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. 3341 _____ Мильхерт А.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Цель работы является изучение и использование регулярных выражений для обработки текстовых данных. Для этого необходимо изучить синтаксис и возможности регулярных выражений, а после применить полученные навыки на практике в ходе решения задачи.

Задание

Вариант 1

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

Подключаются необходимые библиотеки: *stdlib.h*, *stdio.h*, *string.h* и *regex.h*.

В переменную *regexString* записывается необходимое регулярное выражение.

Функция *get_full_text* считывает текст из ввода пользователя, вотом вызывается функцию *split_sentences* для разбиения цельного текста на строки и возвращает их в виде массива строк.

Функция *split_sentences* разделяет текст на отдельные предложения и возвращает их в виде массива строк.

Функция *check_regular* принимает массив предложений, применяет регулярное выражение *regexString* к каждому предложению и выводит предложения, которые соответствуют этому регулярному выражению.

В основной функции *main* происходит вызов этих функций: считывание текста, разделение на предложения и проверка совпадения с регулярным выражением.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	This is simple url: http://www.google.com/track.mp3 Fin.	google.com – track.mp3	Проверка на наличие www перед доменным именем
2.	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	Проверка на валидность выражений с доменами более высокого уровня и на наличие пути до файла
3.	ftp://pepepupu.cheeck/qqwe/qweqw/qwe.avi Fin.	pepepupu.cheeck – qwe.avi	Проверка исправности с протоколом ftp и на наличие пути до файла

Выводы

Цель данной работы заключалась в изучении и практическом применении регулярных выражений для обработки текстовых данных. Были изучены основные синтаксические конструкции и возможности регулярных выражений. Полученные знания были успешно применены для решения практической задачи, демонстрирующей использование регулярных выражений в реальной ситуации. Таким образом, цель данной работы была успешно достигнута.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define BUFFER 1024
#define MAX_LEN 100
#define CHECK 3
#define GROUPS_ID 8

char* get_full_text(int*);
char** split_sentences(char*, int);
void check_regular(char**, int);


int main() {
    int sentences_count = 0;
    char* text = get_full_text(&sentences_count);
    //    printf("%s\n", text);
    //    printf("%d\n", sentences_count);
    char** separation_text = split_sentences(text, sentences_count);
    check_regular(separation_text, sentences_count);

    free(text);
    free(separation_text);
    return 0;
}

char* get_full_text(int* sentences_count) {
    char c;
    int i = 0;
    int capacity = BUFFER;
    char* text = (char*)calloc(capacity, sizeof(char));

    while ((c = getchar()) != EOF) {
        text[i] = c;
        if (c == '\n') {
            ++(*sentences_count);
        }
        if (i == capacity - 1) {
            capacity += BUFFER;
            text = realloc(text, capacity * sizeof(char));
        }
        if (i >= CHECK && text[i] == '.' && text[i - 1] == 'n' &&
text[i - 2] == 'i' && text[i - 3] == 'F') {
            break;
        }
```

```
            i++;
        }
        text[i - 4] = '\0';
        return text;
    }

    char** split_sentences(char* full_text, int count_sentences) {
        int length = 0;
        char**     sentences     =     (char**)calloc(count_sentences,
sizeof(char*));
        char* sentence = strtok(full_text, "\n");

        while (sentence != NULL) {
            if (length >= count_sentences) {
                count_sentences *= 2;
                sentences   =   realloc(sentences,   sizeof(char*)   *
count_sentences);
            }
            sentences[length] = sentence;
            sentence = strtok(NULL, "\n");
            ++length;

        }
        return sentences;
    }

    void check_regular(char** sentences, int sentences_count) {
        char*    regexString    =    "(\\w+\\:\\/\\/)?(www\\.)?(([a-z0-
9\\.]+)?[a-z0-9]+\\.\\w+)\\/(([a-z0-9\\/]+)?\\w+\\/)?([a-z0-9]+\\.\\w+)";
        char** answer = (char**)calloc(sentences_count, sizeof(char*));
        regex_t regexCompiled;
        regmatch_t groups[GROUPS_ID];
        int size = 0;
        int matched_count = 0;
        regcomp(&regexCompiled, regexString, REG_EXTENDED);

        for (int j = 0; j < sentences_count; j++) {
            if (regexec(&regexCompiled, sentences[j], GROUPS_ID, groups,
0) == 0) {
                //answer    =    realloc(answer,    sizeof(char*)    *
(matched_count+1));
                char* final_line = (char*)calloc(100, sizeof(char));
                size = 0;

                for (int i = groups[3].rm_so; i < groups[3].rm_eo; i++)
{
                    final_line[size] = sentences[j][i];
                    ++size;
                }
                final_line[size++] = ' ', final_line[size++] = '-',
final_line[size++] = ' ';
                for (int i = groups[7].rm_so; i < groups[7].rm_eo; i++)
{
                    final_line[size] = sentences[j][i];
                    ++size;
                }

                final_line[size] = '\0';
```

```
                answer[matched_count++] = final_line;
            }
        }
        for (int i = 0; i < matched_count; i++) {
            if (i == matched_count - 1)
                printf("%s", answer[i]);
            else
                printf("%s\n", answer[i]);
        }
    }
```