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 _ Мохаммед А.А.

Преподаватель _ Чайка К.В.

Санкт-
Петербург
2020

Цель работы.

Научиться использовать регулярные выражения в языке программирования си.

Задание.

Вариант 1

На вход программе подается текст, представляющий собой набор предложений с новой строки. Текст заканчивается предложением "**Fin.**" В тексте могут встречаться ссылки на различные файлы в сети интернет. Требуется, используя регулярные выражения, найти все эти ссылки в тексте и вывести на экран пары <название_сайта> - <имя_файла>. Гарантируется, что если предложение содержит какой-то пример ссылки, то после ссылки будет символ переноса строки.

Ссылки могут иметь следующий вид:

- Могут начинаться с названия протокола, состоящего из букв и **://**

после

- Перед доменным именем сайта может быть **www**
- Далее доменное имя сайта и один или несколько доменов

более верхнего уровня

- Далее возможно путь к файлу на сервере
- И, наконец, имя файла с расширением

Пример входных данных:

This is simple url: http://www.google.com/track.mp3 May be more than one upper level domain http://www.google.com.edu/hello.avi Many of them. Rly.

Look at this! http://www.qwe.edu.etu.yahooo.org.net.ru/qwe.q Some other protocols ftp://skype.com/qqwe/qweqw/qwe.avi Fin.

Выполнение работы.

Сначала были созданы функции для считывания текста: readline и readtext. Далее программе на вход подается текст, заканчивающийся предложением «Fin.». Затем было скомпилировано регулярное выражение cFin.». Затем было скомпилировано регулярное выражение с помощью функции regcomp(). Далее запускается цикл for, в котором с помощью функции regexec(), запускается поиск подстроки по шаблону регулярного выражения. Далее происходит вывод групп регулярного выражения под индексами 3 и 7. В конце происходит очистка памяти под выделенный текст и скомпилированное регулярное выражение с помощью функции regfree().

Выводы.

Были изучены регулярные выражения. Создана программа поиска подстрок, с помощью регулярных выражений.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАМЫ

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<regex.h>
char* read_sentence(){
  char* sentence = malloc(1*sizeof(char));
  char current_char;
```

```
  int counter = 0;
  current_char = getchar();
  sentence[0]='\0';

  // remove leading spaces and tabs
  while(current_char==' ' || current_char=='\t'){
    current_char = getchar();
  }
  while(current_char!='\n'){
    sentence = realloc(sentence, (counter+2)*sizeof(char));
    sentence[counter] = current_char;
    counter++;
    sentence[counter] = '\0'; // string termination character
    current_char = getchar();
  }
  return sentence;
}

char** readText(char** text, int* text_length){
  int counter = 0;
  int sentence_length=0;
  int total_chars = 0;
  char* sentence = read_sentence();
  text = malloc(0*sizeof(char*));
  while(sentence!=NULL && strcmp(sentence, "Fin.")!=0){
    text = realloc(text, (counter+1)*sizeof(char*));
    text[counter] = malloc((strlen(sentence)+1)*sizeof(char));
    strcpy(text[counter], sentence);
    counter++;
```

```
   free(sentence);
   sentence = read_sentence();
  }
  *(text_length) = counter;
  return text;
}

void extract_print_links(char** text, int text_length){
  char delimiters[] = " ,!";
  char pattern[] =
"([[:alnum:]]+:\x2F\x2F)?(www\x2E)?([[:alnum:]]+\x2E)+[[:alnum:]]+(\x2F[[:alnum:]]+)+\x2E[
[:alnum:]]+"; // \x2F ==> forward slash /
            // \x2E ==> dot character .
            // [:alnum:] alphanumeric character class
  regex_t regex;

  if (regcomp(&regex, pattern, REG_EXTENDED)!=0) {
   printf("Error in Regular Expression :(\n");
   return;
  }
  // for each sentence
  for (int i=0; i<text_length; i++){
   char* copy_sentence = (char*)malloc((strlen(text[i])+1)*sizeof(char));
   copy_sentence = strcpy(copy_sentence, text[i]);
   char* word = strtok(copy_sentence, delimiters);
   while(word!=NULL){
     // for each word or token
     // web link is a token without delimiters
      if (regexec(&regex, word, 0, NULL, REG_EXTENDED)==0){ // Extended Regular
Expression version
```

```
char* link = (char*) malloc((strlen(word)+1) * sizeof(char));
link = strcpy(link, word);
char* parts[ strlen(link) ];
int index = 0;
char* part =strtok(link, "/:");
while(part!=NULL){
  if (strcmp(part,"")!=0){
    parts[index++] = part;
  }
  part = strtok(NULL, "/:");
}
int domain_index = 0, est_www = 0;
if (strstr(word,":")!=NULL) domain_index= 1; // this is for checking existence of protocol name
if (parts[domain_index]!=NULL && parts[index-1]!=NULL){
  if (strstr(parts[domain_index], "www.")!=NULL) est_www = 1; // for checking existence of www. prefix
  char* temp = (char*) malloc((strlen(parts[domain_index])+1)*sizeof(char));
  temp = strcpy(temp, parts[domain_index]);
  if (est_www==1){ // remove www. from domain name
    int start = 4; // after www. we are at 4th character
    int end = strlen(parts[domain_index]);
    for (int c=start; c<end; c++){
      temp[c-start] = parts[domain_index][c];
    }
    temp[end-start] = '\0';
  }
  printf("%s - %s\n",temp, parts[index-1]);
}else{
  printf("Error\n");
```

```
            return;
          }
        }
      word = strtok(NULL, delimiters);
    }
  }
}

int main(void) {
  char** text=malloc(2*sizeof(char*));
  int textLength=0;
  text = readText(text, &textLength);
  extract_print_links(text, textLength);
  return 0;
}
```