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. 3344 _____ Волков А.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Изучить принцип работы с регулярными выражениями и их применение в языке программирования C.

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 заголовочные файлы *<stdio.h>*, *<stdlib.h>*, *<string.h>*, *<regex.h>*. Создаётся макроопределение *EXTRA_BUF*, которая является размером дополнительного буфера при необходимости выделения дополнительной памяти.

Функции:

1. Функция *readSentence()*:

Эта функция считывает предложение, вводимое пользователем, и возвращает указатель на строку типа *sentence*.

Выделяется память для временного указателя *tmp*. Если выделение памяти прошло успешно, то *tmp* обнуляется. С помощью цикла *while* и функции *gecthar()* происходит считывание символов и при нехватке памяти в динамическом массиве символов выделяется дополнительная память в размере *EXTRA_BUF*. При символе переноса строки (‘*\n*’) цикл останавливается, а в конец добавляется символ окончания строки (‘*\0*’).

2. Функция *readText(int *size)*:

Получает на вход указатель на переменную, в которую надо записать количество предложений втексте. Выделяется память для массива указателей. Используется цикл *while* для считывания каждого предложения с помощью функции *readSentence()* и сохранения указателя на каждое предложение в массив. С помощью функции *realloc()* выделяется память для следующего указателя. Считывание происходить до тех пор, пока не встретится строка "Fin.". Возвращает указатель на двумерный массив строк.

4. Функция *main()*:

Вызывается функция *readText()*, в нее передается адрес переменной *size*, в которой будет хранится количество предложений в тексте.

Создаётся строка *regexString*, которая содержит регулярное выражение, которое будет использоваться для поиска URL – адресов. Оно состоит из нескольких групп:

1. ([a-z]+:\\/{2}|) – протокол URL – адреса (может не быть)

2. (www\\.|) – перед доменным именем (может не быть)

3-4. (([a-zA-Z0-9][a-zA-Z0-9-]*\\.)+) – доменное имя

5. ([a-zA-Z0-9-]+) – домен верхнего уровня

6. ([a-zA-Z0-9_-]+\\/)* - пути в URL – адресе

7. ([a-zA-Z0-9_-]+\\.[a-zA-Z0-9_-]+) – имя файла и расширение

Далее создаётся переменная *maxGroups*, которая определяет максимальное количество групп, которые могут содержаться в регулярном выражение. В *regexCompiled* будет скомпилировано регулярное выражение с помощью функции *regcomp()*. Если компиляция не удалась, то выводится сообщение об ошибке. Каждая строка сравнивается с регулярным выражением с помощью функции *regexec()*. Если соответствие найдено, то выводим название сайта и его название (группа 3, 5, 7). В конце освобождается память, выделенная под текст, а также память, занятая скомпилированным регулярным выражением с помощью функции *regfree()*.

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 domain google.com.edu/hello.q ftp://skype.com/qqwe/qweqw/ qwe.avi Fin.	google.com - track.mp3 google.com.edu - hello.q skype.com - qwe.avi	Верный ответ.
2.	test google..com/hello.mp23 pr http://www.google.com//qwe.exe fw wf fds://google.com/test.exe	google.com - test.exe	Верный ответ (первые два не подходят)

Выводы

Было изучено и практически применено использование регулярных выражений в программировании.

Была разработана программа для поиска и выделения определенных частей URL – адресов с использованием регулярных выражений. Это показало, что регулярные выражения могут использоваться для решения широкого спектра задач, связанных с обработкой текста и данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb_1.c

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

#define EXTRA_BUF 50

char* readSentence(){
    char *tmp = (char*)malloc(sizeof(char));
    char *sentence;
    if (tmp) {
        sentence = tmp;
        tmp = NULL;
    }
    char c;
    int capacity = 1;
    int len = 0;
    while((c = getchar())){
        if(len >= capacity){
            capacity += EXTRA_BUF;
            tmp = (char*)realloc(sentence, capacity*sizeof(char));
            if (tmp){
                sentence = tmp;
                tmp = NULL;
            }
        }

        if (c == '.') {
            sentence[len] = '.';
            sentence[len + 1] = '\0';
            if (strcmp(sentence, "Fin.") == 0)
                return sentence;
            else {
                sentence[len + 1] = ' ';
            }
        }

        if (len >= 1 || (c != ' ' && c != '\t' && c != '\n'))
            sentence[len++] = c;

        if (c == '\n')
            break;
    }
    sentence[len] = '\0';
    return sentence;
}

char** readText(int *size){
    char **text = (char**)malloc(sizeof(char*));
    char *s;
```

```
    int count = 0;
    while((s = readSentence())){
        text[count++] = s;
             text  =  (char**)realloc(text,  sizeof(char*)*(count+1));  //
увеличение массива для нового указателя
        if (strcmp(s, "Fin.") == 0)
            break;
    }
    *size = count;
    return text;
}

int main ()
{
    int size;
    char **text = readText(&size);
    char * regexString = "([a-z]+:\\/{2}|)(www\\.|)(([a-zA-Z0-9][a-zA-Z0-
9-]*\\.)+)([a-zA-Z0-9-]+)\\/([a-zA-Z0-9_-]+\\/)*([a-zA-Z0-9_-]+\\.[a-zA-
Z0-9_-]+)";
    size_t maxGroups = 8;

    regex_t regexCompiled;
    regmatch_t groupArray[maxGroups];

    if (regcomp(&regexCompiled, regexString, REG_EXTENDED))
    {
        printf("Can't compile regular expression\n");
        return 0;
    };
    int flag = 0;
    for (int k = 0;k < size; k++){
        if (regexec(&regexCompiled, text[k], maxGroups, groupArray, 0) ==
0)
        {

            if (flag > 0) {
                printf("\n");
            }
            flag++;
            for(int j=groupArray[3].rm_so;j<groupArray[3].rm_eo;j++)
                printf("%c",text[k][j]);
            for(int j=groupArray[5].rm_so;j<groupArray[5].rm_eo;j++)
                printf("%c",text[k][j]);
            printf(" - ");
            for(int j=groupArray[7].rm_so;j<groupArray[7].rm_eo;j++)
                printf("%c",text[k][j]);

        }
        free(text[k]);
    }
    free(text);
    regfree(&regexCompiled);

    return 0;
}
```