**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 |  | Мурдасов М.К. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Изучение регулярных выражений на языке Си и применение их на практике.

## 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.

## Выполнение работы

Были импортированы библиотеки <*stdio.h>* для работы с вводом-выводом данных, <*stdlib.h*> для работы с памятью, *<string.h>* для работы со строками и *<regex.h>* для использования регулярных выражений.

Для начала была выделена память для входного текста и написан цикл, производящий считывание. Цикл считывает каждую строку с помощью *fgets()* и сохраняет ее в переменную *string*. Строка дописывается в общий текст с помощью *strcat()*, после чего проверяется на соответствие финальной строке “Fin.” с помощью *strcmp()*. Если строка является завершающей, то цикл останавливается. Также была добавлена возможность расширения объема памяти для хранения текста на случай большего кол-ва символов в тексте, чем предусмотрено изначально.

Была создана переменная *regex* типа *regex\_t*, которая используется для хранения скомпилированного регулярного выражения. В переменную type типа *const char\** было записано само регулярное выражение, соответствующее условию, с выделением двух ключевых групп: имя пользователя и команда.

Далее с помощью функции *regcomp()* регулярное выражение *type* сохраняется в переменной *regex*, при этом используя расширенный синтаксис *POSIX* благодаря флагу *REG\_EXTENDED*. При ошибке компиляции информация об ошибке выводится в стандартный поток вывода ошибок.

Используя *strtok()* и *regexec()*, программа по очереди проверяет каждое предложение на соответствие регулярной строке. А именно, создается массив *groupArray* типа *regmatch\_t*, который будет содержать информацию о совпадениях групп. Далее функцией *regexec()* проверяется соответствие текущей строки регулярному выражению. Информация о совпадениях групп сохраняется в *groupArray* и будет использоваться при выводе.

Для вывода имени пользователя и команды подходящей строки в качестве аргументов к *printf()* используются адреса на соответствующие группы в строке и их размеры, полученные с помощью значений *rm\_so* и *rm\_eo*, дающих начало и конец совпадения соответственно относительно начала строки.

В конце программы освобождается память, выделенная для текста и с помощью *regfree()* освобождается память, выделенная функцией *regcomp()* при компилировании регулярного выражения

Таким образом, программа проанализировала каждую строку, и вывела искомые данные из каждой подходящей.

**Тестирование**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | root@84628200cd19: ~ # su box | root - su box | Корректно |
|  | root@5718c87efaa7: ~ # exit | root - exit | Корректно |

## Выводы

Были изучены основы работы с регулярными выражениями и их особенностями. С использованием полученных знаний была написана программа, выделяющая среди входных строк нужные и выводящая на экран только искомые данные.

# Приложение А Исходный код программы

Название файла: Murdasov\_Mikhail\_lb1.c

#include <stdio.h>

#include <stdlib.h>

#include <regex.h>

#include <string.h>

int main() {

char\* text = (char\*)malloc(sizeof(char)\*10000);

int text\_size = 0;

int text\_capacity = 10000;

while(1){

char string[10000];

fgets(string, 10000, stdin);

strcat(text, string);

text\_size+=strlen(string)\*sizeof(char);

if(text\_size >= text\_capacity-1000){

text\_capacity += 10000;

text = (char\*)realloc(text, sizeof(char)\*text\_capacity);

}

if(strcmp(string, "Fin.") == 0){

break;

}

}

regex\_t regex;

const char\* type = "([a-zA-Z0-9\_]+)@[a-zA-Z0-9\_-]+:[ ]\*~[ ]\*# (.+)";

if(regcomp(&regex, type , REG\_EXTENDED)){

fprintf(stderr, "Error: Regular expression compilation failed.");

}

int max\_groups = 3;

char\* token = (char\*)strtok(text, "\n");

while(token){

regmatch\_t groupArray[max\_groups];

if(regexec(&regex, token, max\_groups, groupArray, 0) == 0){

printf("%.\*s - %.\*s\n", (int)(groupArray[1].rm\_eo - groupArray[1].rm\_so), &token[groupArray[1].rm\_so], (int)(groupArray[2].rm\_eo - groupArray[2].rm\_so), &token[groupArray[2].rm\_so]);

}

token = (char\*)strtok(NULL, "\n");

}

free(text);

regfree(&regex);

return 0;

}