Trabajo Práctico 1
Little Enigma
Programación en C

Tecnología Digital II

1. Introducción

A principios del siglo XX se desarrollo lo que se conocería como la Máquina Enigma. Está permitía codificar y decodificar mensajes, encriptandolos de forma que no puedan ser leídos al menos que se posea la clave para desencriptarlos.

Durante la segunda guerra mundial, los alemanes se hicieron de está tecnología y la utilizaron para fines militares. Los británicos por su parte hicieron grandes esfuerzos para decodificar los mensajes enviados por el ejercito alemán, reuniendo en Bletchley Park a científicos, matemáticos y criptógrafos británicos. Entre ellos Alan Turing, uno de los padres fundadores de la computación.

El objetivo de este TP es implementar un conjunto de funciones sobre distintas estructuras a fin de construir una simplificación de la Máquina Enigma. Adicionalmente se deben generar casos de test para comprobar el correcto funcionamiento de cada una de estas funciones de forma independiente. Las funciones a implementar se realizarán sobre tres tipos de datos.

Por un lado el tipo `littleEnigma` que tendrá la información de nuestra máquina y las funciones para encriptar y desencriptar mensajes. Estos mensajes serán *strings* de C, es decir, cadenas de caracteres terminadas en *null*.

La máquina estará compuesta de discos o rotores que serán modelados con listas circulares de nodos, donde cada nodo corresponderá a una letra. Girar o rotar un disco, será equivalente a modificar el puntero al primer nodo de la lista circular, mientras que codificar un carácter corresponderá a tomar la letra en el i-esimo elemento de la lista.

El trabajo práctico debe realizarse en grupos de tres personas. Tienen tres semanas para realizar la totalidad de los ejercicios. **La fecha de entrega límite es el 29 de agosto hasta las 23:59.** Se solicita no realizar consultas del trabajo práctico por canales públicos. Para realizar consultas específicas de su trabajo práctico pueden crear un canal de slack e incluir a todos los docentes y compañeros de grupo. Tengan presente crear el canal de slack con su nombre de grupo, a fin de evitar confusiones.

