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 _____ Хайруллов Д.Л.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью лабораторной работы является изучение применения регулярных выражений в языке программирования C.

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

С помощью функции input_text реализуется считывание текста из стандартного потока ввода и его разделение на предложения с помощью двумерного динамического массива.

Объявляется указатель на строку регулярного выражения, которое затем компилируется. В цикле for перебираются индексы предложений текста, каждое предложение проверяется на наличие совпадений с регулярным выражение с помощью regexec. При нахождении совпадений производится вывод символов, заключенных в определенных группах захвата регулярного выражения.

В конце работы программы с помощью функции clean_memory и regfree очищается ранее занятая динамическая память.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
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.	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 основы работы с регулярными выражениями в языке программирования C. С помощью функций библиотеки regex.h была написана программа, с помощью которого была решена поставленная задача.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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


#define END_OF_THE_TEXT "Fin."
#define MEMORY_ERROR "Error: error reallocating memory\n"
#define COMPILE_REGEX_ERROR "Cant't compile regular expression\n"
#define MEMORY_ALLOCATING_ERROR_PRINT "Error: memory allocating error\n"
#define MEMORY_ALLOCATING_ERROR_EXIT 40

char** text_input(int* num_of_sentences);
void clean_memory(char** text, int* num_of_sen);


char** text_input(int* num_of_sentences){
    int current_symbol_index = 0;
    int current_sentence_index = 0;
    char symbol;
    char** text = (char**)malloc(1*sizeof(char*));
    if(text == NULL)
    {
        printf(MEMORY_ALLOCATING_ERROR_PRINT);
        exit(MEMORY_ALLOCATING_ERROR_EXIT);
    }

    text[current_sentence_index] = (char*)malloc(1*sizeof(char));
    if(text[current_sentence_index] == NULL)
    {
        printf(MEMORY_ALLOCATING_ERROR_PRINT);
        exit(MEMORY_ALLOCATING_ERROR_EXIT);
    }

    while(strstr(text[current_sentence_index], END_OF_THE_TEXT) == NULL){
        symbol = getchar();
        if(symbol == '\n'){
            text[current_sentence_index][current_symbol_index] = '\0';
            current_symbol_index = 0;
            current_sentence_index++;

            text = (char**)realloc(text, (current_sentence_index +
1)*sizeof(char*));
            if(text == NULL)
            {
                printf(MEMORY_ALLOCATING_ERROR_PRINT);
                exit(MEMORY_ALLOCATING_ERROR_EXIT);
            }

            text[current_sentence_index] = (char*)malloc(1*sizeof(char));
            if(text[current_sentence_index] == NULL)
            {
```

```
                printf(MEMORY_ALLOCATING_ERROR_PRINT);
                exit(MEMORY_ALLOCATING_ERROR_EXIT);
            }
        }
        else{
            text[current_sentence_index][current_symbol_index] = symbol;
            current_symbol_index++;
            text[current_sentence_index]                                    =
(char*)realloc(text[current_sentence_index],    (current_symbol_index    +
1)*sizeof(char));
            if(text[current_sentence_index] == NULL)
            {
                printf(MEMORY_ALLOCATING_ERROR_PRINT);
                exit(MEMORY_ALLOCATING_ERROR_EXIT);
            }
        }
    }
    text[current_sentence_index][current_symbol_index] = '\0';
    *num_of_sentences = current_sentence_index;

    return text;
}



void clean_memory(char** text, int* num_of_sen){
    for(int i = 0; i <= *num_of_sen; i++){
        free(text[i]);
    }
    free(text);
}


int main(){
    char** text;
    int num_of_sentences = 0;
    text = text_input(&num_of_sentences);

    char*   regexString   =   "(:\\/\\/)?(www\\.)?(([A-z0-9]+\\.)+[A-z0-
9]+)\\/([A-z0-9]+\\/)*([A-z0-9]+\\.[A-z0-9]+)\0";
    size_t maxGroups = 7;
    regex_t regexCompiled;
    regmatch_t groupArray[maxGroups];
    if(regcomp(&regexCompiled, regexString, REG_EXTENDED)){
        printf(COMPILE_REGEX_ERROR);
        return 0;
    }

    for(int i = 0; i <= num_of_sentences; i++){
        if(regexec(&regexCompiled, text[i], maxGroups, groupArray, 0) ==
0){
            if(groupArray[3].rm_so == -1 || groupArray[6].rm_so == -1){
                break;
            }
            for(int  y  =  groupArray[3].rm_so;  y  <  groupArray[3].rm_eo;
y++){
                printf("%c", text[i][y]);
            }
```

```
            printf(" - ");
            for(int  y  =  groupArray[6].rm_so;  y  <  groupArray[6].rm_eo;
y++){
                printf("%c", text[i][y]);
            }
            printf("%c", '\n');
        }

    }

    clean_memory(text, &num_of_sentences);
    regfree(&regexCompiled);

    return 0;

}
```