**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. 3341 |  | Трофимов В.О. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Цель: Овладеть навыками работы с регулярными выражениями в языке программирования C для поиска и обработки команд в тексте, соответствующих определенному шаблону.

Задачи:

1. Изучить спецификацию задачи и необходимые паттерны регулярных выражений для поиска команд в тексте.

2. Научиться использовать функции библиотеки <regex.h> для работы с регулярными выражениями в C.

3. Написать программу на Си, которая будет считывать текст с входного потока и искать в нем команды, соответствующие заданному шаблону.

4. Настроить регулярные выражения для извлечения имени пользователя и имени команды из найденных примеров.

5. Реализовать вывод пар <имя пользователя> - <имя\_команды> на экран для каждого найденного соответствия.

6. Протестировать программу на различных текстовых данных, включая случаи с несколькими командами в одном предложении.

## 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 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 (Regular Expressions) - это мощный инструмент для работы с текстовой информацией, который позволяет искать, извлекать и изменять текст, соответствующий определенному шаблону. Эти шаблоны могут описывать различные правила поиска текста, включая шаблоны символов, группирование, квантификаторы и многое другое. Вот основные теоретические положения регулярных выражений:

1. Символы: Регулярные выражения могут содержать обычные символы (буквы, цифры, знаки препинания), которые являются литералами и должны точно совпадать с данными символами в тексте.

2. Специальные символы: Регулярные выражения также содержат специальные символы, которые представляют собой метасимволы и используются для задания шаблонов поиска. Некоторые общие специальные символы включают:”.” (точка) - совпадает с любым одиночным символом, кроме символа новой строки. “^” (в начале выражения) - указывает, что совпадение должно начинаться с указанного символа или шаблона. ”$” (в конце выражения) - указывает, что совпадение должно заканчиваться на указанный символ или шаблон. “[]” - набор символов, описывающий диапазон или набор символов для сопоставления. “()” - группировка символов для последующего использования в регулярном выражении.

3. Квантификаторы: Квантификаторы используются для указания количества повторений символов или групп символов. Некоторые распространенные квантификаторы включают: “\*” - 0 или более повторений;

“+” - 1 или более повторений; “?” - 0 или 1 повторение; “{n}” - ровно n повторений. “{n, m}” - от n до m повторений.

4. Логические операторы: Регулярные выражения могут содержать логические операторы для создания более сложных шаблонов поиска. Например, | используется для указания альтернативных вариантов поиска.

## Выполнение работы

В программе объявлены следующие функции:

1) char\* readText();

2) void findCommandsByRegExp(char\* str);

3) void splitText(char\* txt);

1. Функция char\* readText(); используется в качестве считывания текста до предложения Fin. и сохранение текста в динамически выделенной строке.

2. Функция void findCommandsByRegExp(char\* str) принимает строку str, ищет в ней совпадения с заданным регулярным выражением и выводит определенные группы символов из найденного совпадения. Регулярное выражение хранится в переменной regexString

3. Функция void splitText(char\* txt) разбивает входную строку txt на подстроки, используя разделительный символ "\n" (новая строка), и вызывает функцию findCommandsByRegExp() для каждой подстроки. Объявление переменных sep как разделительной строки "\n" и ptr как указателя, в котором хранится результат первого вызова функции strtok() для разбиения исходного текста по разделителю sep. Цикл while(ptr != NULL) для прохода по всем полученным подстрокам: внутри цикла вызов функции findCommandsByRegExp(ptr), которая проверяет на в текущей подстроке. Получение следующей подстроки путем вызова strtok() с аргументом NULL, для продолжения поиска.

В main к переменной char\* text присваивается значение функции readtext(), вызывается функция splitText(text).

## 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 |
|  | root@1: ~ # su boxer  root@2: ~ # leave  root@3: ~ # exittt  Fin. | root - su boxer  root - leave  root - exittt |

## Выводы

Цель работы была достигнута, освоена работа с регулярными выражениями в языке С и изучена теория про регулярные выражения. Получена программа способная считать текст до предложения “Fin.”, также она отбирает по регулярному выражению определённые группы, и выводит на экран пары <имя пользователя> - <имя\_команды>.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <regex.h>

char\* readText(){

int size = 0;

int capacity = 1;

int flag = 0;

char ch = getchar();

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

while (flag != 1){

if (size + 1 >= capacity){

capacity \*= 2;

char\* new\_buf = (char\*) realloc(text,capacity \* sizeof(char));

text = new\_buf;

}

if (ch == '.' && text[size - 1] == 'n' && text[size - 2] == 'i' && text[size - 3] == 'F'){

text[size] = ch;

flag = 1;

}

text[size++] = ch;

ch = getchar();

}

text[size] = '\0';

return text;

}

void findCommandsByRegExp(char\* str){

int maxGroups = 3;

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

regex\_t regexCompiled;

regmatch\_t groupArray[maxGroups];

regcomp(&regexCompiled, regexString, REG\_EXTENDED);

if (regexec(&regexCompiled, str, maxGroups, groupArray,0) == 0){

for (int i = 1; i < maxGroups; i++){

if (groupArray[i].rm\_so != -1){

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

if ((j == groupArray[1].rm\_eo - 1)){

printf("%c - ",str[j]);

}

if ((j == groupArray[2].rm\_eo - 1)){

printf("%c\n",str[j]);

}

if ((j != groupArray[1].rm\_eo - 1 ) && ((j != groupArray[2].rm\_eo - 1))) {

printf("%c",str[j]);

}

}

}

}

}

}

void splitText(char\* txt){

char\* sep = "\n";

char\* ptr;

ptr = strtok(txt,sep);

while(ptr != NULL){

findCommandsByRegExp(ptr);

ptr = strtok(NULL,sep);

}

}

int main(){

char\* text = readText();

splitText(text);

return 0;

}