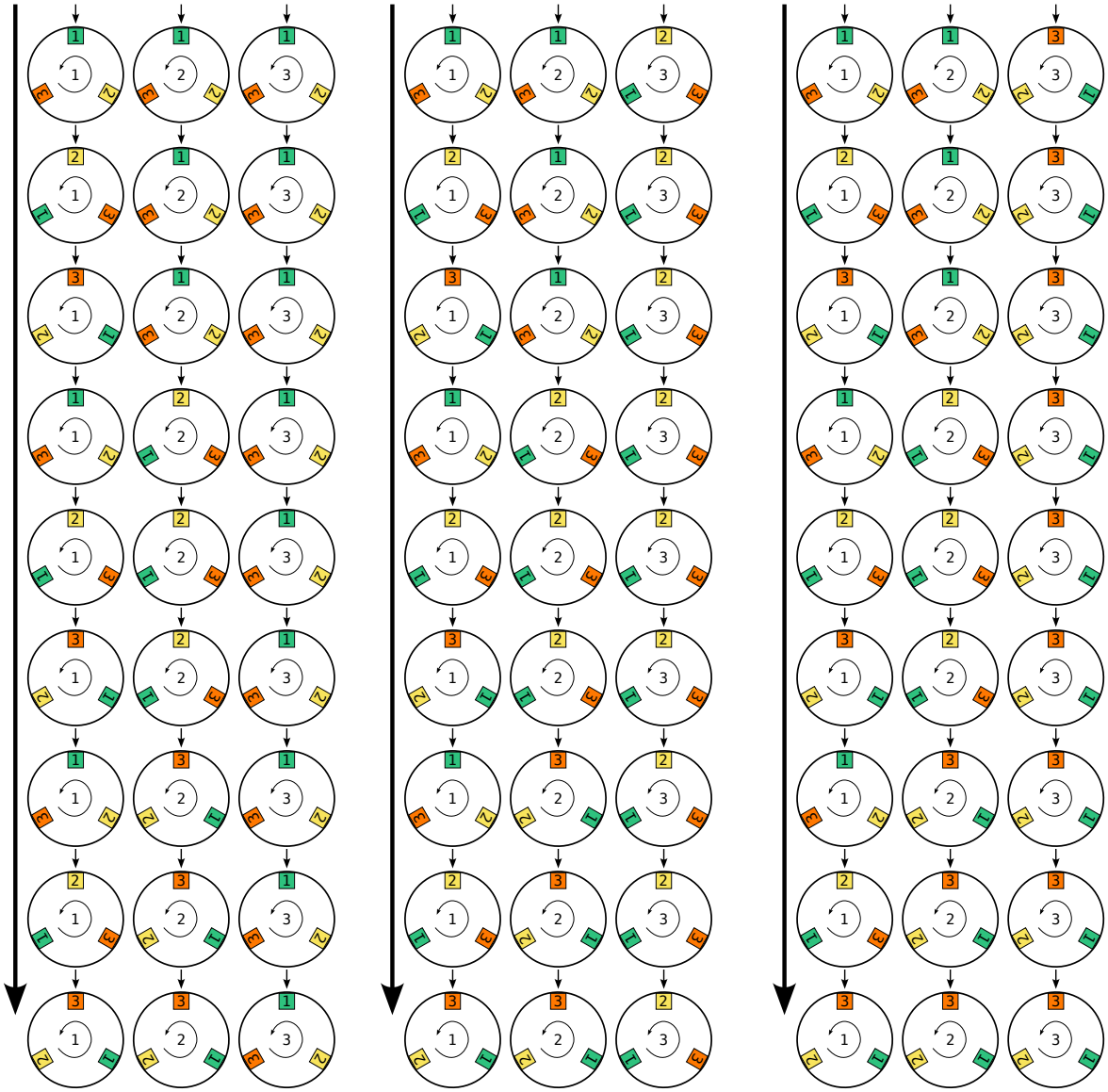
2. Funcionamiento de `littleEnigma`

La máquina cuenta con discos, rotores o ruedas codificadas con una permutación del alfabeto. Su funcionamiento consiste en hacer pasar cada letra del mensaje por cada uno de los discos en orden.

Cada vez que una letra pasa por un disco, se transforma en una nueva letra en cada paso. Al final de este proceso se obtiene la letra codificada.

Para codificar las próximas letras del mensaje, se deberá ir modificando la posición de los dicos. Suponiendo que los discos cuentan con 26 posiciones, equivalente a un alfabeto, se hacen rotar los discos como si se tratara de un contador en base 26.

Para ilustrar este comportamiento se propone la siguiente figura:



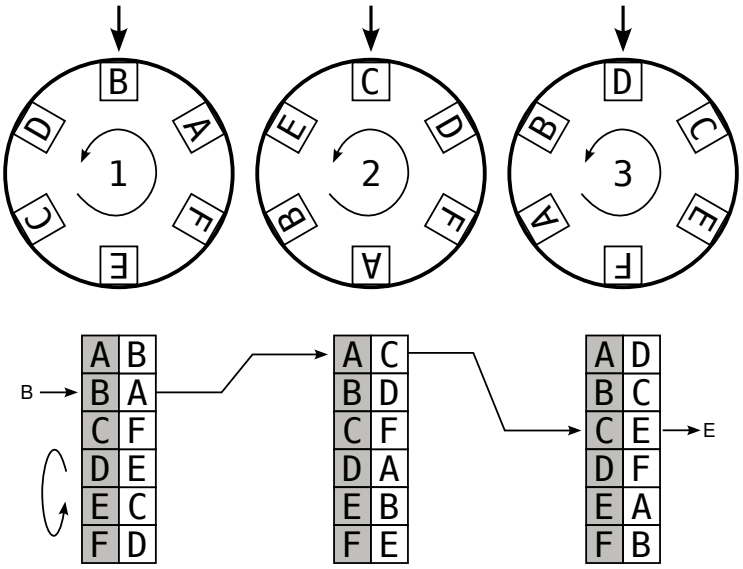
Para simplificar el grafico y poder ilustrar las posiciones de los discos, se suponen discos de solo tres posiciones, pintadas de colores. Inicialmente todos los discos están en una posición determinada, dada por la contraseña de codificación. Supongamos que Inicialmente todos están en `111`. Luego de codificar la primer letra se gira el primer disco una posición dejandolos en `211`, a continuación en la siguiente letra se gira a `311`. Ahora, el primer disco ya dio una vuelta completa, así que se incrementa el segundo disco una vez, además del primero, quedando como `121`.

