**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. 9381 |  | Колованов Р.А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2020

## Цель работы.

Реализация поиска подстрок, удовлетворяющих определенному шаблону, в тексте при помощи регулярных выражений; получение навыков работы с регулярными выражениями в языке 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.

Для удобства хранения, работы с текстом и предложениями были разработаны структуры *Sentence* и *Text*. Для ввода текста были написаны функции *readSentence()* и *readText()*, а после завершения работы с текстом для очищения выделенной динамической памяти написана функция *freeText()*.

Описание данных структур и функций приведено ниже:

* **Структура Sentence**

|  |  |
| --- | --- |
| Название и тип поля | Предназначение |
| *char\* characters* | Хранит адрес первого элемента массива символов. |
| *size\_t charactersNumber* | Хранит количество символов в предложении. |

* **Структура Text**

|  |  |
| --- | --- |
| Название и тип поля | Предназначение |
| *Sentence\* sentences* | Хранит адрес первого элемента массива экземпляров структур. |
| *size\_t sentencesNumber* | Хранит количество предложений в тексте. |

* **Функция readSentence()**

*Sentence readSentence();*

Позволяет считать одно предложение с потока *stdin*. Возвращает структуру *Sentence*.

Для начала создается экземпляр *sentence* структуры Sentence, который потом будет возвращен из функции. Происходит выделение динамической памяти для массива при помощи функции стандартной библиотеки *calloc()*. Указатель на участок выделенной памяти присваивается полю *characters* экземпляра sentence.

Также объявляется переменная *charactersMaxNumber* типа size\_t для хранения размера выделенного динамического массива. Далее происходит посимвольное считывание предложения при помощи функции *fgetc()* до тех пор, пока в потоке не встретится один из символов EOF и ‘\n’. При этом цикле происходит проверка, хватает ли в динамическом массиве места для хранения очередного считанного символа. Если нет, то происходит расширение участка выделенной памяти при помощи функции *realloc()*. После того, как было считано предложение, последний символ (‘\n’ или EOF) заменяется на нулевой символ.

В конце производится освобождение лишней памяти и возвращение предложения из функции.

* **Функция readText()**

*Text\* readText().*

Позволяет считать текст с потока *stdin*. Возвращает указатель на структуру *Text*.

Для начала выделяется память под экземпляр *text* структуры *Text*, указатель на который будет возвращен из функции. Выделение осуществляется при помощи функции *malloc()*. Также происходит выделение участка памяти под массив структур Sentence, указатель на который присваивается полю sentences экземпляра text. Далее объявляется переменная *size\_t sentencesMaxNumber* – хранит размер динамического массива предложений.

После идет цикл, в котором происходит считывание предложений при помощи функции *readSentence()* до тех пор, пока мы не встретим предложение *“Fin."* (означает конец текста). Сравнение предложений происходил при помощи функции *strcmp()*. При этом цикле происходит проверка, хватает ли в динамическом массиве места для хранения очередного считанного предложения. Если нет, то происходит расширение участка выделенной памяти.

В конце производится освобождение лишней памяти и возвращение текста из функции.

* **Функция freeText()**

*void freeText(Text\* text);*

Принимает на вход указатель на структуру Text и освобождает выделенную для нее динамическую память.

**Функция *main().***

Рассмотрим функцию *main()*. Для начала с помощью функции *readText()* с потока *stdin* считывается текст и возвращается экземпляр структуры *Text*.

Далее происходит вызов функции *findRootUserCommands()*, на вход которой подается считанный текст. Данная функция ищет в тексте примеры команд в терминале *Linux*, которые были выполнены в оболочке суперпользователя, при помощи регулярных выражений и выводит на экран пары <имя пользователя> - <имя\_команды>. Для начала в функции создается экземпляр структуры *regex\_t*, которая будет хранить скомпилированное регулярное выражение. При помощи функции *regcomp()* происходит компиляция следующего регулярного выражения: “(\\w+)@(\\w|-)+: ?~ ?# ((.)+)$”. Далее в цикле для каждого предложения в тексте производится поиск подходящей под регулярное выражение подстроки при помощи функции regexec(), на вход которой подается скомпилированное регулярное выражение, предложение, в котором будет производится поиск, а также массив экземпляров структуры *regmatch\_t*. В данную структуру будут записаны индексы начала и конца вхождения для групп регулярного выражения. Далее, если подстрока была найдена, то происходит вывод двух групп регулярного выражения: имя пользователя и команда соответственно. В конце происходит вызов функции *regfree()*, которая очищает выделенную под экземпляр структуры *regex\_t* динамическую память.

