

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 ipsum*' [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(letter_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 leído coincide con el primer carácter de la letra. Si coincide entonces se comprobará si los siguientes caracteres 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 caracteres, 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 búsqueda de la palabra lorem en el archivo `test_text.txt` se usa el siguiente comando.

```
1 ./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 ejecuciones:

```
1 make clean
2 make
3 ./main.out test_text.txt lorem
4 ./main.out test_text.txt sapien
5 ./main.out test_text.txt maecenas
6 ./main.out test_text.txt aliquam
7 ./main.out test_text.txt quam
8 ./main.out test_text.txt tor
9 ./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:

```
1 size = count_simbol(text, simbol)
2 tokens = assign_memory(size)
3 letter = read_character_from_file(text)
4 i = 0
5 while(letter != EOF)
6 {
7     if(letter != simbol)
8     {
9         tokens[i] += letter
10        i += 1
11    }
12    else
13    {
14        tokens[i] += '\0'
15        i = 0
16    }
```

```

17     letter = read_character_from_file(text)
18 }

```

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 probó el programa con tres archivos diferentes ([test1.csv](#), [test2.txt](#) y [test3.txt](#)). La ejecución de los tres archivos se encontrará implementada en el código principal. El programa se encuentra en la carpeta [Problema.1b](#). La compilación fue realizada con el siguiente comando:

```

1 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

Concatene 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:

```

1 len_string1 = obtain_len_string(string1)
2 len_string2 = obtain_len_string(string2)
3 string1 = increase_memory(len_string1 + len_string2)
4 for (i = 0; i < len_string2, i++)
5 {
6     string1[i+len_string1] = string2[i]
7 }

```

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:

```

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

```

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

```

1 > ./main.out abdcf xyz1234
2 Texto 1:    abdcf
3 Texto 2:    xyz1234
4 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.

Al inicio del programa se preguntan las dimensiones de la matriz de datos. Esta matriz es inicializada con números enteros entre 0 y 20. Las matrices que contendran los datos del moving average y la mediana son inicializadas en este mismo proceso con los mismos valores de la matriz original. El algoritmo para obtener el moving average es el siguiente:

```

1 // inputs: matrix, dimensions
2 for (i = 1; i < dimensions[0]; i++)
3 {
4     for (j = 1; j < dimensions[1]; j++)

```

```

5      {
6          sum = 0
7          for(k = i-1; k < i+2; k++)
8          {
9              for(l = j-1; l < j+2; l++)
10             {
11                 sum += matrix[k][l]
12             }
13         }
14         moving_average[i][j] = sum / 9
15     }
16 }

```

Los ciclos que recorren con las variables i, j (líneas 2 y 4) es el recorrido que hará la posición central de la ventana de 3×3 . En la línea 6 se inicializa el cálculo del promedio de los valores en la ventana de los datos. Los ciclos que recorren con las variables k, l (líneas 7 y 9) son usadas para realizar el recorrido en los valores de la matriz que están en la ventana de datos.

El algoritmo para obtener la mediana de cada ventana de datos es el siguiente:

```

1  // inputs: matrix, dimensions
2  median[9]
3  for(i = 1; i < dimensions[0]; i++)
4  {
5      for(j = 1; j < dimensions[1]; j++)
6      {
7          n = 0
8          for(k = i-1; k < i+2; k++)
9          {
10             for(l = j-1; l < j+2; l++)
11             {
12                 median[n] = matrix[i][j]
13                 n += 1
14             }
15         }
16         quick_sort(median)
17         median_matrix[i][j] = median[4]
18     }
19 }

```

Los ciclos que recorren con las variables i, j (líneas 3 y 5) es el recorrido que hará la posición central de la ventana de 3×3 . Los valores de la ventana serán guardados en un arreglo de dimension 9. En la línea 16 los datos de la ventana serán ordenados de menor a mayor valor con un algoritmo de ordenamiento implementado en tareas anteriores. Al ser ordenados los datos es fácil obtener la mediana, en este caso siempre se encontrará en la posición 4 del arreglo. Los algoritmos de la mediana y moving average son muy semejantes, es por ello que en el programa [algorithms.h](#) se escribieron en una sola función llamada [obtain_moving_average_and_median](#). En los dos algoritmos, los valores de la frontera son iguales a los datos iniciales.

El programa se encuentra en la carpeta [Problema_2](#), la compilación del mismo se realizó con el siguiente comando:

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

Uno de los resultados obtenidos al ejecutar el programa es el siguiente:

```

1  > ./main.out 4 7
2
3  _____
4  Datos iniciales
5
6  4.000000    16.000000    14.000000    9.000000
7  8.000000    4.000000    16.000000    7.000000
8  1.000000    7.000000    7.000000    13.000000

```

9	16.000000	12.000000	6.000000	1.000000
10	2.000000	14.000000	6.000000	14.000000
11	2.000000	11.000000	15.000000	14.000000
12	6.000000	14.000000	6.000000	5.000000
13				
14	<hr/>			
15	Moving average			
16				
17	4.000000	16.000000	14.000000	9.000000
18	8.000000	8.555556	10.333333	7.000000
19	1.000000	8.555556	8.111111	13.000000
20	16.000000	7.888889	8.888889	1.000000
21	2.000000	9.333333	10.333333	14.000000
22	2.000000	8.444444	11.000000	14.000000
23	6.000000	14.000000	6.000000	5.000000
24				
25	<hr/>			
26	Median			
27				
28	4.000000	16.000000	14.000000	9.000000
29	8.000000	7.000000	9.000000	7.000000
30	1.000000	7.000000	7.000000	13.000000
31	16.000000	7.000000	7.000000	1.000000
32	2.000000	11.000000	12.000000	14.000000
33	2.000000	6.000000	14.000000	14.000000
34	6.000000	14.000000	6.000000	5.000000

Referencias

- [1] Lorem ipsum text. <https://es.lipsum.com/>.