**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 |  | Иванов Д. М. |
| Преподаватель |  | Глазунов С. А. |

Санкт-Петербург

2024

## Цель работы

Изучить синтаксис регулярных выражений и научиться использовать их в языке C с помощью заголовочного файла regex.h. Реализовать программу, которая принимает на вход команды пользователя в терминале линукс и выводит нужные из них.

## 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

Программа разбита на отдельные функции, каждая из которых выполняет определенные задачи:

void memory\_error() – вывод ошибки выделения памяти

char\*\* read\_text(char\*\* text) – считывание текста до предлоэения “Fin.”. Подается на вход динамических массив строк text. Через getchar() происходит считвание символов, их разбивка на предложения и сохранение в массив. В случае переполнения выполняется перевыделение памяти realloc. Функция возвращает полученный массив.

void find\_commands(char\*\* text) – с помощью цикла обрабатывается каждое отдельное предложение текста. Через команду regexec оно сравнивается с регулярным выражением. Само регулярное выражение: "([a-zA-Z0-9\_]+)@[a-zA-Z0-9\_-]+:[ ]\*[~][ ]\*[#] (.\*)”

1. ([a-zA-Z0-9\_]+) – имя пользователя из букв([a-zA-Z]), цифр([0-9]) и символа \_. + показывает, что символов будет один или более. Нам нужна будет эта информация, так что добавим в группу.
2. @ - дальше идет “собака”
3. [a-zA-Z0-9\_-]+ - то же самое, что и в п. 1. С добавлением символа –
4. :[ ]\*[~]\* - может быть неограниченное количество пробелов, так что поставим \*
5. [#] – нам нужны команды суперпользователя
6. (.\*) – дальше идут любые символы. Тоже добавим их в группу.

После того, как мы нашли нужные команды, через группировку и переменную regmatch\_t matchptr[m] находим индексы нужных символов и выводим их.

Константы:

* TEXT\_MAX\_SIZE – начальный размер предложений
* NUMBER\_OF\_SENTENCES – начальное количество предложений
* REGULAR\_EXPRESSION – регулярное выражение

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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 | Верный вывод |

## Выводы

Была разработана программа, выполнящая считывание команд и вывод нужных из них в соответствии с правилами. Для упрощения этой задачи было составлено регулярное выражение, подключенное с помощью regex.h к С.

# Приложение А Исходный код программы

Название файла: main.c

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <regex.h>

#define TEXT\_MAX\_SIZE 100

#define NUMBER\_OF\_SENTENCES 10

#define REGULAR\_EXPRESSION "([a-zA-Z0-9\_]+)@[a-zA-Z0-9\_-]+:[ ]\*[~][ ]\*[#] (.\*)"

void memory\_error(){

fprintf(stderr, "Error with memory allocation!");

exit(1);

}

char\*\* read\_text(char\*\* text){

int overflow\_sent = 0;

int overflow\_symb = 0;

char\* sent = calloc(TEXT\_MAX\_SIZE, sizeof(char));

if (sent == NULL){

memory\_error();

}

int index = 0;

int len = 0;

char symbol;

while (1){

symbol = getchar();

sent[index++] = symbol;

if ((TEXT\_MAX\_SIZE + overflow\_symb) <= index){

overflow\_symb++;

sent = realloc(sent, (TEXT\_MAX\_SIZE + overflow\_symb) \* sizeof(char));

text[len] = realloc(sent, (TEXT\_MAX\_SIZE + overflow\_symb) \* sizeof(char));

if (sent == NULL || text[len] == NULL){

memory\_error();

}

}

if (strstr(sent, "Fin.") != NULL){

text[len++] = strdup(sent);

free(sent);

break;

}

if (symbol == '\n'){

sent[index - 1] = '\0';

text[len++] = strdup(sent);

overflow\_symb = 0;

index = 0;

free(sent);

sent = calloc(TEXT\_MAX\_SIZE, sizeof(char));

if (sent == NULL){

memory\_error();

}

}

if ((NUMBER\_OF\_SENTENCES + overflow\_sent) <= len){

overflow\_sent++;

text = realloc(text, (NUMBER\_OF\_SENTENCES + overflow\_sent) \* sizeof(char\*));

text[NUMBER\_OF\_SENTENCES + overflow\_sent - 1] = calloc(TEXT\_MAX\_SIZE, sizeof(char));

if (text == NULL || text[NUMBER\_OF\_SENTENCES + overflow\_sent - 1] == NULL){

memory\_error();

}

}

}

return text;

}

void find\_commands(char\*\* text){

regex\_t reegex;

size\_t m = 3;

regmatch\_t matchptr[m];

int value;

int flag = 0;

value = regcomp(&reegex, REGULAR\_EXPRESSION, REG\_EXTENDED);

for (int i = 0; strstr(text[i], "Fin.") == NULL; i++){

value = regexec(&reegex, text[i], m, matchptr, 0);

if (value == 0){

if (flag == 0){

flag = 1;

}

else{

printf("\n");

}

for (int x = 1; x < m; x++)

{

if (matchptr[x].rm\_so == -1)

break;

for(int j=matchptr[x].rm\_so;j<matchptr[x].rm\_eo;j++)

printf("%c",text[i][j]);

if (x == 1){

printf(" - ");

}

}

}

}

regfree(&reegex);

}

int main() {

char \*\*text = malloc(NUMBER\_OF\_SENTENCES \* sizeof(char\*));

if (text == NULL) {

memory\_error();

}

for(int i = 0; i < NUMBER\_OF\_SENTENCES; i++){

text[i] = calloc(TEXT\_MAX\_SIZE, sizeof(char));

if (text[i] == NULL) {

memory\_error();

}

}

text = read\_text(text);

find\_commands(text);

int k = 0;

while (strstr(text[k], "Fin.") == NULL){

free(text[k++]);

}

free(text[k]);

free(text);

}