Tarea 4 - Programación y algoritmos
Giovanni Gamaliel López Padilla

Problema 1

Escribir un programa que genere una caminata aleatoria en una matri de 10x10. El arreglo debe contener inicialmente puntos ".", y debe recorrers basado en el residuo de un número aleatoria (usar srand y rand()) cuyos resultado puede ser {0 (arriba), 1 (abajo), 2 (izq), 3 (der)}, que indican la dirección a moverse. A) Verificar que el movimiento no se salga del arreglo de la matriz, y B) No se pued visitar el mismo lugar más de una vez. Si alguna de estas condiciones intentar moverse hacia otra dirección; si todas las posiciones están ocupadas, finalizar el programa e imprimi el resultado.

Para este problema se definidio como una constante global el tamaño de la matriz y el maximo de letras posibles. Los valores son 10 y 26 respectivamente. Esto porque existen 26 letras en el código ascii ignorando a la letra ñ. El estado inicial de la matriz contiene unicamente puntos, esto puede verse en la figura 1. El programa inicia de dos maneras diferentes, un usuario introduce un punto inicial donde comenzará la caminata o este es escogido aleatoriamente.

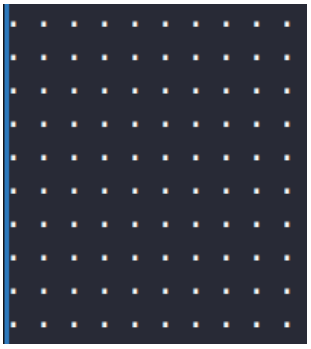

Figura 1: Estado inicial de la matriz

Una casilla se considera disponible si su valor es igual a ".". Antes de obtener el siguiente paso de cada iteracion el programa verifica si sus vecinos estan disponibles. Si existe al menos una casilla disponible entonces el programa sigue ejecutandose (figura 2), si no el programa termina ya que no podrá seguir la caminata (figura 3). El siguinte paso de la iteración se obtiene con la función *rand()*. Se calcula el modulo del número obtenido con 4. Con esto nos aseguramos que el número se encuentre en el conjunto $\{0, 1, 2, 3\}$.

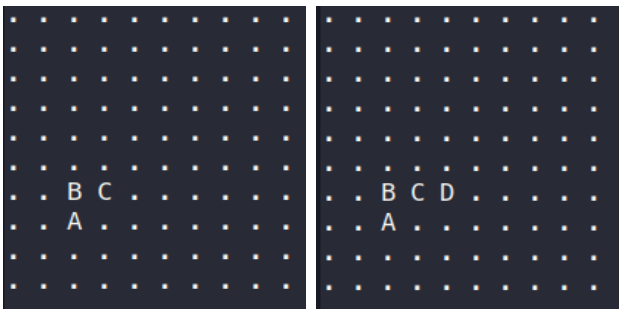

Figura 2: Ejemplo de caminata.

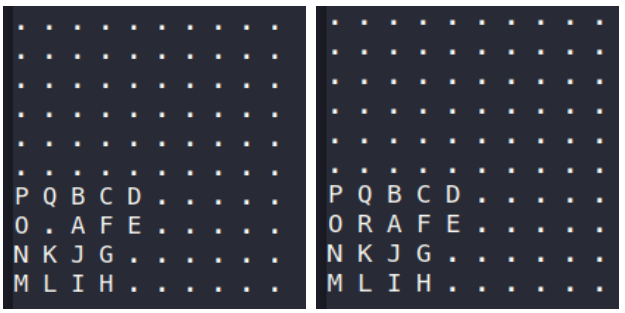

Figura 3: Ejemplo de termino del programa.

Cada paso debe estar contenido dentro del arreglo de 10x10, si el paso sobrepasa la barrera este es rechazado y se calcula otro diferente. Esto es realizado con la función is_in_the_box con el siguiente algoritmo:

```
int is_in_the_box(int pos[])
{
for (int i = 0; i < 2; i++)
{
    if ((pos[i] < 0) || (pos[i] >= size))
    {
        return 0;
    }
}
return 1;
}
```

Si esta dentro de los límites la función devuelve 1, si esta fuera 0. En la figura 4 se muestran diferentes resultados obtenidos por el programa:

