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. 3341 _____ Моисеева А. Е.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Научиться работе с регулярными выражениями в языке Си и использовании их в программном коде.

Для этого необходимо:

- изучить теоретические сведения о регулярных выражениях, их синтаксис
- освоить библиотеку regex.h для работы с регулярными выражениями в языке Си
- научиться применять регулярные приложения в коде программы
- разработать программу, которая будет решать индивидуальные задачи по работе с текстом с помощью регулярных выражений

Задание

Вариант 2

На вход программе подается текст, представляющий собой набор предложений с новой строки. Текст заканчивается предложением "Fin." В тексте могут встречаться примеры запуска программ в командной строке Linux. Требуется, используя регулярные выражения, найти только примеры команд в оболочке суперпользователя и вывести на экран пары <имя пользователя> - <имя_команды>. Если предложение содержит какой-то пример команды, то гарантируется, что после нее будет символ переноса строки.

Примеры имеют следующий вид:

- Сначала идет имя пользователя, состоящее из букв, цифр и символа _
- Символ @
- Имя компьютера, состоящее из букв, цифр, символов _ и -
- Символ : и ~
- Символ $, если команда запущена в оболочке пользователя и #, если в оболочке суперпользователя. При этом между двоеточием, тильдой и $ или # могут быть пробелы.
- Пробел
- Сама команда и символ переноса строки.

Основные теоретические положения

Регулярные выражения – инструмент, использующийся для поиска и обработки текста по определённым шаблонам. Они позволяют описывать условия для поиска текстовых данных.

Регулярные выражения используются для выполнения таких задач, как поиск и замена текста в строках, проверка форматов данных (например, адресов электронной почты, номеров телефонов).

В языке Си для работы с регулярными выражениями используется библиотека regex.h. Её основные функции: regcomp() - компилирует регулярное выражение и готовит его к поиску, regexec() - выполняет поиск по регулярному выражению, regfree() - освобождает память, выделенную под скомпилированное регулярное выражение.

В синтаксисе регулярных выражений используются определённые специальные символы, обозначающие шаблоны для поиска выражений. Вот некоторые из них:

- ^ - начало строки
- $ - конец строки
- . - любой символ
- * - ноль или более повторений предыдущего символа
- + - один или более повторений предыдущего символа
- [abc] - любой символ из последовательности
- [^abc] - любой символ, кроме перечисленных.
- {n} - n повторений предыдущего символа
- | - оператор "или"

Выполнение работы

Подключаются библиотеки stdio.h, stdlib.h, string.h, regex.h для использования текстовых данных, стандартного ввода и вывода, регулярных выражений. С помощью define SIZE_BUFFER задаётся некоторая константа для работы с динамической памятью.

1. Char* input_text(int*)

Эта функция считывает текст из стандартного ввода по одному символу, сохраняя введенный текст в динамически выделенной памяти. Она продолжает считывание до тех пор, пока не встретит последовательность символов "Fin.". Каждый раз, когда вводится символ новой строки ('\n'), увеличивается счетчик предложений. Если буфер заполняется, его размер удваивается.

2. Char **text_sentences(char *, int)

Функция принимает строку текста и количество предложений, а затем разделяет этот текст на предложения с помощью strtok. Для каждого предложения динамически выделяется массив указателей на строки.

2. Void compile_and_match_regex(char**, int)

Функция компилирует регулярное выражение и проверяет каждое предложение на соответствие этому выражению. В случае нахождения совпадения выводится пара: <имя пользователя> - <имя команды>.

4. Void process_text()

Объединяет все предыдущие шаги: считывание текста, разбиение его на предложения и проверку каждого предложения на соответствие регулярному выражению. Очищает выделенную динамически память.

5. Int main()

Запускает функцию process_text и завершает программу.

