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 _____ Иванов Д. М.

Преподаватель _____ Глазунов С. А.

Санкт-Петербург
2024

Цель работы

Изучить синтаксис регулярных выражений и научиться использовать их в языке C с помощью заголовочного файла regex.h. Реализовать программу, которая принимает на вход команды пользователя в терминале линукс и выводит нужные из них.

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

Программа разбита на отдельные функции, каждая из которых выполняет определенные задачи:

void memory_error() – вывод ошибки выделения памяти

char** read_text(char** text) – считывание текста до предлоэения “Fin.”. Подается на вход динамических массив строк text. Через getchar() происходит считвание символов, их разбивка на предложения и сохранение в массив. В случае переполнения выполняется перевыделение памяти realloc. Функция возвращает полученный массив.

void find_commands(char** text) – с помощью цикла обрабатывается каждое отдельное предложение текста. Через команду regexec оно сравнивается с регулярным выражением. Само регулярное выражение: "([a-zA-Z0-9_]+)@[a-zA-Z0-9_-]+:[ ]*[~][ ]*[#] (.*)”

1. ([a-zA-Z0-9_]+) – имя пользователя из букв([a-zA-Z]), цифр([0-9]) и символа _. + показывает, что символов будет один или более. Нам нужна будет эта информация, так что добавим в группу.
2. @ - дальше идет “собака”
3. [a-zA-Z0-9_-]+ - то же самое, что и в п. 1. С добавлением символа –
4. :[ ]*[~]* - может быть неограниченное количество пробелов, так что поставим *
5. [#] – нам нужны команды суперпользователя
6. (.*) – дальше идут любые символы. Тоже добавим их в группу.

После того, как мы нашли нужные команды, через группировку и переменную regmatch_t matchptr[m] находим индексы нужных символов и выводим их.

Константы:

- TEXT_MAX_SIZE – начальный размер предложений
- NUMBER_OF_SENTENCES – начальное количество предложений

- REGULAR_EXPRESSION – регулярное выражение

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
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	Верный вывод

Выводы

Была разработана программа, выполнящая считывание команд и вывод нужных из них в соответствии с правилами. Для упрощения этой задачи было составлено регулярное выражение, подключенное с помощью regex.h к C.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>
#define TEXT_MAX_SIZE 100
#define NUMBER_OF_SENTENCES 10
#define          REGULAR_EXPRESSION          "([a-zA-Z0-9_]+)@[a-zA-Z0-
9_-]+:[ ]*[~][ ]*[#] (.*)"

void memory_error(){
    fprintf(stderr, "Error with memory allocation!");
    exit(1);
}

char** read_text(char** text){
    int overflow_sent = 0;
    int overflow_symb = 0;
    char* sent = calloc(TEXT_MAX_SIZE, sizeof(char));
    if (sent == NULL){
        memory_error();
    }
    int index = 0;
    int len = 0;
    char symbol;
    while (1){
        symbol = getchar();

        sent[index++] = symbol;

        if ((TEXT_MAX_SIZE + overflow_symb) <= index){
            overflow_symb++;
            sent = realloc(sent, (TEXT_MAX_SIZE + overflow_symb) *
sizeof(char));
            text[len] = realloc(sent, (TEXT_MAX_SIZE + overflow_symb) *
sizeof(char));
            if (sent == NULL || text[len] == NULL){
                memory_error();
            }
        }

        if (strstr(sent, "Fin.") != NULL){
            text[len++] = strdup(sent);
            free(sent);
            break;
        }

        if (symbol == '\n'){
            sent[index - 1] = '\0';
            text[len++] = strdup(sent);
            overflow_symb = 0;
            index = 0;
            free(sent);
```

```
            sent = calloc(TEXT_MAX_SIZE, sizeof(char));
            if (sent == NULL){
                memory_error();
            }
        }

        if ((NUMBER_OF_SENTENCES + overflow_sent) <= len){
            overflow_sent++;
            text = realloc(text, (NUMBER_OF_SENTENCES + overflow_sent)
* sizeof(char*));
            text[NUMBER_OF_SENTENCES   +   overflow_sent   -   1]   =
calloc(TEXT_MAX_SIZE, sizeof(char));
            if (text == NULL || text[NUMBER_OF_SENTENCES + overflow_sent
- 1] == NULL){
                memory_error();
            }
        }
    }
    return text;
}

void find_commands(char** text){
    regex_t reegex;
    size_t m = 3;
    regmatch_t matchptr[m];

    int value;
    int flag = 0;
    value = regcomp(&reegex, REGULAR_EXPRESSION, REG_EXTENDED);
    for (int i = 0; strstr(text[i], "Fin.") == NULL; i++){
        value = regexec(&reegex, text[i], m, matchptr, 0);
        if (value == 0){
            if (flag == 0){
                flag = 1;
            }
            else{
                printf("\n");
            }
            for (int x = 1; x < m; x++)
            {
                if (matchptr[x].rm_so == -1)
                    break;

                for(int j=matchptr[x].rm_so;j<matchptr[x].rm_eo;j++)
                    printf("%c",text[i][j]);
                if (x == 1){
                    printf(" - ");
                }
            }
        }
    }
    regfree(&reegex);
}

int main() {
    char **text = malloc(NUMBER_OF_SENTENCES * sizeof(char*));
    if (text == NULL) {
        memory_error();
```

```
    }
    for(int i = 0; i < NUMBER_OF_SENTENCES; i++){
        text[i] = calloc(TEXT_MAX_SIZE, sizeof(char));
        if (text[i] == NULL) {
            memory_error();
        }
    }
    text = read_text(text);
    find_commands(text);
    int k = 0;
    while (strstr(text[k], "Fin.") == NULL){
        free(text[k++]);
    }
    free(text[k]);
    free(text);
}
```