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: Парадигмы программирования.

Студентка гр. 3341 _____ Байрам Э.

Преподаватель _____ Глазунов С.А

Санкт-Петербург

2024

Цель работы

Целью данной работы является разработка программы на языке C, которая выполняет обработку входного текста, представляющего собой набор предложений, заканчивающихся новой строкой и завершающегося предложением "Fin." Программа должна использовать регулярные выражения для поиска всех ссылок на файлы в сети интернет, которые могут встречаться в тексте. После обнаружения ссылок, программа извлекает из них доменное имя сайта и имя файла. В результате работы программа выводит на экран пары в формате `<название_сайта> - <имя_файла>`. Основная цель работы - продемонстрировать навыки работы с вводом и выводом данных, использование регулярных выражений для поиска и извлечения информации, а также умение работать с динамическим выделением памяти в языке C.

Задание

Заданием является разработка программы на языке C, которая принимает на вход текст, представляющий собой набор предложений, каждое из которых заканчивается новой строкой, а текст заканчивается предложением "Fin." В тексте могут встречаться ссылки на различные файлы в сети интернет. Необходимо, используя регулярные выражения, найти все эти ссылки в тексте и вывести на экран пары в формате `<название_сайта> - <имя_файла>`. Гарантируется, что если предложение содержит ссылку, то после нее будет символ переноса строки. Ссылки могут начинаться с названия протокола (например, "http://", "ftp://"), за которым следует доменное имя сайта, один или несколько доменов верхнего уровня, возможно путь к файлу на сервере и имя файла с расширением.

Основные теоретические положения

Основные теоретические положения данной работы включают несколько ключевых концепций, связанных с программированием на языке C и использованием регулярных выражений для обработки текста.

Во-первых, важно понимать принципы работы с вводом и выводом данных в языке C. В данной работе используется функция `getchar()` для чтения входного текста, что позволяет считывать текст посимвольно до достижения конца файла или специальной метки окончания ("Fin."). Это обеспечивает гибкость и возможность обработки текста любой длины.

Во-вторых, значимую роль играет динамическое выделение памяти. В работе используются функции `calloc()` и `realloc()` для динамического выделения и перераспределения памяти. Это необходимо для эффективного хранения и обработки текста произвольного размера.

Следующий важный аспект - это работа с строками. Функции `strtok()` и `strcpy()` применяются для разбиения текста на предложения и копирования строк. Эти функции помогают организовать текст в удобный для дальнейшей обработки вид.

Основной компонент задачи - использование регулярных выражений для поиска ссылок в тексте. Регулярные выражения позволяют формально описать шаблоны, по которым можно искать текстовые фрагменты, соответствующие заданным критериям (в данном случае - ссылки). В программе используется библиотека POSIX для работы с регулярными выражениями (`regex.h`). Функции `regcomp()`, `regexec()` и `regfree()` обеспечивают компиляцию, выполнение и освобождение регулярных выражений соответственно.

Кроме того, важно учитывать принципы работы с массивами и указателями в C, так как программа активно использует эти структуры данных для хранения предложений и результатов обработки.

Наконец, программа демонстрирует основы обработки ошибок и исключений. Проверки корректности входных данных и возвращаемых значений функций помогают избежать некорректной работы программы и возможных сбоев.

В целом, работа объединяет различные аспекты программирования на языке C, от управления памятью и работы со строками до использования регулярных выражений и обработки ошибок, что позволяет эффективно решать задачу поиска и извлечения ссылок из текста.

Выполнение работы

Выполнение работы включает несколько основных этапов, каждый из которых решает определенную часть задачи.

Чтение входного текста: Программа начинает с чтения входного текста, который подается построчно. Для этого используется функция getchar(), которая считывает символы до достижения конца файла или специальной метки окончания текста "Fin.". В процессе чтения текста ведется подсчет количества предложений. Динамическое выделение памяти с помощью calloc() и realloc() позволяет гибко обрабатывать текст любой длины.

Разделение текста на предложения: После того, как весь текст считан, он разделяется на отдельные предложения. Это выполняется с помощью функции strtok(), которая разбивает текст на предложения по символу новой строки. Результатом этого этапа является массив строк, где каждая строка представляет собой отдельное предложение.

Компиляция регулярного выражения: Для поиска ссылок в тексте используется регулярное выражение. Регулярное выражение компилируется с помощью функции regcomp(). Это выражение описывает шаблон ссылок, которые начинаются с названия протокола (опционально), за которым следует доменное имя сайта, путь к файлу и имя файла с расширением.

Поиск и извлечение ссылок: Для каждого предложения из массива строк выполняется поиск ссылок с использованием функции regexec(). Если

в предложении находится ссылка, она разбивается на части: доменное имя сайта и имя файла. Эти части извлекаются из групп совпадений, определенных в регулярном выражении, и сохраняются в массиве строк.

Выводы

В ходе выполнения данной лабораторной работы были достигнуты следующие результаты:

1. Обработка текста:

Программа успешно реализована для чтения текста, который подается построчно и заканчивается специальной меткой "Fin.". Использование функции `getchar()` и динамическое выделение памяти позволили корректно считывать текст любого объема.

2. Разделение текста на предложения:

Текст был успешно разделен на предложения с помощью функции `strtok()`. Это позволило разбить исходный текст на отдельные строки для дальнейшей обработки.

