МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Программирование»

Тема: РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Студент гр. 3341 _____ Костромитин М.М

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

Цель работы

Написать программу, которая на вход принимает текст с примерами запуска программ в командной строке Linux, использует регулярные выражения для нахождения только примеров команд в оболочке суперпользователя и выводит на экран пары <имя пользователя> - <имя_команды>. В ходе написания программы научиться работать с модулем regex.h и составить регулярное выражения для данного задания. Научиться работать с такими функциями как regcomp, regexec, а также научиться отбирать определенные группы из строки, которая подходит регулярному выражению.

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

1. Подключены необходимые библиотеки для вывода и ввода данных, для работы с регулярными выражениями, для работы с памятью.

2.Задано необходимое регулярное выражение в переменной regexBase, а также константа STEP, которая используется для увеличения выделенной памяти для массива.

3. Внутри функции main компилируется необходимое регулярное выражения, и проверяется успешность его компилирования, при неудачной компиляции программы выведет в поток вывода ошибок текст ‘Couldn’t compile’ и завершиться.

4. Вызывается функция regexMatch, в которой вызывается функция fillString, которая динамически выделяет память для строки и заполняет ее из стандартного потока данных пока не встретит символ переноса строки, далее эта функция проверяет является ли строка равной строке ‘Fin.’, если да, то функция возвращает единицу и следовательно функция regexMatch прекращает работать, если же строка не равна строке “Fin.”, то функция regexMatch проверяет строку по имеющимся регулярному выражению, и если строка удовлетворяет этому выражению, то функция печатает необходимые для вывода группы из этой строки с помощью функции printRegexGroup.

5. В конце функции main с помощью стандартной функции regfree из библиотеки regex.h освобождается память отведенная под регулярное выражение.

Код программы – см. Приложение А.

Тестирование

Результаты тестирования представлены в табл. 1

Табл. 1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	Dfgfdgergerg Dgergere rerge rgfg fdg krtgtr@dffgdfg: ~$ cd ../ admin@dfgderhrn:~ # rm -rf /* Fin.	admin - rm -rf /*	Обычный тест.
2	root@dfgdg:~ # cd home dfgdf@sdgdg:~# bruh spike@sdgdg:~ # regfdb ermak@sdgdg: ~# regfdb dfgdfgerg@ffdfg:~ $ dfgbd erggf dergergf wefwgergebyth23g Fin.	root - cd home dfgdf - bruh	Проверка на пробелы между ‘~’ и ‘#’, а также между ‘:’ и ‘~’.
3	@dfgdg:~ # text root@dfgdg:~ # test koren@:~ # wooooow Fin.	root – test	Проверка на наличие имени пользователя и символов после знака ‘@’.

Выводы

В результате выполнения лабораторной работы была разработана программа на языке программирования C, которая использует регулярные выражения из библиотеки regex.h для извлечения примеров запуска команд в оболочке суперпользователя из введенного текста. Программа предоставляет возможность ввода текста с примерами команд в командной строке Linux, а затем выводит на экран пары <имя пользователя> - <имя команды> для найденных примеров команд в оболочке суперпользователя. В результате выполнения лабораторной работы можно сделать вывод, что регулярные выражения – полезный инструмент для нахождения необходимых шаблонов в строках, а также извлечения необходимых кусков из найденных совпадений, но также библиотека не является идеальной для нахождения шаблонов в тексте из-за не самой лучшей производительности, а также высокого уровня абстракции.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <regex.h>
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>


int fillString(char** string);
void regexMatch(regex_t regex);
void printRegexGroup(char* string, regmatch_t group);

#define STEP 2

const char* regexBase = "([a-zA-Z0-9_]+)@[a-zA-Z0-9_-]+: \?~ \?#
(.*)";


int main(){
    regex_t regex;
    int check = regcomp(&regex, regexBase, REG_EXTENDED);

     if (check)
     {
          fprintf(stderr, "Couldn't compile.");
          return 1;
     }

     regexMatch(regex);

     regfree(&regex);

    return 0;
}


int fillString(char** string){
     int capacity = STEP;
     int size = 0;
     *string = (char*)malloc(sizeof(char) * STEP);
     char symb;
     while ((symb = getchar()) != '\n' && symb != EOF)
     {
          if (size + 1 == capacity)
          {
               capacity *= STEP;
               char* buf = (char*)realloc(*string, sizeof(char) *
capacity);
               *string = buf;
          }
          (*string)[size++] = symb;
```

```
	}
	(*string)[size] = '\0';
	if (!strcmp(*string, "Fin.")){
		free(*string);
		return 1;
	}
	return 0;
}


void regexMatch(regex_t regex){
	char* string;
	while(!fillString(&string))
	{
		regmatch_t regexGroups[3];

		if (!regexec(&regex, string, 3, regexGroups, 0))
		{
			printRegexGroup(string, regexGroups[1]);
			printf(" - ");
			printRegexGroup(string, regexGroups[2]);
			printf("\n");
		}

		free(string);
	}

}


void printRegexGroup(char* string, regmatch_t group){
	for (int i = group.rm_so; i < group.rm_eo; i++)
	{
		printf("%c", string[i]);
	}
}
```