**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 |  | Самокрутов А.Р. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

## Цель работы

Целью работы является освоение работы с регулярными выражениями.

Для достижения поставленной цели требуется решить следующие задачи:

* изучить расширенные возможности форматного ввода/вывода в языке Си;

5

* ознакомиться с синтаксисом регулярных выражений;
* изучить способы применения POSIX регулярных выражения в языке Cи;
* написать программу реализующую обработку и поиск подстрок по шаблону в тексте с помощью регулярных выражений.

## 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.

## 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 (*wildcard characters*). Для поиска используется строка-образец, состоящая из символов и метасимволов и задающая правило поиска. Регулярные выражения используются некоторыми текстовыми редакторами и утилитами для поиска и подстановки текста.

Регулярные выражения могут содержать специальные символы, которые обозначают определенные шаблоны символов. Например, символы . или \* могут использоваться для обозначения любого символа или любого количества символов соответственно. Регулярные выражения могут также содержать группировку символов, квантификаторы, альтернативы и другие конструкции для более точного описания шаблонов поиска.

В C для работы с регулярными выражениями обычно используется библиотека regex.h, которая предоставляет функции для компиляции и сопоставления регулярных выражений с текстом.

## Выполнение работы

С помощью директивы *include* подключены следующие библиотеки: *stdlib.h* — для работы с динамической памятью; *stdio.h* — для работы со стандартными потоками ввода-вывода; *string.h* — для работы со строками; *regex.h* — для работы с регулярными выражениями.

В глобальной переменной *regex\_pattern* задано регулярное выражение *"([a-zA-Z0-9\_]+)@[a-zA-Z0-9\_-]+: \*~ \*# (.\*)"*. Данное регулярное выражение начинается с группы, задающей имя пользователя, которая содержит один или более (квантификатор *+*) символ, который может быть латинской буквой, цифрой или символом \_; далее идёт символ @, за которым следует имя компьютера — один или более символов, которыми могут быть латинские буквы, цифры или символы \_ и -. Далее идут символы : *\*~ \*#* — указание на команду от имени суперпользователя, между символами тильды и решётки могут быть пробелы (квантификатор *\**). Затем идёт сама команда — последовательность любых символов *(.\*)*, объединённых в группу.

Объявлены функции *void input(char \*\*)*, *int end\_of\_text(char \*)*, *void check\_string(char \*, regex\_t)*, *void print\_group(char \*, regmatch\_t)*, *void output(char \*, regmatch\_t \*)*.

1. Функция *void input(char \*\*string)*:

Посимвольно считывает вводимую пользователем строку, выделяя динамически память для её хранения и записывает по адресу *char* *\*\*string*.

1. Функция *int end\_of\_text(char \*)*:

Проверяет, содержит ли считанная строка последовательность символов *"Fin."* — указание на конец текста. Если такая подстрока найдена, функция возвращает значение *1*, в ином случае — *0*.

1. Функция *check\_string(char \*string, regex\_t re)*:

Создаёт массив *regmatch\_t groups[]* для хранения групп в строке с совпадением. Далее проверяется выполнение регулярного выражения на строке с помощью функции *regexec()*, при совпадении вызывается функция вывода результата *void output(string, groups)*.

1. Функция *void print\_group(char \*string, regmatch\_t group)*:

Выводит группу *regmatch\_t group*, содержащуюся в строке *char \*string*.

1. Функция *void output(char \*string, regmatch\_t \*groups)*:

С помощью функции *void print\_group()* через дефис выводит первую и вторую группы — имя пользователя и команду.

В функции main() компилируется регулярное выражение и запускается цикл для ввода и проверки строк на совпадение. С помощью функций *free()* и *reg\_free()* освобождается динамически выделенная память.

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 |
|  | root@pc: ~ # cmd | root - cmd | Проверка базовой работоспособности программы |
|  | This is not a valid command |  | Проверка корректной работы программы на неверных входных данных |
|  | test@testpc:~#cmd1  test@testpc: ~ # cmd2  test@testpc: ~ # cmd3 | test - cmd1  test - cmd2  test - cmd3 | Проверка работы программы при отсутствии пробелов или их большом количестве в некоторых позициях |

## Выводы

В ходе работы была написана программа, которая находит во входном тексте с использованием регулярных выражений примеры команды в оболочке суперпользователя и выводит в стандартный поток вывода пары типа *<имя пользователя> - <команда>*.

Изучены регулярные выражения, синтаксис и рассмотрены примеры их использования в программе на языке Си.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <regex.h>

#define STRING\_TERMINATOR '\0'

#define TEXT\_TERMINATOR "Fin."

#define CHUNK 256

const char \*regex\_pattern = "([a-zA-Z0-9\_]+)@[a-zA-Z0-9\_-]+: \*~ \*# (.\*)";

void input(char \*\*);

int end\_of\_text(char \*);

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

void print\_group(char \*, regmatch\_t);

void output(char \*, regmatch\_t \*);

int main(void)

{

regex\_t re;

regcomp(&re, regex\_pattern, REG\_EXTENDED);

char \*string;

while(1) {

string = NULL;

input(&string);

if (end\_of\_text(string))

break;

check\_string(string, re);

if (string != NULL)

free(string);

}

regfree(&re);

return 0;

}

void input(char \*\*string)

{

size\_t size = 0, capacity = 0;

char ch = 0;

while (ch != '\n') {

ch = getchar();

while (size + 1 >= capacity) {

capacity += CHUNK;

\*string = (char \*)realloc(\*string, capacity \* sizeof(char));

}

(\*string)[size++] = ch;

(\*string)[size] = STRING\_TERMINATOR;

if (end\_of\_text(\*string))

break;

}

}

int end\_of\_text(char \*string)

{

return (strcmp(string, TEXT\_TERMINATOR) == 0);

}

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

{

regmatch\_t groups[re.re\_nsub + 1];

if (regexec(&re, string, re.re\_nsub + 1, groups, 0) == 0)

output(string, groups);

}

void print\_group(char \*string, regmatch\_t group)

{

for (size\_t i = group.rm\_so; i < group.rm\_eo; i++)

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

}

void output(char \*string, regmatch\_t \*groups)

{

print\_group(string, groups[1]);

printf(" - ");

print\_group(string, groups[2]);

}