**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. 3342 |  | Епонишникова А.И |
| Преподаватель |  | Глазунов С.А |

Санкт-Петербург

2024

## Цель работы

Целью работы является на практике изучить работу с регулярными выражениями, а также осуществление программы с их использованием.

## Задание

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 слеующий 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

Создается регулярное выражение(regex\_string), количество групп, на которое разбито выражение(max\_groups). Происходит компиляция регулярного выражения.

Далее происходит ввод текст в одномерный массив, ввод заканчивается, когда в тексте присутствует “Fin.”

Затем происходит разделение текста на двумерный массив, отделяется отдельное предложение, и оно проверяется на соответствие регулярному выражению. Если данное предложение подошло, то посимвольно выводится сначала первая группа(имя пользователя), потом третья(команда) таким же образом.

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
|  | 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:  kot@kot-ThinkPad:~$ docker  exec -it stepik "/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 |

## Выводы

На практике научились работать с регулярными выражения, а также их применение в языке программирования Си.

# Приложение А Исходный код программы

Название файла: lab1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <regex.h>

#define BLOCK\_SIZE 10

#define END\_OF\_TEXT "Fin."

int main(){

char\* regex\_string = "([A-Za-z0-9\_]+)@([A-Za-z0-9\_-]+):\\s\*~\\s\*#\\s(.+)";

size\_t max\_groups = 4;

regex\_t regex\_compiled;

regmatch\_t group\_array[max\_groups];

regcomp(&regex\_compiled, regex\_string, REG\_EXTENDED);

int capacity = BLOCK\_SIZE;

char symbol = getchar();

char\* input\_text = (char\*)malloc(capacity);

if(input\_text == NULL){

printf("Memory error");

exit(0);

}

int idx = 0;

while(1){

input\_text[(idx)++] = symbol;

if(idx == capacity - 1){

capacity += BLOCK\_SIZE;

input\_text = (char\*)realloc(input\_text,capacity);

if(input\_text == NULL){

printf("Memory error");

exit(0);

}

}

symbol = getchar();

if(strstr(input\_text, END\_OF\_TEXT)){

break;

}

}

int sentences = 0;

int amount\_of\_char = 0;

int max\_amount\_of\_char = 0;

int amount\_of\_str = 0;

for(int i = 0; i<idx; i++){

if(input\_text[i] == '\n'){

sentences++;

amount\_of\_char++;

if(amount\_of\_char>max\_amount\_of\_char){

max\_amount\_of\_char = amount\_of\_char;

}

amount\_of\_char = 0;

}

else{

amount\_of\_char++;

}

}

char\*\* text = (char\*\*)malloc(sizeof(char\*) \* (sentences+1));

if(text == NULL){

printf("Memory error");

exit(0);

}

for(int i = 0; i<sentences; i++){

text[i] = (char\*)malloc(sizeof(char) \* (max\_amount\_of\_char+2));

if(text[i] == NULL){

printf("Memory error");

exit(0);

}

}

int chars = 0;

for(int i = 0; i<idx; i++){

if(input\_text[i] == '\n'){

text[amount\_of\_str][chars] = '\n';

text[amount\_of\_str][chars+1] = '\0';

if (regexec(&regex\_compiled, text[amount\_of\_str], max\_groups, group\_array, 0) == 0){

for (int i = group\_array[1].rm\_so; i < group\_array[1].rm\_eo; ++i){

printf("%c",text[amount\_of\_str][i]);

}

printf(" - ");

for (int i = group\_array[3].rm\_so; i < group\_array[3].rm\_eo; ++i){

printf("%c",text[amount\_of\_str][i]);

}

}

chars = 0;

}

else{

text[amount\_of\_str][chars] = input\_text[i];

chars++;

}

}

free(input\_text);

for(int i = 0; i<amount\_of\_str; i++){

free(text[i]);

}

free(text);

regfree(&regex\_compiled);

return 0 ;

}