**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. 3342 |  | Романов Е.А. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Освоение работы с регулярными выражениями, их использования в языке программирования C.

## 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

Программа состоит из 2 функций, включая функцию main. Функция read\_text используется для считывания текста из стандартного потока ввода, и его записи в двумерный динамический массив типа char. При помощи цикла while функция посимвольно считывает текст, подаваемый на вход программе и записывает его в одномерные динамические массивы типа char. Считывание завершается, когда полученное функцией предложение идентично предложению “Fin.”. В результате read\_text возвращает указатель на указатель типа char.

В функции main объявляется переменная, хранящая строку с регулярным выражением. При помощи функций стандартной библиотеки это выражение компилируется и записывается в переменную regex\_сompiled. После чего в цикле for программа проверяет соответствие считанных из стандартного потока ввода предложений регулярному выражению, в случае соответствия создаётся переменная, groups\_content в которой хранятся части предложения, соответствующие группам регулярного выражения. В результате соответствующие группы, согласно условию задания, выводятся на экран: программа выводит пользователя, от имени которого совершается команда и текст этой команды.

Переменные используемые в программе:

-int amount\_of\_sentences хранит количество предложений в тексте

-char \*\*text хранит указатели на предложения текста

-regex\_t regex\_compiled хранит скомпилированное регулярное выражение

-regmatch\_t хранит смещение в строке начала подстроки и смещение в строке конца подстроки.

-char \*\*groups\_content хранит подстроки, соответствующие группам регулярного выражения.

Функции стандартной библиотеки, используемые в программе:

-calloc выделяет память под заданное количество блоков

-free освобождает выделенную память

-getchar возвращает символ из стандартного потока ввода

-realloc изменяет размер динамически выделенной области памяти

-regcomp компилирует регулярное выражение

-printf выводит значения через стандартный поток вывода

-regexec проверяет строку на соответствие регулярному выражению

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

В ходе выполнения лабораторной работы были освоены регулярные выражения и группы. А также реализована программа на языке C, в которой на практике применяются полученные навыки.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <regex.h>

#define SUPERUSER\_MODE\_FLAG "#"

#define END\_OF\_TEXT "Fin."

char \*\*read\_text(int \*num\_of\_sentences){

char \*\*text;

int sentence\_size = 0, sentence\_capacity = 1;

text = calloc(1,sizeof(char\*));

if (text == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

text[\*num\_of\_sentences] = calloc(1,sizeof(char));

if (text[0] == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

char ch;

while (1) {

ch = getchar();

if (ch == '\n'){

text[\*num\_of\_sentences] = realloc(text[\*num\_of\_sentences], (sentence\_capacity+1) \* sizeof(char));

if (text[\*num\_of\_sentences] == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

text[\*num\_of\_sentences][sentence\_size] = '\0';

(\*num\_of\_sentences)++;

sentence\_size=0;

sentence\_capacity = 1;

text = (char\*\*)realloc(text, ((\*num\_of\_sentences)+1)\*sizeof(char\*));

if (text == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

text[\*num\_of\_sentences] = calloc(1,sizeof(char));

if (text[\*num\_of\_sentences] == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

continue;

}

if (sentence\_size >= sentence\_capacity) {

sentence\_capacity \*= 2;

text[\*num\_of\_sentences] = realloc(text[\*num\_of\_sentences], sentence\_capacity \* sizeof(char));

if (text[\*num\_of\_sentences] == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

}

text[\*num\_of\_sentences][sentence\_size] = ch;

sentence\_size++;

if (ch =='.'){

if(strstr(text[\*num\_of\_sentences], END\_OF\_TEXT) != NULL){

text[\*num\_of\_sentences] = realloc(text[\*num\_of\_sentences], (sentence\_size) \* sizeof(char));

if (text[\*num\_of\_sentences] == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

text[\*num\_of\_sentences][sentence\_size] = '\0';

return text;

}

}

}

}

int main(){

int amount\_of\_sentences=0;

char\*\* text = read\_text(&amount\_of\_sentences);

size\_t max\_groups\_amount = 5;

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

regex\_t regex\_compiled;

regmatch\_t groups\_array[max\_groups\_amount];

if(regcomp(&regex\_compiled, regex\_str, REG\_EXTENDED)){

fprintf(stderr, "Something went wrong");

return 0;

}

for (int t =0; t<amount\_of\_sentences;t++){

if (regexec(&regex\_compiled, text[t], max\_groups\_amount, groups\_array, 0) == 0) {

char \*\*groups\_content =calloc(max\_groups\_amount, sizeof(char\*));

if (groups\_content == NULL) return 0;

for(size\_t r=0;r<max\_groups\_amount;r++) {

groups\_content[r] = calloc(100, sizeof(char));

if (groups\_content[r] == NULL) {

fprintf(stderr,"Error of memory allocation");

exit(1);

}

}

for (int i = 0; i < max\_groups\_amount; i++) {

if (groups\_array[i].rm\_so == -1) break;

for(int j=groups\_array[i].rm\_so;j<groups\_array[i].rm\_eo;j++)

groups\_content[i][j-groups\_array[i].rm\_so] = text[t][j];

}

if (strcmp(groups\_content[3], SUPERUSER\_MODE\_FLAG) == 0){

printf("%s - %s\n", groups\_content[1], groups\_content[4]);

}

for (int i=0;i<max\_groups\_amount;i++) {

free(groups\_content[i]);

}

free(groups\_content);

}

}

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

free(text[i]);

}

free(text);

regfree(&regex\_compiled);

return 0;

}