Este proceso se repite todo el tiempo, hasta que el segundo disco llegue a dar una vuelta completa, donde se gira una vez el tercer disco. Cuando el último disco da una vuelta completa, el proceso vuelve a comenzar. En el ejemplo tenemos $3 \cdot 3 \cdot 3$ posibilidades de codificación distinta de letras, es decir 27 posibles permutaciones del diccionario. Ahora, si usamos tres discos pero con las 26 letras, las posibilidades aumentan a $26 \cdot 26 \cdot 26$, es decir 17576.

La máquina Enigma original contaba con solo 3 discos, la versiones militares aumentaron este número a 5 discos, incluso llegaron a tener máquinas de 8 discos. Aumentando notablemente la cantidad de combinaciones. Nuestra máquina en cambio soportará una cantidad arbitraria de discos, ya que no contamos con la limitación física de armar los discos.

Sin embargo, la máquina Enigma original no solo utilizaba discos con permutaciones del alfabeto, sino también un tablero de conexiones que permitía alterar una de las etapas de codificación. Para simplificar el desarrollo, nuestra máquina no tendrá este tablero de conexiones.

En la siguiente figura tenemos un ejemplo de una `littleEnigma` de tres discos. A fin de simplificar el esquema, se ilustraron solo 6 letras del alfabeto, pero en el caso general debe contener todas las letras del alfabeto. Considerando de todas formas no tener ni espacios ni símbolos adicionales, solo letras, al igual que la máquina Enigma original.



