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. 3342 Епонишникова А.И

Преподаватель Глазунов С.А

Санкт-Петербург
2024

Цель работы

Целью работы является на практике изучить работу с регулярными выражениями, а также осуществление программы с их использованием.

Задание

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.

Выполнение работы

Создается регулярное выражение(regex_string), количество групп, на которое разбито выражение(max_groups). Происходит компиляция регулярного выражения.

Далее происходит ввод текст в одномерный массив, ввод заканчивается, когда в тексте присутствует “Fin.”

Затем происходит разделение текста на двумерный массив, отделяется отдельное предложение, и оно проверяется на соответствие регулярному выражению. Если данное предложение подошло, то посимвольно выводится сначала первая группа(имя пользователя), потом третья(команда) таким же образом.

Разработанный программный код см. в приложении А.

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

Выводы

На практике научились работать с регулярными выражения, а также их применение в языке программирования Си.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h>
#define BLOCK_SIZE 10
#define END_OF_TEXT "Fin."

int main(){
    char*      regex_string      =      "([A-Za-z0-9_]+)@([A-Za-z0-
9_-]+):\\s*~\\s*#\\s(.+)";
    size_t max_groups = 4;

    regex_t regex_compiled;
    regmatch_t group_array[max_groups];

    regcomp(&regex_compiled, regex_string, REG_EXTENDED);

    int capacity = BLOCK_SIZE;
    char symbol = getchar();
    char* input_text = (char*)malloc(capacity);
    if(input_text == NULL){
        printf("Memory error");
        exit(0);
    }
    int idx = 0;
    while(1){
        input_text[(idx)++] = symbol;
        if(idx == capacity - 1){
            capacity += BLOCK_SIZE;
            input_text = (char*)realloc(input_text,capacity);
            if(input_text == NULL){
                printf("Memory error");
           exit(0);
            }
        }
        symbol = getchar();
        if(strstr(input_text, END_OF_TEXT)){
            break;
        }
    }

    int sentences = 0;
    int amount_of_char = 0;
    int max_amount_of_char = 0;
    int amount_of_str = 0;
    for(int i = 0; i<idx; i++){
        if(input_text[i] == '\n'){
            sentences++;
            amount_of_char++;
            if(amount_of_char>max_amount_of_char){
                max_amount_of_char = amount_of_char;
            }
```

```
                amount_of_char = 0;
            }
            else{
                amount_of_char++;
            }
        }

        char** text = (char**)malloc(sizeof(char*) * (sentences+1));
        if(text == NULL){
            printf("Memory error");
            exit(0);
        }
        for(int i = 0; i<sentences; i++){
            text[i]        =        (char*)malloc(sizeof(char)        *
(max_amount_of_char+2));
            if(text[i] == NULL){
                printf("Memory error");
                exit(0);
            }
        }

        int chars = 0;
        for(int i = 0; i<idx; i++){
            if(input_text[i] == '\n'){
                text[amount_of_str][chars] = '\n';
                text[amount_of_str][chars+1] = '\0';
                if    (regexec(&regex_compiled,    text[amount_of_str],
max_groups, group_array, 0) == 0){
                    for   (int   i   =   group_array[1].rm_so;   i   <
group_array[1].rm_eo; ++i){
                        printf("%c",text[amount_of_str][i]);
                    }
                    printf(" - ");
                    for   (int   i   =   group_array[3].rm_so;   i   <
group_array[3].rm_eo; ++i){
                        printf("%c",text[amount_of_str][i]);
                    }
                }
                chars = 0;
            }
            else{
                text[amount_of_str][chars] = input_text[i];
                chars++;
            }
        }
        free(input_text);
        for(int i = 0; i<amount_of_str; i++){
            free(text[i]);
        }
        free(text);
        regfree(&regex_compiled);
        return 0 ;
    }
```