МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 33

ОТЧЕТ О ПРАКТИКЕ   
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | А.А. Бардовский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ ПО ДИСЦИПЛИНЕ : АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
| ПО ТЕМЕ: «Приёмы работы с текстовыми файлами» |
|  |

ОТЧЕТ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № |  |  |  |  |  |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

**Лабораторная работа № 3.**

**Цель работы:** Изучить приёмы работы с текстовыми файлами.

**Ход работы:**

1. В любой удобной форме (BNF, диаграммы Вирта) описать синтаксис текстового файла, предлагаемого к разбору.

2. Перечислить данные, которые требуется извлечь из файла, описать структуру данных для их хранения.

3. Перечислить тестовые сценарии для функции синтаксического разбора.

4. Написать функцию, выполняющую синтаксический разбор входных данных.

5. Написать функцию, считывающую файл.

6. Написать функцию, реализующую требуемые действия с набором данных.

7. Вывести результат вычислений в стандартный поток вывода.

**Задание:**

2. Вычислить доли используемых браузеров за выбранный период времени (лог Apache).

**Описание работы программы:**

Этот код на C выполняет обработку файла журнала (лог-файла) с данными об использовании интернет-ресурсов. Он извлекает информацию о браузерах, делает подсчет уникальных браузеров и их количества.

Описание работы каждой функции:

1. **freeLogEntries**: Эта функция освобождает память, выделенную для структуры `LogEntry`.

2. **freeBrowsersCount**: Эта функция освобождает память, выделенную для массива BrowserCount.

3. **readFile**: Эта функция открывает указанный файл журнала и читает его построчно. Она разбирает каждую строку, извлекая информацию об IP-адресе, дате и времени, запросе и браузере. Затем сохраняет эти данные в структуру LogEntry. При необходимости увеличивает память для хранения записей о входах.

4. **isValidBrowserName**: Эта функция проверяет, содержит ли строка допустимые символы для названия браузера.

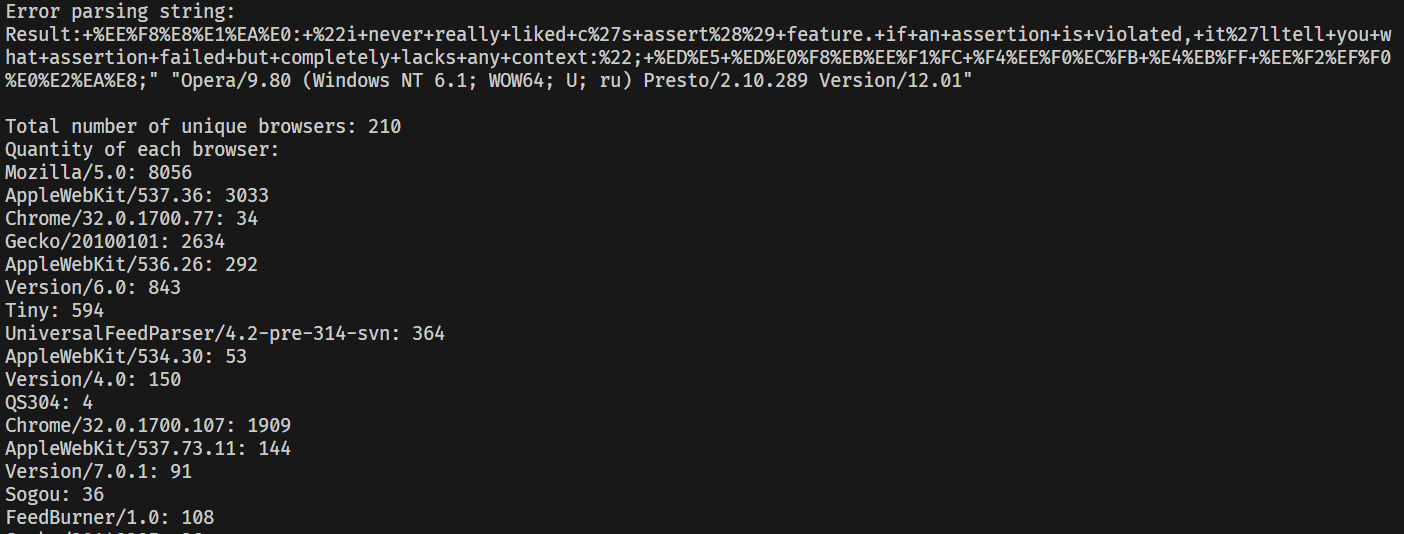
5. **isNewBrowser**: Эта функция проверяет, является ли браузер новым (т.е., ранее не встречавшимся) и допустимым.

