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. 9304 _____ Жорже М.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

Цель работы.

Изучить регулярные выражения. Научиться составлять и применять их в языке программирования Си.

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 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.

Функция «r*ead_sentence()*» с помощью функции «*getchar()*» посимвольно считывает данные записывая их в строку, память под которую выделила функция «c*alloc()*», возвращает указатель на массив. В случае недостатка изначально выделенной памяти, с помощью функции «*realloc()*» выделяется увеличенный объём памяти. «s*tr*» – хранит в себе предложение (массив символов) полученное посимвольным считыванием, «*symb*» – хранит в себе символ из «*getchar()*», «*size*» – хранит в себе количество памяти выделенное под строку, count – индекс для массива «*str*». «*text*»– указатель на массив указателей на массивы символов. Создаём массив *text* и выделяем под него память. Затем с помощью цикла «*do_while()*» считываем текст до предложения «*Fin.*». Потом с помощью цикла *for* и функции *free()* очищается память из под предложений. В конце очищаем память из под массива указателей с помощью *free()*.

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 <user>: 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
2.	kot@kot-ThinkPad:~# docker run -d --name stepik stepik/challenge-avr:latest kot@kot-ThinkPad:~# docker exec -it stepik "/bin/bash" jq rqwrkerwkjhrwehr qwe@asd root@84628200cd19: ~ $ su box eqw q eqe box@84628200cd19: ~ # ^C @@@ box@5718c87efaa7: ~ # exit root@5718c87efaa7: ~ $ exit kot@kot-ThinkPad:~# ^C Fin.	kot - docker run -d — name stepik stepik/challenge-avr:latest kot - docker exec -it stepik "/bin/bash" box - ^C box - exit kot - ^C

Выводы.

Были изучены регулярные выражения. Получены навыки создания и применения регулярных выражений в языке Си.

Была разработана программа, которая находит примеры команд в оболочке суперпользователя и выводит пары вида <имя_пользователя> - <имя_команды> при помощи регулярных выражений и библиотеки <regex.h>.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h>
char *read_sentence()
{
   int size = 10;
   char *str = calloc(size,
sizeof(char));
   char symb = getchar();
   int count = 0;

   if (symb == ' ' || symb ==
'\t')
   {
      symb = getchar();
   }

   while (symb != '\n' &&
symb != '.')
   {
      if (count >= size)
      {
         size += size;
         str = realloc(str, size *
sizeof(char));
      }

      str[count] = symb;
      count = count + 1;
      symb = getchar();
   }

   if (symb != '\n')
   {
      str[count] = symb;
      str[count + 1] = '\0';
   }
   else
   {
      str[count] = '\0';
   }

   return str;
}

int main()
{
   int size = 10;
   char **text = calloc(size,
sizeof(char *));
   int sizeText = 0;
   char *str;
```

```
   do
   {
      str = read_sentence();

      if (sizeText >= size)
      {
         size += size;
         text = realloc(text,
size * sizeof(char *));
      }

      text[sizeText] = str;
      sizeText++;
   } while
(strcmp(text[sizeText - 1],
"Fin."));

   regex_t regexCompiled;
   size_t maxGroups = 3;
   regmatch_t
groupArray[maxGroups];

   if
(regcomp(&regexCompiled,
"(\\w+)@[A-Za-z0-9_-
]+:\\s?~\\s?# (.*)",
REG_EXTENDED))
   {
      printf("Regex not
compiled!\n");
      return 0;
   }

   for (int i = 0; i < sizeText;
i++)
   {
      if
(!regexec(&regexCompiled,
text[i], maxGroups,
groupArray, 0))
      {
         for (int j =
groupArray[1].rm_so; j <
groupArray[1].rm_eo; j++)
         {
            printf("%c",
text[i][j]);
         }

         printf(" - ");

         for (int k =
groupArray[2].rm_so; k <
groupArray[2].rm_eo; k++)
         {
            printf("%c",
text[i][k]);
```

```
            }

            printf("\n");
        }
    }

    regfree(&regexCompiled);

    for (int i = 0; i < sizeText;
i++)
    {
        free(text[i]);
    }

    free(text);

    return 0;
}
```