

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

Перевалов П.И.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Целью работы является освоение работы с регулярными выражениями.

Для достижения поставленной цели требуется решить следующие задачи:

- изучить расширенные возможности форматного ввода/вывода в языке Си;
- ознакомиться с синтаксисом регулярных выражений;
- изучить способы применения POSIX регулярных выражения в языке Си;
- написать программу реализующую обработку и поиск подстрок по шаблону в тексте 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 – это последовательности символов, которые образуют шаблоны поиска в тексте. Они используются для поиска и сопоставления текстовой информации с определенным шаблоном. Регулярные выражения позволяют осуществлять более гибкий и мощный поиск, замену и обработку текста.

Регулярные выражения могут содержать специальные символы, которые обозначают определенные шаблоны символов. Например, символы . или * могут использоваться для обозначения любого символа или любого количества символов соответственно. Регулярные выражения могут также содержать группировку символов, квантификаторы, альтернативы и другие конструкции для более точного описания шаблонов поиска.

Основные элементы синтаксиса регулярных выражений:

1. Литералы: Обычные символы, которые должны точно соответствовать тексту. Например, abc будет соответствовать строке "abc".

2. Специальные символы: Специальные символы используются для создания шаблонов поиска. Например:

- .: соответствует любому одиночному символу, кроме символа новой строки.
- *: соответствует нулю или более вхождениям предыдущего элемента.
- +: соответствует одному или более вхождениям предыдущего элемента.
- ?: делает предыдущий элемент необязательным.

3. Наборы символов: Позволяют указать диапазон или набор символов, которые могут быть найдены в тексте. Например:

- [abc]: соответствует символам 'a', 'b' или 'c'.
- [a-z]: соответствует любому символу в диапазоне от 'a' до 'z'.

4. Группировка: Позволяет группировать части регулярного выражения для применения к ним операторов или квантификаторов. Например, (abc)+ соответствует одному или более повторениям строки "abc".

5. Альтернативы: Позволяют указать несколько альтернативных вариантов для поиска. Например, `cat|dog` будет соответствовать строкам "cat" или "dog".

В С для работы с регулярными выражениями обычно используется библиотека `regex.h`, которая предоставляет функции для компиляции и сопоставления регулярных выражений с текстом.

Выполнение работы

Были подключены библиотеки `stdlib.h`, `stdio.h`, `string.h`, `regex.h` для работы с динамической памятью, стандартным вводом/выводом, строками и регулярными выражениями. В переменной `pattern` задано регулярное выражение.

Объявлены функции `input`, `check`, `print`

1. `Int main()`

В функции объявлена переменная типа `regex_t regex_compiled`, в которой будет храниться скомпилированное регулярное выражение. Далее объявлена пустая строка, которая будет содержать значение вводимых строк. Каждая введенная строка, если она не “Fin.”, обрабатывается с помощью функции `check()` и на экран выводятся нужные данные, если они есть в строке (имя суперпользователя и команда).

2. `Void input (char**)`

На вход функции подается двойной указатель на строку. Далее в цикле производится ввод строки, при этом также есть условие на переполнение – если размер строки больше выделенной памяти – расширяем выделенную память вдвое. Также проверяем строку на соответствие “Fin.”. Если строка совпадает с конечной, выходим из цикла.

3. `Void check (char*, regex_t)`

Функция принимает на вход указатель на строку и переменную `regex_compiled` – скомпилированное регулярное выражение. Если строка соответствует регулярному выражению – она передается функции `print()`.

4. `Void print(char*, regmatch_t)`

Функция принимает на вход указатель на строку и выделенные группы. Она печатает на экран нужные нам пары вида <имя пользователя> - <имя команды>.

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.	<pre>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.</pre>	<pre>root - su box root - exit</pre>	Тест с сайта e.moevm

2.	sun*@set: ~ # flower sun@light: ~ # rise Fin.	sun – rise	Проверка граничного случая – первая строка не подходит, т.к. в имени пользователя есть *
3.	sun@set: ~ # flower sun@light: ~ \$ rise Fin.	sun – flower	Проверка граничного случая – вторая строка не подходит, т.к. команда введена в оболочке пользователя
4.	sun@set*: ~ # flower sun@light: ~ # rise Fin.	sun – rise	Проверка граничного случая – первая строка не подходит, т.к. в имени компьютера символ *
5.	sun@set: ~#flower sun@light: ~ # rise Fin.	sun - rise	Первая строка не подходит, т.к. нет пробелов между ~, # и командой

Выводы

Цель работы достигнута. Написана программа, которая принимает на вход текст, представляющий набор предложений с новой строки и оканчивающийся строков “Fin.”. В тексте с использованием регулярных выражений найдены примеры команд в оболочке суперпользователя и выведены на экран пары в формате <имя пользователя> - <имя_команды>. Путем использования библиотеки `regex.h`, программа компилирует заданное регулярное выражение и применяет его к вводу пользователя с целью нахождения соответствий. Найденные соответствия выводятся на экран в заданном формате.

Изучены регулярные выражения, синтаксис и рассмотрены примеры их использования в программе на языке Си.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

const char* REGEX_PATTERN = "(\\w+)@(\\w|-)+: ?~ ?# (.*)";

void get_text(char**);
void checking(char*, regex_t);
void output(char*, regmatch_t*);

void get_text(char** text) {
    int capacity = 1, len = 0;
    char symbol = getchar();
    while (symbol != '\\n') {
        if (len + 1 >= capacity) {
            capacity = capacity * 2;
            (*text) = (char*)realloc(*text, capacity * sizeof(char));
        }
        (*text)[len++] = symbol;
        (*text)[len] = '\\0';
        if (strcmp(*text, "Fin.") == 0) break;
        symbol = getchar();
    }
}

void checking(char* text, regex_t pattern) {
    regmatch_t group[4];
    if (regexec(&pattern, text, 4, group, 0) == 0) {
        output(text, group);
    }
}

void output(char* text, regmatch_t* group) {
    text[group[1].rm_eo] = '\\0';
    text[group[3].rm_eo] = '\\0';
    printf("%s - %s\\n", text + group[1].rm_so, text + group[3].rm_so);
}

int main() {
    regex_t pattern;
    regcomp(&pattern, REGEX_PATTERN, REG_EXTENDED);
    char* text = NULL;
    get_text(&text);
    while (strcmp(text, "Fin. ")) {
        checking(text, pattern);
        get_text(&text);
    }
    return 0;
}
```

