**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. 3341 |  | Чинаева М.Р. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью работы является освоение работы с регулярными выражениями на языке C.

Для достижения поставленной цели требуется решить следующие задачи:

Ознакомиться с регулярными выражениями;

Разработать программу, которая будет использовать регулярные выражения для поиска примеров команд в оболочке суперпользователя во входном тексте и выводить на экран пары <имя пользователя> - <имя\_команды>.

## Задание

Вариант 4

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.

## Основные теоретические положения

Регулярные выражения — формальный язык, используемый в компьютерных программах, работающих с текстом, для поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

Метасимволы - зарезервированные специальные символы. Для использования метасимвола, как обычного литерала, необходимо экранировать его, для этого нужно поставить «\» непосредственно перед экронируемым метасимволом. Метасимволы разделяют на позиционные (начало/конец строки/текста и т. п.), пробельные, квантификаторы (например, количество вхождений), группирующие, объединяющие (для символьных классов).

## Выполнение работы

Создаётся макрос *#define STR\_SIZE 100*

Создаётся макрос *#define REGEX\_PATTERN "([A-Za-z0-9\_]+)@[A-Za-z0-9\_-]+: ?~ ?# (.\*)"*

Создаётся макрос *#define* END\_INPUT "Fin."

Подключаем заголовочные файлы *stdio.h, stdlib.h, regex.h* и *string.h*

Функции:

1. int main()

Компилируется регулярное выражение типа regex.t. Считывается первая введенная строка. Далее с помощью цикла while (strcmp(END\_INPUT, string) != 0) каждая строка проверяется на соответствие регулярному выражению. Перед считыванием каждой новой строки память из-под старой освобождается.

2. char\* input\_str()

С помощью функции calloc создается строка из 0. Далее с помощью цикла while (check) в созданную строку посимвольно записывается считываемая строка. Если заданной памяти не хватит, то с помощью функции realloc память под строку перевыделится. Выход из цикла происходит если check становится равным 0. Это происходит в двух случаях: 1. Считан символ перевода строки, а значит строка закончилась. 2. Введенная строка соответствует строке, символизирующей конец ввода текста. Функция возвращает указатель на строку.

3. void check\_string(char\* string, regex\_t regex)

Функция проверяет строку на совпадение с регулярным выражением. В случае совпадения выводит имя компьютера и команду в заданном формате.

4. void print\_matches(char\* string, regmatch\_t match)

Принимает на вход строку и нужную нам структуру. Посимвольно выводит на экран нужное нам совпадение.

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 |
|  | 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 | Тест e.moevm |
|  | jslfnl root@84628200cd19: ~ # su box  root@ 84628200cd19: ~ # su box  Fin. | root - su box | Проверка наличия символов перед именем пользователя |
|  | root@ 84628200cd19: ~ # su box  root%@ 84628200cd19: ~ # su box  Fin. | root - su box | Проверка наличия запрещенных символов в имени пользователя |
|  | root@ 84628200cd19: ~ # su box  root@84628200cd19: ~ # su box  Fin. | root - su box | Проверка регулярного выражения на нужное количество пробелов |

## Выводы

Цель работы успешно достигнута. Данная программа осуществляет использование регулярных выражений для поиска примеров команд в оболочке суперпользователя во входном тексте и вывода на экран пар <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, что позволяет пользователям быстро и эффективно получить необходимую информацию из входного текста.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdio.h>

#include <regex.h>

#include <stdlib.h>

#include <string.h>

#define STR\_SIZE 100

#define END\_INPUT "Fin."

#define REGEX\_PATTERN "([A-Za-z0-9\_]+)@[A-Za-z0-9\_-]+: ?~ ?# (.\*)"

char\* input\_str();

void check\_string(char\* string, regex\_t regex);

void print\_matches(char\* string, regmatch\_t match);

int main() {

regex\_t regex;

int check\_comp = regcomp(&regex, REGEX\_PATTERN, REG\_EXTENDED);

if (check\_comp) {

printf("Couldn't compile");

return 1;

}

char\* string = input\_str();

while (strcmp(END\_INPUT, string) != 0) {

check\_string(string, regex);

free(string);

string = input\_str();

}

return 0;

}

char\* input\_str() {

char\* string = (char\*)calloc(STR\_SIZE, sizeof(char));

int check = 1;

int number\_of\_symbol = 0;

while (check) {

if ((number\_of\_symbol + 1) % 100 == 98) {

string = (char\*)realloc(string, (number\_of\_symbol + STR\_SIZE + 3) \* sizeof(char));

}

char new\_symbol = getchar();

if (new\_symbol == '\n' || strcmp(END\_INPUT, string) == 0) {

check = 0;

}

else {

string[number\_of\_symbol] = new\_symbol;

number\_of\_symbol++;

}

}

return string;

}

void check\_string(char\* string, regex\_t regex) {

regmatch\_t matches[3];

if (regexec(&regex, string, 3, matches, 0) == 0) {

print\_matches(string, matches[1]);

printf(" - ");

print\_matches(string, matches[2]);

printf("\n");

}

}

void print\_matches(char\* string, regmatch\_t match) {

for (int i = match.rm\_so; i < match.rm\_eo; i++) {

printf("%c", string[i]);

}

}