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 _____ Романов Е.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Освоение работы с регулярными выражениями, их использования в языке программирования 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

Программа состоит из 2 функций, включая функцию main. Функция read_text используется для считывания текста из стандартного потока ввода, и его записи в двумерный динамический массив типа char. При помощи цикла while функция посимвольно считывает текст, подаваемый на вход программе и записывает его в одномерные динамические массивы типа char. Считывание завершается, когда полученное функцией предложение идентично предложению “Fin.”. В результате read_text возвращает указатель на указатель типа char.

В функции main объявляется переменная, хранящая строку с регулярным выражением. При помощи функций стандартной библиотеки это выражение компилируется и записывается в переменную regex_compiled. После чего в цикле for программа проверяет соответствие считанных из стандартного потока ввода предложений регулярному выражению, в случае соответствия создаётся переменная, groups_content в которой хранятся части предложения, соответствующие группам регулярного выражения. В результате соответствующие группы, согласно условию задания, выводятся на экран: программа выводит пользователя, от имени которого совершается команда и текст этой команды.

Переменные используемые в программе:

-int amount_of_sentences хранит количество предложений в тексте

-char **text хранит указатели на предложения текста

-regex_t regex_compiled хранит скомпилированное регулярное выражение

-regmatch_t хранит смещение в строке начала подстроки и смещение в строке конца подстроки.

-char **groups_content хранит подстроки, соответствующие группам регулярного выражения.

Функции стандартной библиотеки, используемые в программе:

-calloc выделяет память под заданное количество блоков

-free освобождает выделенную память

-getchar возвращает символ из стандартного потока ввода

-realloc изменяет размер динамически выделенной области памяти

-regcomp компилирует регулярное выражение

-printf выводит значения через стандартный поток вывода

-regexec проверяет строку на соответствие регулярному выражению

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

В ходе выполнения лабораторной работы были освоены регулярные выражения и группы. А также реализована программа на языке C, в которой на практике применяются полученные навыки.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define SUPERUSER_MODE_FLAG "#"
#define END_OF_TEXT "Fin."

char **read_text(int *num_of_sentences){
    char **text;
    int sentence_size = 0, sentence_capacity = 1;
    text = calloc(1,sizeof(char*));

    if (text == NULL) {
        fprintf(stderr,"Error of memory allocation");
        exit(1);
    }

    text[*num_of_sentences] = calloc(1,sizeof(char));

    if (text[0] == NULL) {
        fprintf(stderr,"Error of memory allocation");
        exit(1);
    }

    char ch;
    while (1) {
        ch = getchar();
        if (ch == '\n'){
            text[*num_of_sentences]                                    =
realloc(text[*num_of_sentences], (sentence_capacity+1) * sizeof(char));

            if (text[*num_of_sentences] == NULL) {
                fprintf(stderr,"Error of memory allocation");
                exit(1);
            }

            text[*num_of_sentences][sentence_size] = '\0';

            (*num_of_sentences)++;
            sentence_size=0;
            sentence_capacity = 1;

            text               =               (char**)realloc(text,
((*num_of_sentences)+1)*sizeof(char*));

            if (text == NULL) {
                fprintf(stderr,"Error of memory allocation");
                exit(1);
            }
```

```
                text[*num_of_sentences] = calloc(1,sizeof(char));

                if (text[*num_of_sentences] == NULL) {
                    fprintf(stderr,"Error of memory allocation");
                    exit(1);
                }

                continue;
            }
            if (sentence_size >= sentence_capacity) {
                sentence_capacity *= 2;
                text[*num_of_sentences]                                    =
realloc(text[*num_of_sentences], sentence_capacity * sizeof(char));

                if (text[*num_of_sentences] == NULL) {
                    fprintf(stderr,"Error of memory allocation");
                    exit(1);
                }
            }
            text[*num_of_sentences][sentence_size] = ch;
            sentence_size++;

            if (ch =='.'){
                if(strstr(text[*num_of_sentences],    END_OF_TEXT)    !=
NULL){
                    text[*num_of_sentences]                                 =
realloc(text[*num_of_sentences], (sentence_size) * sizeof(char));

                    if (text[*num_of_sentences] == NULL) {
                        fprintf(stderr,"Error of memory allocation");
                        exit(1);
                    }

                    text[*num_of_sentences][sentence_size] = '\0';
                    return text;
                }
            }


        }
    }

    int main(){

        int amount_of_sentences=0;
        char** text = read_text(&amount_of_sentences);


        size_t max_groups_amount = 5;
        char       *regex_str       =       "([A-Za-z0-9_]+)@([A-Za-z0-
9_-]+)\\:\\s*\\~\\s*(\\$|\\#)\\s(.*)";
        regex_t regex_compiled;
        regmatch_t groups_array[max_groups_amount];

        if(regcomp(&regex_compiled, regex_str, REG_EXTENDED)){
            fprintf(stderr, "Something went wrong");
            return 0;
```

```
        }

        for (int t =0; t<amount_of_sentences;t++){
            if  (regexec(&regex_compiled,  text[t],  max_groups_amount,
groups_array, 0) == 0) {
                char     **groups_content     =calloc(max_groups_amount,
sizeof(char*));

                if (groups_content == NULL) return 0;

                for(size_t r=0;r<max_groups_amount;r++) {
                    groups_content[r] = calloc(100, sizeof(char));

                    if (groups_content[r] == NULL) {
                        fprintf(stderr,"Error of memory allocation");
                        exit(1);
                    }
                }

                for (int i = 0; i < max_groups_amount; i++) {
                    if (groups_array[i].rm_so == -1) break;

                    for(int
j=groups_array[i].rm_so;j<groups_array[i].rm_eo;j++)
                        groups_content[i][j-groups_array[i].rm_so]     =
text[t][j];
                }

                if  (strcmp(groups_content[3],  SUPERUSER_MODE_FLAG)  ==
0){
                    printf("%s      -      %s\n",      groups_content[1],
groups_content[4]);
                }

                for (int i=0;i<max_groups_amount;i++) {
                    free(groups_content[i]);
                }
                free(groups_content);
            }
        }
        for (int i=0;i<amount_of_sentences;i++){
            free(text[i]);
        }
        free(text);
        regfree(&regex_compiled);

        return 0;
    }
```