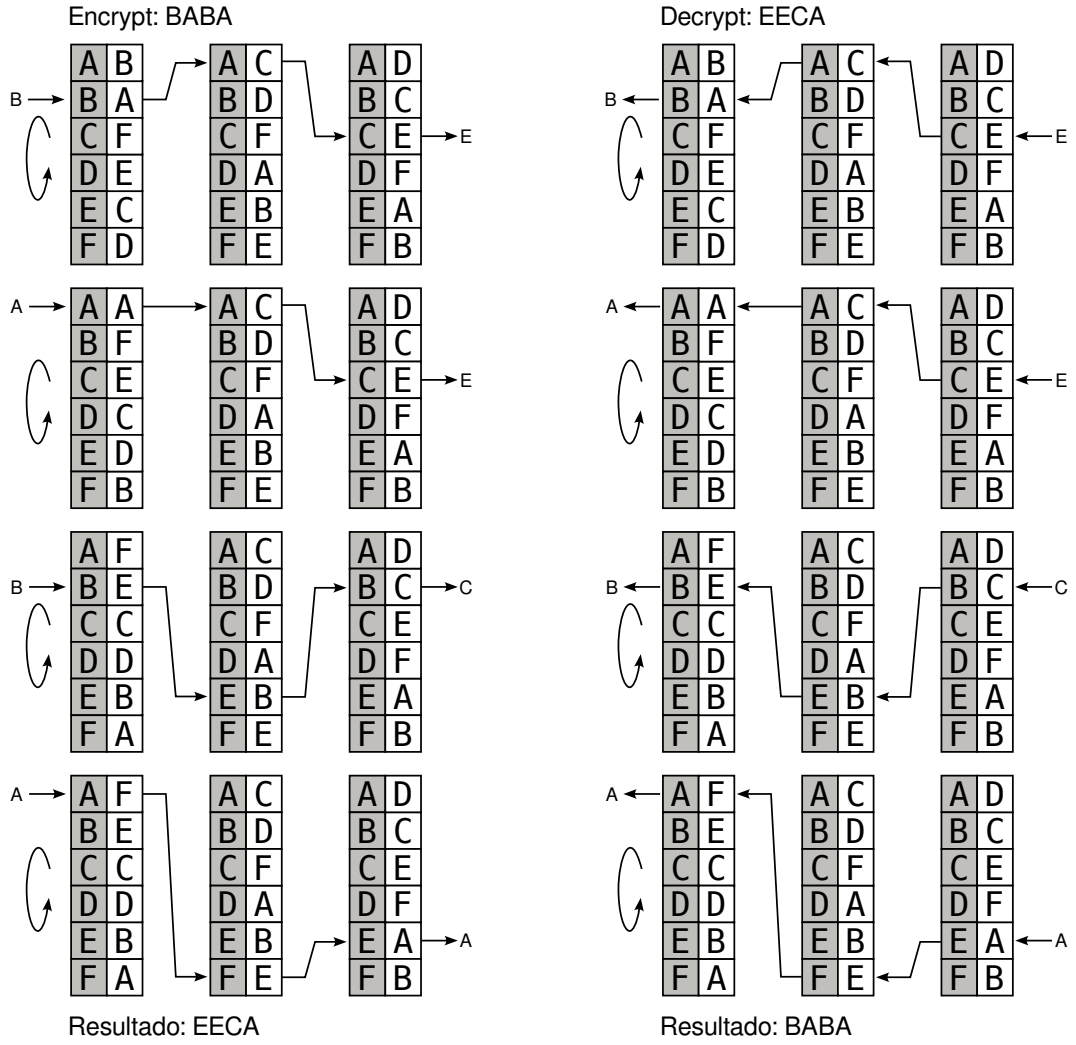
En esté ejemplo se está codificando la letra `B`, que pasa por el primer disco y se transforma en `A`, para luego pasar por el segundo disco y transformarse en `C`. Por último pasa por el tercer disco y se transforma en `E`. La letra entonces codificada es la `E`.

En la siguiente figura se gráfica paso a paso la codificación y decodificación de la palabra ejemplo `BABA`. Inicialmente se codifica el carácter `B`, resultando en `E`. Luego se codifica el carácter siguiente, es decir `A`. Para esto se rota el primer disco, ahora, la letra `B` se codifica como una `F`, mientras que la `A`, que antes se codificaba como una `B`, ahora se codifica como una `A` en el primer disco. Esta letra pasa por los restantes discos resutando en una `E` para el último disco. De aquí podemos observar que las primeras dos letras `BA` se codificaron como `EE`, es decir, dos letras iguales. Esto es perfectamente válido, ya que todos las letras se codifican con caracteres diferentes a medida que rotan los discos.

Continuando con la ilustración, las siguientes dos letras se codifican como `C` y `A` respectivamente. Aquí tenemos otra observación, también puede pasar que una letra quede codificada en la misma letra que era originalmente. La codificación final de `BABA` entonces queda como `EECA`.

La decodificación es el proceso equivalente, pero en el sentido inverso. Partiendo de la máquina con los discos en la misma posición que a la hora de codificar, se parte desde el último disco y se realiza la transformación a la inversa. En el ejemplo, para la primera letra, esta se busca dentro del disco. Cuando se encuentra la letra `E`, se traduce a la letra desde donde se llego, que en este caso es la letra `C`. Luego se vuelve hacer el proceso para el siguiente disco y así sucesivamente hasta llegar a el disco número 1. La letra resultante será la decodificada.