6. **freeBrowserCount**: Эта функция освобождает память, выделенную для массива BrowserCount.

7. **calculateBrowserUsage**: Эта функция анализирует данные о браузерах, извлеченных из журнала. Она подсчитывает количество уникальных браузеров и их встречаемость, игнорируя браузеры, имена которых содержат менее трех символов. Затем выводит общее количество уникальных браузеров и их количество каждого.

В основной функции **main** происходит вызов readFile для чтения файла журнала и calculateBrowserUsage для анализа данных о браузерах. Кроме того, предусмотрено освобождение памяти, выделенной под структуру LogEntry (вызов функции freeLogEntries).

**Вывод программы:**



**Вывод:**

В ходе выполнения лабораторной работы было реализована фильтрация содержимого входного файла.

**Листинг программы:**

Файл main.c:

/\*

 \* Представление структуры файла логов

 \* [IP адрес] - - [Дата] "[HTTP метод] [Запрошенный ресурс]" [Код состояния] [Размер ответа] "[Заголовок Referer]" "[Заголовок User-Agent]"

 \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LINE\_LENGTH 1024

struct LogEntry

{

    char \*ip;

    char \*dateTime;

    char \*request;

    char \*browser;

};

struct BrowserCount

{

    char \*name;

    int count;

};

void freeLogEntries(struct LogEntry \*logEntries, int numEntries)

{

    for (int i = 0; i < numEntries; i++)

    {

        free(logEntries[i].ip);

        free(logEntries[i].dateTime);

        free(logEntries[i].request);

        free(logEntries[i].browser);

    }

    free(logEntries);

}

void freeBrowsersCount(struct BrowserCount \*browsersCount, int numUniqueBrowsers)

{

    for (int i = 0; i < numUniqueBrowsers; i++)

    {

        free(browsersCount[i].name);

    }

    free(browsersCount);

}

void readFile(const char \*filename, struct LogEntry \*\*logEntries, int \*numEntries)

{

    FILE \*file = fopen(filename, "r");

    if (file == NULL)

    {

        printf("Error opening file.\n");

        exit(1);

    }

    char line[MAX\_LINE\_LENGTH];

    \*numEntries = 0;

    \*logEntries = NULL;

    size\_t allocatedEntries = 0;

    while (fgets(line, sizeof(line), file))

    {

        char ip[50], dateTime[50], request[255], browser[255];

        if (sscanf(line, "%s %\*s %\*s [%[^]]] \"%\*[^\"]\" %\*s %\*s \"%[^\"]\" \"%[^\"]\"",

                   ip, dateTime, request, browser) >= 3)

        {

            if (\*numEntries >= allocatedEntries)

            {

                // Increase the allocated memory for logEntries

                allocatedEntries += 10; // You can adjust the increment as needed

                \*logEntries = realloc(\*logEntries, allocatedEntries \* sizeof(struct LogEntry));

                if (\*logEntries == NULL)

                {

                    printf("Memory allocation failed.\n");

                    exit(1);

                }

            }

            (\*logEntries)[\*numEntries].ip = strdup(ip);

            (\*logEntries)[\*numEntries].dateTime = strdup(dateTime);

            if (sscanf(line, "%\*s %\*s %\*s [%\*[^]]] \"%\*[^\"]\" %\*s %\*s \"%\*[^\"]\" \"%\*[^\"]\" %s", request) == 1)

            {

                (\*logEntries)[\*numEntries].request = strdup(request);

            }

            else

            {

                (\*logEntries)[\*numEntries].request = NULL;

            }

            (\*logEntries)[\*numEntries].browser = strdup(browser);

            (\*numEntries)++;

        }

        else

        {

            printf("Error parsing string:\n%s\n", line);