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.	Run docker container: kot@kot-ThinkPad:~$ docker run -d --name stepik stepik/challenge-avr:latest You can get into running /bin/bash command in interactive mode: "/bin/bash" Switch user: su : root@84628200cd19: ~ # su box box@84628200cd19: ~ $ ^C Exit from box: box@5718c87efaa7: ~ $ exit exit from container: root@5718c87efaa7: ~ # exit kot@kot-ThinkPad:~$ ^C Fin.	root - su box root - exit	Строки root@84628200cd19: ~ # su box и root@5718c87efaa7: ~ # exit (после ~ идёт знак #) введены в оболочке суперпользователя - программа выводит имя пользователя и имя команды.
2.	chi9#@pi7: ~ # chapa chi9@pi#: ~ # chapa chi9@pi7: ~ $ chapa chi9@pi7: ~ # chapa chi9@pi_7: ~ # chapa Fin.	chi9 - chapa chi9 - chapa	Первая строка не подходит, поскольку имя пользователя не может содержать #, второе – так как имя компьютера не может содержать #, третья

			команда введена не в оболочке суперпользователя (после ~ идёт $), оставшиеся две команды подходят – выводится имя пользователя и команда.

Выводы

Цель работы достигнута. Изучен теоретический материал по синтаксису регулярных выражений, библиотеке regex.h и работы с регулярными выражениями в коде. В результате с помощью полученных знаний реализована программа, которая принимает на вход текст, представляющий набор предложений с новой строки и оканчивающийся строкой “Fin.”. Через регулярные выражения найдены строки текста, представляющие собой примеры команд в оболочке суперпользователя и выведены на экран пары в формате <имя пользователя> - <имя_команды>.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1.c

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

#define SIZE_BUFFER 1024

char *input_text(int *num_sentences) {
    char *text = malloc(SIZE_BUFFER * sizeof(char));
    int len = 0;
    char c;
    while (!(text[len-4] == 'F' && text[len-3] == 'i' && text[len-
2] == 'n' && text[len-1] == '.')) {
        c = getchar();
        text[len++] = c;
        if (c == '\n') {
            (*num_sentences)++;
        }
        if (len == SIZE_BUFFER) {
            text = realloc(text, SIZE_BUFFER*2);
            if (!text) return NULL;
        }
    }
    text[len - 4] = '\0';
    return text;
}

char **text_sentences(char *text, int num_sentences) {
    char **sentences = malloc(num_sentences * sizeof(char *));
    if (!sentences) return NULL;

    int i = 0;
    char *sentence = strtok(text, "\n");
    while (sentence != NULL) {
        sentences[i++] = sentence;
        sentence = strtok(NULL, "\n");
    }
    return sentences;
}

void compile_and_match_regex(char **sentences, int num_sentences){
    regex_t regex;
    int count = 0;
    const char *pattern = "([a-zA-Z0-9_]+)@([a-zA-Z0-9_-]+):[ +]?~[
+]?#[ +]?(.+)";
    regcomp(&regex, pattern, REG_EXTENDED);
    for (int i = 0; i < num_sentences; i++) {
        regmatch_t pmatch[4];
        if (regexec(&regex, sentences[i], 4, pmatch, 0) == 0) {
            if (count != 0){
```

```
                    printf("\n");
                }
                count++;
                for (int z = pmatch[1].rm_so; z < pmatch[1].rm_eo; z++)
{
                    printf("%c", sentences[i][z]);
                }
                printf(" - ");
                for (int w = pmatch[3].rm_so; w < pmatch[3].rm_eo; w++)
{
                    printf("%c", sentences[i][w]);
                }
            }
        }
        regfree(&regex);
    }

    void process_text() {
        int num_sentences = 0;
        char *text = input_text(&num_sentences);
        char **sentences = text_sentences(text, num_sentences);
        if (sentences == NULL) {
            free(text);
            return;
        }
        compile_and_match_regex(sentences, num_sentences);
        free(text);
        free(sentences);
    }

    int main() {
        process_text();
        return 0;
    }
```