La contraseña de codificación para la Máquina Enigma consistía en varias tablas con información de toda la configuración de la máquina, tanto del tablero de conexiones como también de la posición de los discos (estos eran intercambiables). En nuestro caso, a fin de simplificar el modelo, consideraremos la contraseña como solamente la posición de rotación de los discos. Está será configurada por medio de una función de nuestra máquina.



3. Tipos de datos: `littleEnigma`

A partir de las siguientes estructuras se define una `littleEnigma`:

```
struct littleEnigma {
    struct wheel** wheels;
    int wheelsCount;
};
```

```
struct wheel {
    struct letter* first;
    char* alphabet;
    int count;
};
```

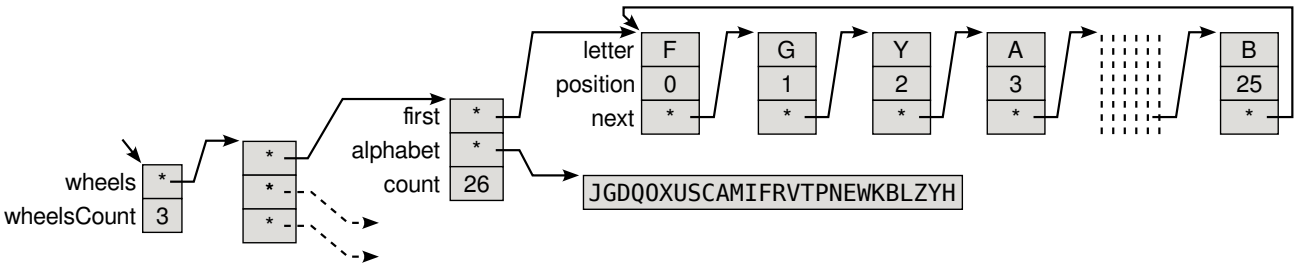
```
struct letter {
    char letter;
    int  position;
    struct letter* next;
};
```

La estructura `littleEnigma`, contiene un puntero a un arreglo de `wheels`, definido en la estructura como un doble puntero. Para conocer la cantidad de `wheels` que tiene la máquina se tiene el campo `wheelsCount`.

Una `wheel` es una estructura similar a una lista circular, donde cada nodo de la lista es una estructura `letter`. La estructura `wheel` contendrá una copia de la permutación del afabeto en el campo `alphabet` y un campo `count` con la cantidad de letras en este alfabeto. Además contendrá un puntero a la primer letra, indicado por el campo `first`.

Las letras por su parte serán los nodos de la lista circular. La estructura `letter` tendrá tres campos, el primero `letter`, para guardar la letra, el segundo `position` para indicar la posición de la letra dentro de la permutación, y el tercero como puntero a la letra siguiente denominado `next`.

A continuación se ilustra un ejemplo de la estructura:



En este caso se puede observar como de la primer estructura se apunta a un arreglo que contiene los punteros a las tres `wheels` definidas. Dentro de cada estructura `letter`, se tiene la letra, y la posición en la permutación del alfabeto.

4. Enunciado

Ejercicio 1

Implementar las siguientes funciones sobre *strings*.

- `int strLen(char* src)`
  Calcula la longitud de una *string* pasada por parámetro, indicando como respuesta la cantidad de caracteres de esta.
- `char* strDup(char* src)`
  Duplica un *string*. Debe contar la cantidad de caracteres totales de *src* y solicitar la memoria equivalente. Luego, debe copiar todos los caracteres a esta nueva área de memoria. Además, como valor de retorno se debe retornar el puntero al nuevo *string*. Esta función NO debe borrar la *string* original.

Ejercicio 2

Este ejercicio consiste en implementar funciones sobre el tipo de datos `wheel`. Para simplificar el desarrollo, se provee un conjunto de funciones útiles ya implementadas.

