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 _____ Львов А.В.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью данной работы является ознакомление с регулярными выражениями и их реализацией на языке C.

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.

Выполнение работы

В начале компилируется регулярное выражение в соответствии с условием работы. В случае ошибки программа завершается.

Далее пользователем вводится текст, который сохраняется в переменную txt, представляющую собой структуру Text, которая, в свою очередь, хранит массив предложений, разделённых символом переноса строки и их количество.

В функции getText() сначала выделяется память для хранения предложений и для первого предложения. Далее при помощи цикла for производится чтение текста и запись результата в переменную result типа Text.

После того, как пользователь ввёл текст, с помощью цикла for перебираются все предложения. Если предложение удовлетворяет регулярному выражению, то выводится информация об имени пользователя и введённой им команде.

После завершения вывода информации память, выделенная для хранения регулярного выражения освобождается функцией regfree().

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	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

Выводы

Было проведено ознакомление с регулярными выражениями и функциями библиотеки regex.h языка C для работы с ними.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUF_SIZE 1024
#define EOT "Fin.\n"
#define PATTERN "([a-zA-Z0-9_]+)@[a-zA-Z0-9_-]+: ?~ ?# (.*)"

typedef struct Text{
    char ** text;
    int size;
} Text;

Text getText();

int main() {
    regex_t regex;
    size_t maxGroups = 3;
    regmatch_t groupArray[maxGroups];

    int err = regcomp(&regex, PATTERN, REG_EXTENDED);

    if (err != 0) {
        printf("Error!");
        return 0;
    }

    Text txt = getText();

    for (int i = 0; i < txt.size; i++) {
       err = regexec(&regex, txt.text[i], maxGroups, groupArray, 0);
        if (err == 0) {
           for (int j = groupArray[1].rm_so; j < groupArray[1].rm_eo;
j++) {
                printf("%c", txt.text[i][j]);
            }
            printf(" - ");
           for (int k = groupArray[2].rm_so; k < groupArray[2].rm_eo;
k++) {
                printf("%c", txt.text[i][k]);
            }
        }
        free(txt.text[i]);
    }
    free(txt.text);
    regfree(&regex);
    return 0;
}
```

```
Text getText() {
    Text result;
    char ** text = (char **)malloc(BUF_SIZE * sizeof(char *));
    if (text == NULL) {
        exit(1);
    }

    int size = 0;
    int currBuf = BUF_SIZE;

    char * currSentence = (char *)malloc(BUF_SIZE * sizeof(char));
    if (currSentence == NULL) {
        exit(1);
    }
          while    (fgets(currSentence,    BUF_SIZE,    stdin)    &&
strcmp(currSentence, EOT)) {
        text[size++] = strdup(currSentence);
        if (text[size - 1] == NULL) {
            exit(1);
        }
        if (size == BUF_SIZE - 1) {
            currBuf += BUF_SIZE;
            text = realloc(text, currBuf * sizeof(char *));
            if (text == NULL) {
                exit(1);
            }
        }
        free(currSentence);
        currSentence = (char *)malloc(BUF_SIZE * sizeof(char));
        if (currSentence == NULL) {
            exit(1);
        }
    }
    free(currSentence);

    result.text = text;
    result.size = size - 1;
    return result;
}
```