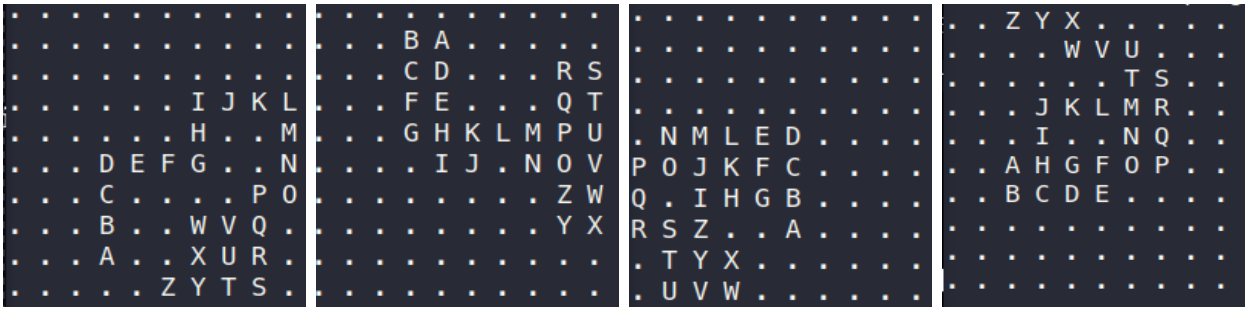

Figura 4: Diferentes output del programa

El programa se encuentra en la carpeta Problema_1. Para compilar el programa se uso el siguiente comando:

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

Problema 2

Dado un archivo de entrada, escribir un programa que encuentre los siguiente:

- **Probabilidad (P) de aparición de cada una de las letras del alfabeto (no haga diferencia entre minúsculas y mayúsculas)**
- **P(x|y), para (x,y) ∈ (a,...,z,A,...,Z) de las 10 letras (x) más frecuentes.**

Al querer obtener la probabilidad de una letra x en el texto ($P(x)$) y la probabilidad que una letra y se encuentre después de una letra x ($P(y|x)$), entonces el programa se ideo para encontrar los casos de tener una letra continua a la otra, ignorando signos de puntuación y saltos de linea. Esta operación se realiza en la función valid_second_letter, la cual se encuentra en el archivo algorithms.h. La función tiene la siguiente estructura:

```
    void valid_second_letter(FILE *text_file, char *letter1, char *letter2,
int probabilities[number_letters])
{
    while (!is_a_letter(*letter2) && (*letter2 != EOF || *letter1 != EOF))
    {
        *letter1 = fgetc(text_file);
        while (!is_a_letter(*letter1) && *letter1 != EOF)
        {
```

```
8                   *letter1 = fgetc(text_file);
9               }
10              *letter2 = fgetc(text_file);
11              count_individual_data(*letter1,
12                                    probabilities);
13          }
14      }
```

donde las variables letter1 y letter2 son apuntadores a caracteres, y letter1 siempre es el caracter antes de letter2 en el texto. Al inicio de verificara que letter2 sea una letra y letter1 y letter2 sean diferentes al final del texto. Si letter2 no es una letra y letter 1 y letter2 no son el final del texto, entonces se buscará que letter1 sea una letra, en este espacio de busqueda puede darse el caso que letter1 sea una letra pero letter 2 no, entonces se introduce el contador para las posibles letras que entren en este caso (linea 11). Ya que letter1 es una letra, entonces se leera el siguiente caracter del texto que será guardado en letter2, si es una letra entonces el proceso de validación termina. Si no se reinicia el proceso de busqueda de letter1.
Al verificar que letter1 y letter2 son letras, entonces se sumaran contadores para ese evento, esto será guardado en un arreglo de 26x26. Todo el proceso de lectura, validación y conteo de probabilidades se encuentra en la función obtain_data en el archivo algorithms.h. La función esta descrita de la siguiente manera:

```
1       char letter1, letter2;
2       letter1 = fgetc(text_file);
3       letter2 = fgetc(text_file);
4       while (letter1 != EOF && letter2 != EOF)
5       {
6           // Validacion de la segunda letra
7           valid_second_letter(text_file,
8                               &letter1,
9                               &letter2,
10                              probabilities);
11          if (letter1 != EOF && letter2 != EOF)
12          {
13              // Conteo de las probabilidades condicionales
14              obtain_conditional_probability(letter1,
15                                             letter2,
16                                             data);
17              // Intercambio de las letras
18              letter1 = letter2;
19              letter2 = fgetc(text_file);
20              count_individual_data(letter1,
21                                    probabilities);
22          }
23      }
```

En la linea 12 se realiza la validación de las variables letter1 y letter2 no sean el final del archivo, ya que como acaban de salir de la validación de letras aqui podemos asegurar que es una letra o es el final del archivo. En las lineas 18 y 19 se realiza la variable letter1 toma el valor de letter2, como letter2 era una letra entonces realizamos el conteo individual de esta letra (linea 20). Este proceso se repetira hasta que letter1 o letter2 sean el final del archivo.
El output producido con el archivo *don_quijote.txt* es el contenido en el archivo output_don_quijote.txt. Además de los archivos sample_spcae.csv y probabilities.csv, los cuales contienen las probabilidades condiciones y probabilidades de cada letra respectivamente.
El programa se encuentra en la carpeta Problema_2. El comando para ejecutar el programa es el siguiente:

```
1       ./main.out don_quijote.txt
```