        }

    }

    fclose(file);

}

// Function to check if the browser name contains valid characters

int isValidBrowserName(const char \*browser)

{

    while (\*browser != '\0')

    {

        if (!(isalnum(\*browser) || \*browser == '/' || \*browser == ' ' || isalpha(\*browser) || \*browser == '\_' || \*browser == '.' || \*browser == '-'))

        {

            return 0; // Not a valid character

        }

        browser++;

    }

    return 1; // Valid browser name

}

// Function to check if it's a new valid browser

int isNewBrowser(const char \*browser, struct BrowserCount \*browsersCount, int numUniqueBrowsers)

{

    if (!isValidBrowserName(browser))

    {

        return 0; // Not a valid browser name

    }

    for (int j = 0; j < numUniqueBrowsers; j++)

    {

        if (strcmp(browser, browsersCount[j].name) == 0)

        {

            return 0; // Not a new valid browser

        }

    }

    return 1; // New valid browser

}

void freeBrowserCount(struct BrowserCount \*browsersCount, int numUniqueBrowsers)

{

    for (int i = 0; i < numUniqueBrowsers; i++)

    {

        free(browsersCount[i].name);

    }

    free(browsersCount);

}

void calculateBrowserUsage(struct LogEntry \*logEntries, int numEntries)

{

    struct BrowserCount \*browsersCount = NULL;

    int numUniqueBrowsers = 0;

    size\_t allocatedBrowsers = 0;

    for (int i = 0; i < numEntries; i++)

    {

        int inParentheses = 0;

        char \*tempBrowser = strdup(logEntries[i].browser);

        int diff = 0;

        for (int j = 0; tempBrowser[j] != '\0'; j++)

        {

            if (tempBrowser[j] == '(')

            {

                inParentheses = 1;

            }

            else if (tempBrowser[j] == ')' && inParentheses)

            {

                inParentheses = 0;

                diff = j + 2;

                j += 2;

            }

            else if (!inParentheses)

            {

                if (isspace(tempBrowser[j]))

                {

                    tempBrowser[j] = '\0';

                    if (isNewBrowser(tempBrowser + diff, browsersCount, numUniqueBrowsers))

                    {

                        if (numUniqueBrowsers >= allocatedBrowsers)

                        {

                            allocatedBrowsers += 10; // Increment by 10 or adjust as needed

                            browsersCount = realloc(browsersCount, allocatedBrowsers \* sizeof(struct BrowserCount));

                            if (browsersCount == NULL)

                            {

                                printf("Memory allocation failed.\n");

                                exit(1);

                            }

                        }

                        browsersCount[numUniqueBrowsers].name = strdup(tempBrowser + diff);

                        browsersCount[numUniqueBrowsers].count = 1;

                        numUniqueBrowsers++;

                    }

                    else

                    {

                        for (int k = 0; k < numUniqueBrowsers; k++)

                        {

                            if (strcmp(tempBrowser + diff, browsersCount[k].name) == 0)

                            {

                                browsersCount[k].count++;

                                break;

                            }

                        }

                    }

                }

            }

        }

        // free(tempBrowser);

    }

    int diff = 0;

    for (int i = 0; i < numUniqueBrowsers; i++)

    {

        if (strlen(browsersCount[i].name) < 3)

        {

            diff++;

        }

    }

    printf("Total number of unique browsers: %d\n", numUniqueBrowsers - diff);

    printf("Quantity of each browser:\n");

    for (int i = 0; i < numUniqueBrowsers; i++)

    {

        if (strlen(browsersCount[i].name) >= 3)

        {

            printf("%s: %d\n", browsersCount[i].name, browsersCount[i].count);

        }

    }

    freeBrowserCount(browsersCount, numUniqueBrowsers);

}

int main()

{

    struct LogEntry \*logEntries;

    int numEntries = 0;

    readFile("C:\\Users\\smaug\\Desktop\\University\\3\_semester\\algorithms\\algotithms\_university\\lab3\\access.log", &logEntries, &numEntries);

    calculateBrowserUsage(logEntries, numEntries);

    // freeLogEntries(logEntries, numEntries);

    return 0;

}