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 _____ Смирнова Е.С.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Цель работы — изучить и освоить работу с регулярными выражениями, а также реализовать программу с их использованием.

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 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.

Основные теоретические положения

regex.h – библиотека, используемая для работы с регулярными выражениями в Си программах.

regcomp() – функция, используемая для компиляции регулярного выражения в структуре *regex_t*.

regexec() – функция, выполняющая поиск соответствия между заданной строкой и регулярным выражением, скомпилированным с помощью функции *regcomp()*.

regfree() – функция, освобождающая память, выделенную для скомпилированного регулярного выражения.

strstr() – функция ищет подстроку в строке.

Выполнение работы

Программа создает регулярное выражение *(regex)*, количество групп, на которое разбито выражение *(regex_group)* и строку, содержащую регулярное выражение, затем компилирует его с помощью *regcomp()*.

Далее происходит ввод текста, заканчивающийся, когда в тексте присутствует предложение *“Fin.”,* что проверяется функцией *strstr().*

Затем каждое предложение проверяется на соответствие регулярному выражению с помощью *regexec()*. Если данное предложение подошло, то посимвольно выводится сначала первая группа (имя пользователя), потом вторая (команда) таким же образом.

После с помощью *regfree()* очищается память, выделенная на регулярное выражение.

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

Изучены и освоены работа с регулярными выражениями, а также реализована программу с их использованием.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: menu.c

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

int main()
{
	regex_t regex;
	regmatch_t regex_group[3];
	char*	pattern	=	"([_A-Za-z0-9]+)@[-_A-Za-z0-
9]+:\\s?~\\s?#\\s(.+\n)";
	regcomp(&regex, pattern, REG_EXTENDED);

	char sentence[1000];
	char fin_sentence[5] = "Fin.";

	while (strstr(sentence,fin_sentence) == 0)
	{
		fgets(sentence,1000,stdin);

		if (regexec(&regex, sentence, 3, regex_group, 0) == 0)
		{
			for	(int	i	=	regex_group[1].rm_so;	i	<
regex_group[1].rm_eo; i++)
			{
				printf("%c", sentence[i]);
			}
			printf(" - ");
			for	(int	j	=	regex_group[2].rm_so;	j	<
regex_group[2].rm_eo; j++)
			{
				printf("%c", sentence[j]);
			}
		}
	}
	regfree(&regex);
	return 0;
}
```