Разработанный программный код см. в приложении А.

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
|  | 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 |

## Выводы.

Было изучена работа с регулярными выражениями в языке Си; получены навыки работы с ними.

Разработана программа, которой на вход подается набор предложений, заканчивающихся переводом на новую строку. Программа ищет примеры команд в терминале *Linux*, которые были выполнены в оболочке суперпользователя, при помощи регулярных выражений и выводит на экран пары <имя пользователя> - <имя\_команды>.

Для осуществления поиска по регулярным выражениям были использованы функции *regcomp(), regexec()* и *regfree()* из библиотеки *regex.h*. Для хранения текста списка использовались функции выделения, изменения размера выделенного участка и освобождения динамической памяти *malloc(), calloc(), realloc() и free().* В программе использовались операторы if-else, циклы for и do-while, указатели и структуры.

# Приложение А Исходный код программы

Название файла: main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <regex.h>

typedef struct Sentence {

char\* characters;

size\_t charactersNumber;

} Sentence;

typedef struct Text {

Sentence\* sentences;

size\_t sentencesNumber;

} Text;

Sentence readSentence() {

size\_t charactersMaxNumber = 20;

Sentence sentence;

sentence.characters = calloc(charactersMaxNumber, sizeof(char));

sentence.charactersNumber = 0;

do {

if (sentence.charactersNumber >= charactersMaxNumber - 1) {

charactersMaxNumber += 20;

sentence.characters = realloc(sentence.characters, charactersMaxNumber \* sizeof(char));

}

sentence.characters[sentence.charactersNumber++] = fgetc(stdin);

} while (sentence.characters[sentence.charactersNumber - 1] != '\n' && sentence.characters[sentence.charactersNumber - 1] != EOF);

sentence.characters[sentence.charactersNumber - 1] = '\0';

if (sentence.charactersNumber + 1 < charactersMaxNumber) {

sentence.characters = realloc(sentence.characters, sentence.charactersNumber \* sizeof(char));

}

return sentence;

}

Text\* readText() {

Text\* text = malloc(sizeof(Text));

text->sentences = calloc(5, sizeof(Sentence));

text->sentencesNumber = 0;

size\_t sentencesMaxNumber = 5;

do {

if (text->sentencesNumber >= sentencesMaxNumber) {

sentencesMaxNumber \*= 2;

text->sentences = realloc(text->sentences, sentencesMaxNumber \* sizeof(Sentence));

}

text->sentences[text->sentencesNumber++] = readSentence();

} while (strcmp(text->sentences[text->sentencesNumber - 1].characters, "Fin.") != 0);

if (text->sentencesNumber < sentencesMaxNumber) {

text->sentences = realloc(text->sentences, text->sentencesNumber \* sizeof(Sentence));

}

return text;

}

void findRootUserCommands(Text\* text) {

regex\_t regexPattern;

regcomp(&regexPattern, "(\\w+)@(\\w|-)+: ?~ ?# ((.)+)$", REG\_EXTENDED);

for (size\_t i = 0; i < text->sentencesNumber - 1; i++) {

regmatch\_t regexMatch[4];

int matchResult = regexec(&regexPattern, text->sentences[i].characters, 4, regexMatch, 0);

if (matchResult == 0) {

for (size\_t j = regexMatch[1].rm\_so; j < regexMatch[1].rm\_eo; j++) {

printf("%c", text->sentences[i].characters[j]);

}

printf(" - ");

for (size\_t j = regexMatch[3].rm\_so; j < regexMatch[3].rm\_eo; j++) {

printf("%c", text->sentences[i].characters[j]);

}

printf("\n");

}

}

regfree(&regexPattern);

}

void freeText(Text\* text) {

for (int i = 0; i < text->sentencesNumber; i++) {

free(text->sentences[i].characters);

}

free(text->sentences);

free(text);

}

int main() {

Text \*text = readText();

findRootUserCommands(text);

freeText(text);

return 0;

}