3. Использование регулярных выражений:

Регулярные выражения, скомпилированные с использованием библиотеки POSIX (`regex.h`), позволили эффективно находить ссылки в тексте. Регулярное выражение было разработано таким образом, чтобы учитывать различные варианты ссылок, включая протоколы, доменные имена и пути к файлам.

4. Извлечение и вывод ссылок:

Программа корректно извлекала доменные имена сайтов и имена файлов из найденных ссылок. Это достигалось путем анализа групп

совпадений в регулярных выражениях. Найденные пары `<название_сайта> - <имя_файла>` были успешно выведены на экран.

5. Работа с динамической памятью:

В процессе работы программы использовалось динамическое выделение и перераспределение памяти, что позволило гибко обрабатывать текст и результаты поиска. Это включало выделение памяти под строки и массивы строк.

6. Проверка корректности:

Программа была протестирована на различных входных данных, включая различные варианты ссылок и их отсутствие. В результате тестирования подтверждена корректная работа программы в соответствии с поставленной задачей.

В целом, работа продемонстрировала навыки работы с вводом и выводом данных в языке C, использование регулярных выражений для поиска и извлечения информации, а также эффективное управление памятью. Программа выполняет поставленные задачи и может быть использована для поиска и анализа ссылок в текстовых данных.

Тестирование

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарии
1.	This is simple url: http://www.google.com/track.mp3 Fin.	google.com – track.mp3	Проверка на наличие www перед доменным именем
2.	This is simple url: http://www.google.com/track.mp3 May be more than one upper level	google.com - track.mp3 qwe.edu.etu.yahooo.org.net.ru - qwe.q	Проверка на валидность выражений с доменами более высокого уровня и на наличие пути до файла

	domain http://www.qwe.edu.etu.yahoo.org.net.ru/qwe.q Some other protocols		
3.	ftp://pepepupu.cheeck/qqwe/qweqw/qwe.avi Fin.	pepepupu.cheeck – qwe.avi	Проверка исправности с протоколом ftp и на наличие пути до файла

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

#define BUFFER 1024
#define MAX_LEN 100
#define CHECK 3
#define GROUPS_ID 8
```

```
char* get_full_text(int*);
char** split_sentences(char*, int);
void check_regular(char**, int);


int main() {
    int sentences_count = 0;
    char* text = get_full_text(&sentences_count);
    //    printf("%s\n", text);
    //    printf("%d\n", sentences_count);
    char**     separation_text     =     split_sentences(text,
sentences_count);
    check_regular(separation_text, sentences_count);

    free(text);
    free(separation_text);
    return 0;
}

char* get_full_text(int* sentences_count) {
    char c;
    int i = 0;
    int capacity = BUFFER;
    char* text = (char*)calloc(capacity, sizeof(char));

    while ((c = getchar()) != EOF) {
        text[i] = c;
        if (c == '\n') {
            ++(*sentences_count);
        }
        if (i == capacity - 1) {
            capacity += BUFFER;
            text = realloc(text, capacity * sizeof(char));
        }
        if (i >= CHECK && text[i] == '.' && text[i - 1] == 'n'
&& text[i - 2] == 'i' && text[i - 3] == 'F') {
```

```
                break;
            }
            i++;
        }
        text[i - 4] = '\0';
        return text;
    }

    char** split_sentences(char* full_text, int count_sentences) {
        int length = 0;
        char**    sentences    =    (char**)calloc(count_sentences,
sizeof(char*));
        char* sentence = strtok(full_text, "\n");

        while (sentence != NULL) {
            if (length >= count_sentences) {
                count_sentences *= 2;
                sentences  =  realloc(sentences,  sizeof(char*)  *
count_sentences);
            }
            sentences[length] = sentence;
            sentence = strtok(NULL, "\n");
            ++length;

        }
        return sentences;
    }

    void check_regular(char** sentences, int sentences_count) {
        char*   regexString   =   "(\\w+\\:\\/\\/)?(www\\.)?(([a-z0-
9\\.]+)?[a-z0-9]+\\.\\w+)\\/(([a-z0-9\\/]+)?\\w+\\/)?([a-z0-
9]+\\.\\w+)";
        char**     answer     =     (char**)calloc(sentences_count,
sizeof(char*));
        regex_t regexCompiled;
        regmatch_t groups[GROUPS_ID];
        int size = 0;
```

```
        int matched_count = 0;
        regcomp(&regexCompiled, regexString, REG_EXTENDED);

        for (int j = 0; j < sentences_count; j++) {
            if (regexec(&regexCompiled, sentences[j], GROUPS_ID,
groups, 0) == 0) {
                //answer = realloc(answer, sizeof(char*) *
(matched_count+1));
                char* final_line = (char*)calloc(100, sizeof(char));
                size = 0;

                for (int i = groups[3].rm_so; i < groups[3].rm_eo;
i++) {
                    final_line[size] = sentences[j][i];
                    ++size;
                }
                final_line[size++] = ' ', final_line[size++] = '-',
final_line[size++] = ' ';
                for (int i = groups[7].rm_so; i < groups[7].rm_eo;
i++) {
                    final_line[size] = sentences[j][i];
                    ++size;
                }

                final_line[size] = '\0';
                answer[matched_count++] = final_line;
            }
        }
        for (int i = 0; i < matched_count; i++) {
            if (i == matched_count - 1)
                printf("%s", answer[i]);
            else
                printf("%s\n", answer[i]);
        }
    }
```