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. 3341 Мальцев К.Л.

Преподаватель Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является освоение работы с регулярными выражениями на языке C.

Для достижения поставленной цели требуется решить следующие задачи:

Ознакомиться с регулярными выражениями;

Разработать программу, которая будет использовать регулярные выражения для поиска примеров команд в оболочке суперпользователя во входном тексте и выводить на экран пары <имя пользователя> - <имя_команды>.

Задание

Вариант 4

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

Регулярные выражения (Regular Expressions) - это специальные последовательности символов, используемые для поиска и манипуляции текстовой информацией. Они представляют собой мощный инструмент для работы с текстом и позволяют осуществлять поиск, замену, извлечение и проверку текстовой информации.

Основные теоретические положения по регулярным выражениям включают в себя следующие концепции:

1. Метасимволы: Регулярные выражения содержат метасимволы, которые представляют собой специальные символы или комбинации символов, используемые для задания шаблонов поиска. Например, "." может соответствовать любому символу, а "*" может соответствовать нулю или более повторениям предыдущего символа.

2. Классы символов: Регулярные выражения позволяют определить классы символов, такие как цифры, буквы, пробелы и другие. Например, [0-9] соответствует любой цифре, а \w соответствует любой букве, цифре или символу подчеркивания.

3. Квантификаторы: Регулярные выражения могут содержать квантификаторы, которые определяют количество повторений предыдущего элемента. Например, "+" соответствует одному или более повторениям, а "?" соответствует нулю или одному повторению.

4. Группы и альтернативы: В регулярных выражениях можно создавать группы символов и использовать оператор "|" для создания альтернативных вариантов поиска.

5. Якоря: Регулярные выражения позволяют использовать якоря, такие как "^" (начало строки) и "$" (конец строки), для определения местоположения шаблона в тексте.

Выполнение работы

Ход работы по данному коду:

1. Подключены необходимые библиотеки: stdlib.h, stdio.h, string.h, regex.h, для работы с динамической памятью, стандартным вводом/выводом, строками и регулярными выражениями.

2. Задано регулярное выражение в переменной pattern.

3. Созданы функции solve, inputString, inputInterruption, checkString, printGroup.

4. Функция main запускает функцию solve.

5. Функция solve: компилирует регулярное выражение, запускает цикл для ввода строк с обработкой с помощью функций inputString и checkString.

6. Функция inputString читает ввод пользователя вводом посимвольно, используя динамическое выделение памяти при необходимости.

7. Функция inputInterruption проверяет, содержит ли строка "Fin.", и возвращает 1, если да, иначе 0.

8. Функция checkString принимает строку и регулярное выражение, и если выполнение регулярного выражения на строке успешно, выводит на экран пары <имя пользователя> - <имякоманды>.

9. Функция printGroup выводит найденную подстроку из регулярного выражения.

Этот код представляет собой небольшую программу, которая читает ввод пользователя, ищет в нем строки, удовлетворяющие условиям заданного регулярного выражения, и выводит на экран найденные соответствия в формате <имя пользователя> - <имякоманды>.

Разработанный программный код см. в приложении А.

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	тест e.moevm
2.	lmdlld root_@cd19: ~ # try1 lmlld rt@-: ~ # try2 lmlld rt-@-: ~ # try3 Fin.	Fin.root_ - try1 rt - try2	1) проверка наличия пробелов между знаком :, ~ и #. 2) проверка наличия символов до имени пользователя.

Выводы

Цель работы успешно достигнута. Данная программа осуществляет использование регулярных выражений для поиска примеров команд в оболочке суперпользователя во входном тексте и вывода на экран пар <имя пользователя> - <имя_команды>. Путем использования библиотеки regex.h, программа компилирует заданное регулярное выражение и применяет его к вводу пользователя с целью нахождения соответствий. Найденные соответствия выводятся на экран в заданном формате, что позволяет пользователям быстро и эффективно получить необходимую информацию из входного текста.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

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

void inputString(char**);
int inputInterruption(char*);
void checkString(char*, regex_t);
void checkString(char*, regex_t);
void printGroup(char*, regmatch_t);


int main() {
	regex_t regexCompiled;
	regcomp(&regexCompiled, pattern, REG_EXTENDED);

	while (1) {
		char* string = NULL;
		inputString(&string);
		checkString(string, regexCompiled);
		if (inputInterruption(string)) {
			break;
		}
	}
}

void inputString(char** string) {
	int size = 0, capacity = 0;
	char ch = 0;
	while (ch != '\n') {
		ch = getchar();
		while (size + 1 >= capacity) {
			if (capacity == 0) {
				capacity = 1;
			}
			capacity *= 2;
			(*string) = (char*) realloc(*string,
capacity*sizeof(char));
		}
		(*string)[size++] = ch;
		(*string)[size] = '\0';
		if (inputInterruption(*string)) break;
	}
}

int inputInterruption(char* string) {
	return (strcmp(string, "Fin.") == 0);
}
```

```
void checkString(char* string, regex_t regexCompiled) {
      regmatch_t groups[3];
      if (regexec(&regexCompiled, string, 3, groups, 0) == 0) {
            printGroup(string, groups[1]);
            printf(" - ");
            printGroup(string, groups[2]);
      }
}

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