- `int letterToIndex(char letter)`
  Toma una letra y la convierte en un índice entre 0 y 26. Esta función se debe utilizar para determinar que elemento de la lista circular se debe usar para convertir las letras.
- `char indexToletter(int index)`
  Toma un indice y lo convierte al carácter correspondiente. Se utiliza para la decodificación de mensajes, cuando se debe hacer el proceso inverso.
- `void setWheel(struct wheel* w, int position)`
  Dada una estructura `wheel`, hace rotar la lista circular hasta llegar a la posición indicada por el parámetro. Esta función es utilizada para configurar la posición inicial de las ruedas, la que se utiliza como codificación inicial de la máquina.
- `void wheelPrint(struct wheel* w)`
  Permite imprimir en pantalla una estructura de tipo `wheel`.

Se pide entonces implementar las siguientes funciones:

- `struct wheel* makeWheelFromString(char* alphabetPermutation)`
  Esta función toma una *string* que contiene una permutación del alfabeto y retorna una estructura `wheel` creada en base a la permutación pasada por parámetro. Utiliza cada letra en el orden en que llegaron y arma una lista circular de nodos de tipo `letter`. Cada nodo de la lista circular contará con una letra y un indice o posición dentro del disco, numerado de 0 a la cantidad de letras del alfabeto.
  La estructura `wheel` apuntará a esta lista a partir del puntero `first`, mientras que el campo

`alphabet` tendrá una copia del alfabeto pasado por parámetro y el campo `count` será completado con la cantidad de letras totales del alfabeto.

- `void rotateWheel(struct wheel* w, int steps)`
  Mueve el puntero `first` de la lista la cantidad de `steps` pasados por parámetro. No retorna ningún valor, ya que modifica la estructura `w`.

- `void rotateWheels(struct wheel** w, int count)`
  Toma un arreglo de punteros a `wheels` y los hace girar con la siguiente lógica. Hace girar el primero, si este llego al final, entonces hace girar también hace girar el segundo. Si el segundo llego al final, también hace girar al tercero y así sucesivamente. Para determinar si se llego al final de una vuelta, se utiliza el campo `position` dentro de nodo `letter`. El campo `count` indica la cantidad de `wheels` pasadas por parámetro. Para resolver está función puede utilizar la función `rotateWheel`.

- `void wheelDelete(struct wheel* w)`
  Recorre la estructura de una `wheel` y borra toda la memoria solicitada por la misma. Tener en cuenta que se deben borrar los nodos de la lista y el alfabeto.

Ejercicio 3

Este ejercicio consiste en implementar funciones sobre el tipo de datos `wheel`. Para simplificar el desarrollo, se provee un conjunto de funciones útiles ya implementadas.

- `char encryptWheel(struct wheel* w, char letter)`
  Convierte la letra `letter` en la letra codificada por la estructura `wheel`. Para esto, convierte la letra en un indice usando `letterToIndex`, y obtiene la nueva letra recorriendo la lista y retornando la letra en la i-esima posición de la lista.

- `char decryptWheel(struct wheel* w, char letter)`
  Realiza el proceso inverso a `encryptWheel`. Para esto busca en la lista la letra pasada por parámetro y calcula su indice dentro de la lista. Este indice lo convierte nuevamente en una letra usando la función `indexToletter`.

- `void littleEnigmaPrint(struct littleEnigma* le)`
  Imprime en pantalla

Se pide entonces implementar las siguientes funciones:

- `struct littleEnigma* littleEnigmaNew(char** alphabetPermutation, int count)`
  Genera una nueva máquina en la estrucutura `littleEnigma`. Para esto recibe como parámetro arreglo de punteros a *strings*. Cada *string* corresponde a una permutación del alfabeto que será utilizada para contruir las estructuras *wheel* de la máquina. Para contruir las estructuras `wheel` se debe utilizar la función `makeWheelFromString`. El parámetro `count` indica la cantidad de elementos del arreglo.

