Tarea 5 - Programación y algoritmos
Giovanni Gamaliel López Padilla

Problema 1

Problema 1a

Encuentre si la palabra 'p' esta presente en un string 'str', de otra manera regresa el número de veces que se encontro.
Se descargaron 20 parrafos de un documento de prueba llamado '*Lorem lipsum*' [1]. Este documento es guardado en la carpeta Problema_1a con el nombre test_text.txt. Se creo una función llamada find_word. Esta función recibe como parámetros un puntero del tipo FILE y la palabra a buscar en el texto. El algoritmo que sigue la función find_word es el siguiente:

```
1    // input text, word
2    // output count
3    count = 0
4    letter_behind = ' '
5    letter_file = read_character_from_file(text)
6    letter_word = word[0]
7    letter_final_word = word[-1]
8    while(letter!=EOF)
9    {
10       if(lettler_file == letter_word)
11       {
12           if(letter_behind_is_not_a_word)
13           {
14               letter_behind = letter_file
15               letter_file = read_character_from_file(text)
16               i = 1
17               while(letter_file == word[i])
18               {
19                   letter_file = read_character_from_file(text)
20                   i += 1
21               }
22               if(i == word_size and letter_is_not_a_character)
23               {
24                   count += 1
25               }
26           }
27       }
28       letter_file = read_character_from_file(text)
29   }
```

En la linea 10 se comprueba si el carácter leido coincide con el primer carácter de la letra. Si coincide entonces se comprobará si los siguientes carácteres coinciden. En la linea 12 se comprueba si el carácter anterior a la primer coincidencia no es una letra, esto para no encontrar coincidencias en palabras distintas a las buscadas. En la linea 17 comienza la verificación de los demás carácteres, si coincidieron todos entonces la variable i debe ser del mismo valor que el tamaño de la palabra. En la linea 22 se comprueba si la variable i es del mismo tamaño que la palabra y si la letra siguiente no es una letra.
El programa se encuentra en la carpeta Problema_1a, el programa se compilo con el siguiente comando:

```
1    gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```

Se usaron las palabras lorem, sapien, maecenas, aliquam, quam, tor y tor para probar el programa. Los resultados son los mostrados en la tabla 1.

Palabra	Resultados
lorem	9
sapien	10
maecenas	6
aliquam	18
quam	8
tor	0
tortor	6

Tabla 1: Resultados del programa 1a.

La manera de ejecutar el programa es la siguiente: Para hacer la busqueda de la palabra lorem en el archivo test_text.txt se usa el siguiente comando.

```
./main.out test_text.txt lorem
```

Se creo un script en bash el cual ejecuta el programa con las palabras mencionadas en la tabla 1. El script contiene las siguientes ejecucciones:

```
make clean
make
./main.out test_text.txt lorem
./main.out test_text.txt sapien
./main.out test_text.txt maecenas
./main.out test_text.txt aliquam
./main.out test_text.txt quam
./main.out test_text.txt tor
./main.out test_text.txt tortor
```

Problema 1b

Separe un string en tokens de acuerdo a un carácter especial dado como entrada. Debe regresar un arreglo que apunte a cada uno de los tokens encontrados.
Se creo una función llamada split la cual recibe como parámetros el nombre del archivo donde se leera el string, el símbolo de separación, un puntero que la cantidad de tokens y un doble puntero que guardará los tokens. El algoritmo que sigue esta función es el siguiente:

```
size = count_simbol(text, simbol)
tokens = assign_memory(size)
letter = read_character_from_file(text)
i = 0
while(letter != EOF)
{
    if(letter != simbol)
    {
        tokens[i] += letter
        i += 1
    }
    else
    {
        tokens[i] += '\0'
        i = 0
    }
    letter = read_character_from_file(text)
}
```

El tamaño de cada elemento de los tokens esta definido en 40, esto en la linea 1 del archivo size_per_word. Puede implemntarse una función para obtener el tamaño máximo de los tokens para designar esta variable, pero esto requeriría leer e identificar a los tokens dos veces.

Se probo el programa con tres archivos diferentes (test1.csv, test2.txt y test3.txt). La ejecución de los tres archivos se encuentrá implementada en el código principal. El programa se encuentra en la carpeta Problema_1b. La compilación fue realizada con el siguiente comando:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```

El output esperado del programa se encuentra en el archivo output.txt.

Problema 1c

Concadene dos strings creados con memoria dinámica. Regrese el resultado en el primer string.
Se creo una función llamada join_strings la cual recibe como parámetros dos punteros a strings. El algoritmo que implementa es el siguiente:

```
len_string1 = obtain_len_string(string1)
len_string2 = obtain_len_string(string2)
string1 = increase_memory(len_string1 + len_string2)
for(i = 0; i< len_string2, i++)
{
    string1[i+len_string1] = string2[i]
}
```

En la linea 3 se aumenta la memoria del primer string para poder almacenar los datos del segundo string.
El programa se encuentra en la carpeta Problema_1c. La compilación se realizo con el siguiente comando:

```
gcc -Wall -Wextra -Werror -pedantic -ansi -o main.out main.c -std=c11
```

Probando con los strings *abcdf* y *xyz1234* se obtiene el siguiente output:

```
> ./main.out abdcf xyz1234
Texto 1:    abdcf
Texto 2:    xyz1234
Union:      abdcfxyz1234
```

Problema 2

Programa que calcule el moving average y la mediana de una matriz de tamaño $L_1 \times L_2$. Reemplazar los valores de la matriz en su valor central.

Referencias

[1] Lorem lipsum text. https://es.lipsum.com/.