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. 9383 _____ Корсунов А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Изучить регулярные выражения, освоить библиотеку regex.h.

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.

Описание функций:

1) **InputSentence();**

Данная функция принимает на вход текст (посимвольно) с помощью функции getchar() пока не будет встречен символ перевода строки или предложение «Fin.». Предложение сохраняется в массив символов Sentence типа *char, функция возвращает этот массив.

2) **InputText(int *count);**

Данная функция на вход принимает указатель *int (он нужен, чтобы выяснить, сколько всего поступило предложений на вход). Сама же функция вызывает функцию 1), чтобы сохранить предложения в массив предложений типа **char. Функция работает, пока не встретит предложение «Fin.». Возвращает созданный ранее массив.

3) **main();**

В функции main() происходит вызов функции 2), чтобы работать с веденным текстом. Далее создается массив символов типа char, в котором записано регулярное выражение. Затем с помощью функций regcomp() и regexec() происходит соответственно компиляция регулярного выражения и сравнение предложений с этим выражением, попутно выводя на экран нужные по заданию данные. В конце происходит очистка памяти.

Тестирование.

№ 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	root - su box root - exit

	Exit from box: box@5718c87efaa7: ~ $ exit exit from container: root@5718c87efaa7: ~ # exit kot@kot-ThinkPad:~$ ^C Fin.	
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

Вывод.

В ходе проделанной работы были изучены регулярные выражения и работа с ними. Разработана требуемая по заданию программа. Изучена библиотека regex.h.

Приложение А
Исходный код программ

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

char* InputSentence();
char** InputText(int *count);

int main()
{
   int n = 0;
   char** SentenceS = InputText(&n);
   char* regexs = "([A-Za-z0-9_]+)@[A-Za-z0-9_-]+: ?~ ?# (.+)\n$";
   size_t maxGroups = 3;
   regmatch_t Groups[maxGroups];
   regex_t regexCompiled;
   if(regcomp(&regexCompiled, regexs, REG_EXTENDED))
   {
      return 0;
   }
   for(int i = 0; i < n; i++)
   {
      if(regexec(&regexCompiled, SentenceS[i], maxGroups, Groups, 0) == 0)
      {
         for(int j = Groups[1].rm_so; j<Groups[1].rm_eo; j++)
         {
            printf("%c", SentenceS[i][j]);
         }
         printf(" - ");
```

```
            for(int j = Groups[2].rm_so; j<Groups[2].rm_eo; j++)
            {
               printf("%c", SentenceS[i][j]);
            }
            printf("\n");
         }
      }
      regfree(&regexCompiled);
      for(int i = 0; i < n; i++)
      {
         free(SentenceS[i]);
      }
      free(SentenceS);
      return 0;
}

char* InputSentence()
{
      int size = 15;
      int index = 0;
      char* Sentence = malloc(size * sizeof(char));
      char key = getchar();
      Sentence[index] = key;
      index++;
      while(key != '\n')
      {
         key = getchar();
         Sentence[index] = key;
         index++;
         if (strcmp(Sentence,"Fin.") == 0)
            break;
         if(index+1 == size)
         {
           size += 15;
           Sentence = realloc(Sentence, size * sizeof(char));
         }
      }
      return Sentence;
}

char** InputText(int *count)
{
      int index = 0;
      int size = 10;
```

```
    char** SentenceS = malloc(size * sizeof(char*));
    while(1)
    {
      SentenceS[index] = InputSentence();
      if(strcmp(SentenceS[index], "Fin.") == 0)
      {
         break;
      }
      index++;
      if(index+1 == size)
      {
         size += 10;
         SentenceS = realloc(SentenceS, size * sizeof(char*));
      }
    }
    *count = index;
    return SentenceS;
}
```