- `void littleEnigmaSet(struct littleEnigma* le, int* positions)`
  Configura la posición de cada `wheel` de la máquina utilizando el arreglo de `positions` pasado por párametro. La configuración se debe realizar por medio de la función `setWheel`.

- `char* littleEnigmaEncrypt(struct littleEnigma* le, char* text)`
  Genera una nueva *string*, resultado de la codificación de la *string* pasada por parámetro. Esta última no debe ser modificada. Aplica a cada caracter de la función `encryptOneLetter`.

- `char* littleEnigmaDecrypt(struct littleEnigma* le, char* code)`
  Genera una nueva *string*, resultado de la decodificación de la *string* pasada por parámetro. Esta última no debe ser modificada. Aplica a cada caracter de la función `decryptOneLetter`.

- `void littleEnigmaDelete(struct littleEnigma* le)`
  Borra la estructura `littleEnigma` y todas las estructuras apuntadas. Para esto se debe utilizar la función `wheelDelete`.

- `char encryptOneLetter(struct littleEnigma* le, char letter)`
  Utilizando la función `encryptWheel` y `rotateWheels` se debe codificar una letra. Para esto se debe recorrer disco a disco transformando la letra de entrada. Luego, al finalizar, se debe incrementar en una posición los discos de la máquina.

- `char decryptOneLetter(struct littleEnigma* le, char letter)`
  Utilizando la función `decryptWheel` y `rotateWheels` se debe decodificar una letra. Para esto se debe recorrer disco a disco en el orden inverso, transformando la letra de entrada. Luego, al finalizar, se debe incrementar en una posición los discos de la máquina, al igual que en el proceso de codificación.

Ejercicio 4

A continuación, se enumera un conjunto mínimo de casos de test que deben implementar dentro del archivo `main`:

- `strLen`
  1. String vacío.
  2. String de un carácter.
  3. String que incluya todos los caracteres válidos distintos de cero.

- `strDup`
  1. String vacío.
  2. String de un carácter.
  3. String que incluya todos los caracteres válidos distintos de cero.

- `makeWheelFromString`
  1. Generar una estructura `wheel` con un alfabeto de 1 caracter.
  2. Generar una estructura `wheel` con un alfabeto de 10 caracteres.
  3. Generar una estructura `wheel` con un alfabeto de 26 caracteres.

- `rotateWheel`
  1. Rotar una estrucutura `wheel` de 26 caracteres una posición.
  2. Rotar una estrucutura `wheel` de 26 caracteres 26 posiciones.
  3. Rotar una estrucutura `wheel` de 5 caracteres 26 posiciones.
  4. Rotar una estrucutura `wheel` de 1 caracteres 26 posiciones.

- `littleEnigmaEncrypt` y `littleEnigmaDecrypt`
  1. Encriptar y desencriptar un mensaje de 0 caracteres.
  2. Encriptar y desencriptar un mensaje de 35 caracteres iguales con una máquina de un solo disco.
  3. Encriptar y desencriptar un mensaje de 10 caracteres con una máquina de tres discos.
  4. Encriptar y desencriptar un mensaje de 10 caracteres con una máquina de cinco discos.
  5. Encriptar y desencriptar un mensaje de 10 caracteres con una máquina de ocho discos.

Entregable

Para este trabajo práctico no deberán entregar un informe. Sin embargo, deben agregar comentarios en el código que expliquen su solución. No deben comentar qué es lo que hace cada una de las instrucciones sino cuáles son las ideas principales del código implementado y por qué resuelve cada uno de los problemas.
La entrega debe contar con el mismo contenido que fue dado para realizarlo más lo que ustedes hayan agregado, habiendo modificado **solamente** los archivos `utils.c` y `main.c`. Es requisito para aprobar entregar el código correctamente comentado.