Y la compilación del mismo con el siguiente comando:

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

Problema 3

Dado una lista de nombres (strings) de N personas (apellido paterno, apellido materno, Nombre(s)), escribir un programa que ordene los nombres alfabéticamente usando un arreglo de apuntadores. Los nombres pueden tener distintas longitudes; cuando un nombre sea prefijo de otro, considerar al nombre mas corto como menor. El ordenamiento debe ser a través de una función que reciba el arreglo de apuntadores.

Se creo una estructura la cual contiene tres elementos para el apellido paterno, apellido materno y nombre(s). Cada linea del archivo de nombres será una persona diferente. El orden que se opto fue primero por apellido paterno, si se encuentra que son iguales pasa a ordenar comparando el apellido materno y por último el nombre de la persona. Se utilizo la función sorted creada en tareas anetriores. Se modifico para comparar nombres con la función compare_names y order_names. La función order_names es la siguiente:

```
1   int order_names(char name1[], char name2[])
2   {
3       int i = 0;
4       int compare;
5       while (name1[i] != '\0' && name2[i] != '\0')
6       {
7           compare = name1[i] - name2[i];
8           if (compare != 0)
9           {
10              return compare;
11          }
12          i = i + 1;
13      }
14      if (name1[i] == '\0' && name2[i] != '\0')
15      {
16          return -name2[i];
17      }
18      if (name2[i] == '\0' && name1[i] != '\0')
19      {
20          return name1[i];
21      }
22      return 0;
23  }
```

Los parámetros name1 y name2 son introducidos a partir de un puntero hacia la estructura. En la linea 10 se devuelve el producto de la "resta" de caracteres de la linea 7. Si este valor es menor a 0 entonces name2 tiene un orden alfabético mayor que name1. Si son iguales entonces el valor de su resta es 0. En el caso en que un nombre o apellido sea prefijo de otro entonces el ciclo iniciado en la linea 5 terminará, ya que uno de los nombres habra llegado al final. Si name1 que llego a su final, entonces se devolverá el negativo de la i-esima letra de name2 (linea 16), si name2 llego a su final, entonces se devolverá la i-esima letra de name1 (linea 20).
Se uso el archivo names.txt que contiene los siguientes nombres:

```
1   Venere Bookamer Art Art
2   Paprocki Kampa Lenna Lenna
3   Paprocki Kampa Lenna Le
4   Dar Biddy Josephine Josephine
5   Darakjy Biddy Josephine Josephine
6   Foller Gillaspie Donette Donette
7   Butt Motley James James
8   Morasca Harabedian Simona Simona
9   Paprocki Kampa Lenna Lo
10  Tollner Hixenbaugh Mitsue Mitsue
11  Dilliard Oles Leota Leota
12  Foller Gilla Donette Donette
13  Paprocki Kampa Le
14  Wieser Ankeny Sage
```

El programa se encuentra en la carpeta Problema_3. Para ejecutar el programa se uso el siguiente comando:

```
./main.out names.txt
```

El cual produce el siguiente output:

```
_____________________________________

Nombres desordenados

Venere Bookamer Art Art
Paprocki Kampa Lenna Lenna
Paprocki Kampa Lenna Le
Dar Biddy Josephine Josephine
Darakjy Biddy Josephine Josephine
Foller Gillaspie Donette Donette
Butt Motley James James
Morasca Harabedian Simona Simona
Paprocki Kampa Lenna Lo
Tollner Hixenbaugh Mitsue Mitsue
Dilliard Oles Leota Leota
Foller Gilla Donette Donette
Paprocki Kampa Le
Wieser Ankeny Sage


_____________________________________
Nombres ordenados

Butt Motley James James
Dar Biddy Josephine Josephine
Darakjy Biddy Josephine Josephine
Dilliard Oles Leota Leota
Foller Gilla Donette Donette
Foller Gillaspie Donette Donette
Morasca Harabedian Simona Simona
Paprocki Kampa Le
Paprocki Kampa Lenna Le
Paprocki Kampa Lenna Lenna
Paprocki Kampa Lenna Lo
Tollner Hixenbaugh Mitsue Mitsue
Venere Bookamer Art Art
Wieser Ankeny Sage
```

La compilación el programa se uso el siguiente comando:

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

Problema 4

Programa que encuentre los tres números mayores de un arreglo de enteros, especificando su posición (índice) original en el arreglo de entrada.