[maxGroups];
    regex_t regexCompiled;
    if(regcomp(&regexCompiled, regexString, REG_EXTENDED)){
        printf("Could not compile regular expression.\n");
        return 0;
    }
    char** text = malloc(1*sizeof(char*));
    int n = read_til_terminal(&text, TERM);

    for(int i = 0; i < n; i++){
        if(regexec(&regexCompiled, text[i], maxGroups,
groupArray, 0) == 0){
            for(int j = groupArray[3].rm_so; j <
groupArray[3].rm_eo; j++){
                printf("%c", text[i][j]);
            }
            printf(" - ");
            for(int k = groupArray[7].rm_so; k <
groupArray[7].rm_eo; k++){
                printf("%c", text[i][k]);
            }
            printf("\n");
        }
    }
    regfree(&regexCompiled);
    for(int i = 0; i < n; i++){
        free(text[i]);
    }
    free(text);
    return 0;
}
int read_til_terminal(char*** totext, char* terminal){
    int n = 0;
    char** totexttest;
    do{
```

```
            totexttest = realloc(*totext, (++n)*sizeof(char*));
            if(!totexttest){
                printf("Could not allocate memory for text\n");
                free(*totext);
                return 0;
            }
            *totext = totexttest;
            *(*totext+n-1) = get_sentence();
        }while(strcmp(*(*totext+n-1), terminal));
        return n;
    }

    char* get_sentence(char*** totext){
        char* newsntc = malloc(ADDSIZE*sizeof(char));
        char* newsntctest;
        char c;
        int size = ADDSIZE;
        int i = 0;
        while((c = getc(stdin)) && (c != '\n')){
            if(i == size-2){
                size += ADDSIZE;
                newsntctest = realloc(newsntc, size*sizeof(char));
                if(!newsntctest){
                    printf("Could not allocate memory for sentence\
n");
                    free(newsntc);
                    free(*totext);
                    return 0;
                }
                newsntc = newsntctest;
            }
            newsntc[i] = c;
            i++;
            if(strcmp(newsntc, TERM) == 0){
                newsntc[i] = '\0';
                return newsntc;
            }
        }
        newsntc[i] = '\0';
        return newsntc;
    }
```