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 Си;
- 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*.

2. 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*.

3. 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)*.

4. Функция *void print_group(char *string, regmatch_t group)*:

Выводит группу *regmatch_t group*, содержащуюся в строке *char *string*.

5. 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
1.	root@pc: ~ # cmd	root - cmd	Проверка базовой работоспособности программы
2.	This is not a valid command		Проверка корректной работы программы на неверных входных данных
3.	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]);
}
```